INVENTORY MANAGEMENT DASHBOARD FOR TRACKING OF SPORTS EQUIPMENT AND FACILITIES IN A SECONDARY SCHOOL'S SPORT CENTRE

YAP RUI YA

UNIVERSITI TUNKU ABDUL RAHMAN

INVENTORY MANAGEMENT DASHBOARD FOR TRACKING OF SPORTS EQUIPMENT AND FACILITIES IN A SECONDARY SCHOOL'S SPORT CENTRE

YAP RUI YA

A project report submitted in partial fulfilment of the requirements for the award of Bachelor of Software Engineering (Honours)

Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

September 2025

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Name : YAP RUI YA

ID No. : 2107016

Date : 28/4/2025

COPYRIGHT STATEMENT

This final year project report is submitted in partial fulfilment of the requirements for the degree of Software Engineering at Universiti Tunku Abdul Rahman (UTAR). This final year project report represents the work of the author, except where due acknowledgement has been made in the text. No part of this final year project report may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ACKNOWLEDGEMENTS

I would like to extend my heartfelt appreciation to everyone who have contributed to accomplish the milestones of this project. This report was prepared as part of the academic requirements for the Lee Kong Chian Faculty of Engineering and Science at Tunku Abdul Rahman University (UTAR to help students complete their graduation studies.

I would like to express my gratitude to Ms Gunavathi a/p Duraisamy as my research supervisor for her valuable advice, sensible opinions, guidance and enormous patience across all of the research's timelines. In this case, the research would not have been completed as required within the set parameters without her careful guidance and suggestions.

I would also like to express my sincere gratitude to my parents, relatives and all the people who have support me in one-way, moral support and insights along the way. The respect and trust they placed in me motivates me to keep striving forward.

Lastly, I wish to show my appreciation for the volunteers who took part within the user acceptance testing. Their enthusiastic participation and insightful feedback were crucial to improve the overall quality of the project which greatly attributed to its success.

ABSTRACT

Manual inventory management in secondary school sports centres often results in misplaced equipment, overbooked facilities, and inefficient maintenance tracking. This project aims to address these issues by developing an Inventory Management Dashboard that digitalizes the management of sports equipment and facilities. The system integrates QR code-based tracking, real-time updates, and a centralized booking platform to enhance operational accuracy and accountability. Using a prototyping methodology, the system was iteratively designed, developed, and refined through continuous user feedback from stakeholders, including administrators, quartermasters, teachers, and students. The solution was implemented as a web-based application using React.js for the frontend, Laravel for the backend, and a MySQL database for persistent storage. Key features include role-based access control, QR code generation and scanning for equipment check-in and check-out, real-time inventory and facility booking, maintenance scheduling, and an analytics dashboard for performance insights and decision-making. The system underwent comprehensive unit testing, integration testing, and user acceptance testing, confirming its functionality, usability, and effectiveness in meeting user requirements. Results demonstrate significant improvements in inventory accuracy, booking transparency, and maintenance monitoring. The developed dashboard offers a scalable and user-friendly solution that reduces manual workload, minimizes errors, and promotes data-driven management of sports resources. Future improvements may include the integration of predictive maintenance, automated reporting, and multi-school scalability to broaden its impact across educational institutions.

Keywords: inventory management; sports facilities; QR code tracking; web application; React.js; Laravel; MySQL; prototyping methodology

Subject Area: QA76.76 Computer software

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 General Introduction

The main purpose of this project is to develop an Inventory Management Dashboard for tracking sports equipment and facilities in a secondary school's sports centre. This system will enhance the efficiency of inventory management for stakeholders to efficiently manage sports inventory by ensuring the availability and proper utilization of equipment and facilities. The project aims to replace traditional manual record-keeping methods, which have very low efficiency rates due to ineffective tracking and higher chances of error to a digital solution.

This dashboard will provide a centralized platform for monitoring inventory levels, tracking equipment usage, and scheduling maintenance activities. It will also enhance transparency, streamline operations, and improve resource allocation within the school's sports centre.

Thus, the project's background, problem statement, objectives, scope, suggested solution, and methodology will be covered in detail in Chapter 1.

1.2 Project Background

Nowadays, sports and other extracurricular activities are integrated into the educational process, which makes the management of sports facilities and equipment become more important. This cause Effective inventory management become more crucial in order to ensuring the availability and proper utilization of resources in various fields especially education and sports. However, many schools still rely on manual inventory tracking systems such as paper-based records and spreadsheets. These systems are prone to errors, inefficiencies, and difficulties in tracking equipment availability and maintenance needs. This outdated approach leads to operational inefficiencies, unaccounted losses, and delays in equipment allocation, ultimately affecting the quality of sports programs. Therefore, inventory management is an important function for any organization dealing with physical commodities to ensure the right quantity of products are available at the right time to avoid the problems of overstocking and understocking (Madamidola et al., 2024).

According to Macadamidola et al. (2024), the development of inventory management systems has greatly improved inventory tracking accuracy and operational efficiency, which has introduced modern technologies such as barcode scanning, Radio Frequency Identification (RFID), and the Internet of Things (IoT). Due to the continue increasing of efficiency demands, school have to turned into digital solutions to improve inventory tracking and overcome overstocking, understocking, and poor tracking accuracy problems. However, many schools have yet to adopt these innovations, the main reason is due to a lack of awareness or the absence of a customised solution designed for educational institutions.

The motivation of this project is to bridge the gap between traditional inventory tracking and modern digital solutions. The purpose of this project is to develop an **Inventory Management Dashboard for Tracking of Sports Equipment and Facilities in a Secondary School's Sport Centre** which will provide a centralized digital platform that allows secondary schools can effectively track their sports resources, monitor facility usage, and schedule maintenance activities.

1.3 Problem Statement

1.3.1 Inefficient Equipment Tracking System

Many secondary schools have adopted digital teaching and learning solutions, but sports equipment management is still largely neglected. The research shows that the current investigations into digital technologies within physical education are focused mainly on areas such as gamified teaching, wearable devices, and collaborative learning but fewer studies addressing resource and facility management (Jastrow, Greve, Thumel, Krieger, & Süßenbach, 2022). Ideally, the school sport centre will have a real-time and automated system to track all sport equipment so that each equipment can be clearly to be tracked and easily accessed to teachers and students. However, the sports centre still relies on manual tracking methods such as paper logs or spreadsheets to track sports equipment which are very time consuming and error prone.

Besides, the manual tracking method also makes it difficult for staff to locate equipment when needed as the equipment is often lost or damaged. This inefficiency can lead to communication issues and human errors which result in delayed events, frustration among students and staff, and increased costs due to lost or damaged equipment (Ko, Azambuja, & Lee, 2016). Without real-time visibility of equipment

availability, physical activity and training sessions are unnecessarily disrupted. Even though many secondary schools have integrated digital tools into their classrooms to improve learning, the adoption of digital solutions for managing sports facilities and equipment has been largely disregarded. To address this issue, an inventory management tracking system using QR codes can be implemented. This system will provide real-time updates on equipment availability which can significantly reduce manual tracking errors, improve inventory management, and create a smoother experience for both teachers and students.

1.3.2 Overbooking and Underutilization of Facilities

In Malaysia, the overall utilisation rate of sports facilities stands at approximately 46.9% of capacity has indicating significant underutilisation (Aman et al., 2020). The school sports centre should have a centralized scheduling system that ensures optimal utilization of all equipment. This system would prevent overbooking, underutilization, minimize conflicts, and ensure that every equipment is used efficiently. However, a lot of schools continue to use manual tracking method which can cause the scheduling disputes for teachers and students. For example, popular facilities such as basketball courts are often fully booked during peak hours, while other facilities such as tennis courts are not fully utilized. Even with sufficient sports facilities, these are often underutilised due to less awareness (Sadiq et al., 2023). The lack of structured inventory management system may increase the operating costs and resulting in a waste of resources.

Besides, the manual tracking methods makes it difficult to track historical booking data which may prevent the administrative from optimising the use of facilities based on demand. As a result, it makes the staffs and students difficult to plan their activities efficiently, and the last-minute changes or cancellations further disrupt the scheduling process. At the same time, it may also limit the potential for school training sessions and events. To solve this problem, a dashboard that provides real-time availability and booking status of facilities is required to optimize usage and improve user experience.

1.3.3 Lack of Maintenance Tracking

Inspecting sports equipment is important for preventing costly losses or serious damage. The most effective time for inspection is before beginning of each sporting season to

ensure that the equipment compiles with safety standards and protecting the users and spectators' safety when using it (Morrow, 2018). Sports equipment and facilities often require maintenance to ensure safety, longevity and optimal performance, but the current system fails to track maintenance schedules effectively. This leads to delayed repairs, the equipment is unsafe, and the resources have a shortened lifespan. The research shows that the rising of maintenance costs has driven the use of computerised models to enhance equipment utilisation and reduce expenditure in comparison to reactive or temporary maintenance procedure (Sayyed, 2015). Therefore, a sports centre should have a proactive maintenance tracking system that can monitor maintenance schedules and tracks the condition of equipment and facilities. This advanced system would mitigate the chances of breakdown disasters, enhance safety, and save money on expensive repairs in the future.

The use of manual tracking method may pose major risks because facilities and equipment can be ignored until they reach a critical failure point. Okirie AJ, Barnabas M, Adagbon JE (2024) mentioned that the manual tracking can result in ineffective maintenance resource allocation, decreased equipment reliability, and higher maintenance expenses. This negative approach puts the health and safety of the students and staff at risk by preventing physical activity and simultaneously increases operating costs through emergency repairs or premature replacement of equipment. Moreover, the lack of a structured maintenance tracking system also means that facilities staff have a lack of knowledge about the condition of sports resources, making it difficult to plan for necessary repairs or replacements. The absence of essential safety inspection systems, equipment management protocols and regular maintenance programmes may compromise safety and accelerate equipment deterioration (Wu, Lu and Ma, 2025).

To address these challenges, a module that tracks maintenance tasks and ensures regular inspection and upkeep of sporting equipment must be implemented to assist facilities and expand their capabilities. Preventive maintenance is better than reactive maintenance. This is because preventive maintenance plays important role in facility upkeep which help to prevent unexpected equipment failures and costly repairs and extends the service life of assets and system (Hawkes, 2025). In addition, the integration of QR codes on each piece of equipment will allow staff to quickly check maintenance status and report issues instantly to ensure a safe sporting environment for students and staff.

1.3.4 Poor Reporting and Decision-Making

Effective management of sports centres requires accurate data on equipment usage, facility bookings and maintenance activities. However, allocation of resources and budgeting for repairs, equipment purchases, and other activities is challenging for staff as they lack reliable reports that are based on analysis of usage trends. In sports management, big data and analytics have become essential in guiding decision-makers because they allow interpreting large datasets to drive operational strategies (Watanabe et al., 2021). Therefore, sports centres need to be equipped with holistic reporting and analytics systems that generate real-time insights to support data-driven decision-making and improve overall efficiency.

Besides, data recorded in manually may be inconsistently and leading to errors and incomplete information. Without a clear understanding of how often equipment is used, which facilities are in high demand or when maintenance is required, decisions tend to be made on assumptions instead of facts. Such inefficiencies can lead to misallocation of resources, unnecessary expenditure and lack of improvement in the functioning of the sports centres. Therefore, a reporting and analysis module should be incorporated into the inventory management system. The module will provide detailed analysis of the equipment and other resources used, facilities booked, and maintenance done, and provide real-time accurate data to assist administrators in decision making and resource allocation. According to Peter Drucker's well-known quote, "What gets measured gets managed,", data allows system to make better decisions and promote efficient decision (7 Ways Data Can Drive Better Facilities Management Decisions, 2025). By integrating data visualisation tools such as interactive dashboards and trend analysis charts, the system will be able to make better decisions by identifying peak usage times and forecasting equipment replacement needs. In addition, automated report generation will streamline administrative tasks, reduce manual workload and improve operational efficiency. The management in the sports centres will make better decisions in the facility's utilization to increase sustainability and user satisfaction long term with such a system in place.

1.4 Aim and Objectives

This project intends to achieve the following objectives:

1. To develop a web application with dashboard on tracking of the movement and inventory of the sport equipment and facilities.
2. To develop equipment tracking and maintenance features using QR code.
3. To develop a mobile application with QR code scanning feature for the equipment whenever there's an in & out movement from the sport center.
4. To evaluate the effectiveness proposed system by conducting the user acceptance testing with the selected school.

1.5 Project Scope

This project aims to develop a **web-based Inventory Management Dashboard and mobile app for scanning** to streamline the management of sports equipment and facilities at a secondary school's sports centre. The system will centralize equipment tracking, automate facility scheduling, enable proactive maintenance management, and provide data-driven insights for decision-making.

1.5.1 Target User

The intended users of this system are:

- **Teachers & Students**: Primary users who can rent sports equipment.
- **Administrators**: Authorized personnel responsible for managing sports equipment and overall operations within the school's sports.
- **Quarter masters**: The authorized personnels (students) who are responsible for managing the inventory, do stock check, tracking the equipment's in and out, and manage the booking.

1.5.1.1 Functionality for Teachers & Students

1.5.1.1.1 Equipment Reservation & Rental

Teachers and students can view the availability of sports equipment in real time through a user-friendly interface. This feature ensures that users can quickly determine which equipment us available at any given time. Users can also reserve equipment for specific time slots. For example, during class hours or extracurricular activities. This ensures that users have all the equipment they needed for their planned activities. Besides, users can track the equipment when equipment was borrowed and view the return dates through booking history.

1.5.1.1.2 Equipment Return & Notifications

Users can mark equipment as “check in” or “check out” in the system and the system will updating its availability status automatically once the users have finished using the booked equipment. This feature ensures that equipment is returned to the pool of available equipment in a timely manner and preventing overbooking. In addition, the system will automatically send reminders to notify users about the upcoming due dates and alert them when the equipment is overdue. This notification will be sent in advance so that users have sufficient time to return the equipment.

1.5.1.1.3 QR Code scanning for reservation and booking

Users can scan the QR code by using their mobile phone to make reservations and booking. After scanning, users need to login or sign up to the website. After login, it will navigate to specific equipment page to allow users to “check in” or “check out”. The system will update the availability of the equipment immediately.

1.5.1.2 Functionality for Administrators and Quarter masters

1.5.1.2.1 Inventory Management

Administrators and Quarter masters have full control over the inventory management of sports equipment. They can add, view, update and delete equipment details such as the condition and quantity of equipment. This feature ensures that inventory always up-to-date and avoid discrepancies or confusion over equipment availability. Administrators can also update the equipment records by using QR code scanning which making the inventory management become faster and efficient. Besides,

administrators can track and change the equipment status in real time. The status can be categorized as ‘available’, ‘rented’ or ‘under maintenance’ to allow users to view information.

1.5.1.2.2 Role-Based Access Control

Administrators can assign specific permissions to different users to ensure only authorised user can access to certain features. For example, only administrators can limit the amount of equipment booking and view the analysed result displayed on the dashboard, while other users without the permission will only have access to basic functions which is reserving and returning the equipment. This ensures that only appropriate administrative privileges have access to sensitive data on equipment usage, booking trends and overall system performance in order to avoid misuse.

1.5.1.2.3 User Access Control

Administrators can assign specific roles and permissions to users as needed. For example, teachers can prioritize the use of certain equipment and extend the rental time based on the teaching and activities needs, while students will be restricted on maximum rental time to ensure that equipment are returned on time and available for others to use.

1.5.1.2.4 QR code generation

Only administrators can generate QR code for each category of equipment or facility. By using QR code, it can direct users to a centralized system page which can reduce manual process that may cause mistakes made by human errors. Sharing QR codes for grouped items can simplify management and allow more accurate tracking of equipment bookings.

1.5.2 Project Out-of-Scope

There are some features are outside the scope of the project. Firstly, the system does not include the financial tracking or budget management for the purchase of new equipment. This is because the main focus of the project is on inventory tracking rather than financial management. Secondly, the system does not support automated

assessment or predictive maintenance as the inspection and maintenance decisions still require manual intervention. Thirdly, the project was only limited to secondary school sport centre and was not designed for broader multi-school or district-wide use. Lastly, the system will only rely on QR code scanning with the existing mobile devices without using the hardware such as barcode scanner. It will also not track the equipment location via GPS.

1.6 Project Solution

The project is focus on developing an inventory management dashboard tailored for secondary school's sport centre. The objective of the project is to address the inefficiencies in equipment tracking, facilities reservation and maintenance management. The system will combine with the proven technology with innovative features to ensure scalability, real time updates and user-centered design.

The frontend of the inventory management dashboard will be developed using React.js, a JavaScript framework known for its flexibility and real-time rendering capabilities. React's component-based architecture facilitates the creation of reusable UI components, thereby enhancing the maintainability and scalability of the code (Gackenheimer, 2015). However, Visual Studio Code (VS Code) will serve as the primary integrated development environment (IDE) in order to provide a strong ecosystem of extensions to streamline the development process. Besides, Axure Rp will be used to create system prototype. It is a tool that can create wireframes, models and interactive prototypes without writing a line of code and user can simulate complex user interactions and interface behaviour before development work starts (Krahenbuhl, 2015).

The backend of the inventory management system will use MySQL, an open-source relational database management system (RDBMS) to store and manage data (Erickson, 2024). It will use Node.js to handle API request, authentication and others. Firebase Realtime Database will store device details and facilities schedules in order to provide real-time synchronization to avoid overbooking and ensure accurate inventory tracking. It is a cloud-hosted NoSQL database that allows administrators to store and transfer data across users in real-time to make sure that users have access to up-to-date information (Firebase, 2025). Firebase which contains authentication features will also enables secure role-based access control. This will result in the different between users and administrators' permission.

1.7 Project Approach

The project will use an iterative, user-centred development methodology combined with modern full-stack technologies to address the inefficiency of tracking sports equipment in secondary school sports centres. Prototyping methodology will be used for this project in order to solve the inefficiency of the tracking sports equipment in secondary school sports centers. Prototyping methodology is selected for this project because it allows early development of working system models, gathering continuous user feedback and iterative improvement of features. Therefore, it makes it suitable for this project where the requirements are changing during the development process.

The project will be divided into several sprints and each of them will focus on specific functionality. For example, equipment booking, inventory updates, user access control and so on. This will allow for iterative improvements and early testing of individual components. The Agile methodology will start with requirements gathering and analysis where the pain points will be identified through surveys. So that, it will ensure that the system aligns with user expectations and meet school requirements. Besides, prototyping and design validation will be carried out by using Axure RP to produce interactive wireframes that simulated key functionality. Users should take part in usability testing to make sure that the design was user-friendly and meet their requirements. According to Camburn et al. (2017), prototyping can clarify an ambiguous or changing requirement through iterative feedback between users and developers to ensure that the final system is aligned with the user needs.

The project will start with requirements gathering and analysis. At this phase, the problems and requirements will be identified by using surveys and interviews to solve the issues faced by the users. Based on the collected requirements, an early low-fidelity prototype will be created using Axure RP which focus on basic layout, user flows and essential functionality. This prototype will be shared with users to gather usability feedback at an early stage.

After the early feedback is collected, the prototype will be refined and improved over multiple cycles. Each version will add more detailed functionality and gradually leading to high-fidelity model that closely matches to the final system. After each iteration, users will involve in usability testing to verify the design, test navigation flows and provide suggestions for improvements.

Throughout the process, functional feedback will be collected after each prototype evaluation phase to identify usability issues and improve the system design.

Each iteration will be tested for early detection, and the issues will be solved to ensure that the final product is reliable, intuitive and user-friendly. As Horváthová and Voštinár mentioned that feedback is important for improving system performance as it allows users to learn from their mistakes.

Lastly, the project will ensure that the developed inventory management control panel is user-centred and flexible to respond to changing requirements and able to solve the inefficiencies of manual sports equipment tracking effectively by using prototyping methodology in order to deliver a high-quality functional system that meets the needs of users.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter reviews the existing literature on developing an inventory management system for tracking secondary school sports equipment and facilities. It starts with outlining the current scope and issues that emphasize the limitations of manual inventory management systems and the need for proposed solutions. It also explores the Software Development Life Cycle (SDLC) to outline systematic processes and ensure it is aligned with industry standards and enhance project efficiency. The chapter also covers the relevant techniques and approaches to evaluate the effectiveness of the system. Besides, the chapter studies the development tools and system architectures commonly used in similar projects. Lastly, this chapter will examined systems and compared it with similar functionality to understand current trends, capabilities, and limitations. These elements provide a solid theoretical and practical foundation for the creation of the proposed system.

2.2 Challenges of Manual Inventory Management

Secondary schools use inefficient manual systems to track and manual sports equipment and facilities which often leading to misplaced items, maintenance delays and poor budget management. According to a report by AssetPanda, it has mentioned that educational institutions has lose up to $250,000 (RM 1,101,874.46) per year due to improper asset tracking. A study by Link Labs also found that 5 to 10% of recorded assets are “ghost assets” which are assets still recorded but are actually missing or unusable. This problem occurs is due to poor tracking or record keeping that always rely on manual inventory management methods.

2.3 Software Development Methodology (SDLC)

2.3.1 Overview of SDLC

Software Development Life Cycle (SDLC) is the structured process that guides the development of a software system. It consists of 6 phases which is planning, defining, designing, building, testing and deployment.

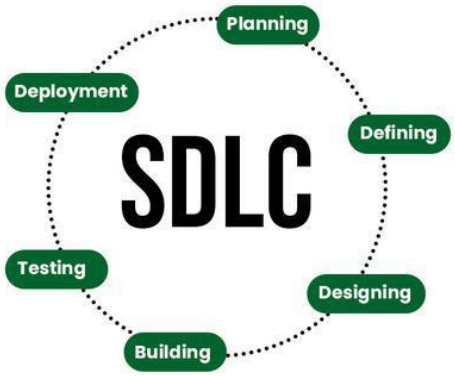


Figure 2.1: Software Development Life Cycle (SDLC) (GeeksforGeeks, 2020)

SDLC can reduce the risk of project failure as it will ensure that the system development is systematic and meet the user requirement (Jindal, Gulati & Rohilla, 2015). According to Gurung et al. (2020), each phase in SDLC is critical to produce high-quality software systems and provides opportunities for verification and validation. Furthermore, Kute and Thorat (2014) mentioned that using the correct SDLC can increase clarity of project scope, improve communication between stakeholders and better managing schedules and cost. Therefore, SDLC is important as it determines how the projects adapt to change, meet the customer expectations and maintains quality assurance at the same time not overbudget.

2.3.2 Waterfall Development Methodology

The waterfall methodology is a traditional and linear software development methodology where each phase must be fully completed before progressing to the next. The idea of the "waterfall" of development activities was first proposed by Royce (1970, cited in Bell and Thayer, 1976) which emphasized the structured, top-down nature of the progression of each distinct development phase, where the output of one phase becomes the input to the next. It follows a linear and sequential path where each phase must be completed before the next phase start. It contains six sequential phases: Requirements Gathering & Analysis, System Design, Implementation, Testing, Deployment, and Maintenance.

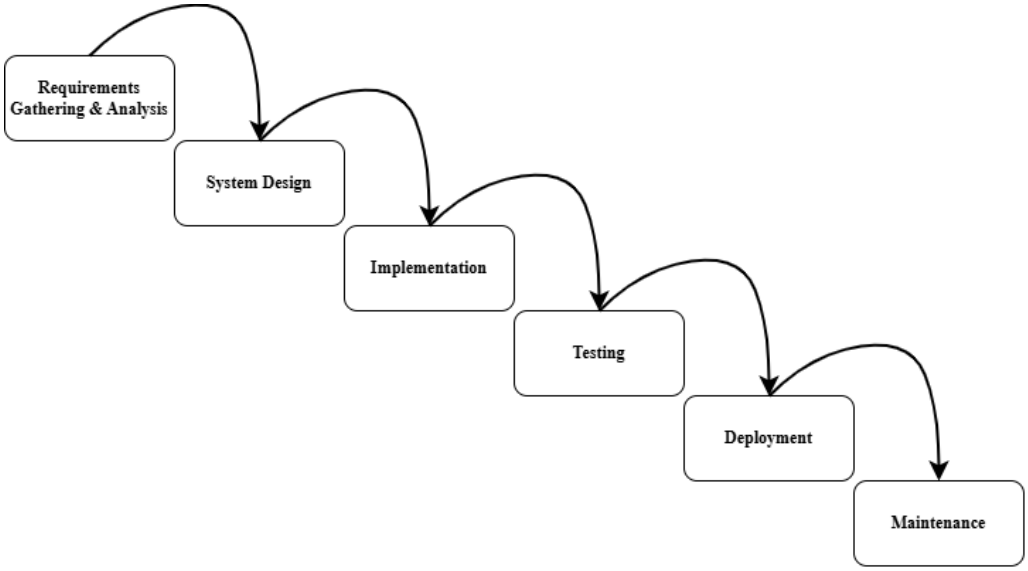


Figure 2.2: Waterfall Model

The first and primary step of the waterfall approach is Requirements Gathering and Analysis phase. All the functional and non-functional requirements of system will be collected in this phase in order to gain a comprehensive understanding of the intended functionality of the system. Functional requirements are defined as use cases that explain the user's interactions with the system and contain aspects such as the system's purpose, scope, functionality, interface requirements, and database requirements while the non-functional requirements are defined as a constraints and quality features such as dependability, scalability, testability, and performance (Bassil, 2012). At this phase, the detailed insights will be gathered through conducting interviews, surveys or workshops and then recorded in Software Requirements Specification (SRS). Software Requirements Specification (SRS) is a detailed explanation about the development of software behaviour (Bassil, 2012). According to Chemuturi (2013), a document will not be 'frozen' before it gets authorization by appropriate body and sometimes such approval is obtained after a rigorous internal review and quality control process. It means that once the SRS has been reviewed and approve, the requirements are considered as 'frozen' which means there is no further modification can be made. Therefore, at this phase would outline the features like QR-code scanning and role-based access for sports equipment.

The system design phase takes place after the requirements specifications has been completed. Software design is a unique and critical step that separates requirements from coding and helps to ensure that development proceeds in a structured

and systematic manner (Royce, 1987). At this phase, software developers and designers require to create a solution proposal such as database design, user interface layout, data model and data structure. They also require creating logical and physical design diagrams to visualise system components and their interactions such as Entity Relationship Diagrams (ERDs) and Unified Modelling Language diagrams (UMLs). The ERDs has been preferred to use in conceptual modelling due to it "easy to understand, effectively models real-world problems, and is easily converted into database schemas." (Song, Evans, & Park, 1995). However, UMLs will provide standardized approach to modelling the structure and behaviour of a system which will improve communication among teams and help them more clearly specify, visualize and record system documents (Visual Paradigm, 2019).

The implementation phase is a phase that transforms the system design specification, blueprints and business requirements into an actual environment by using the programming language and development tools (Bassil, 2012). Therefore, developers will use a modular programming approach to implement components that defined in the design phase and implement functionality outlined in the system blueprint. Unit testing also plays an important role in verifying the functionality of individual components during the implementation phase. It usually carried out concurrently with development to ensure the software quality and detect the problem early. It uses automated tools to effectively detect and address issues early in the implementation phase to maintain the code quality and prevent downstream defects (TestFyra, 2023).

Besides, the testing phase is a process that will completely validate the system once the system is built to ensure that it meets all the requirements and has no bugs. It is also a process of confirming that a software solution satisfies the initial requirements and specifications and achieves its intended purpose (Bassil, 2012). During this phase, both unit testing and system testing are carried out. Unit testing checks individual components to verify that each unit perform as expected (GeeksforGeeks, 2019). While system testing evaluates the entire workflow to ensure that entire system integrate smootly (GeeksforGeeks, 2019). For example, testing that if the inventory updates properly when a student checks out the equipment. This step is important as it must identify and resolve any issues before the system is deployed into live use.

Maintenance phase is a last important step in waterfall methodology as it ensures the performance and long-term functioning of the system. It is the process that

resolving any problems or bugs reported by user and ensuring that the system remains functioning and meets the requirements of users. It may also have some additional maintenance such as strengthening software stability, meeting new user requirements, and adapting the software to the environment (Bassil, 2012). Therefore, maintenance phase supports long-term success by ensuring that the system remains up-to-date, reliable and meets user expectations.

According to the above explanations, it shows that the projects with clear, well-defined requirements that are unlikely to change during development are suitable for the waterfall methodology. Therefore, it is ideal for the projects with predictable scope and tight deadlines or budgets. This is because the water methodology is a linear progression where each phase must be completed before moving on to the phase and there is no overlap between the phases (Senarath, 2021). Therefore, waterfall methodology provides a structured approach that ensures that all specifications are fully documented and locked down early in order to minimize the changing risk during the project lifecycle. If project has fixed project scope and non-negotiable requirements such as data privacy laws, the waterfall methodology is a reliable choice.

Table 2.1: Advantage and Disadvantage of Waterfall Methodology

Advantages	**Disadvantages**
Simple and Structured: Waterfall is easy to understand and implement as it is a clear and step-by step process.Each phase has clearly defined milestones which makes it easier to monitor progress and ensure the project is completed on time.	Inflexible to Changes: Difficult to make changes once a phase is completed
Detailed documentation: Detailed deliverables for each phase (e.g., SRS, design documents) to ensure that the stakeholders have a clear understanding of the project scope, requirements and objectives.	Late Testing: Bugs are often discovered only after coding is complete.
Predictability: Helps maintain control of monitoring projects when costs,	High Risk: Waterfall assumes all requirements are understood from the

timelines, and resources are defined at the beginning of the project.	start, which make it difficult to predict all requirement in the early stage as the requirement may change frequently.
Minimal customer involvement: Stakeholders are involved only during the requirements phases which can simplify communication and avoid overwhelming stakeholders.	Longer time to market: The linear nature of Waterfall may slow down delivery when a project needs to be delivered more quickly.
Clear Milestones: Projects progress through different phases (requirements, design, coding, testing, deployment) that provide clear checkpoints and approvals can help teams focus on delivering one phase before moving on to the next and reducing the risk of scope creep.	Limited Flexibility: Difficult to adapt to new requirements or changes.

2.3.3 Agile Development Methodology

In 2001, a lightweight and adaptive development methods were born and flourished after gathering of 17 software engineering experts This meeting resulted in the Agile Manifesto which outlines the core values and guides the principles that is used to improve software agility and system development (Al-Saqqa, Sawalha & AbdelNabi, 2020). Fundamentally, Agile emphasizes individual and interactions over processes and tools as it recognizes that successful collaboration and communication are more important to success than rigid workflows. It emphasizes working software over comprehensive documentation that highlights the prioritization of functional software over extensive paperwork (Apke, 2016). Agile methodology also promotes collaboration with customers rather than contract negotiation which encourages customers to provide ongoing feedback to ensure the product meets the changing needs. Lastly, it promotes reacting to change rather than following a fixed plan which allow the developers to adjust according to shifting project needs. These values are realized through various Agile frameworks such as Scrum, Kanban and Extreme Programming

(XP) and each of them may provide tools and practices to help developer teams implement Agile values on real projects effectively.

Scrum is the most widely used Agile framework that divides the development into iterative cycles called sprints which usually last 2-4 weeks. The process will start with the project vision which sets the overall goal. Next, the sprint backlog is created to prioritize a list of all required features and fixes in the form of user stories. For example, “As a user, I want to scan QR codes to book equipment, so that I can reserve it quickly without manual check-in”. Therefore, the sprint backlog outlines the tasks selected for the sprint and lists out all the requirements that the developer will concentrate on during the development cycle (Srivastava, Bhardwaj & Saraswat, 2017).

Before start of each sprint, the developers will conduct sprint planning meetings to decide what backlog items will be completed during that sprint. Each sprint day starts with daily standup meetings where the team members should answer questions: What work can be accomplished in this sprint and how will the selected work be accomplished? Daily standup meetings offer a chance to share important information which may facilitate continuous improvement. Through meetings, teams can enhanced communication as everyone knows what everyone else is working on, which avoids duplication and misunderstandings.

During sprint execution, the developers design, code, and test selected features. At the end of the sprint, they hold a sprint review to show the working product to stakeholders and get feedback. After that, they conduct a sprint retrospective to reflect on what went well and what could be improved in the next sprint. This methodology repeats in cycles which allow the developers to constantly adapt and improve the product based on feedback and changing needs.

Agile methodology also involves several key roles to ensure the process runs smoothly. The product owner represents the stakeholder and is responsible for managing and prioritising the product backlog which ensure that the developers remain focused on delivering the product with the most valuable features. The Scrum Master is responsible for driving the Agile process which enabled the developers follow the Scrum practices and removing any barriers that may be blocking progress. Development teams are also one of the key roles which are cross-functional and that consisting of developers, testers, and designers who collaborate to build and deliver working increments of the product.

In conclusion, Agile methodology is best suited for projects where requirements are unclear or may change over time. For examples, school staff requesting new features when developing a sports centre dashboard. It is also well suited to complex projects that require frequent feedback and fast-paced environments that require rapid incremental delivery of functional software. Below is the advantages and disadvantages of Agile methodology.

Table 2.2: Advantage and Disadvantage of Agile Methodology

Advantages	Disadvantages
Flexibility: Agile accommodates changing requirements even late in the project.	High Customer Involvement Required: Frequent interactions can be time-consuming and demanding for users.
Faster Value Delivery: Working software is delivered in short, regular intervals.	Unpredictable Timelines: Evolving scope can cause delays or shifting deadlines.
Early and Continuous Testing: Bugs are identified and fixed early through regular testing.	Less Emphasis on Documentation: May lead to confusion if team members change or details are unclear.
High Stakeholder Involvement: Regular feedback ensures the product aligns with user needs.	Scope Creep Risk: Without strong control, frequent changes can lead to uncontrolled growth of features.
Increased Transparency and Visibility: Progress is tracked through sprints and reviews.	Not Ideal for Fixed-Requirement Projects: May not be suitable where full specifications must be defined upfront.

2.3.4 Prototyping Development Methodology

Prototyping is a development methodology that involves building a simplified working model of a system or specific feature in order to better understand and improve the final product. Prototyping is really useful for the project with unclear or changing requirements as it is very helpful to clarify expectations early in the process. Prototyping also allows developers and stakeholders to test usability, explore design concept and gather valuable feedback before moving to full development. The process can also expose potential defects or usability issues that may not be obvious from

documentation itself. For example, the prototype can use clickable mock-up interface that allows user can interact with it to test its usability and provide suggestions before the actual system is built. As Camburn et al. (2017) mentioned that prototyping plays an important role in validating requirements, revealing critical design issues and identifying design changes that enhance performance.

The prototyping process is a structured approach for building and refining early models of a system to clarify requirements and improve the final product. According to Camburn et al. (2017), prototyping helps to clarify ambiguous or changing requirement through iterative feedback between users and developers. The prototype methodology starts with requirements gathering phase. In this phase the developers and designers will collaborate with stakeholders to determine the core requirements of the system even the requirements are not yet complete or still changing. After that, the developers build a simplified version of the system which include low-fidelity prototypes like paper sketches or wireframes that focus on layout and navigation flow, or high fidelity interactive digital models that more accurately represent user experience and visual design.

In the third phase, prototype evaluation will take place which involves testing the prototype with real users to observe how the users interact with the prototype and identify usability problems. The fourth phase is refinement phase. In this stage, the developers will modify the prototype based on user feedback. This may involve moving unclear sections, improving the design or adding useful functionality. If the significant issues remain, a new prototype iteration is made for more testing in the fifth phase which called iterate or continue. The process will move to full development when the stakeholders are satisfied. This iterative process helps to ensure that the final product is both functional and user-friendly.

Prototyping is more useful in the projects where the requirements are unclear and changing. This is due to the stakeholders are difficult to accurately express their needs for the system. It is also suitable for the system that involve complex user interactions such as the equipment check-out or facility booking flows in a sports centre dashboard. Early testing of these process helps the developers able to ensure a smooth and intuitive user experience. Prototyping is very beneficial in high-risk projects where the errors could be expensive during complete development. This is because it allows developers to identify and resolve potential issues early. In addition, prototyping

supports the creation of user-centered systems by involving actual users in the design process and ensuring that the product meets their expectations.

Table 2.3: Advantage and Disadvantage of Prototyping Methodology

Advantages	**Disadvantages**
Early Feedback: Users can identify usability or functionality issues before development.	Scope Creep: Continuous user feedback may lead to never-ending changes and feature requests.
Reduced Risk – Helps catch potential problems early, reducing the chance of expensive fixes later.	Time or Cost Overhead: Creating and refining multiple prototypes can slow the development timeline.
Enhanced Stakeholder Understanding: Prototypes help stakeholders visualize the system, making it easier to communicate ideas and confirm requirements.	Misleading Expectations: Users might assume the prototype represents the final, complete system.
Requirements Validation: Help to clarify ambiguous or misunderstood requirements.	Technical Limitations: Prototypes might not reflect real-world performance or security concerns.
Improved Communication: Enhances collaboration between users and developers through tangible examples.	Limited Functionality: Some important backend or integration elements might be left out.

2.3.5 Comparison of the Evaluated Development Methodologies

Table 2.4: Comparisons Between Different Methodologies

	Waterfall Methodology	**Agile Methodology**	**Prototyping Methodology**
Structure	Linear and sequential.	Iterative and incremental.	Iterate using early simplified models.
Flexibility	Low – difficult to change the requirements once development starts	High – allow to change even late in the process.	Moderate – changes can be made after every iteration.
Customer Involvement	Minimal – mainly during requirements and delivery phases	Continuous – regular feedback through sprints and reviews	High – users interact with prototypes to provide early feedback
Documentation	Extensive – detailed documentation at each phase	Minimal – focuses on working software over comprehensive documentation	Varies – documentation may be limited, focusing on the prototype itself
Risk Management	Identifies risks early but addresses them late in the process	Continuous evaluation and adjustments of risks throughout the project	Identify usability and design issues early through user interaction with prototypes
Best Suited For	Projects with well-defined requirements and scope	Projects with evolving requirements and the need for frequent feedback	Projects with unclear requirements or

			complex user interactions
Delivery Timeline	Long – delivers at the end of the project	Short – deliver functional software periodically.	Varies – depends on the number of iterations and refinements
Cost Implications	High	Low	Variable

2.3.6 Conclusion of Methodology

The Prototyping Methodology is the most suitable for the developers who are working alone on a sports equipment management system. Its iterative nature allows rapid development and improves the system based on continuous feedback. Before the formal development, the concepts can be tested, usability problems can be found, and requirements can be clarified by creating early prototypes. It also can reduce the possibility of expensive errors and ensure that the final product is aligned with the user needs. Prototypes methodology also supports changes in requirements and priorities which enable for flexibility and quickly modifications during development.

The waterfall methodology is not suitable for this project is because it is a rigid and linear approach that requires each phase to be fully completed before moving on to the next phase. This makes it difficult to adjust once the development process is start. The lack of flexibility in the waterfall methodology also causes usability issues or mistakes in design cannot be solve in a right way as the feedback and changes can only be made after the project is completed. Besides, all requirements must have a clearly understanding when using waterfall methodology are not suitable for the projects that require user feedback and always changing requirements.

In addition, agile methodology is not suitable for this project as it requires more resources and collaboration with others which may be challenge when the project is lacks consistent stakeholder involvement or has limited resources. This is because Agile methodology is relied on continuous communication, regular feedback, and active collaboration between developers and users throughout the development process. If developers work alone or have few opportunities for frequent reviews with stakeholders, it may become difficult to implement agile practices such as sprint

reviews and backlog refinement. It may also be a challenge for independent developers to work without a team and may cause scope creep, unclear deadlines, and inconsistent progress when not managed well. This is because an individual developer is hard to manage multiple iterations and respond to changing requirements.

2.4 Web Application Framework

2.4.1 React Native

React Native is a popular open-source framework developed by Meta (formerly Facebook) that allows developers to build mobile applications using JavaScript and React. Unlike traditional native development which requires separate code bases for iOS and Android, React Native enables cross-platform development with a single code base. React has evolved to support web development through tools like React Native for Web which allows developers to use a unified code base across mobile and web platforms. React Native is a powerful solution for creating high quality and responsive mobile or web apps. This is because it can access to native APIs and many community-supported libraries. The performance and user experience are very similar with the fully native app as it uses native components rather than web views. Therefore, its component-based architecture that fosters code reusability making it become an effective choice for developers who want to build scalable, maintainable mobile or web applications as it provides easy maintenance, consistent user interface, and faster development speed.

React Native has many advantages. One of its main advantages is code reusability which allows developers to write single code base for both web and mobile applications. React Native allows developers to reuse up to 90% of the code between iOS and Android platforms (Leed Software Development, 2024). This result in the decreases in the development time and effort. It also provides a smoother user experience by using native components to deliver near-native performance. Besides, React Native has a rich ecosystem and strong community support that provides a variety of libraries, plugins and tools to accelerate development and simplify tasks such as navigation and API integration. It also supports hot reloading which can increases the productivity by enabling developers to observe the changes without the need to rebuild the entire program (Leed Software Development, 2024).

Even though React Native has many advantages, it also has some disadvantages. One of that main limitation is its performance limitations for complex applications that require intensive calculations or complex animations. The responsiveness of the application may be affected by latency as the framework relies on JavaScript bridge to communicate with native modules (Singh, 2023). Furthermore, React Native rely on third-party libraries which can lead to compatibility issues and security vulnerabilities if these libraries are not maintain regularly. The framework also does not have full access to native API's (Leed Software Development, 2024). This result in the custom native modules need to be developed in order to implement some specific features but these may increase the complexity to the development process. Due to the interaction between JavaScript and native code, it causes debugging React Native become more challenges than fully native apps as developers must expertise in both areas.

Table 2.5: Advantage and Disadvantage of React Native

Advantages	**Disadvantages**
Code can be reused across web and mobile apps.	Not so effective for apps that require heavy processing or complex animations.
Saves time by sharing up to 90% of code between platforms.	Latency may occur due to communication through a JavaScript bridge.
Provides near-native performance using native components.	Relies on third-party libraries, which may cause security or compatibility issues.
Has many libraries, tools, and a strong developer community.	Some native features require custom modules, which adds complexity.
Hot reloading speeds up development by showing changes instantly.	Debugging is more difficult and requires knowledge of both JavaScript and native code.

2.4.2 Vue.js

Vue.js is an incremental JavaScript framework that is used to create user interface. Due to its incremental nature, developers can gradually integrate it into projects. Vue.js is focuses on view layer which make it easy to integrate with other libraries or existing

projects. Its core library is lightweight and provides responsive data binding and component-based architecture which similar to frameworks like React and Angular.

Vue.js has many advantages. One of the main advantages is its beginner-friendly and has a mild learning curve for those who are familiar with HTML, CSS, and JavaScript (Johnson, 2023). It is because it provides a declarative and component-based component that extends the use of standard HTML, CSS, and JavaScript (Vue.js, no date). Its two-way data binding makes it easy to synchronize between models and views. Vue.js also provides documentation that makes the developers to easily get started and solve problems quickly. The component-based architecture improves code reusability, ensure faster load times and better performance. Its flexibility also makes it suitable for any size of the project from small to large (Epifany Bojanowska, 2018).

Besides, Vue.js has some disadvantages. One of the main disadvantages is it has smaller community and ecosystem than React or Angular. Therefore, it has less third-party tools, libraries, and job opportunities.Vue.js is difficult to extend as it may become more complex for larger projects that do not have strong architectural guidance. Additionally, plugins or support for enterprise-level features can be inconsistent if Vue.js is under development. Lastly, the over-flexibility can lead to differences in coding styles within a team if the strict standards are not followed (Johnson, 2023).

Table 2.6: Advantage and Disadvantage of Vue.js

Advantages	**Disadvantages**
Beginner-friendly with a gentle learning curve.	Smaller community compared to React and Angular.
Uses standard HTML, CSS, and JavaScript with a declarative approach.	Fewer third-party tools and libraries available.
Two-way data binding simplifies model-view synchronization.	Difficult to scale for large projects with poor structure.
Detailed documentation helps developers get started quickly.	Can be complex to maintain in large applications.
Flexible and suitable for both small and large projects.	

2.4.3 Laravel

Laravel is a popular open-source PHP web application framework known for its elegant syntax, powerful features and developer-friendly tools. It follows the Model-View-Controller (MVC) architectural pattern and is designed to improve routine operations such as routing, authentication, sessions, and caching (Neelam Menariya, 2022). As Laaziri et al. (2019) mentioned that Laravel can avoid the common mistake of "spaghetti code" by developing the PHP code in neatly and easy way. Laravel also comes with built-in support for Blade templates, the Eloquent ORM for database administration, and the powerful Artisan CLI for the automation of the repetitive tasks which is suitable for building scalable and maintainable web applications.

Laravel has many advantages. It provides clean and readable code that make it easier to maintain and expand applications. The MVC architecture promote code organization and separation of concerns. Laravel also has built-in security tools such as protection against SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF). Besides, Laravel has a large community and accurate documentation (Laaziri et al. ,2019). This allows Laravel can provide a variety of packages and learning resources. Laravel Mix also makes it easy to compile assets and integrate with front-end tools.

However, Laravel also has some disadvantages. One of the main disadvantages is the learning curve can be high for beginners who are not familiar with object-oriented programming or MVC architecture. Laravel applications can be heavy and need to be optimized for high performance needs. If the features or packages are not structured properly, it may lead to messy code even though Laravel provides a lot of flexibility. Lastly, Laravel hosting requirement may limit its deployment on older server as it too relies on modern PHP features (Neelam Menariya, 2022). Additionally, it also has slow performance compared to other frameworks such as Node.js or ASP.NET. Therefore, this result in Laravel is not suitable for applications that require high performance or real-time features.

Table 2.7: Advantage and Disadvantage of Laravel

Advantages	Disadvantages
Clean and readable code, easier to maintain and expand.	Steep learning curve for beginners unfamiliar with OOP or MVC.
MVC architecture promotes code organization and separation of concerns.	Laravel applications may require optimization for high performance.
Built-in security tools: protection against SQL injection, XSS, and CSRF.	Misstructured features or packages can lead to messy code.
Large community and detailed documentation for support and resources.	Hosting requirements may limit deployment on older servers.
Laravel Mix simplifies asset compilation and front-end integration.	Slower performance compared to frameworks like Node.js or ASP.NET, not suitable for real-time applications.

2.4.4 Express.js

Express.js is a fast, open, and minimalist Node.js web framework. It provides a range of powerful features for web and mobile applications that simplify the process of building web servers and APIs. Express.js is widely used to build backend services. It is also well known for its ease of use, flexibility, and performance. It uses the non-blocking, event-driven features of Node.js to handle multiple requests which makes it become a popular choice for creating scalable and high-performance web applications.

One of the main advantages of Express.js is its simplicity and minimalism which allows developers to build web applications and APIs in a faster way and without unnecessary overhead. Its modular structure gives the developers freedom to create custom frameworks and add middleware to meet developer's requirements. Express.js also provide many plug-ins and frameworks that make it easier to expand its functionality. It also provides powerful routing capabilities that allow developers to easily define and manage routes. It is also a developer-friendly framework due to it large community support and documentation

Express.js has some disadvantages. One of the main disadvantages is minimal functionality which means that the developers need to write more boilerplate code for features that are pre-built in other frameworks. It also does not have built-in solutions such as authentication or data validation. Therefore, the developers need to integrate

third-party tools or libraries. Besides, Express.js also inherits some of the Node.js limitations such as inability to handle CPU-intensive operations which can affect some of the use case performance.

Table 2.8: Advantage and Disadvantage of Express.js

Advantages	Disadvantages
Simple, minimalist, and fast web application/API development.	Limited functionality, requires more boilerplate code.
Modular structure, supports custom frameworks and middleware.	Lacks built-in features like authentication and data validation.
Provides numerous plugins and frameworks for extended functionality.	Requires third-party tools/libraries to implement more features.
Powerful routing function, easy to manage routing.	Inherits Node.js limitations and struggles with CPU-intensive tasks.
Large community and comprehensive documentation	

2.4.5 Conclusion of Web Application Framework

For the frontend, React Native is the most suitable for this project as it can build reusable components that make it easy to manage complex user interface. It also suitable for interactive dashboards and real-time data changes such as booking status or equipment availability. React's virtual DOM also can improves the performance and provide smooth delivery and responsive user experience. Furthermore, React has large ecosystem that contains useful libraries for routing and state management. React's component-based architecture makes it simple to divide the user interface into manageable, reusable parts and speeding up development and maintenance.

For the backend, Laravel is most suitable for this project as it provides a clear and structured foundation for creating scalable web applications because it is a PHP framework with an MVC design. Laravel's built-in tools like Eloquent ORM simplify database management, while its Blade templating engine ensures clear separation of frontend and backend logic. The framework also provides strong security features, protecting against SQL injection, cross-site scripting (XSS), and cross-site request

forgery (CSRF). Laravel's large community and detailed documentation make it easier for developers to troubleshoot and extend the application's functionality which can provide a smooth development process for the sports equipment management system.

2.5 Existing Similar Application

2.5.1 Odoo

The Odoo Inventory Management System is a comprehensive open-source solution designed to simplify and automate inventory operations for businesses of all sizes. The system integrates with other Odoo applications such as sales, manufacturing, and accounting which will provide a single method to manage the entire supply chain. The system also provides real-time visibility of inventory levels that enable business to make decisions and reduce stockouts.

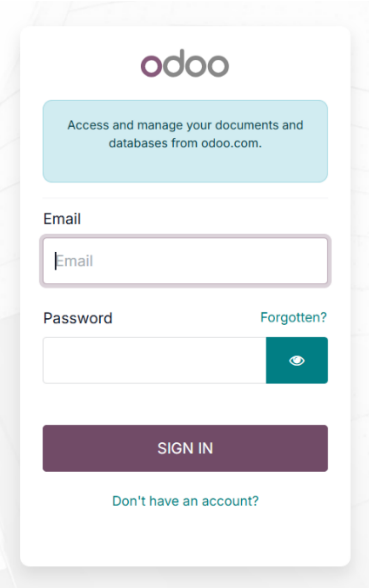


Figure 2.3: Login Page

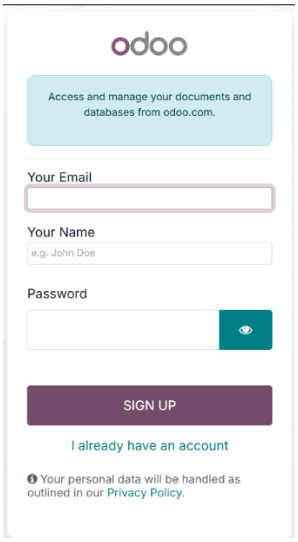


Figure 2.4: Sign Up Page

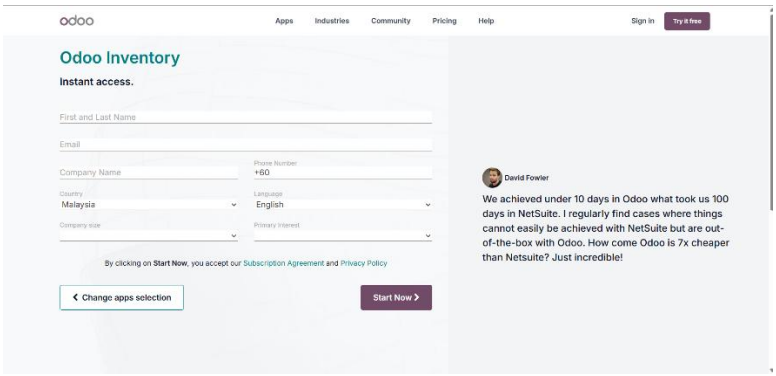


Figure 2.5: Instant Access Page

The figure show login and sign-up page which allows users to fill in their details. The sign-up page includes email field, name field and password field while the login only required user to fill in their email and password. It also includes the instant Access Page which is a quick access to demo inventory management system. After signing up, user can access to the inventory management system such as menus, fields, navbar buttons, chatter, report actions, and multi-action views.

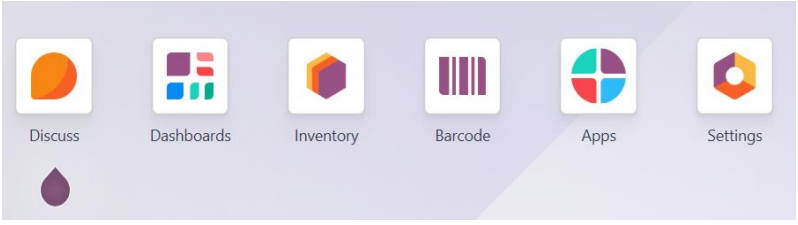


Figure 2.6: Main Page

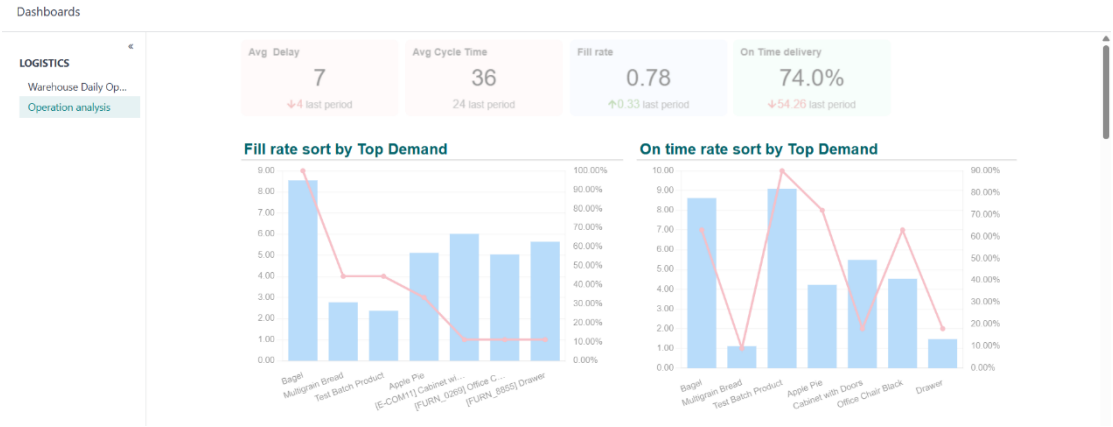


Figure 2.7: Dashboards Page

Figure 2.7 shows the dashboards pages which shows the KPIs (Key Performance Indicators) such as inventory levels, sales orders and financial metrics. Users can the charts, graphs, and lists based on their roles.

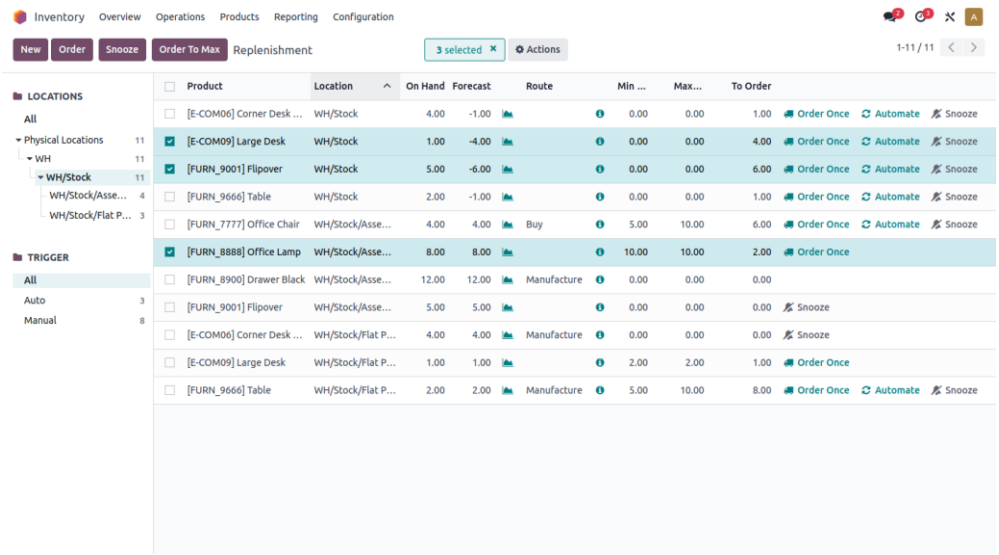


Figure 2.8: Inventory Page

The figure shows the Inventory Page. It allows users to select, create, delete and update the inventory based on user roles. It contains location tracking so that users can know where the item is stored. It also allows user to set minimum and maximum quantity levels for items. The real-time inventory tracking allows users to get the immediate results into the stock levels across multiple warehouses to ensure the information is up-to-date and accurate.

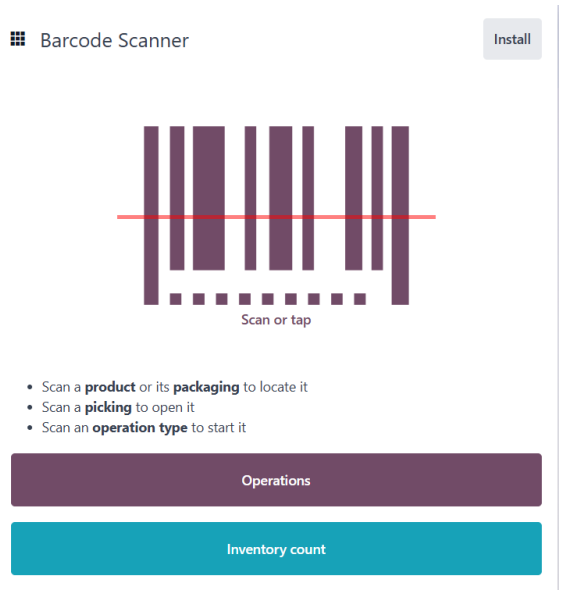


Figure 2.9: Barcode Page.

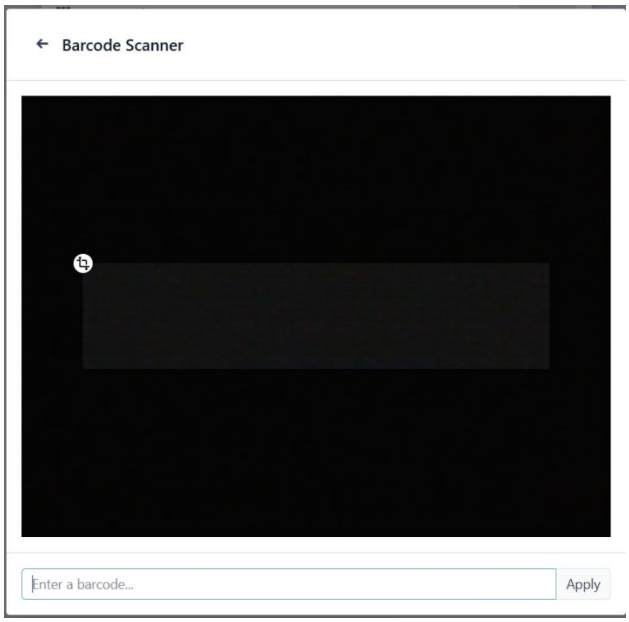


Figure 2.10: Barcode Scanner Page

Figure 2.9 shows the Barcode Page which allow user to scan the barcode, and it also allows user to see the operation such as receipt or delivery option or allows user to check the amount of inventory by clicking the inventory count button. Figure 2.10 shows the Barcode Scanner Page which allows user to scan the barcode.

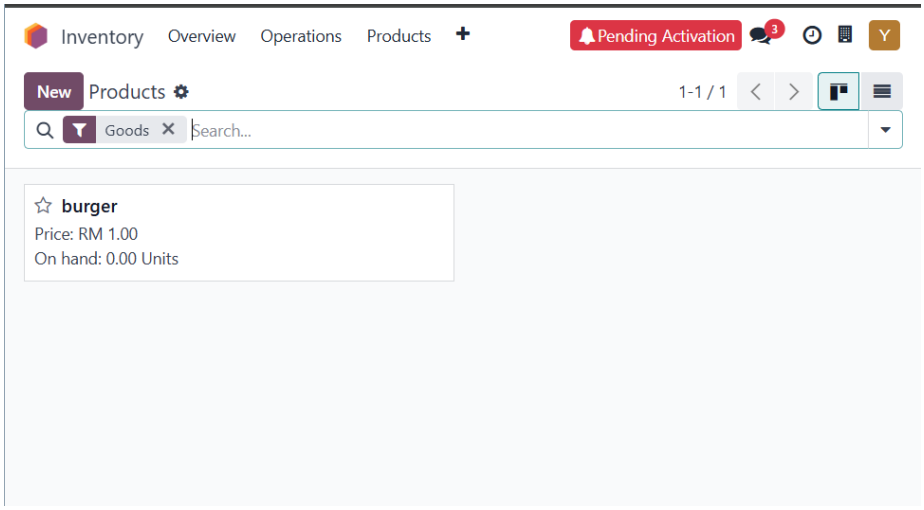


Figure 2.11: Product Page

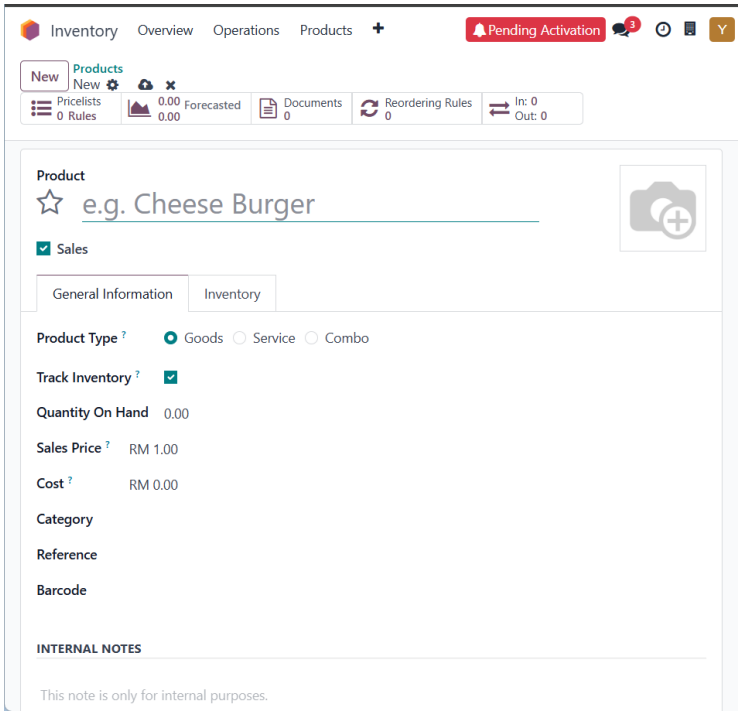


Figure 2.12: Add new product Page

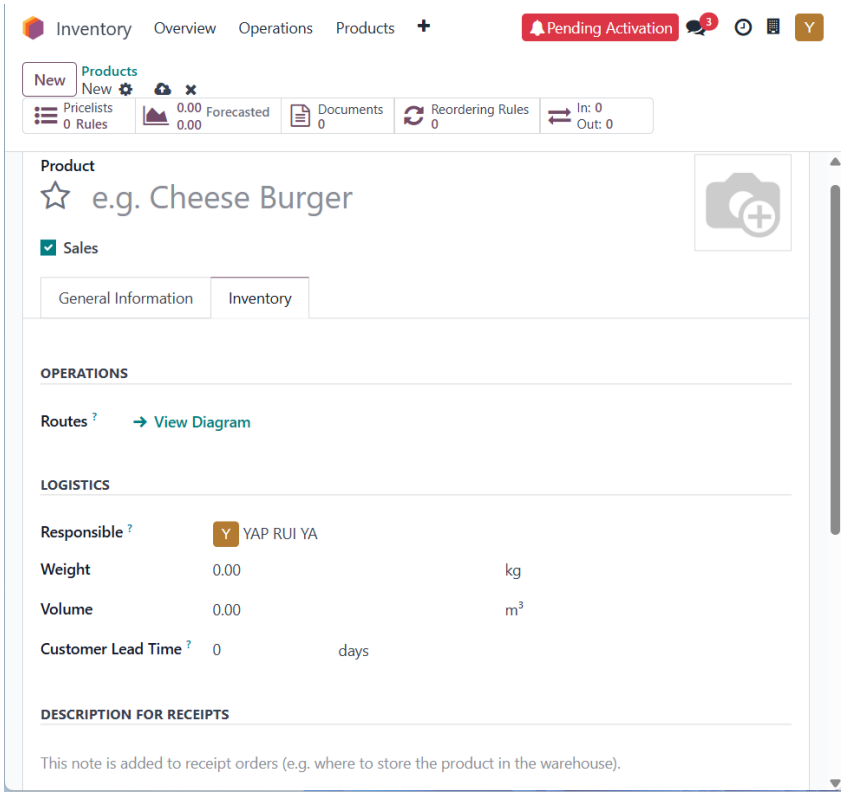


Figure 2.13: Add new product Page- quantity

Figure 2.12 and Figure 2.12 shows the product page and add new product page. In add new product page, it allows user to select the product type and track inventory, quantity on hand, sales price, cost, category, reference, barcode, add images, and internal note. It also allows user to choose the inventory and fill in the logistics and description for receipt section. In logistic section, user can fill in the weight, volume and customer lead time. After creating a new product, it will display in the product page and it can be editing the product information by clicking the frame as shown in Figure 2.11.

Table 2.9: Advantage and Disadvantage of Odoo

Advantages
Odoo provides real-time visibility into inventory levels, movements, and locations, enabling businesses to make informed decisions and reduce out-of-stocks.
When inventory levels is low, the system automatically generates a purchase order to ensure timely replacement and preventing inventory overstocking.
Integration with barcode and RFID technology improves the accuracy and efficiency of inventory handling, reduces manual errors and speeds up the process.

Odoo provides advanced reporting tools that provide insights into inventory performance that can help businesses identify trends and make data-driven decisions.
Disadvantages
The initial configuration of Odoo's inventory module can be complex and time-consuming which requiring technical expertise or external assistance.
User interface complexity.
Limited Support for Non-Standardized Processes
Limited availability of features in some regions

2.5.2 Dashcode

Dashcode is a front-end development tool that designed to create interactive and visually appealing dashboards easily. It offers a user-friendly drag-and-drop interface that allows developers to quickly design and prototype custom dashboards without extensive coding knowledge. It also provides a variety of customizable widgets, such as charts, tables, and graphs that make it easier for displaying real-time data and key metrics. It allows faster development of small projects and prototypes but it requires additional tool or system for more complex features such as back-end integration or real-time updates.

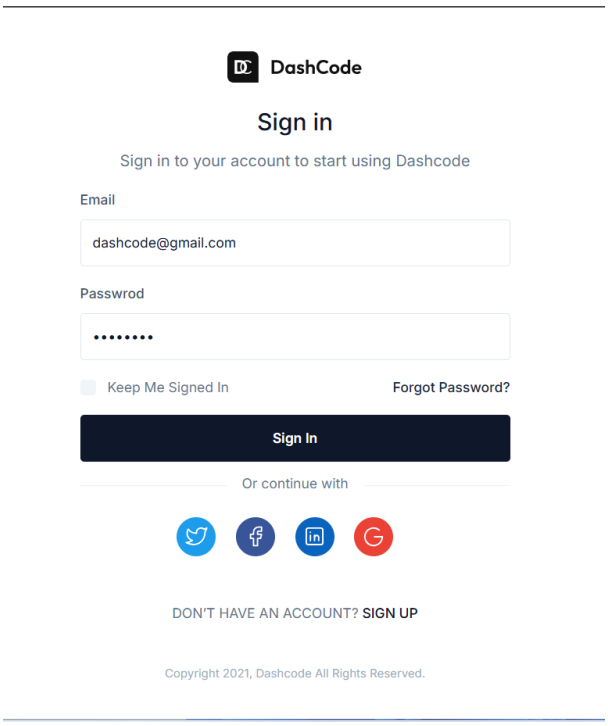


Figure 2.14: Log in Page

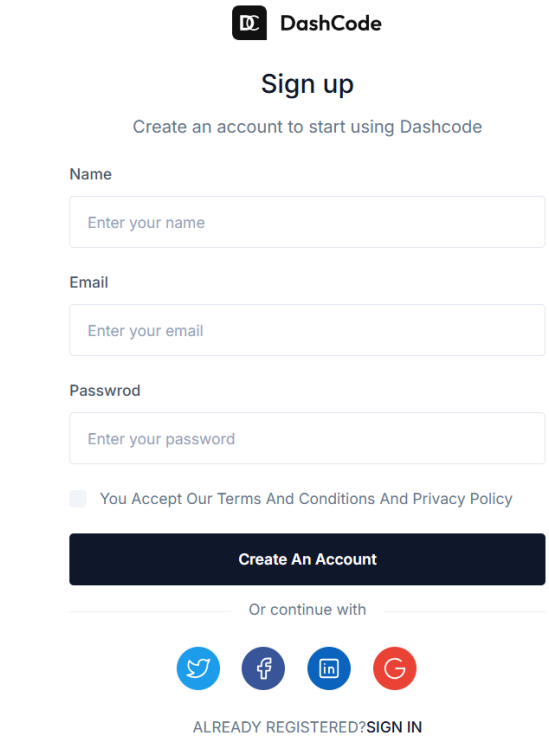


Figure 2.15: Sign up Page

Above figure shows the sign up and login page. The sign up page required users to fill in their name, email and password while the login page required users to fill in their name and password.in the sign up page, users must make sure they tick the terms and conditions before they enter the create an account button.

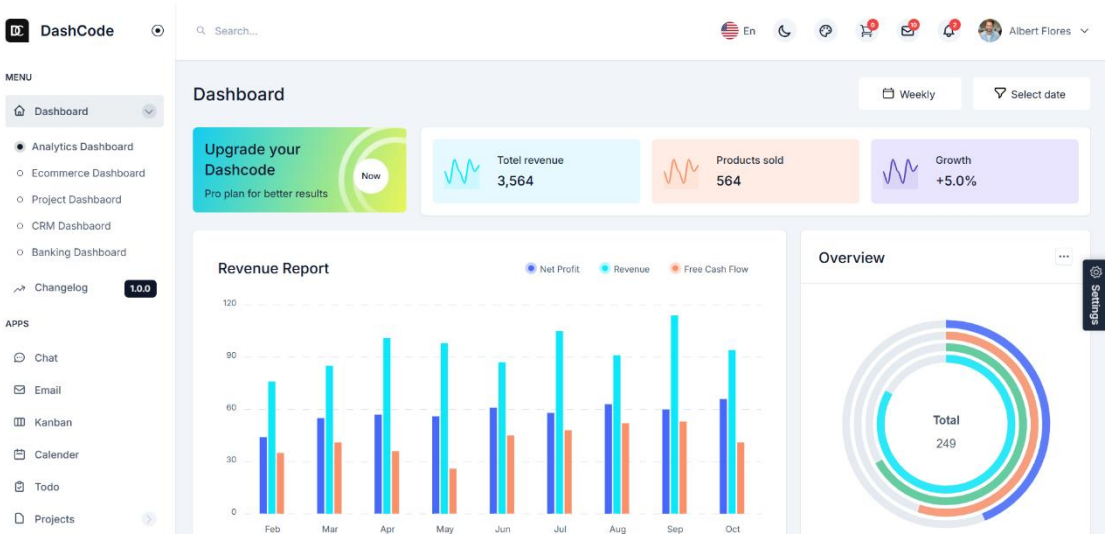


Figure 2.16: Analytics Dashboard Page

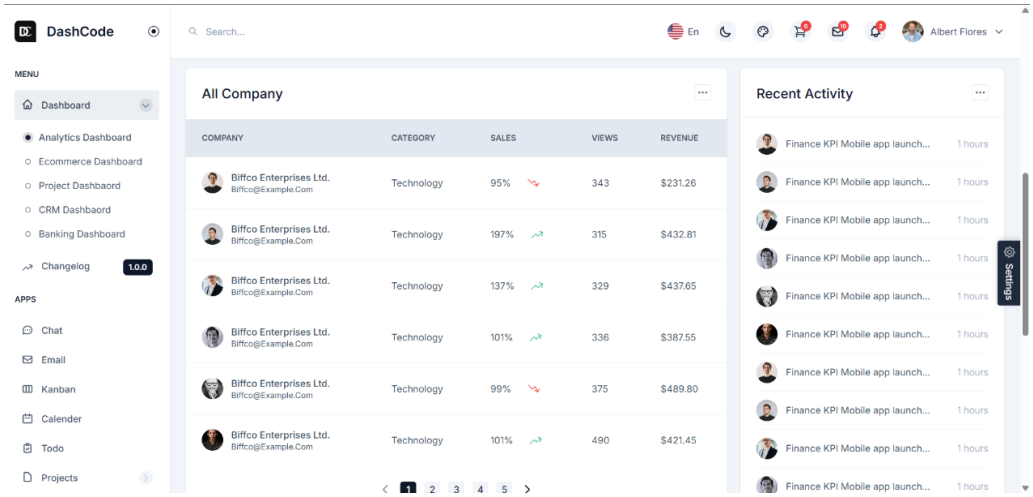


Figure 2.17: Analytics Dashboard Page-2

Figure 2.16 and Figure 2.17 shows the analytics dashboard Page. It allows users to interact with data easily. At the left side, it contains sidebar which allow users to easily navigate between different sections or pages within dashboards. At the tops it contains a search bar to allow users to filter and search the data based on specific criteria.

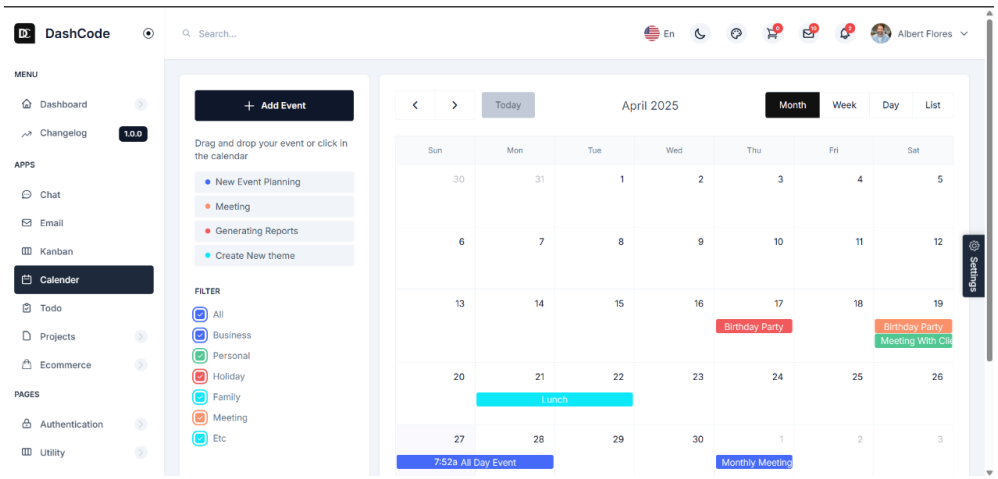


Figure 2.18: Calendar Page

Figure 2.18 shows the calendar page. It allows users to schedule the event by clicking on the specific date and the events can be added directly. Calendar Page also allows users to navigate between months, weeks, and days to view or add scheduled events. Besides, clicking on a specific day or event within the calendar can reveal more details about that event and the events on the calendar can be color-coded or labelled to indicate their type. Dashcode also allow users to set up recurring events in the calendar.

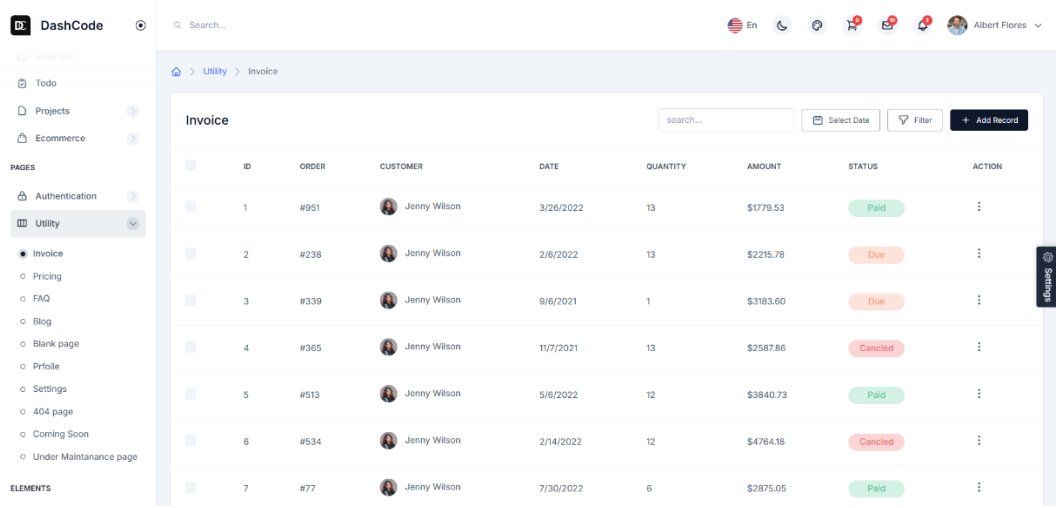


Figure 2.19: Invoice Page

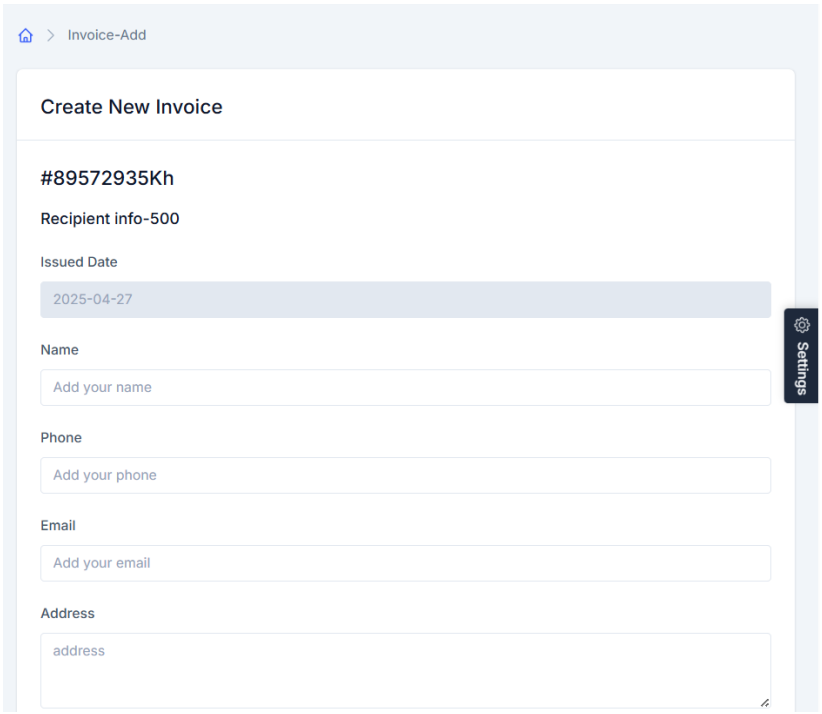


Figure 2.20: Add Invoice Page

Figure 2.19 is an invoice page. It is designed to help manage and generate invoice for different transactions such as purchases, sales, or rentals. It displays a list of all past invoices with details such as invoice number, date, customer name, total amount, and status. It also has an add record button that enables users to generate a new invoice. After clicking the button, it navigates to add invoice page as shown in Figure 2.20.it has a form where users can input all details of invoices such as customer information, invoice items, date and payment term. Besides, it displays the status of the invoice, such as "Pending," "Paid," "Overdue," or "Cancelled.

Table 2.10: Advantage and Disadvantage of Dashboard

Advantages
Easy to navigate and use.
Customizable dashboard
Provides detailed insights.
Reduces manual work and errors.
Provides detailed insights.
Works well with various business tools.
Disadvantages
Need third-party integrations.
Requires stable connectivity.
Can be overwhelming for small businesses.
May not scale well for large businesses.
Slow response times during peak hours.
May not integrate with niche platforms easily.

2.5.3 ECOUNT (Inventory / Barcode Software)

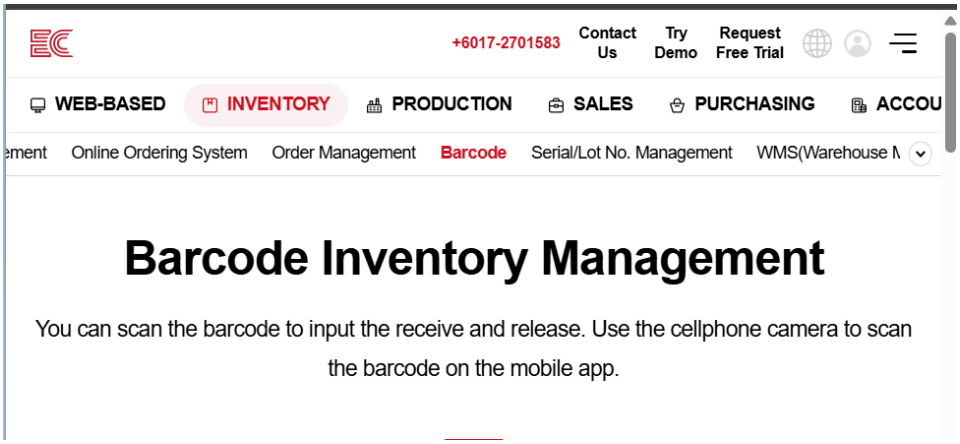


Figure 2.21: Barcode Inventory Management

ECOUNT is a cloud-based ERP system that includes inventory, barcode, warehouse management system (WMS), sales, purchasing, accounting and other modules. It is designed to help businesses manage stock levels, simplify warehouse operations and improve the accuracy of inventory tracking by using barcode scanning. The system

allows users to generate and print barcodes, scan the barcodes using mobile devices or external barcode scanners and manage inventory across multiple outlets in real-time. Therefore, ECOUNT can handles the inventory control and reporting efficiently.

Figure 2.22: Connect Barcode Scanner using OTG Cable

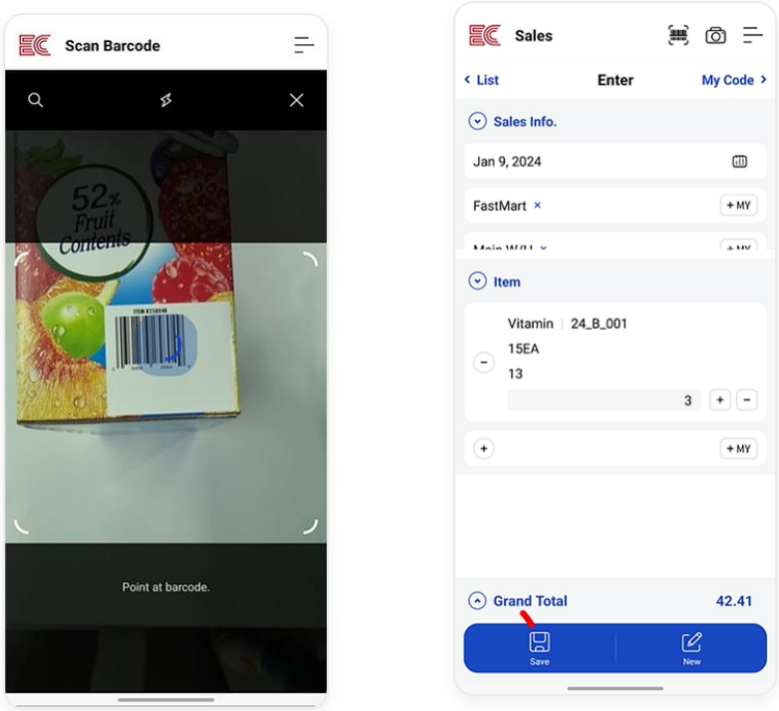


Figure 2.23: Scan Barcode using Mobile Application

The figure 2.22 and 2.23 shows that the ECOUNT allows users to scan the barcodes using either mobile application or external devices that connect by using USB or OTG. After scanning, the system updates stock-in or stock-out records automatically to reduce the manual input and errors.

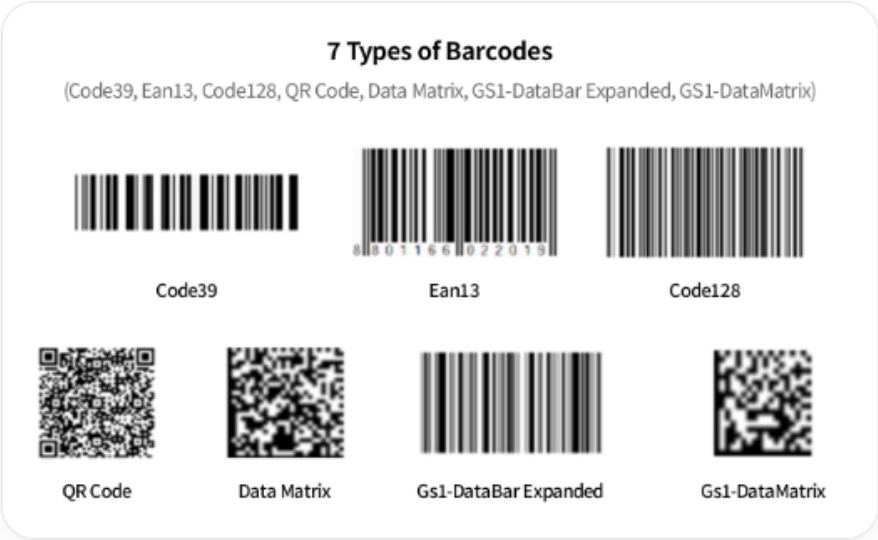


Figure 2.24: 7 types of Barcodes

Figure 2.24 **s**hows that ECOUNT allows user to generate custom barcodes for products that lack of manufacturer codes. Users can create their own barcode format by using the combination of items codes, name or batch numbers. It also supports label printing that including key details such as price, SKU and company logo.

Table 2.11: Advantage and Disadvantage of React Native

Advantages
Real-time inventory tracking
Supports barcode scanning with mobile or external devices
Multi-warehouse and location management
Barcode generation and label printing
Integrated with other ERP modules like sales, purchasing, and accounting
Cloud-based access from any device
Supports serial/lot tracking and safety stock alerts
Disadvantages

Requires stable internet connection
Initial setup can be time-consuming
Some advanced features only available in paid modules
Performance may vary depending on hardware and scanner compatibility
Limited custom workflows for niche industries

2.5.4 Comparison of Existing Similar Application

Table 2.12: Comparison of Existing Similar Application

Feature	**Odoo Inventory**	**Dashcode**	**ECOUNT**
Purpose	Full-fledged ERP with inventory management capabilities	Front-end development tool for creating dashboards	Cloud-based ERP with strong inventory and barcode management features
Ease of Use	User-friendly with a variety of modules, but may require setup	Easy to use for creating simple UI-based dashboards	Easy to use for basic inventory and barcode features.
Customization	Highly customizable with powerful tools for inventory tracking	Limited to front-end UI design, requires external systems for backend	Moderate customization available; supports custom barcodes, forms, and reports
Inventory Tracking	Advanced tracking of stock levels, movements, locations, and reordering	Basic tracking via custom-designed widgets, requires additional setup	Real-time tracking of stock levels, serial/lot numbers, and warehouse transfers
Reporting and Dashboards	Provides automated and dynamic inventory reports,	Supports simple dashboards but lacks advanced	Built-in inventory reports with export features (Excel/PDF);

	analytics, and KPIs	reporting capabilities	dashboard includes alerts and summaries
Integration	Strong integration with other Odoo modules (e.g., sales, purchases) and external systems	Limited integration capabilities, especially for backend or complex systems	Integrates with other ECOUNT modules (sales, purchase, accounting); API available for third-party systems
Multi-location Support	Yes, tracks multiple storage locations and warehouses	Needs custom design for multi-location management	Yes, supports multi-warehouse/location tracking, transfers, and inventory balance per location
Stock Movement	Automated and real-time stock movements (e.g., borrowed/returned items)	Requires manual setup to track stock movements and updates	Barcode-based automated stock-in/out, real-time updates, and support for multiple movement types
Barcode Scanning	Full barcode support for quick stock management and tracking	Can be implemented with additional tools, but not built-in	Built-in barcode scanning via mobile camera or external scanners; supports label generation and printing
User Roles & Access Control	Granular user role management for access control and permissions	Limited user access control, focused more on visual elements	Role-based access for modules, menus, and operations; user permissions are customizable
Scalability	Scalable for small to enterprise-level	Primarily suited for small to medium	Scales for SMEs with multi-location support; may need

	needs with complex features	projects, may not scale easily	enhancements for enterprise-grade operations
Speed of Development	Faster for implementing out-of-the-box inventory solutions	Rapid prototyping of simple inventory dashboards, but more manual effort for complex systems	Fast setup for core inventory/barcode features; full ERP setup may take longer
Support & Documentation	Extensive documentation and a large community, official support available	Limited documentation, community support mainly for UI development	Good official documentation; email and phone support; resources targeted toward SME users
Advanced Features	Automated restocking, alerts, supplier management, maintenance tracking	Limited to basic features like displaying stock data and creating visual reports	Includes barcode generation, inventory alerts, invoice integration, basic WMS; some features limited in free version

2.5.5 Conclusion of existing similar applications

Through the analysis of the existing applications such as Dashcode, Odoo and ECOUNT, it shows that they provide a strong inventory control and asset management solutions. Dashcode may perform well with features such as barcode scanning, real-time updates, and maintenance scheduling that make it suitable for industries that need to track and manage assets in real time, but it may lack some of the advanced customization.

Odoo provides a full set of integrated modules that make it become a multifunctional solution for organizations to manage all aspects of their business such as inventory, sales and purchasing. Odoo also provides a powerful automation and

reporting tools that can help to provide the insightful analytics on equipment usage, stock levels, and maintenance activities.

ECOUNT also provides a cloud-based ERP solution with strong support for barcode scanning, multi-location inventory tracking and automated stock movements. Its user-friendly interface, built-in barcode label generation, and integration with modules like purchasing and sales make it effective for small to medium enterprises (SMEs) looking for a cost-effective and functional inventory system.

In conclusion, it is possible to build a tracking management system that allows for real-time monitoring, easy management of equipment across locations, and efficient maintenance scheduling through selecting and integrating these features from Dashcode Odoo and ECOUNT Inventory. However, it may face a challenge such as complex user interfaces and the need for further customization and these challenges can be solved by using custom modifications to meet the needs of the project.

CHAPTER 3

METHODOLOGY AND WORK PLAN

3.1 Introduction

This chapter describes the methodology and work plan for developing an inventory management dashboard for tracking sport equipment and facilities at a secondary school sport centre. The prototyping methodology was selected due to its early creation of functional system models which allows continuous user feedback and iterative refinement. The flexibility of prototyping methodology also ensures that the stakeholder's requirements are clearly understand and meets their needs. Besides, a detailed Work Breakdown Structure (WBS) will be created to divide the project into smaller tasks in various phases such as initiation, design, prototyping, testing, deployment, and closure. The combination of prototyping methodologies and structured planning promotes well-organized progress, effective use of resources, and the capacity to adapt to changes.

3.2 Prototyping Methodology

This project will use Prototyping Methodology which is an iterative approach and user will involve in this project to gradually refine the system requirement and functionality. Prototyping methodology will be chosen as it is suitable for the projects where the requirements are unclear or always changing. It also focuses on the early user interaction to build a usable and effective system. Besides, the project is divided into multiple prototype iterations, and each prototype iteration will include planning, building, user testing, and improvement activities that allow the system to evolve based on real user feedback.

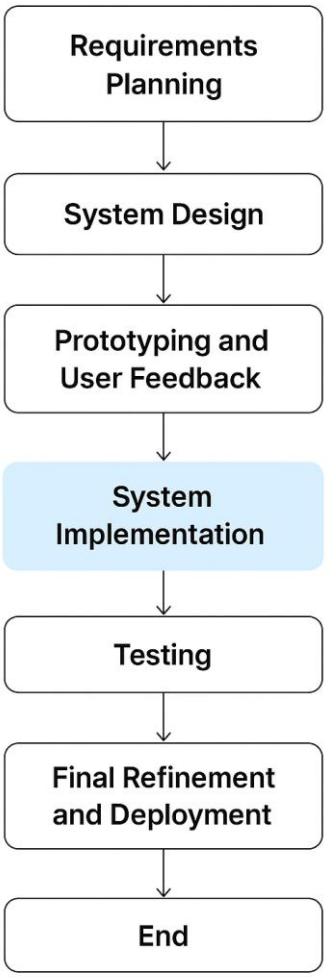


Figure 3.1: End-to-End Methodology Flow

3.2.1 Requirements Planning

Prototyping Methodology is a flexible and iterative development that can adapted to changes in requirements throughout the project life cycle. It needs to collaborate with the stakeholders such as the teachers, students and sport centre administrators to gather information about Tracking Sports Equipment and Facilities Inventory Management System. Its gathers the requirements through conducting interviews and questionnaire to identify which features is required to implement in the system such as equipment tracking and booking system. In addition, literature review will be carried out to explore existing systems and identify best practices in inventory management. The results will be recorded as initial user requirement which will influence early prototypes and guide continuous improvements based on stakeholder feedback. Therefore, the prototyping

methodology can ensure that the system remains aligned with user requirements throughout the development process.

3.2.1.1 Quantitative Methodology

Quantitative methodology was used to better understand the needs and the challenges related to Tracking Sports Equipment and Facilities Inventory Management System. A questionnaire will distribute to collect information from relevant stakeholders. The purpose of the questionnaire was to gather information from questions that related to equipment availability, maintenance tracking, and overall inefficiencies in the manual process. There are 8 respondents provided valuable feedback that allow developer to identify the essential features and functionality required to implement for the new system. These insights provided the basis for developing the functional and non-functional requirements of the system.

3.2.1.2 Literature Review and Existing Systems Analysis

A literature review was conducted to explore similar inventory management systems in sports sectors. The review included comparing the advantages and disadvantages of various system such as ease of use, equipment tracking capabilities and integration with other systems. The review also identified industry best practices that could be implemented in tracking sports equipment and facility inventory management systems. Lessons learned from the systems can help to improve system requirements and structure for more efficient and user-friendly solutions.

3.2.1.3 Requirement Specification

The requirements for a sports equipment and facility inventory management system can be identified after collecting data and reviewing the literature. The system will include some features such as equipment tracking, maintenance scheduling, equipment booking, and a user-friendly interface for users. The requirements were categorized into functional and non-functional requirements. The use case diagrams were created to visualize the system interactions and workflows to provide a clear understanding of system functionality.

3.2.2 Design Process Using Prototyping Methodology

The inventory management system designed using a prototyping approach was well suited for this project. This is because usability is important the user requirements may change over time. This methodology allows users to be included in early development and collects feedback on iterative design models. Each improved prototype version shows the user feedback has been considered to ensure that the final system is fully functional, user-friendly and meets the requirements of the secondary school's sport centres. The development process consisted of 3 major prototyping iterations.

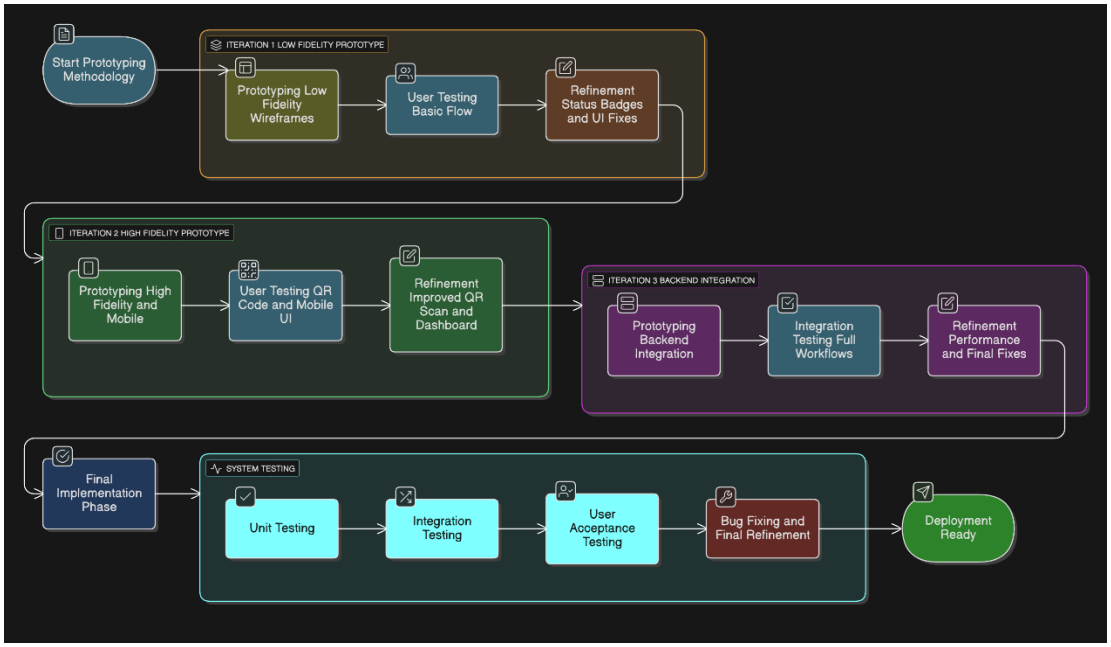


Figure 3.2: End-to-End Prototyping and Testing Flowchart

3.2.2.1 Iteration 1: Low-Fidelity Prototype and Basic Interaction Flow

3.2.2.1.1 Prototyping

In the first iteration, a low-fidelity prototype was developed to demonstrate the main features of the system such as equipment tracking, inventory management, login process and user roles. Axure RP will be used to create interactive wireframes that representing the main navigation flows. This early model emphasized basic navigation flow without full visual design details in order to quickly produce a working structure that could be evaluated by users. At this stage does not include any backend logic.

3.2.2.1.2 User Testing

The first iteration of the prototype was tested by a small group of stakeholders, such as quarter master and administrators. They will explore the wireframes and mocking up tasks such as viewing equipment and logging in. Feedback was collected based on system usability, functionality, and clarity. The common feedback included the need for a clearer visualisation of ‘available’ and ‘rented’ equipment. Unit tests were conducted on specific features to ensure that basic functionality worked as expected.

3.2.2.1.3 Refinement

The first prototype was refined based on user feedback and it will solve the user interface’s problems and new functionalities are planned to be added in future iterations. For example, add status badges for equipment and simple instructions for check in or check out equipment.

3.2.2.2 Iteration 2: High-Fidelity Prototype with Key Features

3.2.2.2.1 Prototyping

In second iteration, a high-fidelity prototype was created with expanded features and a more realistic interface. It was expanded the system by adding new features such as real-time equipment availability interface, equipment reservation with time slot selection, and booking history tracking to help users monitor past and upcoming reservations. A mobile version of the system was also developed to ensure that users can access to the platform by using their smartphones. The QR code functionality will include two key components which are QR code generation and QR code scanning. Administrators will be able to generate unique QR codes for each item during the inventory entry process. These QR codes will be printed and attached to the specific items. For users, they will be able to scan these QR codes by using their mobile phones to book the equipment. Administrator and quarter master are also can access to QR code scanning. In addition, role-based access will be introduced to differentiate user permissions based on their roles.

3.2.2.2.2 User Testing

This version of the system was tested by both sports centre staff and end users. The testing will focus on the new features introduced in this iteration such as the mobile responsiveness, QR code scanning efficiency and overall system performance. Administrators will test the QR code generation feature to ensure that QR code can be create, print, and assign to equipment correctly. They also tested others function such as equipment status updates, condition editing, and QR code generation. Feedback also will be collected to evaluate how users and administrators interact with the QR code system.

3.2.2.2.3 Refinement

The mobile interface will be improved based on the feedback from end users for better accessibility and ease of use. The QR code scanning process was simplified to ensure quick access to booking function. On the administrative side, the QR code generation will be improved to makes it easy to distribute and print. Dashboard and reporting will also be improved to provide summary of equipment usage. The user interface was also restructured to make it more intuitive and user-friendly, especially for first-time users.

3.2.2.3 Iteration 3: Backend Integration and Final Functional Testing

3.2.2.3.1 Prototyping

In the third iteration, the front-end system was integrated with the back-end database to ensure that the data can retrieved through backend. The system was designed to provide real-time data updates on equipment availability such as QR-based check-in and check-out operations that can directly updated equipment status in the database. Dashboard will also be introduced to provide analytics on equipment usage trends that allows administrator to make a true decision making and only administrators can view analytics data. This prototype also included full user role functionality.

3.2.2.3.2 User Testing

Integration testing was conducted to ensure that the data flow between the front-end and back-end systems were accurately. Feedback was also collected from administrators and end user to ensure that the system met their requirements and expectations. Users tested full workflows, such as logging in, scanning equipment QR

codes, booking facilities, and checking equipment status. Performance under concurrent user loads was also tested. In addition, final usability testing was conducted to validate the complete system functionality.

3.2.2.3.3 Refinement

Final improvements were made to optimize system performance, resolve integration issues, and ensure that the system was prepared for deployment. After final feedback was collected through User Acceptance Testing (UAT), the final version of the system was ready for deployment.

3.3 Final Implementation Phase

After the third iteration, the project will proceed to the final implementation phase. At this phase, a high-fidelity and user approved prototype will be converted into fully functional web and mobile application. All frontend components developed using React.js and it will link to the MySQL database backend. This project will only test in localhost. Therefore, a localhost server is used to deploy and test the entire system on a localhost environment.

3.4 System Testing

A structured testing strategy is implemented to ensure that the reliability, usability, and correctness of the inventory management dashboard for tracking of sports equipment and facilities system. The testing process involves multiple phases and each of them focuses on different aspects of system functionality.

3.4.1 Unit Testing

Unit testing is an important aspect of software development as it allows for early defect detection and improve code quality by using automated frameworks (Daka and Fraser, 2014). Unit testing was carried out before and after development iteration to validate system components to ensure that each function worked as expected. This isolation testing approach helped to detect and resolve logic-related errors early in the development process to ensure that it will function properly for later integration.

3.4.2 Integration Testing

Integration testing is important in object-oriented systems where the method executions are connected by messages across multiple components and errors only occur when units are composed together (Jorgensen and Erickson, 1994). Therefore, integration testing was focused on verified the communication between frontend and backend. At this phase, developers will test the functionality and also focused on the navigation between data and system modules to ensure that all components were worked as a cohesive unit.

3.4.3 User Acceptance Testing (UAT)

The system will release to users for user acceptance testing (UAT) Once integration testing was complete. These users were asked to perform user acceptance testing to evaluate usability and functionality. As Davis and Venkatesh (2004) mentioned that hand on usability testing of new system is usually conducted near the end of a system development project when the system development process is nearing its final operation state. At this phase, developers will ensure that the system was user-friendly and aligned with the user requirements and expectations that had been identified during the planning stage.

3.4.4 Bug Fixing and Final Refinements

Developers will record any bugs that had been found during all testing phases and categorized them according to severity in order to maintain system stability. In the final phase, developers made improvements to enhance the consistency of user interface, resolve remaining logic issues and optimize backend queries for better performance. The system will finalize and ready for deployment after all bugs were solve.

3.5 Project Plan

3.5.1 Work Breakdown Structure (WBS)

The project tasks are broken down into phases and subtasks for the Inventory Management Dashboard for Tracking Sports Equipment and Facilities:

Tracking Sports Equipment and Facilities Inventory Management System

1. Project Preparatory

1.1. Conduct research on sports equipment tracking and facility management systems

1.2. Discuss proposal ideas and refine scope with supervisor

1.3. Confirm final FYP title with supervisor

2. Requirements Planning

2.1. Project Initiation

2.1.1. Define project background and motivation

2.1.2. Identify key problems (overbooking, missing equipment)

2.1.3. Define specific project objectives

2.1.4. Define project scope and limitations

2.1.5. Propose system solution (Dashboard-based tracking and booking system)

2.1.6. Define project approach (Prototyping methodology)

2.1.7. Create Work Breakdown Structure (WBS)

2.1.8. Develop Gantt chart for project scheduling

2.2. Requirements Gathering

2.2.1. Develop questionnaire

2.2.1.1. Design close-ended and open-ended questions

2.2.1.2. Identify target respondents (teachers, students, sport centre staff)

2.2.1.3. Validate questionnaire through supervisor

2.2.2 Submit ethical clearance

2.2.3 Distribute questionnaire and collect responses

2.2.3.1 Analyse questionnaire results to identify user needs

2.2.4. Literature Review

2.2.4.1. Study software development methodologies (Waterfall, Agile, Prototyping)

2.2.4.2. Conclude relevant approaches and technologies

2.2.4.3. Research web frameworks (React, Laravel, MySQL)

2.2.4.4. Review existing inventory and booking applications (e.g., Odoo, Dashcode)

2.3. Define Requirements

2.3.1. Develop functional requirements (e.g., booking, reporting, CRUD operations)

2.3.2. Define non-functional requirements (e.g., accessibility, performance)

2.3.3. Create use case diagram and descriptions

2.4. Develop Low-Fidelity Prototype

2.4.1. Design User Management Page

2.4.2. Design Equipment Management Page

2.4.3. Design Reservation Management Page

2.4.4. Design Booking Management Page

2.4.5. Design Track Equipment Page

2.4.6. Design Dashboard Page

2.4.7. Design Stock Check Page

2.4.8. Design Booking Page

2.4.9. Design View History Page

2.4.10. Design Member Page

2.4.11 Design Login Page

2.4.12. Design QR Code Generation and Scanning Interface

3. User Design and Iteration

3.1. First Iteration

3.1.1. Prototyping Phase 1

3.1.1.1. Implement homepage with login functionality

3.1.1.2. Create basic booking module interface

3.1.1.3. Develop equipment listing and inventory views

3.1.1.4. Build reporting module structure (dashboard)

3.1.1.5. Prototype QR Code Generation UI for equipment

3.1.1.6 Create ERD diagram

3.1.2. Conduct User Testing and Evaluation 1

3.1.3. Refine Prototype 1 based on feedback

3.2. Second Iteration

3.2.1. Prototyping Phase 2

3.2.1.1. Improve inventory management with edit/delete features

3.2.1.2. Improve booking management and tracking features

3.2.1.3. Add QR code scanning functionality for equipment check-in/out

3.2.2. Conduct User Testing and Evaluation 2

3.2.3. Refine Prototype 2

3.3. Third Iteration

3.3.1. Prototyping Phase 3

3.3.1.1. Finalize UI/UX enhancements

3.3.1.2. Test full QR code functionality

3.3.2. Link frontend and backend components

3.3.3. Conduct User Testing and Evaluation 3

3.3.4. Final refinement of prototype Construction

4.1. Coding implementation of all modules Authentication, Booking, Inventory, Reporting, Maintenance, Dashboard

4.2. Conduct system walkthrough and informal user evaluation

5. Deployment

5.1. System Testing

5.1.1. Unit testing of individual functions (e.g., CRUD, login)

5.1.2. Integration testing (database ↔ backend ↔ frontend)

5.1.3. User Acceptance Testing (UAT) with target users

5.2 Deployment

5.3 Write report

5.4 Presentation and Demonstration

3.5.2 Work Plan

3.5.2.1 FYP 1 Gantt Chart

The FYP1 Ghantt Chart is attached in the Appendix B for reference.

3.5.2.2 FYP 2 Gantt Chart

The Fyp2 Ghantt Chart is attached in the Appendix B for reference.

3.6 Development Tools

This project utilized a combination of design, development and database tools to ensure an efficient and well-structured development process. The selected tools supported front-end and back-end development, interface prototyping and system modelling. These tools were important to the successful implementation of an inventory management dashboard for tracking sports equipment and facilities.

Figure 3.3: Development Tools

3.6.1 Visual Studio Code

Visual Studio Code (VS Code) is the main source code editor will be used during the development of this project. This is because it supports different types of programming languages and technologies such as HTML, CSS, JavaScript, PHP and MySQL integration. It also provides built-in Git support, intelligent code completion, debugging tools and an extensive marketplace for extensions which allowed developers to customize the development environments to improve efficiency. It is also suitable for novice and experienced developers due to its user-friendly interface and cross-platform availability.

3.6.2 Axure RP

Axure RP is used to create interactive wireframes and prototypes of the system during the design and planning phase. It helps to visualize the layout and functionality of user interfaces before the development starts. Axure RP allows the developers and stakeholders to interact with mock versions of system components. This tool is

important in a prototyping methodology as it allows early feedback from user and helps to develop the interface design and improve user experience.

3.6.3 React

React is used to create the frontend of the application. React is a JavaScript library that used to create user interface and provides responsive, dynamic UI, reusable components, and effective DOM rendering. React's component-based architecture makes it easier to manage the complexity of user interaction workflows and ensures modularity and maintainability in code.

3.6.4 Laravel

Laravel is the backend framework that used to handling server-side logic, user authentication, and API development. The Model-View-Controller (MVC) architecture of Laravel can help to organize the application structure and separate appearance from logic. Laravel is suitable for creating scalable and safe backend due to its built-in security features, middleware, and route management. Laravel's Artisan CLI also makes repetitive tasks become easier such as database migrations and controller generation.

3.6.5 MySQL

MySQL is a relational database management system which will store and retrieve all persistent data in the system. MySQL is used in this project due to its reliability, ease of use, and compatibility with Laravel. Through Laravel's Eloquent ORM, MySQL tables can be queried and updated using expressive, readable PHP code which makes data handling become easier and safety.

3.6.6 Enterprise Architect

Enterprise Architect is used for advanced UML modelling and system architecture documentation. It supports the development of comprehensive system design models such as use case diagrams, class diagrams, sequence diagrams, and component diagrams. It is a useful for formal software engineering documentation as it can supports traceability from requirements to implementation and shows the complex system behaviour, workflows, and interaction logic between modules.

3.6.7 WampServer

WampServer is used as a local web server environment to host and run the Laravel backend during development and testing. It provides an integrated stack of Apache, MySQL, and PHP that allows applications to run on the local machine without the need for an external server. WampServer is important for the projects that need to run on localhost.

3.6.8 phpMyAdmin

phpMyAdmin is a web-based interface that used to manage the MySQL database. It comes together with WampServer to carry out database operations such as creating tables, performing queries, editing records, and exporting data. It is useful for users who need to inspect or maintain their database without writing SQL commands.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Introduction

This chapter shows the requirements specification of the proposed Inventory Management Dashboard for the school sports centre. It provides a detailed overview of the functional and non-functional requirements. This chapter also provides a use case diagram that highlights the roles and activities of the main stakeholders such as teachers, students, quarter master and administrators. This chapter also provides screenshots of early prototypes to present the user interface and fact findings.

4.2 Fact Findings

To better understand the existing challenges and needs of the school's sports equipment and facilities management, the school staff has provided some evidence and information that shows their current workflow. The equipment inventory count process is manual which they use handwritten form to track their equipment. The bookings for sport equipment and facilities are also recorded manually in a book by staff which may cause a lot of human error, mistakes and inconsistency.

For inventory records, the school uses a basic Microsoft Excel spreadsheet with only 3 columns which is equipment name, instock and damage. The inventory quantities are updated manually by staff. They also mentioned that all inventory records, booking requests and tracking processes are maintained in hard copy which relying on handwritten logs and physical documentation. These limitations mention the needs for a centralized and digital system that can improve overall process in order to increase its efficiency, transparency in sport equipment management system. An image illustrating the current manual stock check and booking method used by the school is attached in Appendix A for reference.

4.2.1 Analysis

Section A: School Profile

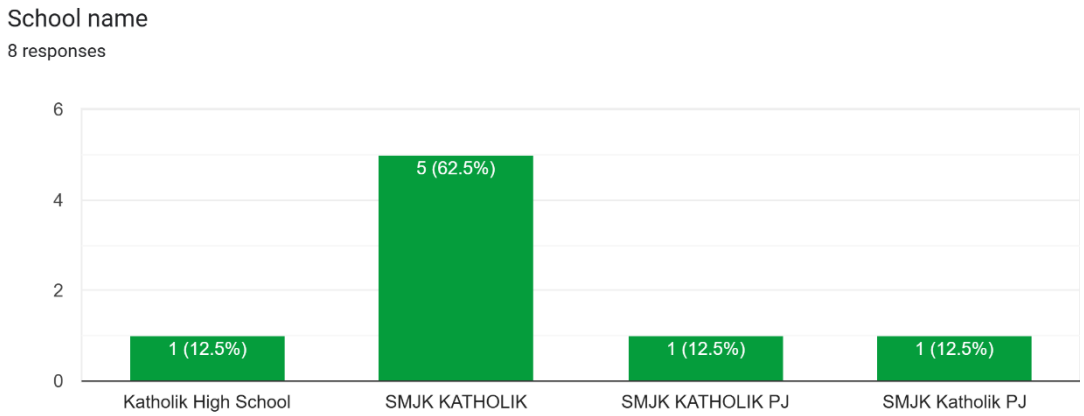


Figure 4.1: Current Method of Tracking Sports Equipment

Based on the above pie chart, the respondents are all from the same school.

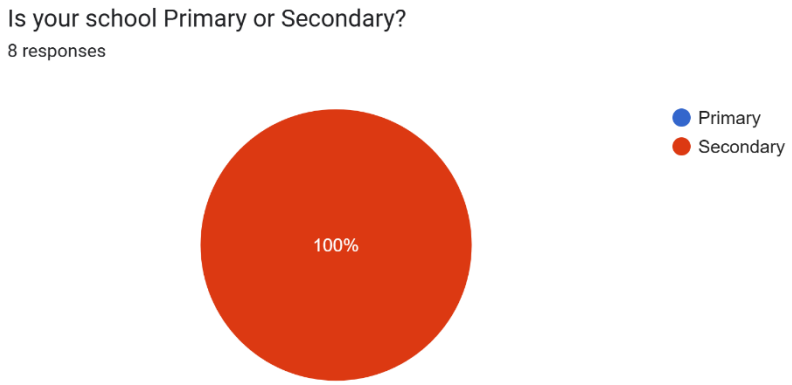


Figure 4.2: School of Respondents

Figure 4.1 **s**hows that 100% of the respondents come from the secondary school.

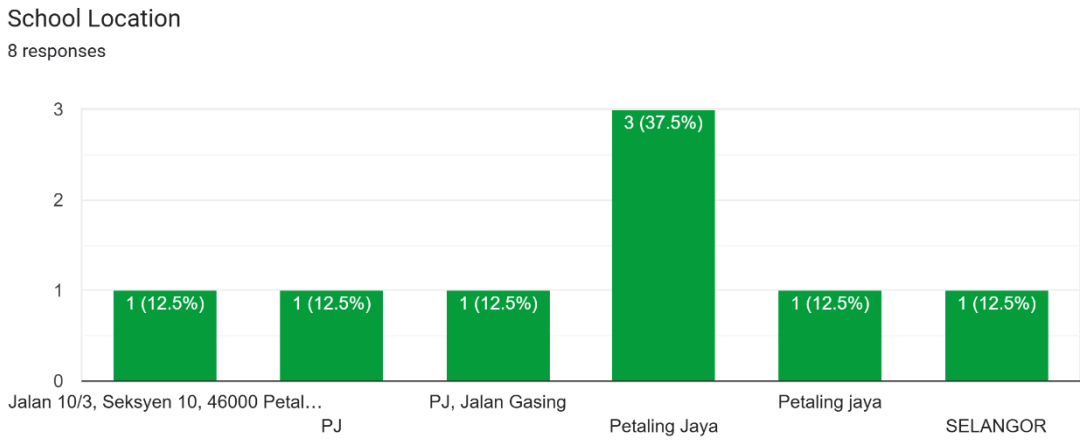


Figure 4.3: School Location of Respondents

It was found that the school was located in Petaling Jaya even the respondents entered different location.

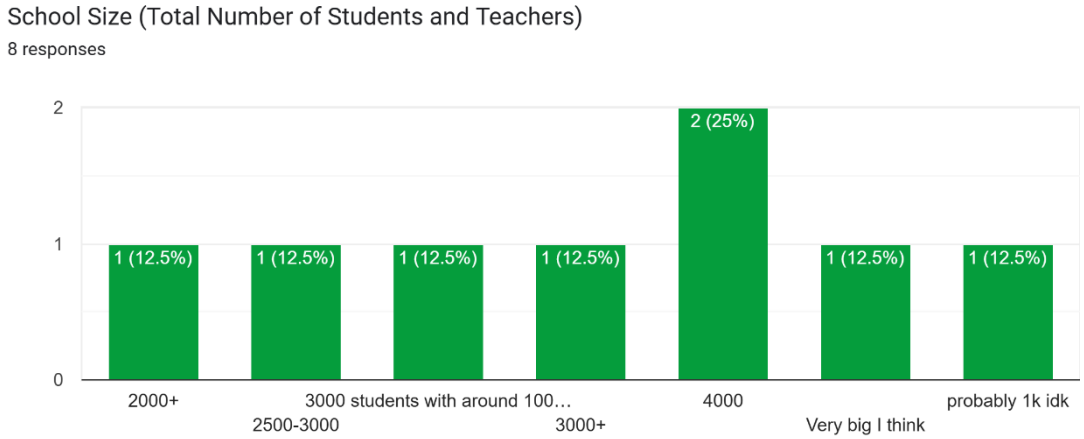


Figure 4.4: School Size of Respondents

There are 8 respondents answer about the school size where 25% of respondents answer 4000 which is the highest, 12.5% of respondents answer 2000+,12.5% of respondents answer 2500-3000, 12.5% of respondents answer 3000 students with around 100 teachers, 12.5% of respondents choose very big I think and 12.5% of respondents answer probably 1k.

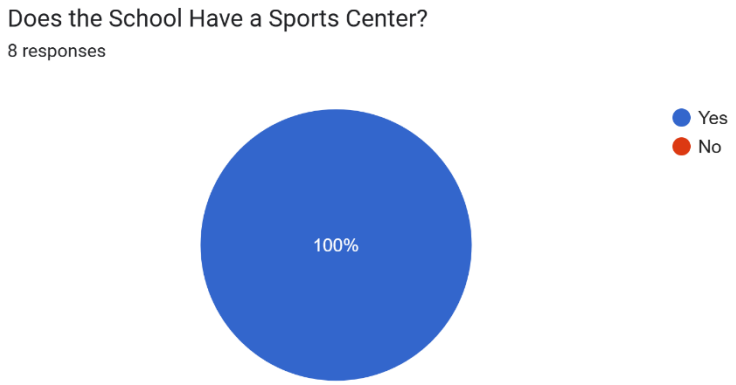


Figure 4.5: Availability of a Sports Centre

There are 100% of respondents choose yes which shows that all respondents school have sports centre. This finding supports the relevance and importance of improving sports facility and equipment management systems.

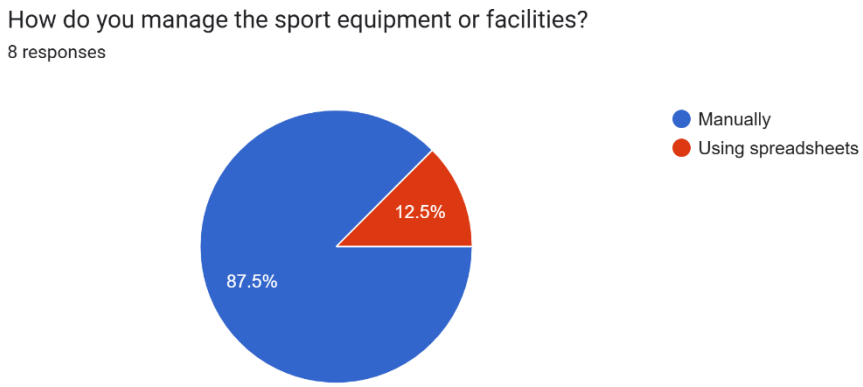


Figure 4.6: Current Method of Managing Sports Equipment

Many respondents (87.5%) manage sports equipment manually and only 12.5% use spreadsheets. This shows that most of the school rely on traditional paper-based methods.

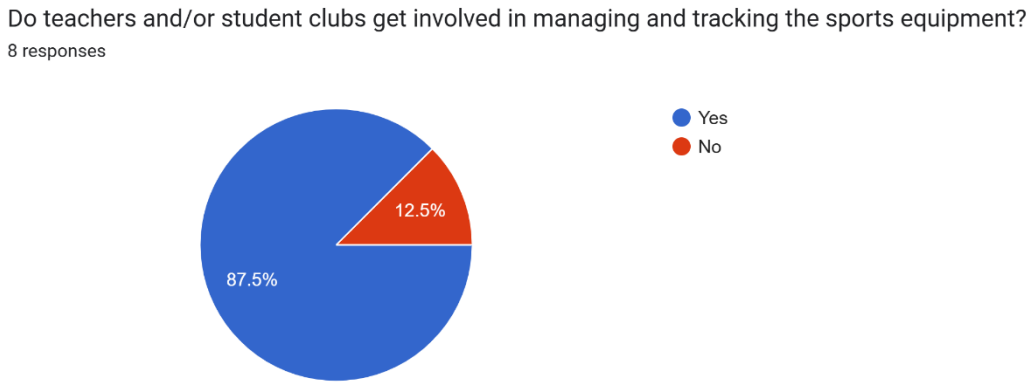


Figure 4.7: Involvement of Teachers or Student Clubs in Equipment Management

Most respondents (87.5%) indicated that teachers and student clubs are involved in managing and tracking sports equipment and only 12.5% shows that there are no teacher and/or student involves in equipment management. This shows that the school encourage a collaborative environment where both students and staff contribute to resource management.

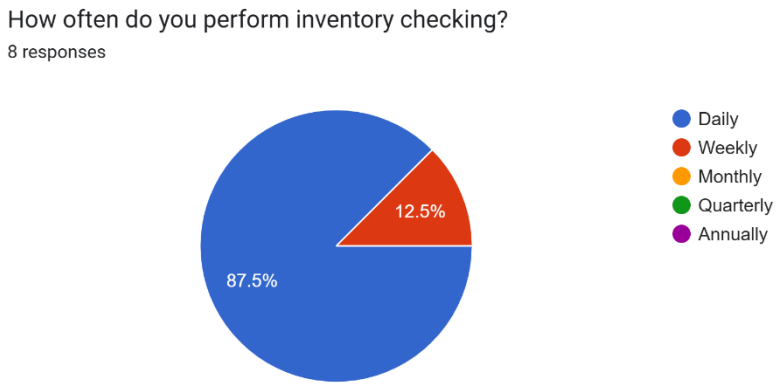


Figure 4.8: Frequency of Inventory Checking

Daily inventory checking is conducted by 87.5% of respondents, while the remaining 12.5% check weekly. This high frequency of checking implies a strong effort to maintain accurate inventory even though the manually process still lead to inefficiencies.

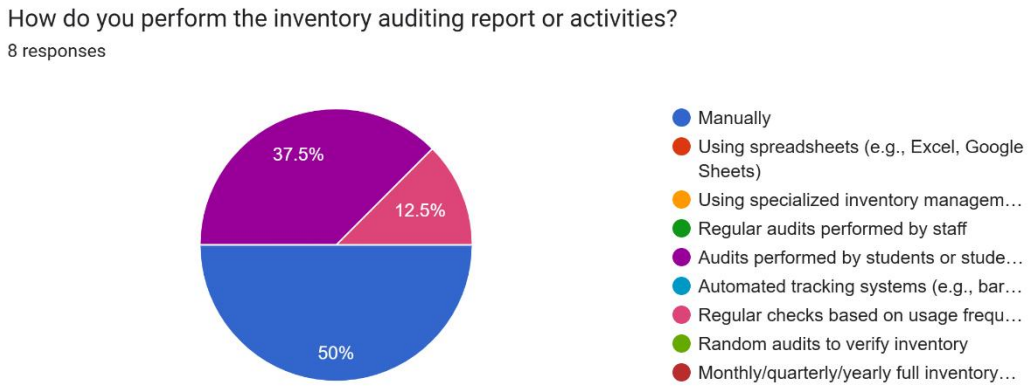


Figure 4.9: Methods Used for Inventory Auditing

Half of the respondents (50%) stated they perform inventory audits manually, while 37.5% rely on audits by student clubs, and 12.5% conduct regular checks based on equipment usage frequency. This shows a lack of formal auditing tools and an over-reliance on labour-intensive methods.

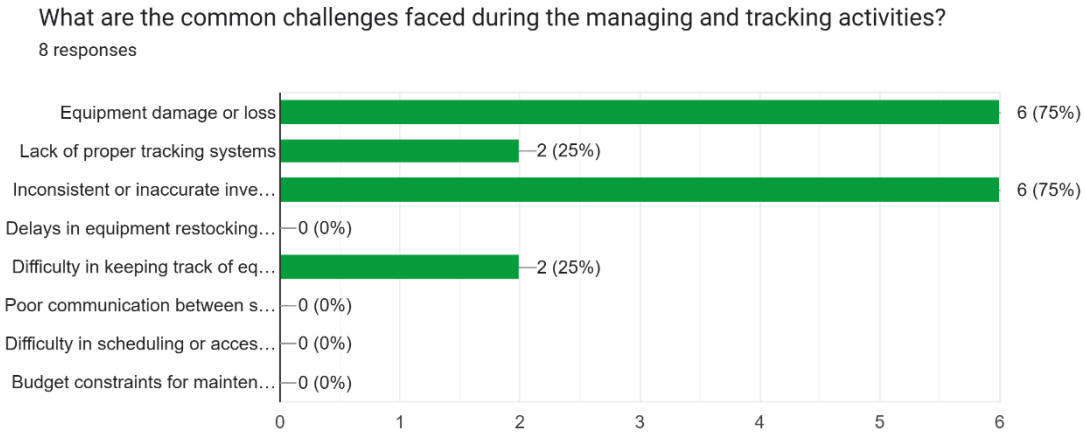


Figure 4.10: Common Challenges in Equipment Management

The most commonly challenges faced were equipment damage or loss which contains 75% of respondents choose this challenges, inconsistent or inaccurate inventory records have also 75% of respondents choose, and difficulty in keeping track of equipment usage only have 25%. These findings suggest critical gaps in the current tracking system that contribute to inefficiency and potential loss of resources.

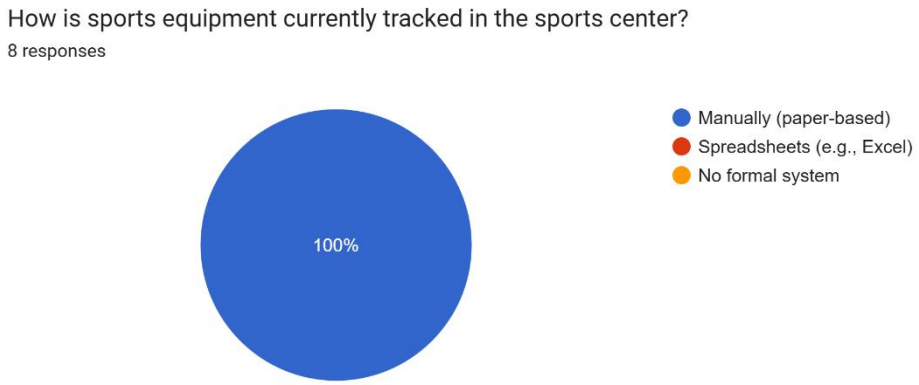


Figure 4.11: Current Method of Tracking Sports Equipment

All respondents (100%) stated that tracking is done manually by using paper-based methods. This stronger mention that the need for a digital solution to improve accuracy, efficiency, and traceability.

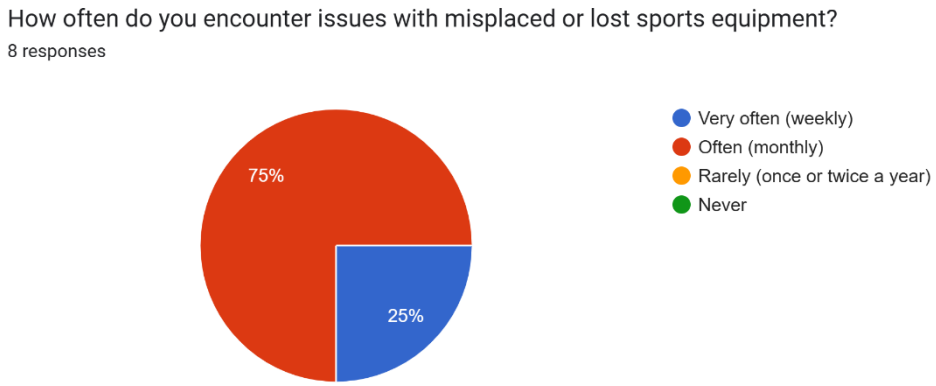


Figure 4.12: Frequency of Misplaced or Lost Equipment

There are 75% of respondents reported that encounter issues with misplaced or lost equipment on a monthly basis, while 25% face this issue weekly. This high frequency mentions the effect of relying on manual systems.

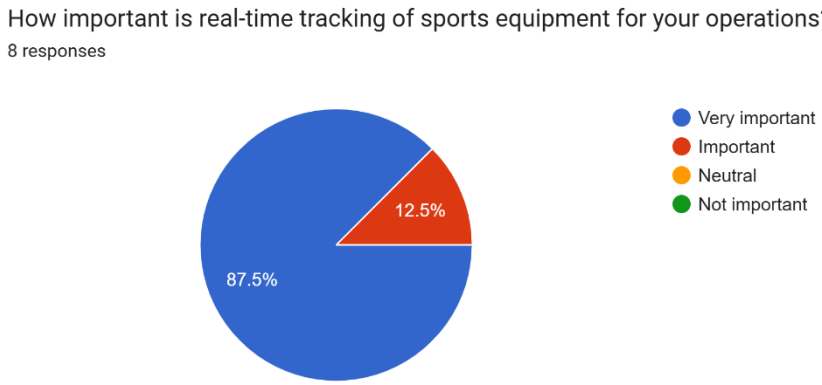


Figure 4.13: Importance of Real-Time Equipment Tracking

There are 87.5% of respondents believe that real-time tracking is very important, with the remaining 12.5% rating it as important. This shows the high demand and need for an updated system that allows the real time tracking of equipment.

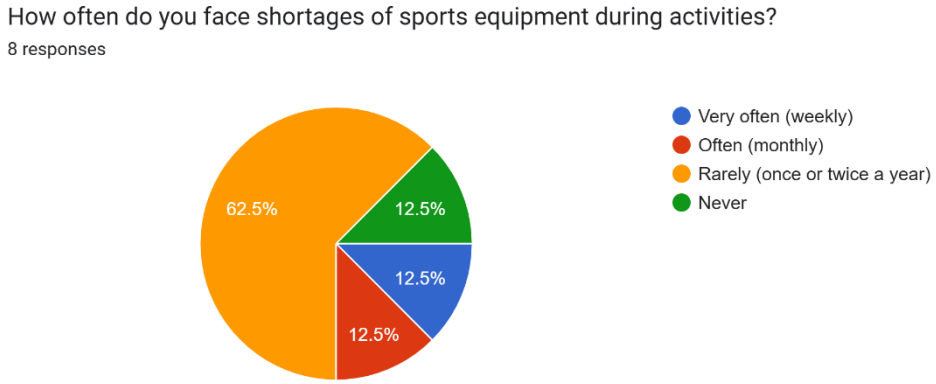


Figure 4.14: Frequency of Equipment Shortages

There are 62.5% of respondents said they rarely experience shortages, while 12.5% said they very often experience shortages. It also has 12.5% of respondents said they are never and often experience shortages.

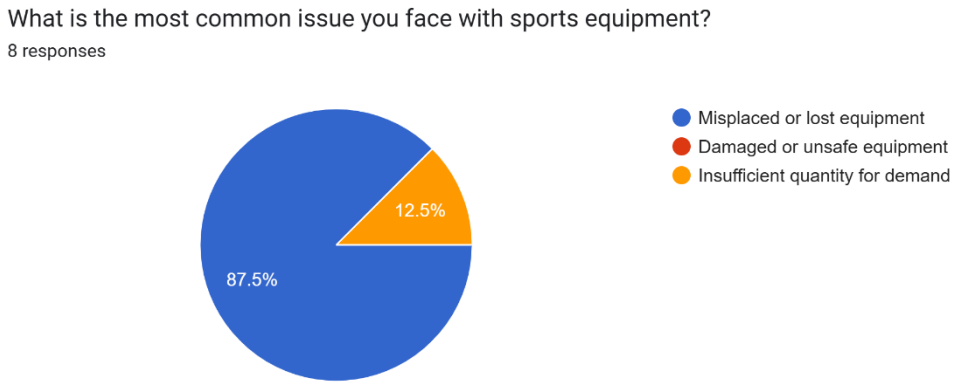


Figure 4.15: Most Common Issue with Sports Equipment

There are 87.5% of respondents reported that they faced misplaced or lost equipment and 12.5% of respondents reported that they faced insufficient quantity for demand.

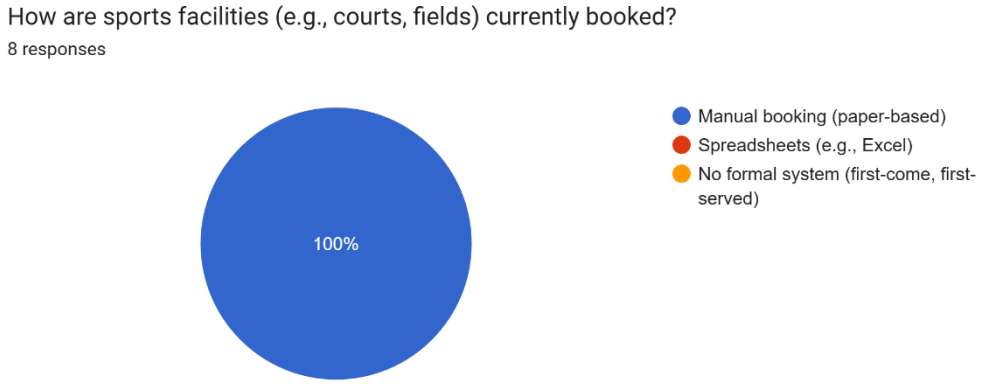


Figure 4.16: Current Method of Booking Sports Facilities

All respondents (100%) reported that sports facilities are booked manually by using paper-based methods. This indicates that there is no digital system in use, and an inventory management system can be implemented to improve efficiency.

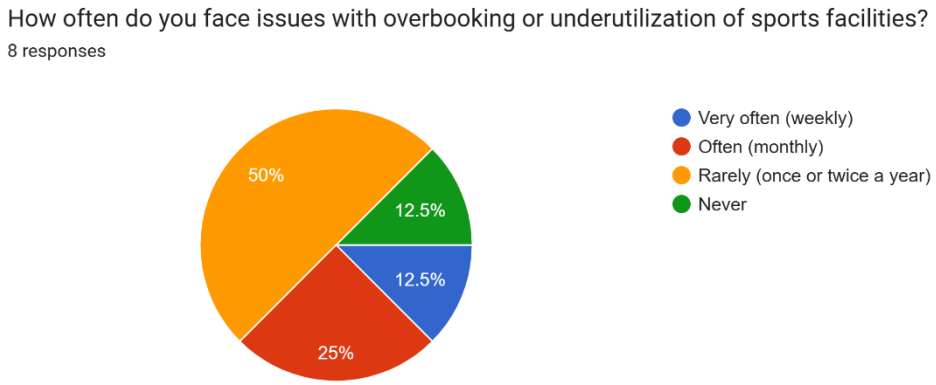


Figure 4.17: Frequency of Booking Conflicts or Underutilization

Half of the respondents (50%) stated they rarely face issues with overbooking or underutilized, 25% of respondents stated they encounter them monthly, 12.5% of respondents experience them monthly and 12.5% of respondents experience them weekly.

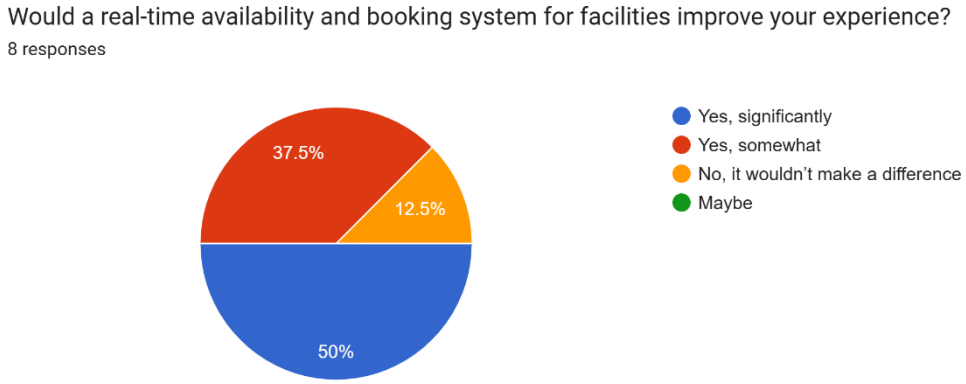


Figure 4.18: Benefits of a Real-Time Booking System

There are 50% of respondents stated that the real-time booking system can improve experience significantly and 37.5% saying somewhat. Only 12.5% of respondents stated that it would not make a difference.

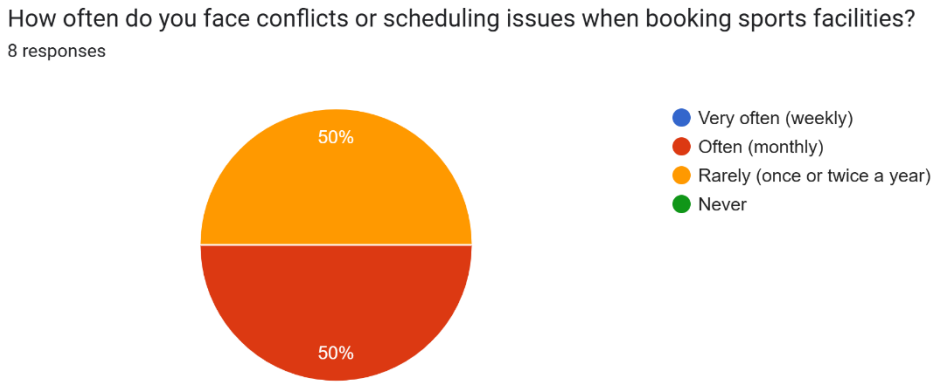


Figure 4.19: Frequency of Scheduling Conflicts

There are 50% of respondents facing scheduling conflicts monthly and 50% of respondents rarely. This shows that booking conflicts occur frequently and there is a need to implement a better system.

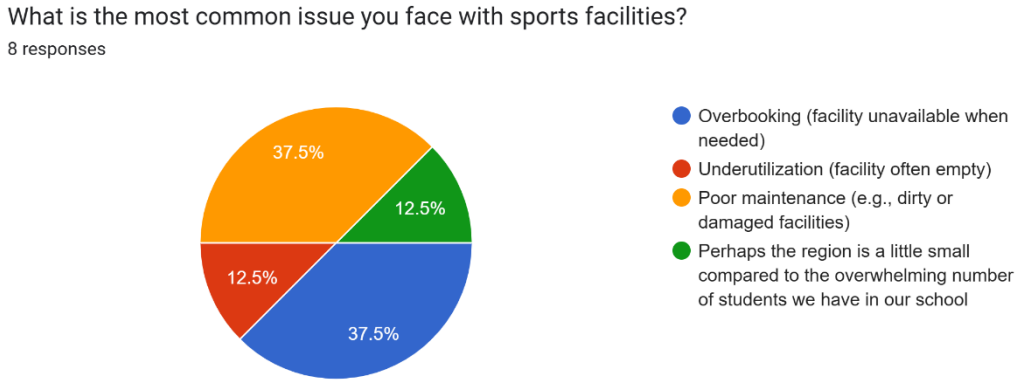


Figure 4.20: Most Common Facility-Related Issues

Overbooking and poor maintenance were each reported by 37.5% of respondents as the most common issues face with sport facilities. 12.5% of respondents shows that the facilities are underutilization and 12.5% of respondents shows that the facilities are too small for the student population.

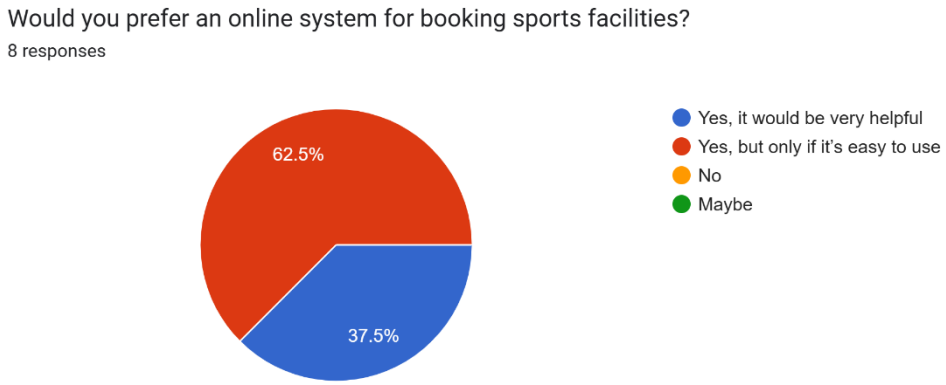


Figure 4.21: Interest in an Online Facility Booking System

All respondents expressed interest in an online booking system, with 62.5% stating it would be very helpful and 37.5% preferring it only if it is easy to use. This suggests that user-friendliness will be key to successful adoption.

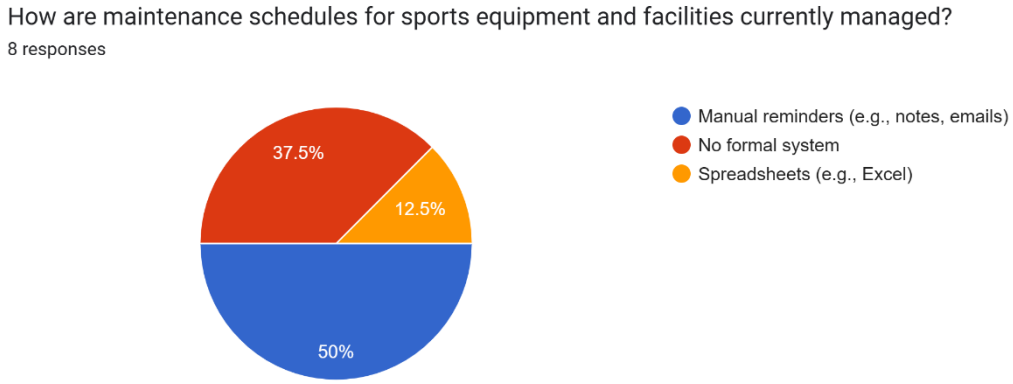


Figure 4.22: Current Method for Managing Maintenance Schedules

Half of the respondents (50%) use manual reminders like notes or emails, 37.5% have no formal system, and only 12.5% use spreadsheets. These results show that maintenance planning is unstructured and need to improve.

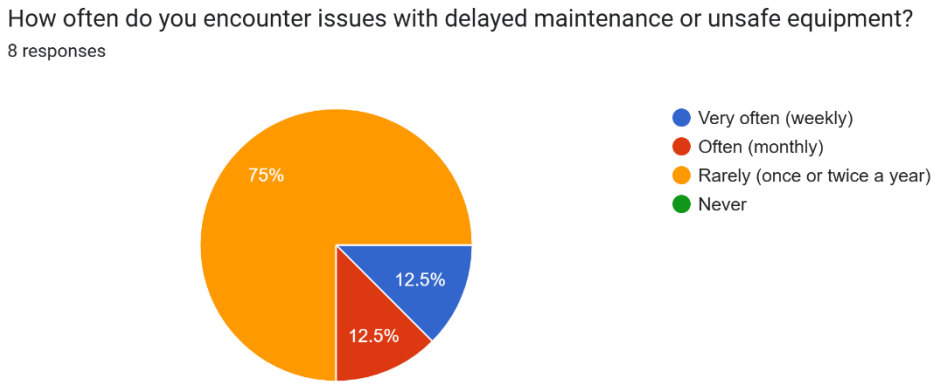


Figure 4.23: Frequency of Delayed Maintenance or Unsafe Equipment

75% of respondents stated that delayed maintenance or unsafe equipment issues are rarely encountered. Each of 12.5% of respondents stated that delayed maintenance or unsafe equipment issues are often and very often encountered.

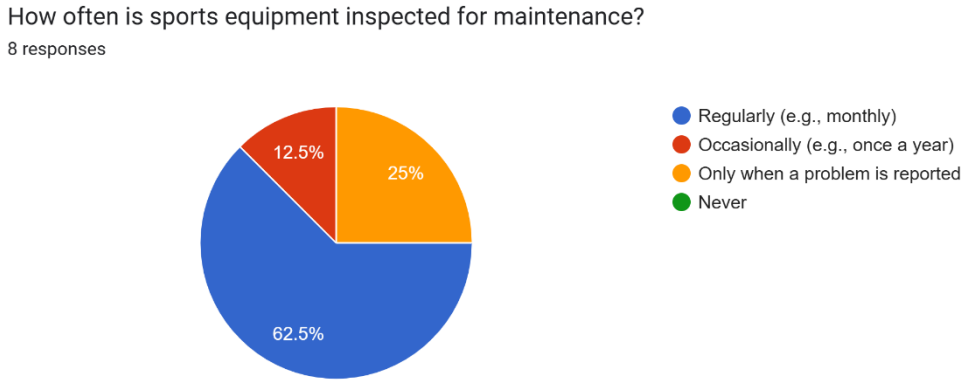


Figure 4.24: Frequency of Equipment Maintenance Inspections

There are 62.5% of respondents stated that regular sport equipment inspected are conducted. 25% of respondents stated that equipment maintenance inspections are only conducted when the problem is reported. 12.5% of respondents stated that equipment maintenance inspections performed regularly.

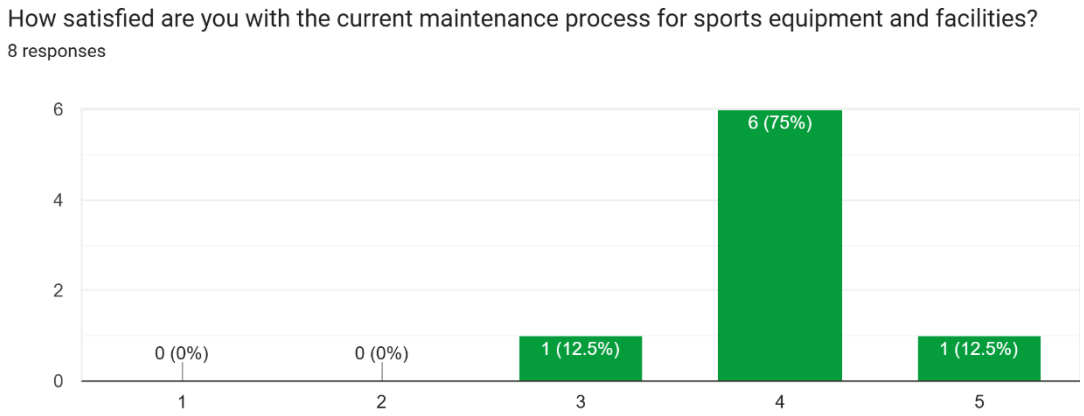


Figure 4.25: Satisfaction Level with the Current Maintenance Process

There are 75% of respondents rated their satisfaction level as 4 out of 5 which shows that they are sarisfied with the current maintenance process for sport equipment and facilities. Only 1 respondent gives lower score of 3 and 1 respondent gives 5.

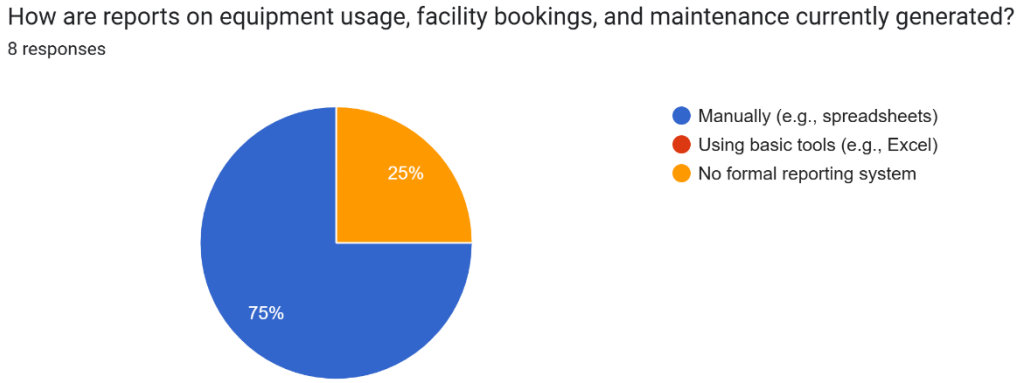


Figure 4.26: Current Method of Report Generation

There are 75% of respondents stated that reports are created manually, while 25% reported having no formal reporting system. This shows that current reporting methods are not standardized and efficient.

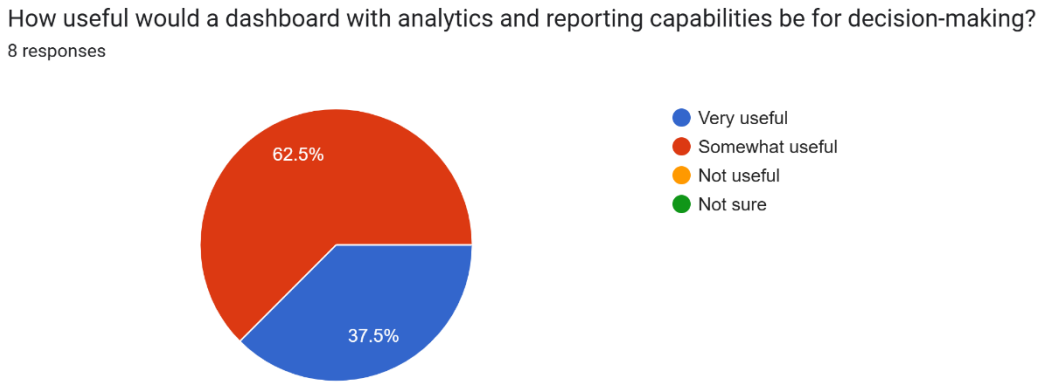


Figure 4.27: Usefulness of a Dashboard

All respondents agreed on dashboard usefulness, with 62.5% rating it as "very useful" and 37.5% as "somewhat useful." This shows a strong interest in using a centralized reporting and decision-making tool.

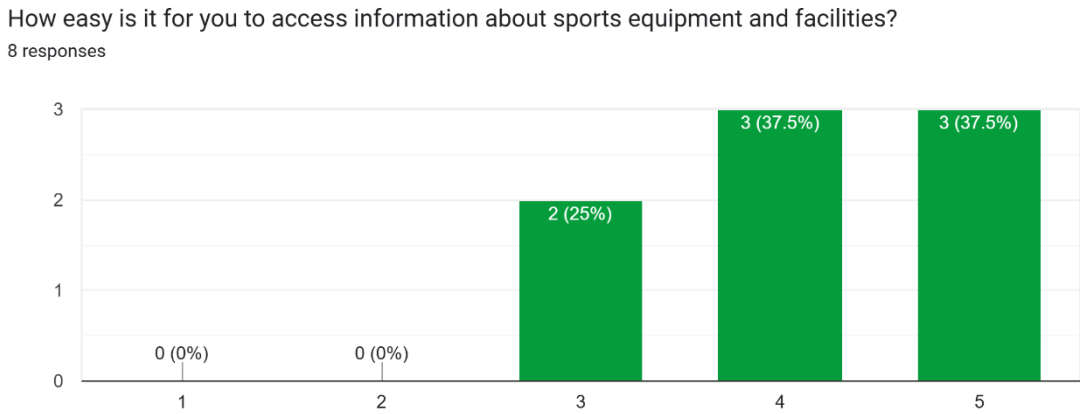


Figure 4.28: Easy Access to Sports Equipment and Facility Information

There are 37.5% of respondents agree that it is easy to access to sports equipment and facilities information with each of them rating it 4 and 5. 25% of respondents choose 3 out of 5 which shows it is normal for them. This shows that the access is good.

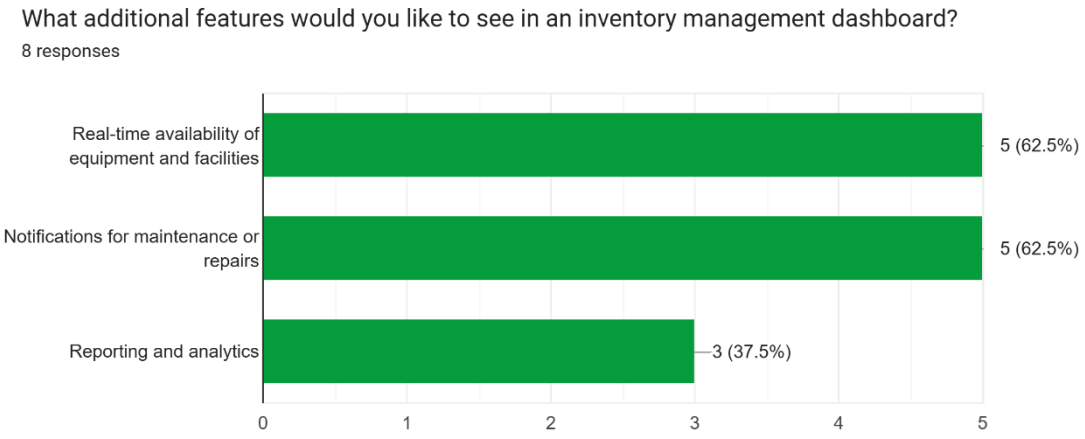


Figure 4.29: Desired Additional Features in a Management Dashboard

Figure 4.29 **s**hows that 62.5% of the respondents desired to see the real time availability of equipment and facility features in an inventory management dashboard. There are also have 62.5% of respondents desired to see the notifications for maintenance or repairs in the inventory management dashboard system. Reporting and analytics feature has 37.5% of respondents desired to implement in the inventory management dashboard system.

4.3 Requirements Specification

4.3.1 Functional Requirements

The functional requirements describe the specific operations the system must perform to fulfill its intended use. These are organized by user roles to clarify the access rights and responsibilities of each type of user: teachers/students, administrators, and quarter masters.

Users (Teacher and Student):

1. Users shall be able to log in using credentials (e.g. username and password) created by the Administrator.
2. Users shall be able to view available equipment.
3. Users shall be able to book equipment
4. Users shall be able to make equipment reservations through the system.
5. Users shall be able to view the status of their own reservations.

6. Users shall be able to view their booking history.
7. Users shall be able to scan a QR codes to access equipment information.
8. Users shall be able to scan a QR code by using mobile device to check equipment in or out directly through the system.
9. Users shall be able to receive notifications about reservation approvals, rejections, booking status and other relevant updates.
10. Users shall be able to log out of the system at any time.

Administrators

1. The Administrator shall be able to log in independently without a created account
2. Administrators shall be able to create, update, edit, and delete user accounts and assign roles (e.g., teacher, student, quarter master)
3. Administrators shall be able to add, edit, update, and delete equipment data with detailed information such as name, type, quantity and location.
4. The Administrator shall be able to scan QR codes to access equipment details or validate check-ins/check-outs.
5. The Administrator shall be able to generate a QR code after creating equipment.
6. The Administrator shall be able to print the generated QR code for existing equipment.
7. The Administrator shall be able to view all reservation and booking requests submitted by users.
8. The Administrator shall be able to approve, reject, or cancel any reservation.
9. The Administrator shall be able to approve, reject, or cancel any booking.
10. The Administrator shall be able to track equipment status, including check-in/check-out history.
11. Administrators shall be able to manually update equipment status to reflect “available,” “rented,” or “under maintenance” or “out of stock”.

12. The Administrator shall be able to perform stock checks to verify and update the current availability and condition of all equipment.
13. The Administrator shall be able to access to dashboard.
14. The Administrator shall be able to log out of the system securely at any time.

Quarter Masters

1. The Quartermaster shall be able to log in using their assigned credentials.
2. The Quartermaster shall be able to view equipment information and status.
3. The Quartermaster shall be able to update the status of equipment.
4. The Quartermaster shall be able to scan QR codes to access equipment records or validate check-ins/check-outs.
5. The Quartermaster shall be able to print QR codes for equipment.
6. The Quartermaster shall not be able to create or delete equipment records.
7. The Quartermaster shall be able to approve, reject, or cancel any booking.
8. The Quartermaster shall be able to approve, reject, or cancel any reservation.
9. The Quartermaster shall be able to track equipment status, including check-in/check-out history
10. The Quartermaster shall be able to perform stock checks to inspect, verify, and update the availability and physical condition of equipment.
11. The Quartermaster shall be able to log out of the system securely at any time.

4.3.2 Non-Functional Requirements

Usability:

1. The system shall provide a user-friendly interface that is easy to navigate for all user roles.
2. The system shall provide clear and descriptive error messages to help users correct input mistakes.
3. The system shall support responsive design for usability on website and mobile devices.

Performance Requirements

1. The system shall respond to user actions within 2 seconds for 95% of interactions.
2. The system shall generate QR codes within 3 second of request.
3. The system shall load equipment inventory and booking data efficiently even if the data set is large.

Security Requirements

1. The system shall require all users to authenticate with a valid username and password before accessing the system.

4.4 Use Case Modelling

4.4.1 Use Case Diagram

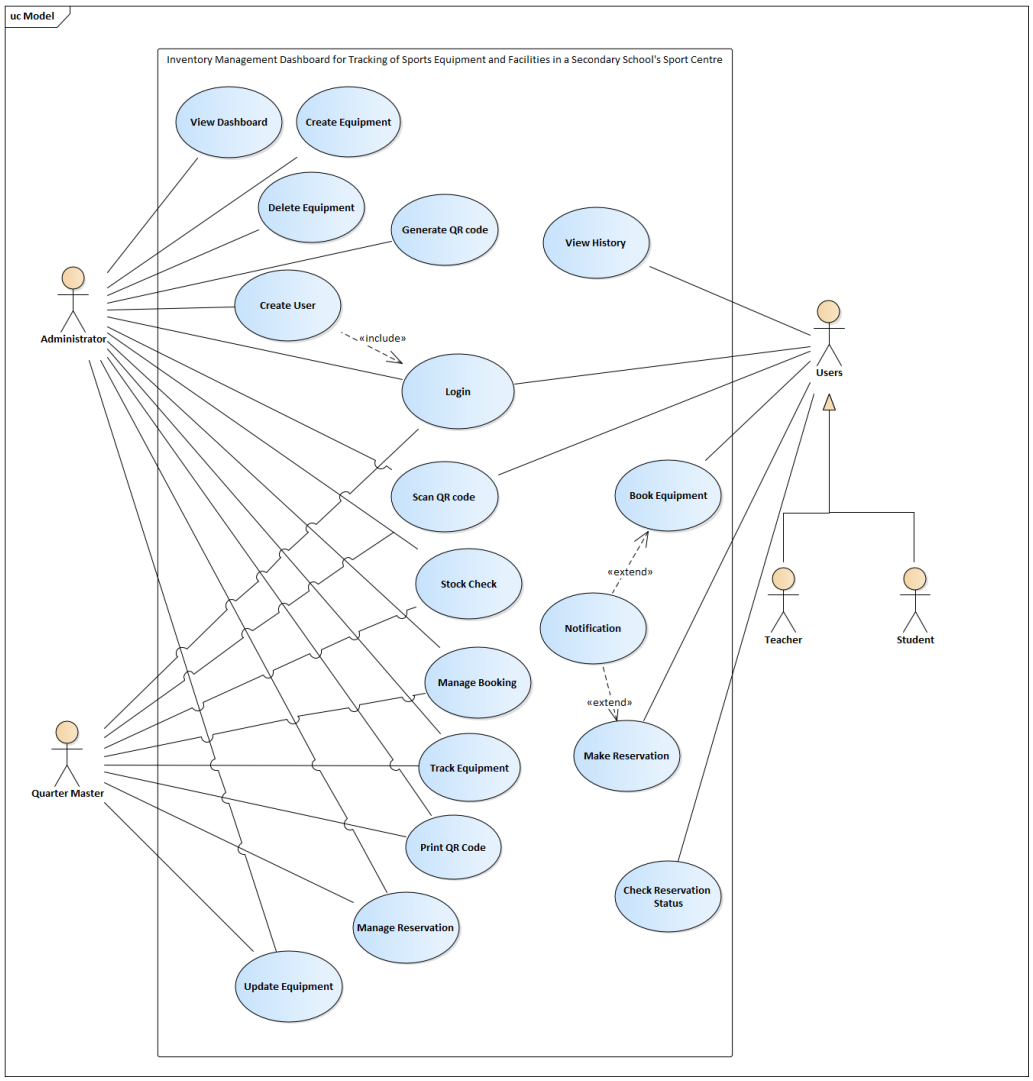


Figure 4.30: Use Case Diagram

4.4.2 Use Case Description

4.4.2.1 Login

<table>
<tr><td colspan="2">Use Case Name: Login</td><td>ID: UC001</td><td>Importance Level: High</td></tr>
<tr><td colspan="2">Primary Actor: Administrator, Quarter master, Users (Teachers & Students)</td><td colspan="2">Use Case Type: Detailed, Essential</td></tr>
<tr><td colspan="4">Stakeholders and Interests:
Administrator, Quarter master, Users (Teachers & Students) – Login to access the system</td></tr>
<tr><td colspan="4">Brief Description: This use case describes the process of login by the administrator, quarter master and users (teacher and student).</td></tr>
<tr><td colspan="4">Trigger: Administrator, Quarter master or Users who want to login and after scanning QR code.</td></tr>
<tr><td colspan="4">Relationships:
Association : Administrator, Quarter master, Users (teachers and students)
Include : Create User
Extend : -
Generalization : -</td></tr>
<tr><td colspan="4">Normal Flow of Events:
1. The user scans the QR code.
2. The user navigates to system login page.
3. The system prompts the customer to enter personal information which includes username and password.
4. The user confirms his/her personal information before submitting. Perform 4.1 and 4.2</td></tr>
</table>

5. The user login successfully and the system redirects users to booking page, equipment management page or admin dashboard page based on the user's role. *Perform 5.1,5.2 and 5.3*
Sub-flows: 4.1 If user entered wrong personal information, the system displays a notification to user indicating that personal information incorrectly. Back to flow no.3. 4.2 If user entered personal information correctly. Flow no.5 continues. 5.1 If user role is administrator, it will redirect to admin dashboard page. 5.2 If user role is quarter master, it will redirect to equipment management page. 5.3 If user role is members, it will redirect to admin dashboard page.
Alternate/Exceptional Flows: A. The user must have a valid login to the system. B. The quarter master and members (teacher and student) only can login after the administrator creates an account for them. C. If account does exist or is inactive, please contact administrator.

4.4.2.2 Create User

Use Case Name: Create User	ID: UC002	Importance Level: High
Primary Actor: Administrator	Use Case Type: Detailed, Essential	
Stakeholders and Interests: Administrator - Create user account for teacher, student or quarter master so that they can access to system.		
Brief Description: This use case describes the process of creating a user by the administrator.		
Trigger: An administrator who wants to create users.		
Relationships:		

Association : Administrator Include : - Extend : - Generalization : -
Normal Flow of Events: 1. The administrator logs into the system. 2. The administrator chooses “create user” button. 3. The system displays a form to enter user details such as username, password and role. 4. The administrator fills in the required information. 5. The system validates the input data. *Perform 5.1,5.2 and 5.3* 6. The system creates the new user account and stores it in the database. 7. The new user account creates successfully.
Sub-flows: 5.1 If administrator fill in the required information correctly and match with the validation. Flow no.6 continues. 5.2 If administrator fill in the required information in a wrong way such as weak password and empty required field. Back to flow no.3. 5.3 If administrator entered the username has already existed, the system will notify the administrator. Back to flow no.3.
Alternate/Exceptional Flows: A. The administrator must have a valid login to the system. B. The username cannot be repeated.

4.4.2.3 View Dashboard

Use Case Name: View Dashboard	ID: UC003	Importance Level: High
Primary Actor: Administrator	Use Case Type: Detailed, Essential	

Stakeholders and Interests: Administrator – view visual representation of data in dashboard
Brief Description: This use case describes the process of viewing dashboard by the administrator.
Trigger: An administrator who wants to view dashboard.
Relationships: Association : Administrator Include : - Extend : - Generalization : -
Normal Flow of Events: 1. The administrator logs into the system. 2. The administrator selects "Dashboard" from the navigation bar. 3. The system retrieves summary data and displays the data visually in charts, table or count. 4. The administrator view dashboard. *Perform 4.1.*
Sub-flows: 4.1 If there are no usage records, the dashboard will display no data available.
Alternate/Exceptional Flows: A. The administrator must have a valid login to the system.

4.4.2.4 Create Equipment

Use Case Name: Create Equipment	ID: UC004	Importance Level: High
Primary Actor: Administrator	Use Case Type: Detailed, Essential	
Stakeholders and Interests: Administrator – wants to create equipment.		
Brief Description: This use case describes the process of creating equipment by the administrator.		

Trigger: An administrator who wants to create equipment.
Relationships: Association : Administrator Include : - Extend : - Generalization : -
Normal Flow of Events: 1. The administrator logs into the system and navigates to Equipment Management. 2. The administrator selects “Create Equipment.”. 3. The system displays a form to enter equipment details such as equipment name, type, quantity, location. 4. The administrator fills in the required information. 5. The system validates the input. *Perform 5.1 and 5.2* 6. The system saves the new equipment record into the database. 7. The equipment creates successfully.
Sub-flows: 5.1 If any required fields are left empty or contain invalid input, the system prompts the administrator an error messages until it fills in correctly. Back to flow no.3. 5.2 If the required fields are validated, flow no.6 continues.
Alternate/Exceptional Flows: A. The administrator must have a valid login to the system.

4.4.2.5 Delete Equipment

Use Case Name: Delete Equipment	ID: UC005	Importance Level: High
Primary Actor: Administrator	Use Case Type: Detailed, Essential	

Stakeholders and Interests:

Administrator – have ability to delete equipment to keep inventory accurate.

Brief Description: This use case describes the process of deleting equipment by the administrator.

Trigger: An administrator who wants to delete equipment.

Relationships:

Association : Administrator

Include : N/A

Extend : N/A

Generalization: N/A

Normal Flow of Events:

1. The administrator login to the system. *Perform 1.1 and 1.2*
2. The administrator navigates to equipment page.
3. The system displays a list of equipment details
4. The administrator selects the equipment item to be deleted.
5. The system prompts the confirmation messages for administrator to delete the equipment.
6. The administrator confirms the deletion.
7. The system shows a confirmation message that the equipment has been successfully deleted.

Sub-flows:

1.1 If the login is valid, the flow no.2 continues.

1.2 If the login is invalid, the system informs the user is not login and prompts them to login again. Then repeat flow no.1.

Alternate/Exceptional Flows:

A. The administrator must have a valid login to the system.
B. If the equipment is currently reserved or checked out, the system prevents deletion.

4.4.2.6 Generate QR code

Use Case Name: Generate QR code	ID: UC006	Importance Level: High
Primary Actor: Administrator	Use Case Type: Detailed, Essential	
Stakeholders and Interests: Administrator – QR code is generated to simplify tracking and identification.		
Brief Description: This use case describes the process of generating QR code by the administrator.		
Trigger: An administrator who wants to generate QR code.		
Relationships: Association : Administrator Include : N/A Extend : N/A Generalization: N/A		
Normal Flow of Events: 1. The administrator logs in to the system. 2. The administrator navigates to "product" section. *Perform 2.1* 3. The administrator clicks "generate QR code" button. 4. The system generates unique QR code based on different equipment.		
Sub-flows: 2.1 If the QR code is missing or damaged, the administrator can re-generate a new one without altering the equipment record.		
Alternate/Exceptional Flows: A. The administrator must have a valid login to the system.		

4.4.2.7 Scan QR code

Use Case Name: Scan QR code	ID: UC007	Importance Level: High
Primary Actor: Users (Teachers and Students), Quarter master, Administrator	Use Case Type: Detailed, Essential	
Stakeholders and Interests: Users (Teachers and Students) – can book the equipment by scanning a QR code.		
Brief Description: This use case describes the process of scanning QR code by the Users (Teachers and Students), Quarter master and Administrator.		
Trigger: A user who wants to scan QR code.		
Relationships: Association : Users (Teachers and Students), Quarter master, Administrator Include : – Extend : – Generalization: –		
Normal Flow of Events: 1. The user open QR code scanning features in mobile. 2. The user scans the QR code attached to the equipment. 3. The system reads the QR code. 4. The user navigates to login page and ask user to login. 5. The user can access to booking page.		
Sub-flows:		
Alternate/Exceptional Flows: A. The user must have a valid login to the system. B. The user must access to the camera.		

C. If the QR code is unreadable or not linked to any equipment, the system displays an error message.

4.4.2.8 Stock Check

Use Case Name: Stock Check	ID: UC008	Importance Level: High
Primary Actor: Administrator, Quarter master	Use Case Type: Detailed, Essential	
Stakeholders and Interests: Quarter master – need to do stock checking for the equipment. Administrator – need to do stock checking for the equipment.		
Brief Description: This use case describes the process of stock checking by the administrator and quarter master.		
Trigger: An administrator and quarter master who wants to do stock checking.		
Relationships: Association : Administrator, Quarter master Include : – Extend : – Generalization: –		
Normal Flow of Events: 1. The quarter master or administrator logs into the system 2. The quarter master or administrator navigates to the “Stock Check” section. 3. The system will display a list of all equipment. 4. The quarter master or administrator needs to calculate the amount of equipment such as how many instock, damage and record it into the system. *Perform 4.1* 5. A stock check result is generated and saved.		
Sub-flows:		

4.1 If the stock check amount is not same with the actual amount, it will prompt error message for administrator.
Alternate/Exceptional Flows: A. The administrator and quarter master must have a valid login to the system.

4.4.2.9 Manage Booking

Use Case Name: Manage Booking	ID: UC009	Importance Level: High
Primary Actor: Administrator, Quarter master	Use Case Type: Detailed, Essential	
Stakeholders and Interests: Administrator and Quarter master – allows to view, approve, reject, or cancel equipment bookings to control equipment usage and resolve scheduling conflicts.		
Brief Description: This use case describes the process of managing booking by the administrator and quarter master.		
Trigger: An administrator and quarter master who wants to manage booking.		
Relationships: Association : Administrator, Quarter master Include : – Extend : – Generalization: –		
Normal Flow of Events: 1. The user login the system. *Perform 1.1 and 1.2* 2. The user navigates to “manage booking” section. 3. The system displays a list of all booking records with statuses, id, name of equipment, check in and check out dates. *Perform 3.1* 4. The user can create, edit, delete, and update booking. 5. The system updates the booking status.		

Sub-flows: 1.1 If the login is valid, flow no.2 continues. 1.2 If the login is invalid, the system informs the user is not login and prompts them to login again. Then repeat flow no.1. 3.1 The sdministrator can filter bookings by status, user, or date range to manage efficiently.
Alternate/Exceptional Flows: A. The user must have a valid login to the system.

4.4.2.10 Track Equipment

Use Case Name: Track Equipment	ID: UC010	Importance Level: High
Primary Actor: Administrator, Quarter master	Use Case Type: Detailed, Essential	
Stakeholders and Interests: Quarter master and administrator – needs to know which items are currently checked out, who owns them, and when they are returned.		
Brief Description: This use case describes the process of tracking equipment by the administrator and quarter master.		
Trigger: An administrator and quarter master who wants to track the equipment.		
Relationships: Association : Administrator, Quarter Master Include : – Extend : – Generalization: –		
Normal Flow of Events: 1. The user logs into the system. 2. The user accesses the “Track Equipment” section.		

3. The system displays a list of all equipment with status aavailable, check in, reserve, checked out, under maintenance, missing or damaged. *Perform 3.1* 4. The user selects an item to view its tracking details.
Sub-flows: 3.1 If an item has no usage or tracking history, the system displays a message indicating that no records are available.
Alternate/Exceptional Flows: A. The administrator and quarter master must have a valid login to the system.

4.4.2.11 Print QR code

Use Case Name: Print QR code	ID: UC011	Importance Level: High
Primary Actor: Administrator	Use Case Type: Detailed, Essential	
Stakeholders and Interests: Administrator – ability to print QR codes when equipment is first added or when a QR is damaged or lost.		
Brief Description: This use case describes the process of printing QR code by the administrator.		
Trigger: An administrator who wants to print the QR code.		
Relationships: Association : Administrator Include : – Extend : – Generalization: –		
Normal Flow of Events: 1. The administrator logs into the system. 2. The administrator navigates to the equipment page. 3. The administrator selects an equipment item.		

4. The administrator clicks “Print QR code” button. 5. The QR code is displayed in a printable format. 6. The administrator prints the QR code.
Sub-flows:
Alternate/Exceptional Flows: A. The administrator must have a valid login to the system. B. If the QR code doesn’t exist, the system prompts the user to generate one before printing.

4.4.2.12 Manage Reservation

Use Case Name: Manage Reservation	ID: UC012	Importance Level: High
Primary Actor: Administrator, Quarter master	Use Case Type: Detailed, Essential	
Stakeholders and Interests: Administrator and quarter master – who allow to approve, reject, or modify reservation requests.		
Brief Description: This use case describes the process of managing reservations by the administrator and quarter master.		
Trigger: An administrator and quarter master who wants to manage a reservation.		
Relationships: Association : administrator, quarter master Include : – Extend : – Generalization: –		
Normal Flow of Events:		

1. The administrator or quarter master can access the reservation of the system. 2. The system displays a list of recent reservations in the system, including username, ID, status and details. *Perform 2.1* 3. The administrator or quarter master selects the order needs to manage. 4. The system displays the details of the selected order which includes quantity. 5. The administrator or quarter master selects an equipment in the order to manage its reservation status, including cancelled, approved, rejected and. *Perform 5.1* 6. The administrator confirms the changes of an order. 7. The system updates the reservation in the database.
Sub-flows: 2.1 The system only displays the list of orders for this month. 5.1 If a customer requests to cancel the order, the administrator needs to cancel the order and update the order status again.
Alternate/Exceptional Flows: A. The administrator and quarter master must have a valid login to the system.

4.4.2.13 Update Equipment

Use Case Name: Update Equipment	ID: UC013	Importance Level: High
Primary Actor: Administrator, Quarter master	Use Case Type: Detailed, Essential	
Stakeholders and Interests: N/A		
Brief Description: This use case describes the process of updating equipment by the customer.		
Trigger: An administrator and quarter master who wants to update the equipment.		

Relationships: Association : Administrator, Quarter master Include : – Extend : – Generalization: –
Normal Flow of Events: 1. The user needs to login to the system.*Perform 1.1 and 1.2* 2. The user access to the "equipment section". 3. The system displays a list of equipment data, including name, id quantity, status and location. 4. The user chooses the equipment he/she wants to modify. 5. The user edits the information. 6. The user confirms the information update after the information is modified. 7. The system updates the equipment information in the database. 8. The system displays new equipment information in the system.
Sub-flows: 1.1 If the login is valid, flow no.2 continues. 1.2 If the login is invalid, the system informs the user is not login and prompts them to login again. Then repeat flow no.1.
Alternate/Exceptional Flows: A. The administrator or quarter master must have a valid login to the system.

4.4.2.14 View History

Use Case Name: View History	ID: UC014	Importance Level: High
Primary Actor: Users (teacher and student)	Use Case Type: Detailed, Essential	
Stakeholders and Interests: N/A		
Brief Description: This use case describes the process of viewing history by the users.		

Trigger: A user who wants to view History.
Relationships: Association : Users Include : – Extend : – Generalization: –
Normal Flow of Events: 1. The member chooses the view history option. 2. The system displays the booking history list.
Sub-flows:
Alternate/Exceptional Flows: A. The users must have a valid login to the system.

4.4.2.15 Book Equipment

Use Case Name: Book Equipment	ID: UC015	Importance Level: High
Primary Actor: Users (teacher and student)	Use Case Type: Detailed, Essential	
Stakeholders and Interests: N/A		
Brief Description: This use case describes the process of booking equipment by the users.		
Trigger: A user who wants to book equipment.		
Relationships:		

Association : Users Include : Notification Extend :- Generalization: -
Normal Flow of Events: 1. The user scans the QR code that attach on specific equipment. 2. The system asks user to login. *Perform2.1 and2.2.* 3. The system displays the equipment name and check in or check out button. 4. User needs to select the quantity of the equipment. 5. User clicks the check in /check out button.
Sub-flows: 2.1 If the login is valid, flow no.3 continues. 2.2 If the login is invalid, the system informs the user is not login and prompts them to login again. Then repeat flow no.2.
Alternate/Exceptional Flows: A. The user must have a valid login to the system.

4.4.2.16 Make Reservation

Use Case Name: Make Reservation	ID: UC016	Importance Level: High
Primary Actor: Users (teacher and student)	Use Case Type: Detailed, Essential	
Stakeholders and Interests: -		
Brief Description: This use case describes the process of making reservations by the users.		
Trigger: A user who wants to make a reservation.		
Relationships: Association : Users (teacher and student)		

Include : - Extend : Notification Generalization: -
Normal Flow of Events: 1. The user logs into the system and choose the “Reserve Equipment” button. 2.The system will display the equipment name, quantity, date and time and reserve button. 3. The user selects the quantity of equipment need, date and time that he/she prefers. 4. The user clicks the make reservation button. 5. The reservation request is submitted and display successful message and display pending status in history page.
Sub-flows:
Alternate/Exceptional Flows: A. The user must have a valid login to the system.

4.4.2.17 Notification

Use Case Name: Notification	ID: UC017	Importance Level: High
Primary Actor: Users (teacher and student)	Use Case Type: Detailed, Essential	
Stakeholders and Interests: N/A		
Brief Description: This use case describes the process of sending notification to the users.		
Trigger: A system event occurs (e.g., reservation approved, reservation rejected).		
Relationships: Association : Users(teacher and student)		

Include : – Extend : – Generalization: –
Normal Flow of Events: 1. The member can login to the system. 2. Once the member login, he/she will receive notification about his/her booking or reservation status.
Sub-flows:
Alternate/Exceptional Flows: A. The administrator must have a valid login to the system. B. Notification will be trigger when user make booking and reservation.

4.4.2.18 View Reservation Status

Use Case Name: View Reservation Status	ID: UC018	Importance Level: High
Primary Actor: Users (Teacher and Student)	Use Case Type: Detailed, Essential	
Stakeholders and Interests: Users – who want to view the reservation status.		
Brief Description: This use case describes the process of view reservation status by the user.		
Trigger: A user who wants to view reservation status.		
Relationships: Association : Users (Teacher and Student) Include : – Extend : – Generalization : –		

Normal Flow of Events: 1. The user scan QR code. 2. The user login the system. *Perform 2.1 and 2.2* 3. The user chooses “my reservation”. 4. The user retrieves the user submitted reservations from database. 5. The user displays the reservation status such as approved or rejected. 6. The user views the reservation status in history.
Sub-flows: 2.1 If the login is valid, flow no.3 continues. 2.2 If the login is invalid, the system informs the user is not login and prompts them to login again. Then repeat flow no.2.
Alternate/Exceptional Flows: A. The user must have a valid login to the system. B. The reservation record show in history.

4.5 Prototype Screenshot

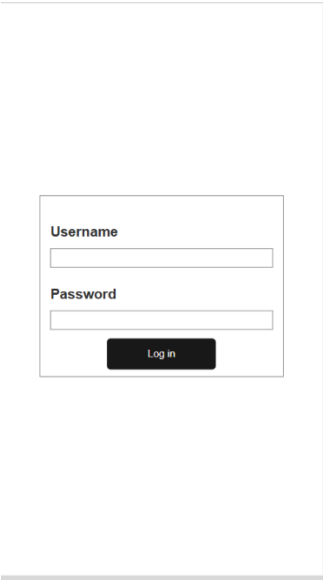


Figure 4.31: Login Module

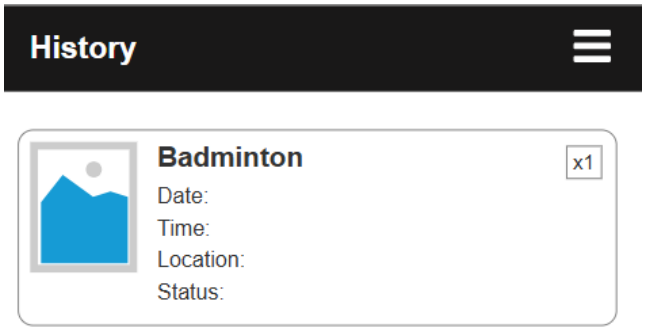


Figure 4.32: View History Module

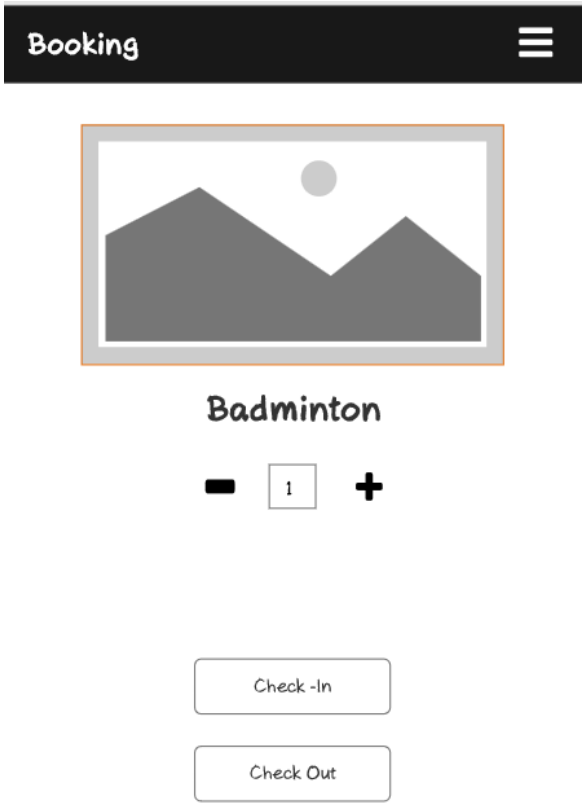


Figure 4.33: Booking Module

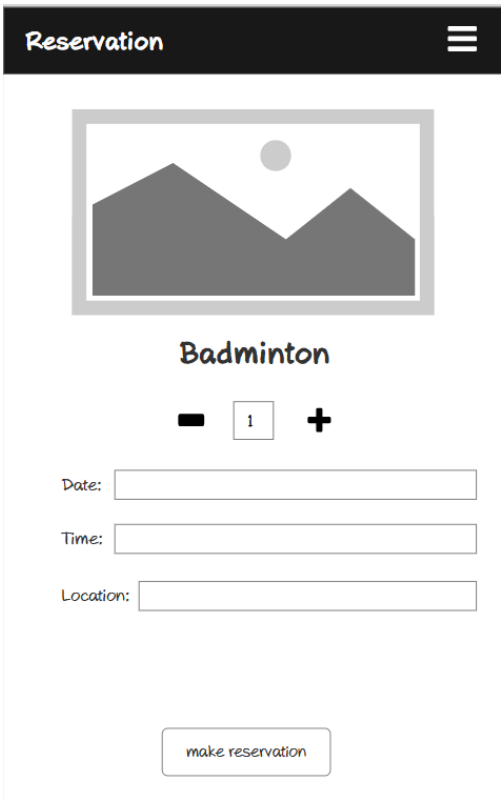


Figure 4.34: Make Reservation Module

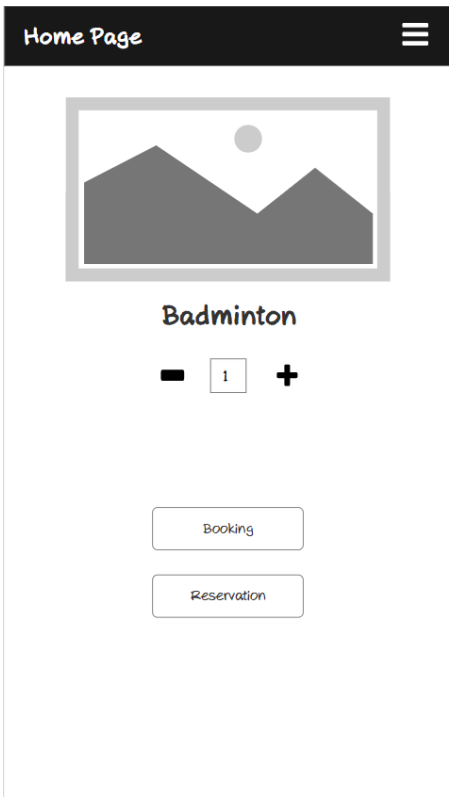


Figure 4.35: Home Page

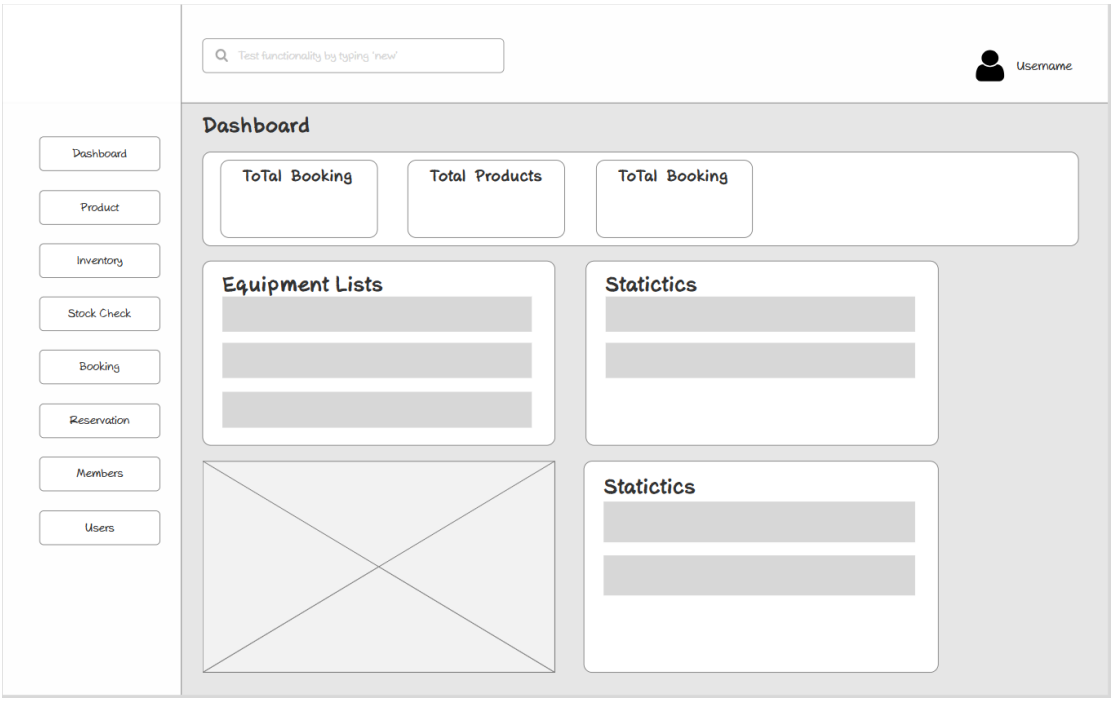


Figure 4.36: Dashboard Module

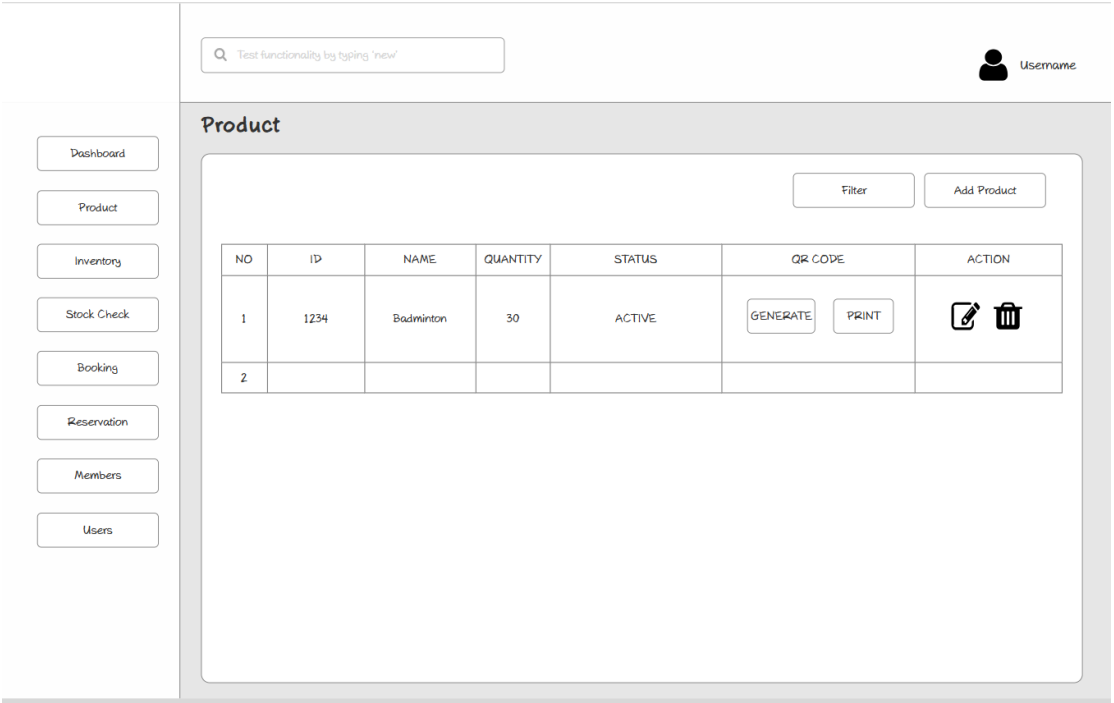


Figure 4.37: Product Management Module

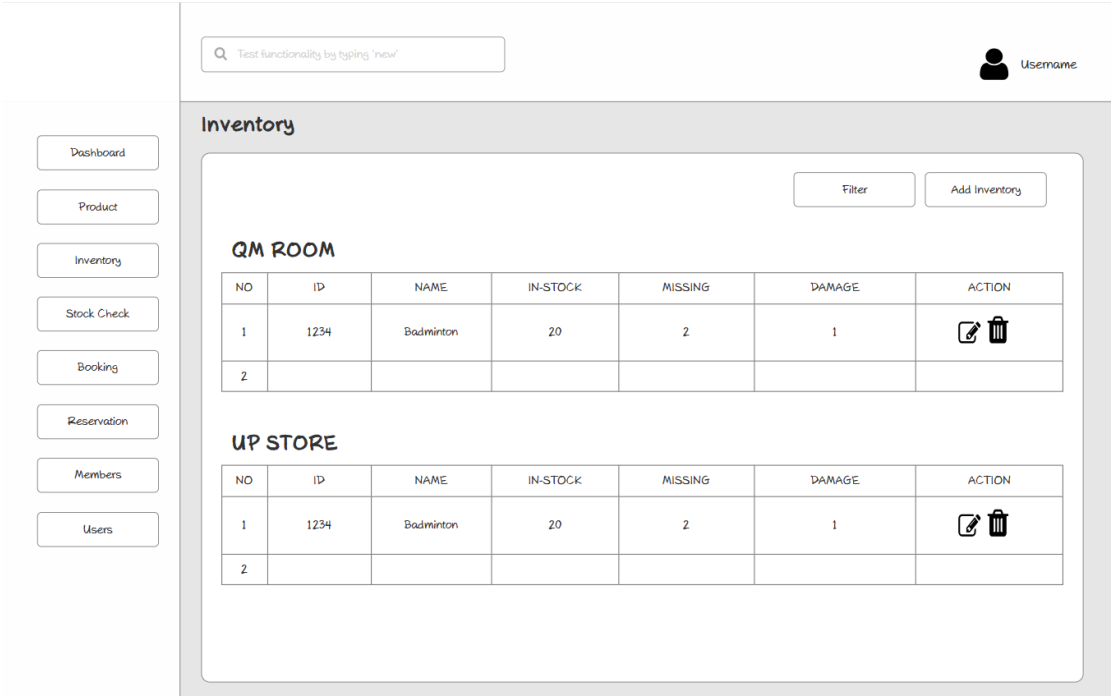


Figure 4.38: Inventory Management Module

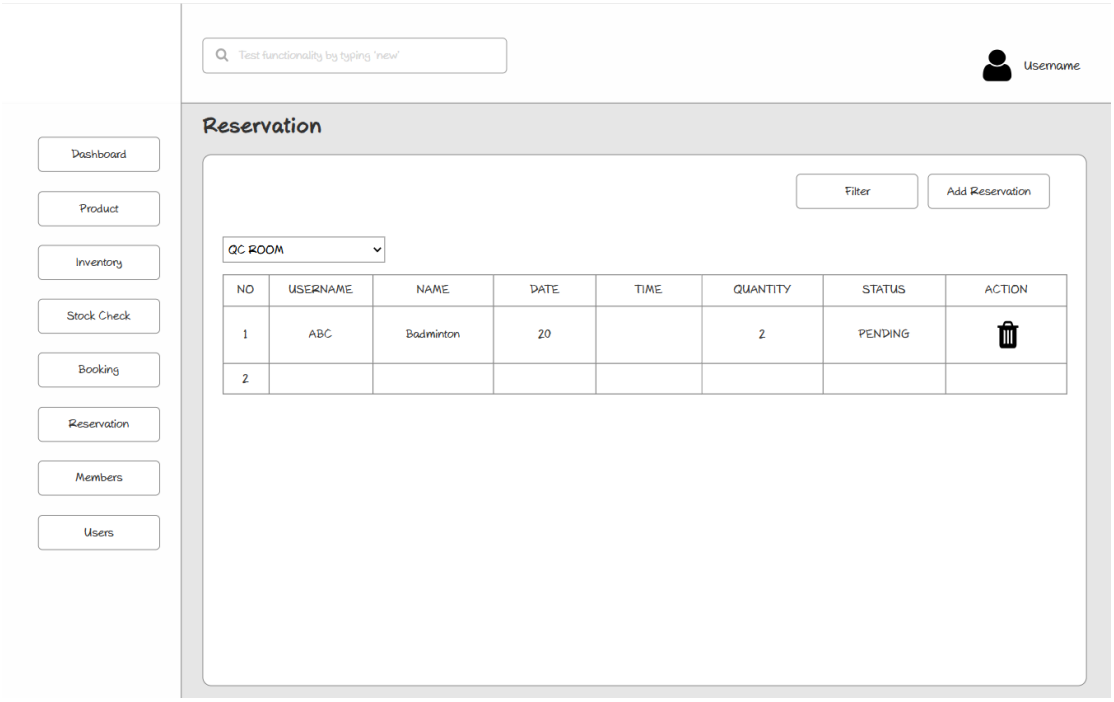


Figure 4.39: Reservation Management Module

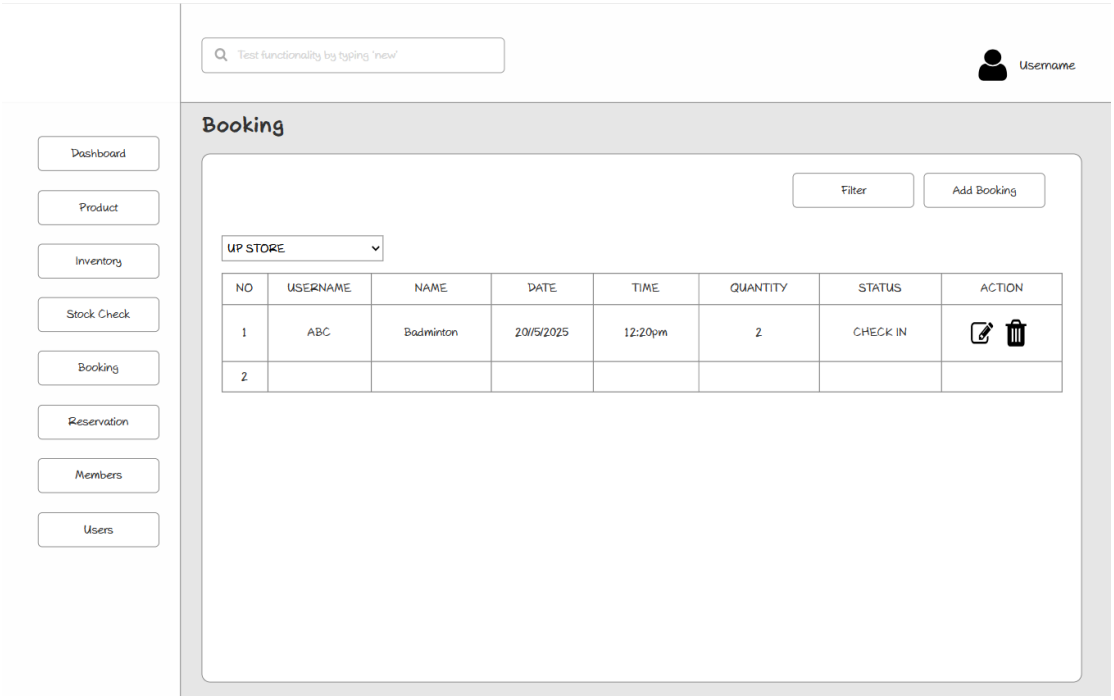


Figure 4.40: Booking Management Module

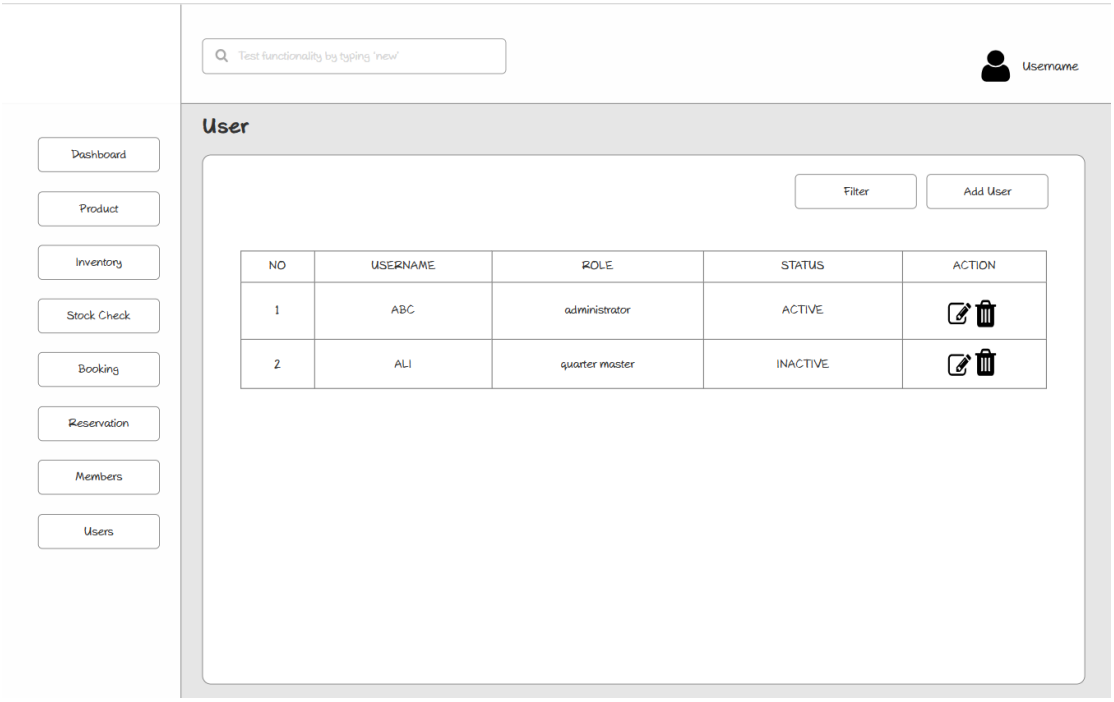


Figure 4.41: User Management Module

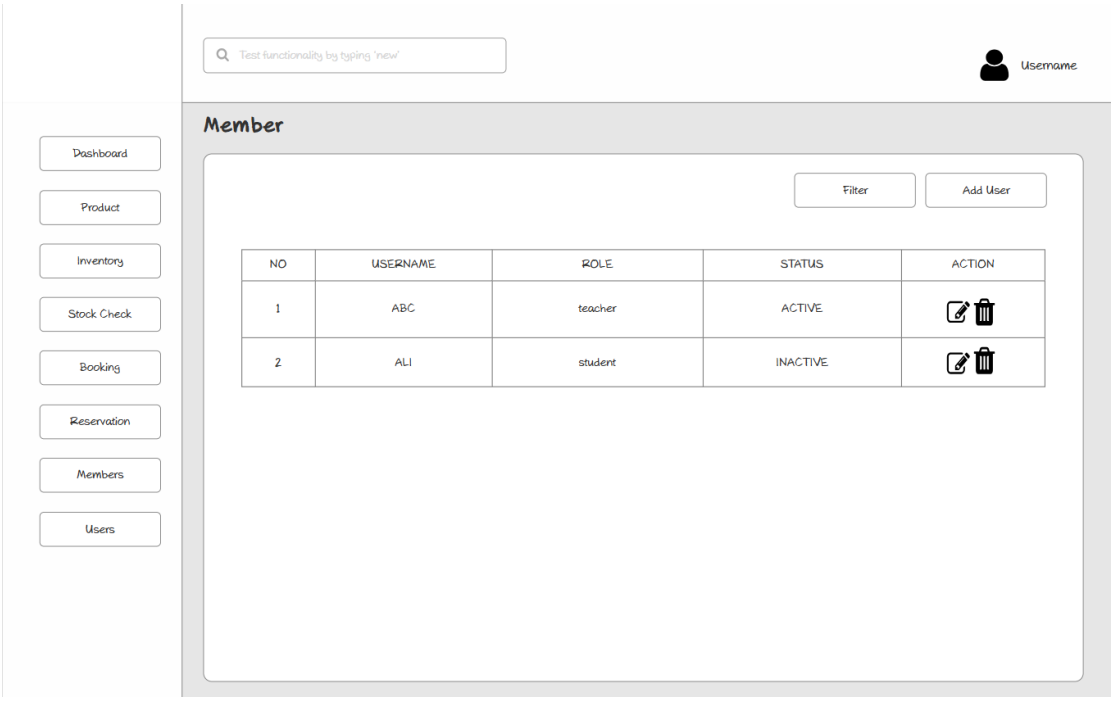


Figure 4.42: Member Management Module

CHAPTER 5

SYSTEM DESIGN

5.1 Introduction

This chapter describes the system design for the Sports Centre Management System that covers the application architecture, database design, data modelling (ERD), data dictionary and data flow diagram (DFD). The design adopts 3-tier architecture to manage users/members, products, bookings and so on, which defines the core entities and relationships. This chapter will explain the system architecture, model the data using an ERD and Data Dictionary derived from tables, and illustrate the system's processes with a Data Flow Diagram (DFD). The design ensures the system meets its goal of efficient tracking for sports equipment and facilities.

5.2 System Architecture Design

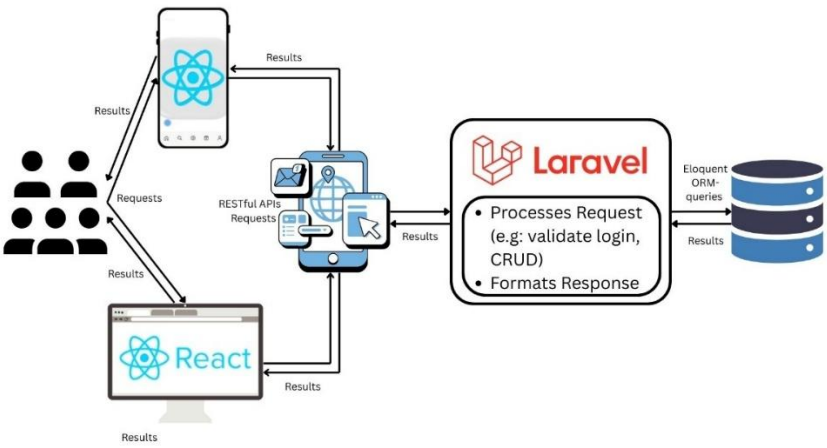


Figure 5.1: Overview of System Architecture Design

The system architecture implemented in this project follows a three-tier structure consisting of the presentation layer, the application layer, and the data layer.

5.2.1 Presentation Layer

The presentation layer is developed using ReactJS, which is a JavaScript library creates a dynamic and responsive user interfaces for both web and mobile devices. It builds as a Single Page Application (SPA) that allows user seamless navigation and real-time updates without full page reloads. The ReactJS frontend can communicate with the Laravel backend to allow efficient data exchange through RESTful APIs using JSON. React.js is a component-based architecture starts with the root component that renders entire application and enhances maintainability and scalability through reusable UI components such as user authentication, inventory management, reservations, and booking.

JSX allows developers to write HTML-like syntax within JavaScript and simplifies the components creation which is used to design user interfaces. The Virtual DOM also optimizes rendering performance to ensure a responsive experience that update in real time. UI library likes Ant Design or Material UI can act as third-party libraries integrated into React.js components as it provides pre-built components for polished UI. This ensures consistency and responsive across web and mobile access.

5.2.2 Application Layer

The application layer is developed using Laravel on WAMPServer. It processes requests from the frontend, perform CRUD (create, read, update, delete) rules, validates data, and interacts with the database. It is a Model-View-Controller (MVC) architecture that handles server-side logic which separates into three different layers to ensure the scalability and maintainability of system (GeeksforGeeks, 2023):

- **Model**: Manages the data in the MySQL database using Laravel's Eloquent ORM.
- **View**: Provides the response in JSON format that the frontend (ReactJS) can interpret and render.
- **Controller**: Handles incoming requests, applies business rules, validates data, and coordinates communication between the model and the view.

Laravel also provides built-in features such as authentication, middleware, and role-based access control. Role-based access control is used to differentiate permission between users and members to ensure secure and each type of user and member has

corrected permissions access to system functionalities. For example, administrators can manage the entire system, but the quarter master can only manage several functions of the system.

The Laravel backend interacts with a MySQL database hosted on WAMPServer. Eloquent ORM also simplifies database interactions with MySQL as it allows flexible PHP code for querying and updating data. Laravel's built-in security features also protect against vulnerabilities like SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF) (Discipline Infotech, 2023). The backend also processes frontend requests through RESTful APIs, such as retrieving device details, handling bookings, or generating QR codes. Laravel's Artisan command-line tool automates tasks like database migration and seeding which simplifies the development workflow.

5.2.3 Data Layer

The data layer is developed using MySQL, a reliable open-source relational database management system, hosted on WampServer for local development and testing. WampServer provides an integrated stack of Apache, MySQL, and PHP that allows local testing without relying on external servers (Y, 2018). The phpMyAdmin included with WampServer offers a web-based MySQL management interface, facilitating operations such as table creation, data inspection, and query execution. Laravel's Eloquent ORM also supports high-performance operations to ensure efficient interaction with MySQL. The database is intended to manage massive datasets and maintain responsiveness while achieving efficient operation through optimised queries.

5.3 Modelling Diagram

5.3.1 Entity Relationship Diagram (ERD)

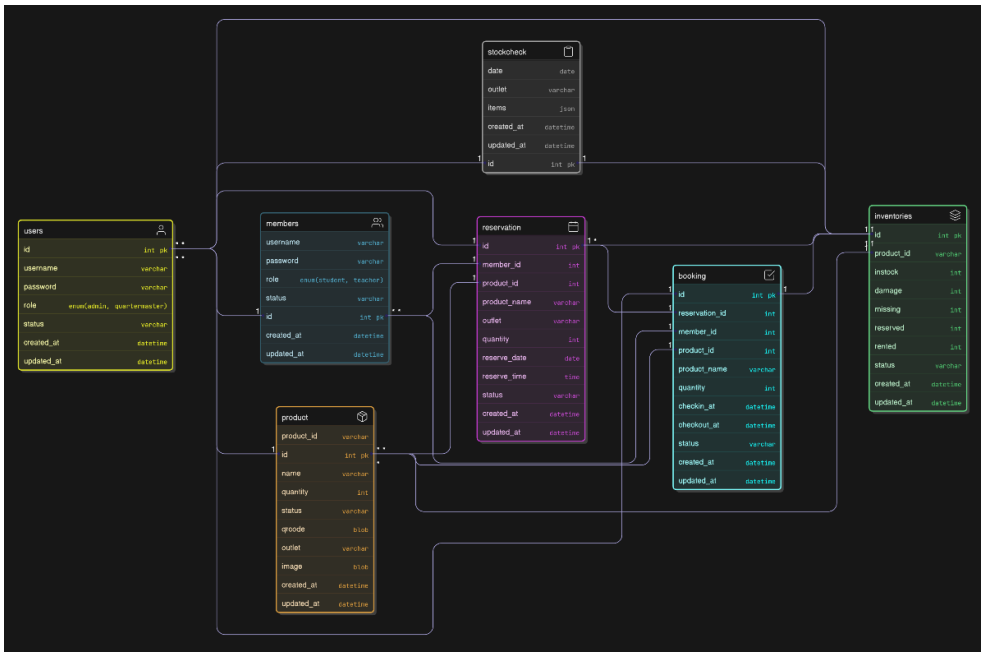


Figure 5.2: Entity Relationship Diagram

5.3.2 Entity Relationship

The purpose of entity relationship is to provide a clear and comprehensive definitions for each entity within the database. It facilitates an understanding of the function, attributes, and entity relations to ensure a shared comprehension of the database's structure.

Table 5.1: Entities Description Table

Entity	**Description**
Users	Stores all the user (administrator / quarter master) details including login credentials
Members	Stores all the member (student / teacher) details including login credentials
Products	Stores all the products
Inventories	Stores all the reservations made by members for future pickup or usage.
StockChecks	Stores all the periodic stock check details.
Bookings	Stores all the booking and accepted reservation details
Reservation	Stores all the reservation details

5.3.3 Data Dictionary

Data dictionary is a centralized repository that store detailed information about the data used in a database system (Uhrowczik, 1973). It specifies and explains the structure, format, and meaning of each data element, including tables, fields, data types, relationships, and constraints. The data dictionary serves as a reference for developers, database administrators, and users. It promotes consistency, accuracy, and a clear understanding of how data is organized and managed within the system. It plays a crucial role in database design, maintenance, and documentation.

Table 5.2: Data Dictionary for Product Table

Column Name	**Description**	**Data Type**	**Key**	**Nullable**
id	Unique identifier for product	Bigint (20)	Primary	No
product_id	Product code	varchar(10)	Unique	No
name	Product name	varchar (255)	-	No
qrcode	QR code data	longtext	-	No
quantity	Product quantity	int (11)	-	No
outlet	Outlet name	Varchar (255)	-	no
image	Product image path	Varchar (255)	-	no
Status	Product status (active, inactive)	enum	-	no
created_at	Record creation timestamp	timestamp	-	Yes
updated_at	Record update timestamp	timestamp	-	Yes

Table 5.3: Data Dictionary for Booking Table

Column Name	Description	Data Type	PK/FK	Nullable
id	Unique identifier for booking	Bigint (20)	Primary	No
product_id	Product identifier	varchar (255)	-	No
product_name	Name of the product booked	varchar (255)	-	No
reservation_id	References a related reservation	Bigint (20)	Foreign (reservations)	Yes
member_id	References the member making the booking.	Bigint (20)	Foreign (members)	No
username	Username of member	varchar (255)	-	No
quantity	Quantity booked	int (11)	-	No
checkin_at	Date & time of check-in	timestamp	-	Yes
checkout_at	Date & time of check-out	timestamp	-	Yes
status	Booking status (accepted, checkin, checkout, closed)	Enum ('accepted', 'checkin', 'checkout', 'closed')	-	No
created_at	Record creation timestamp	timestamp	-	Yes
updated_at	Record update timestamp	timestamp	-	Yes

Table 5.4: Data Dictionary for Inventories Table

Column Name	Description	Data Type	PK/FK	Nullable
id	Unique identifier for each inventory record	Bigint (20)	Primary	No
product_id	Product identifier	varchar (255)	-	No
instock	Number of items available	int (11)	-	No
damage	Number of damaged items	int (11)	-	No
missing	Number of missing items	int (11)	-	No
reserved	Number of reserved items	int (11)	-	No
rented	Number of rented items	Int (11)	-	No
status	Inventory status (active, inactive)	Enum ('active', 'inactive'	-	No
created_at	Record creation timestamp	timestamp	-	Yes
updated_at	Record update timestamp	timestamp	-	Yes

Table 5.5: Data Dictionary for Reservation Table

Column Name	**Description**	**Data Type**	**Key**	**Nullable**
Id	Unique identifier for reservation	Bigint (20)	Primary	No
Member_id	Member making reservation	Bigint (20)	Foreign (members)	No
Username	Username of member	Varchar (255)	-	No
Product_id	Product identifier	Bigint (20)	-	No
Product_name	Product name	Varchar (255)	-	No
Outlet	Outlet name	Varchar (255)	-	Yes
Quantity	Quantity reserved	Int (11)	-	No
Reserve_date	Reservation date	Date	-	No
Reserve_time	Reservation time	Time	-	No
status	Reservation status (pending, accepted, rejected)	Enum ('pending', 'accepted', 'rejected')	-	No
Created_at	Record creation timestamp	Timestamp	-	Yes
Updated_at	Record update timestamp	timestamp	-	Yes

Table 5.6: Data Dictionary for Stockcheck Table

Column Name	Description	Data Type	Key	Nullable
Id	Unique identifier for stock check	Bigint (20)	Primary	No
Date	Date of stock check	date	-	No
Outlet	Outlet name	Varchar (255)	-	No
items	List of items checked (JSON format)	Json	-	No
Created_at	Record creation timestamp	Timestamp	-	Yes
Updated_at	Record update timestamp	Timestamp	-	Yes

Table 5.7: Data Dictionary for Users Table

Column Name	Description	Data Type	Key	Nullable
Id	Unique identifier for user	Bigint (20)	Primary	No
Username	Login username	Varchar (191)	-	No
Password	Encrypted password	Varchar (191)	-	No
Role	Role of user (administrator, quarter master)	Varchar (191)	-	No
status	account status (Active, Inactive)	Varchar (191)	-	No
Created_at	Record creation timestamp	Timestamp	-	Yes
Updated_at	Record update timestamp	Timestamp	-	Yes

Table 5.8: Data Dictionary for Members Table

Column Name	Description	Data Type	Key	Nullable
Id	Unique identifier for member	Bigint (20)	Primary	No
Username	Login username	Varchar (255)	-	No
Password	Encrypted password	Varchar (255)	-	No
Role	Role of member (student, teacher)	ENUM('student', 'teacher')	-	No
status	account status (Active, Inactive)	ENUM ('active', 'inactive')	-	No
Created_at	Record creation timestamp	Timestamp	-	Yes
Updated_at	Record update timestamp	Timestamp	-	Yes

5.4 User Interface Design

5.4.1 Login Module

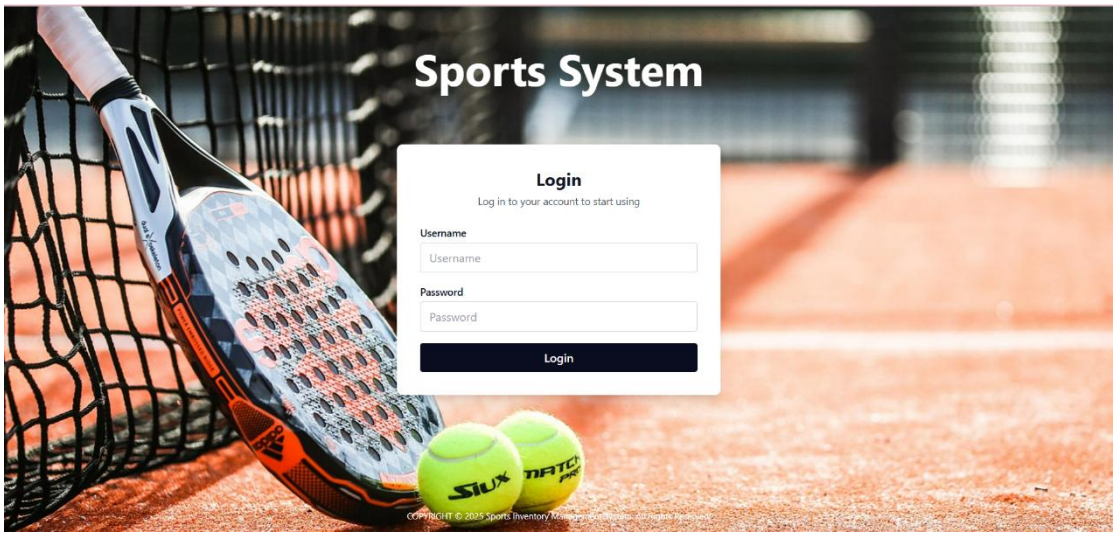


Figure 5.3: Login page

5.4.2 Dashboard Module

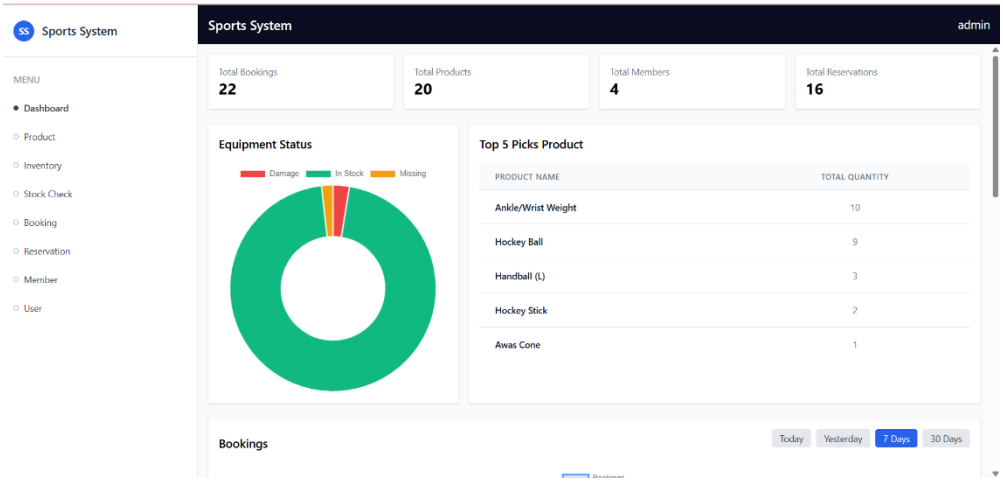


Figure 5.4: Dashboard page – Part 1

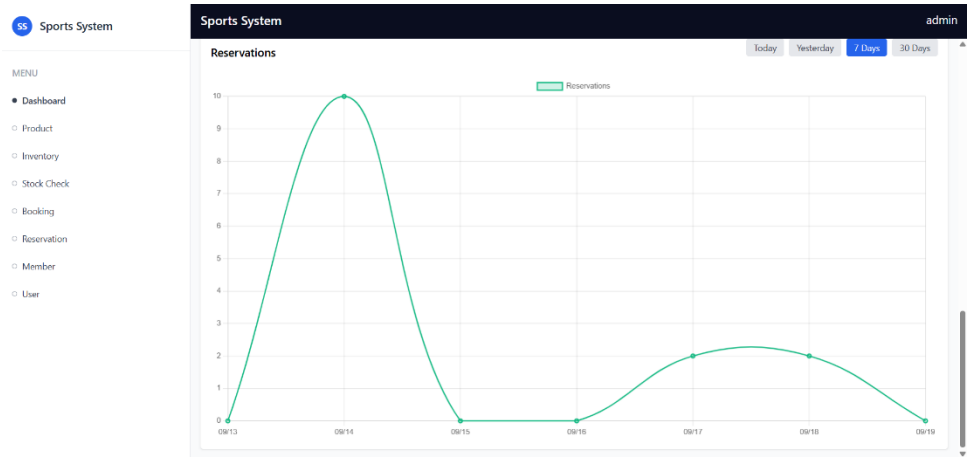


Figure 5.5: Dashboard page – Part 2

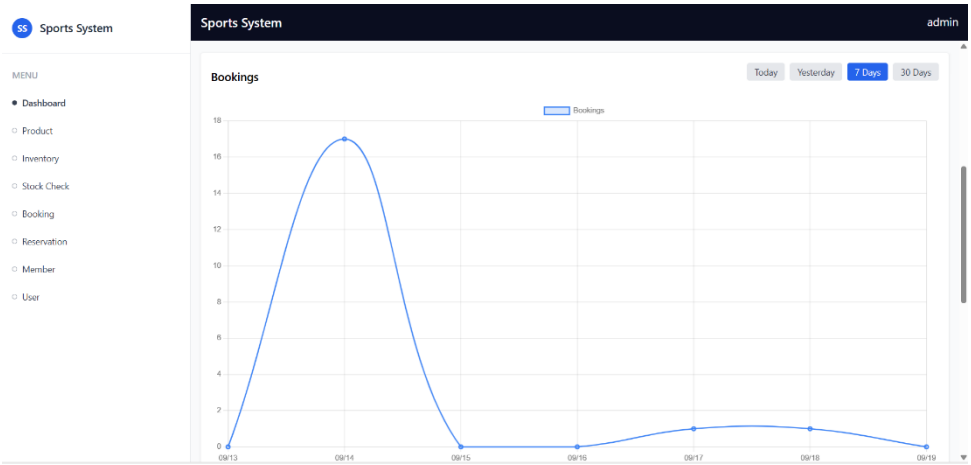


Figure 5.6: Dashboard page – Part 3

5.4.3 Product Management Module

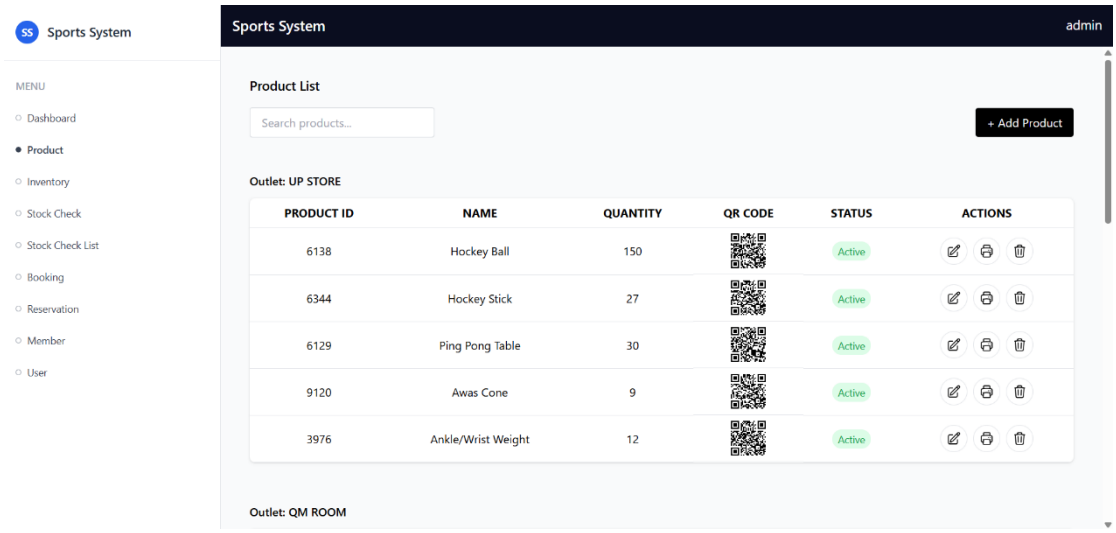


Figure 5.7: Product List Page

Figure 5.8: Product Page - Print Icon

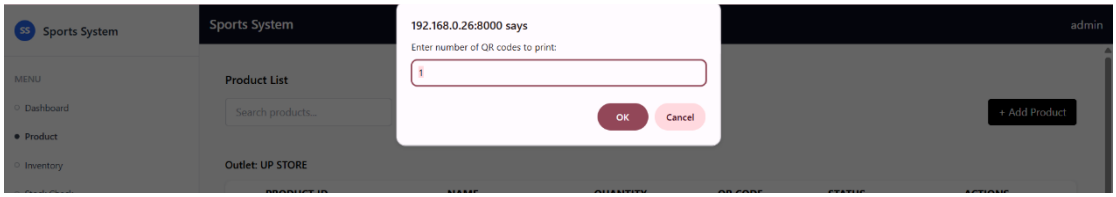


Figure 5.9: Product Page - Print Quantity Input Field

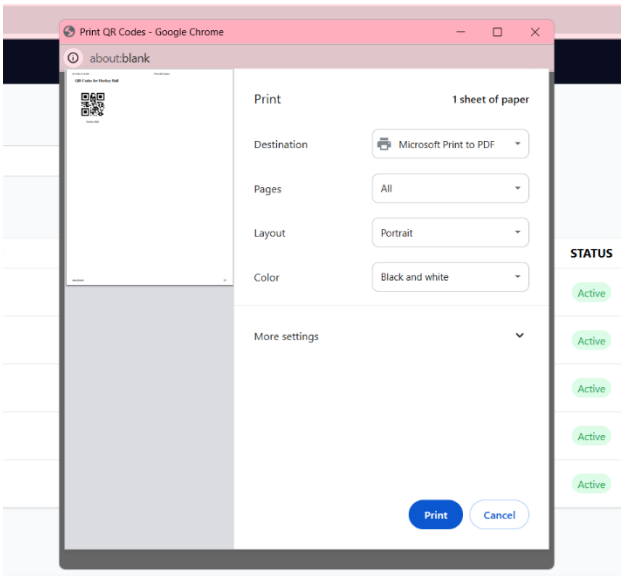


Figure 5.10: Product Page - Printing Page

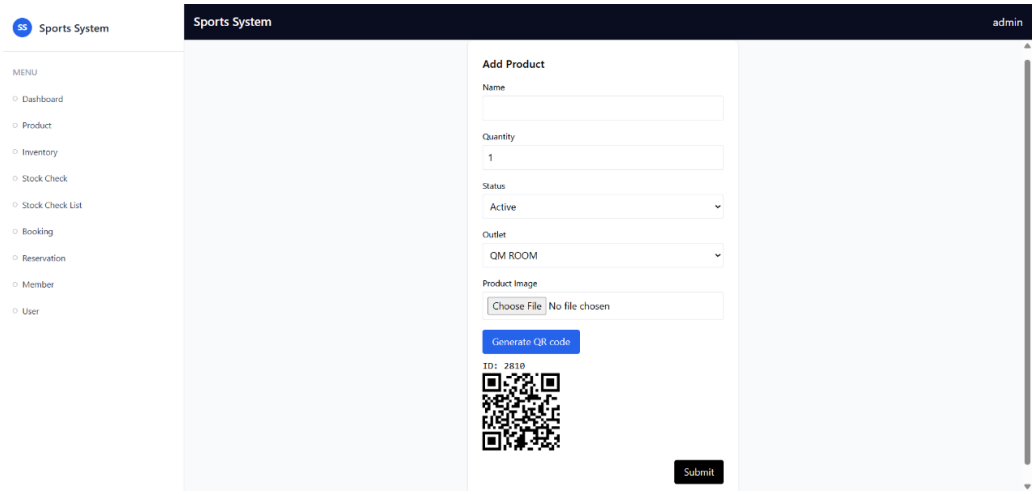


Figure 5.11: Product Add Page

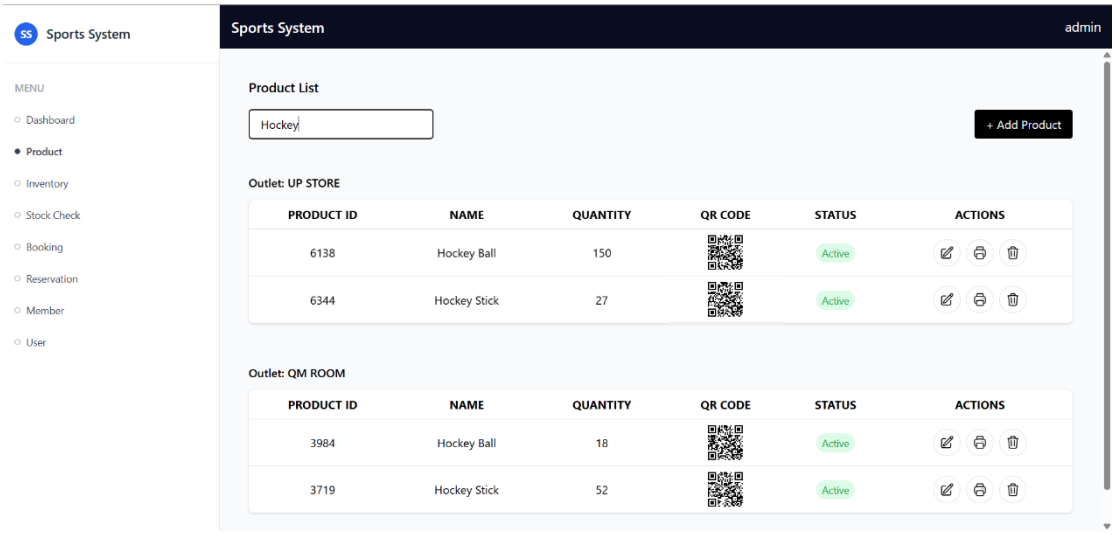


Figure 5.12: Product Page Filter Function

Figure 5.13: Product Page - QR Code Pop Up Modal

Figure 5.14: Product Page – Edit Icon Button

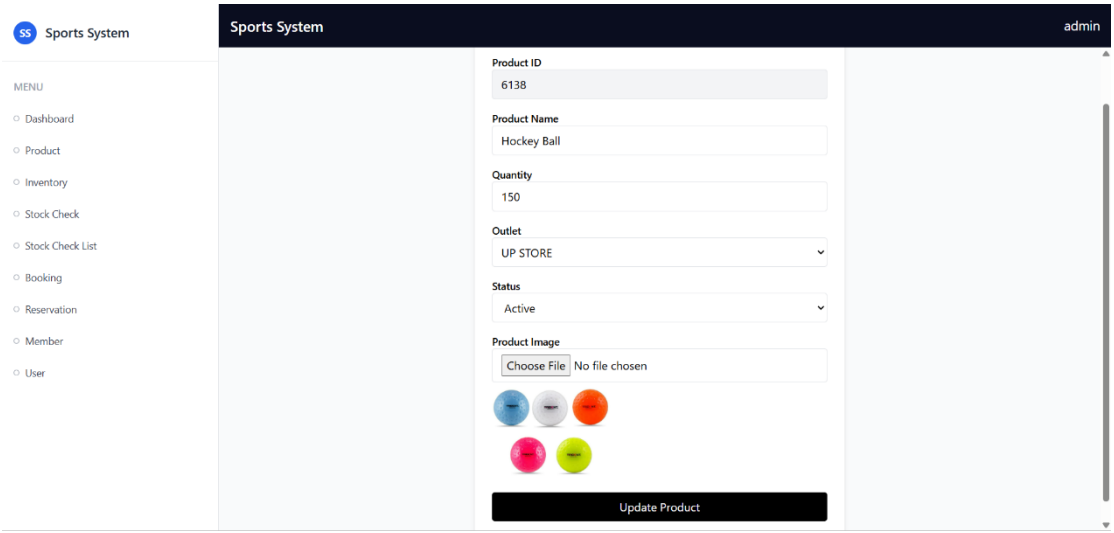


Figure 5.15: Product Edit Page

Figure 5.16: Product Page- Delete Icon

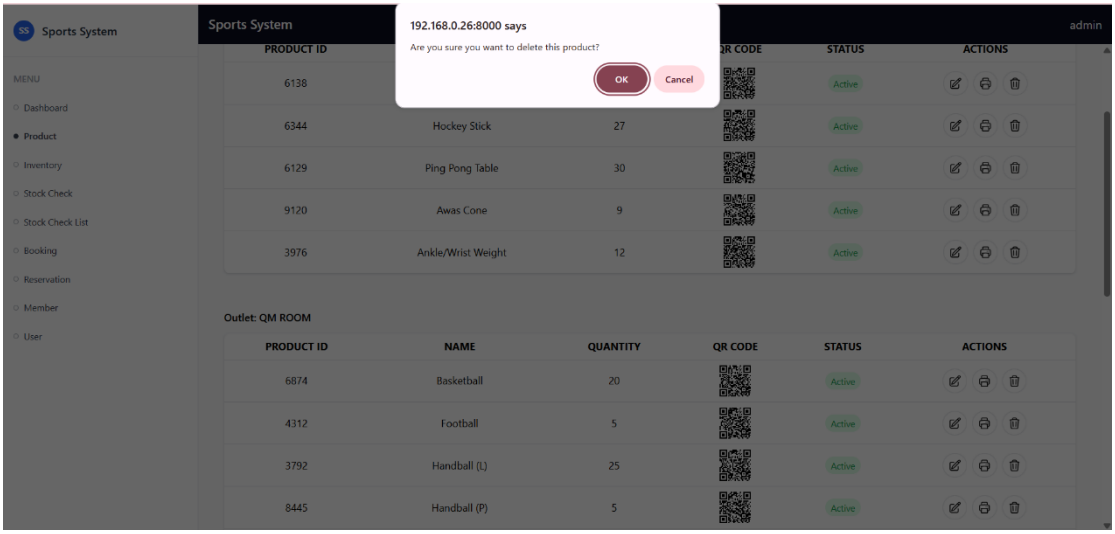


Figure 5.17: Product Page-Delete Confirmation Prompts

5.4.4 Inventory Management Module

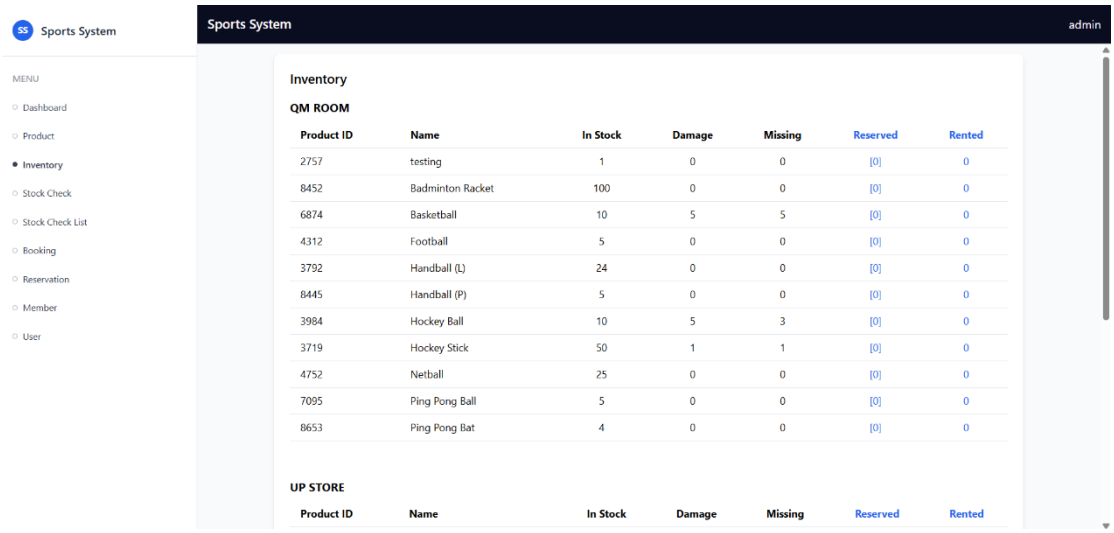


Figure 5.18: Inventory List page

5.4.5 Stock Check Module

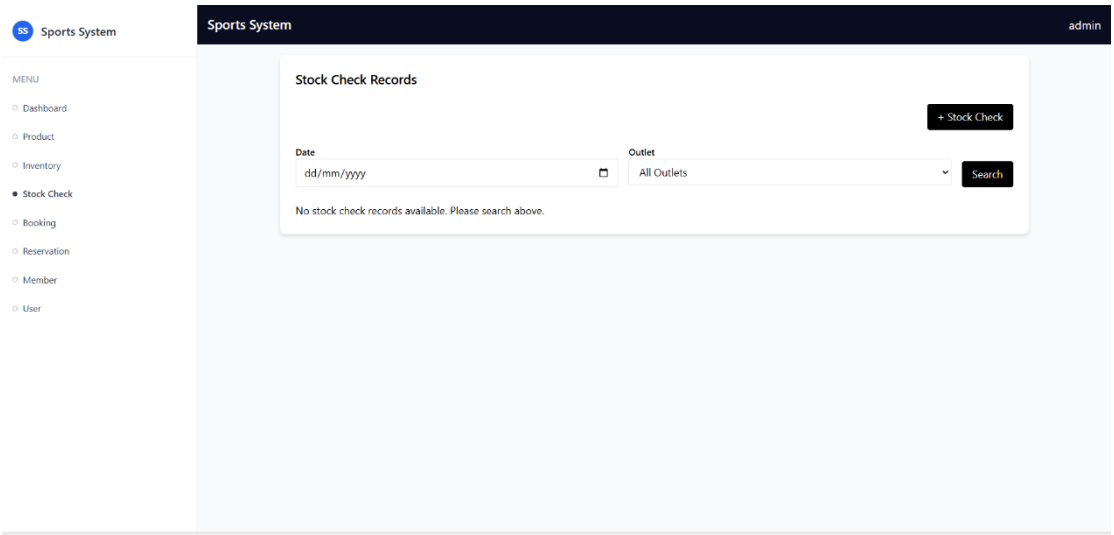


Figure 5.19: Stock Check List

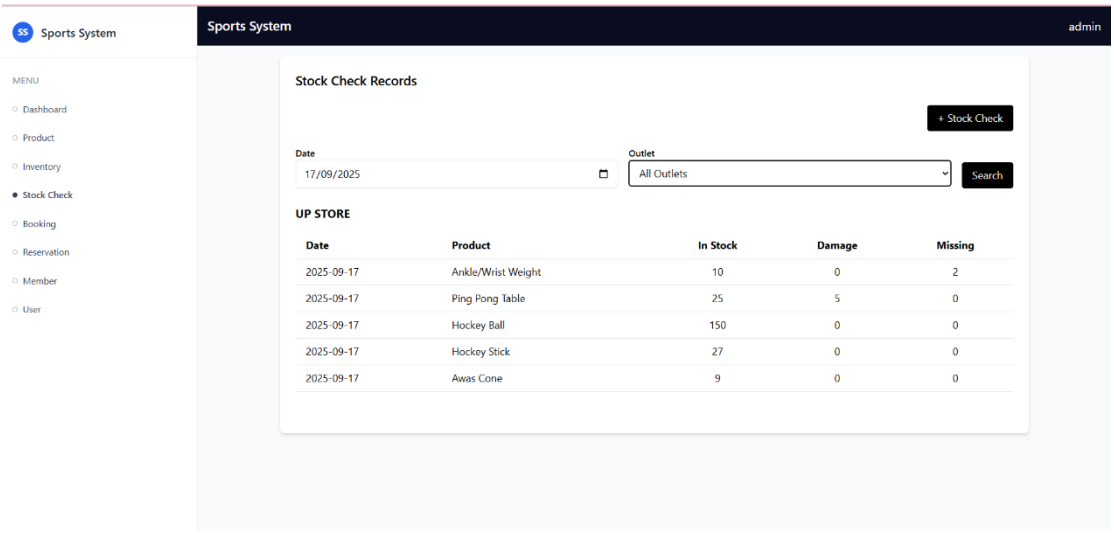


Figure 5.20: Stock Check List - Display Data Based on Date and Outlet.

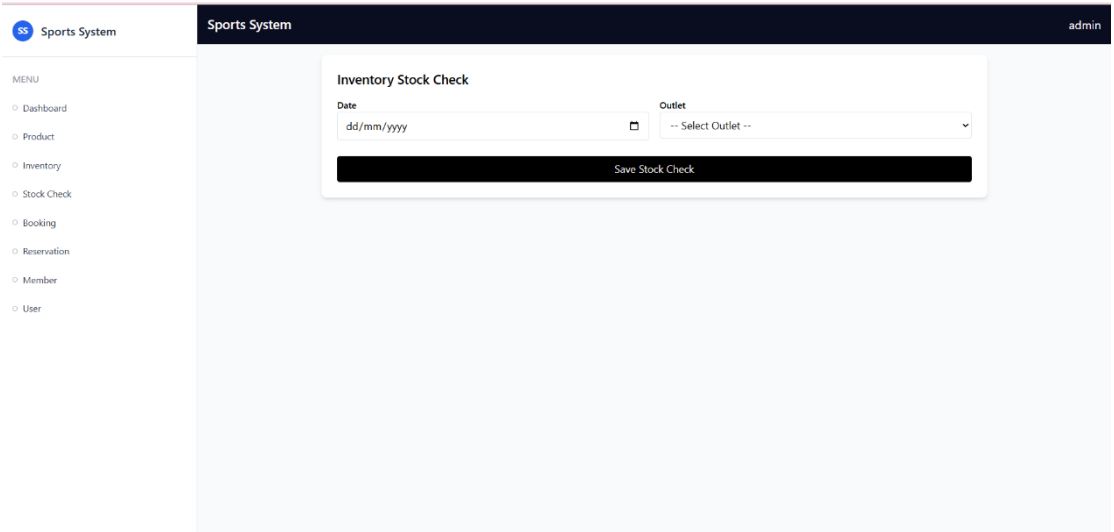


Figure 5.21: Add Stock Check

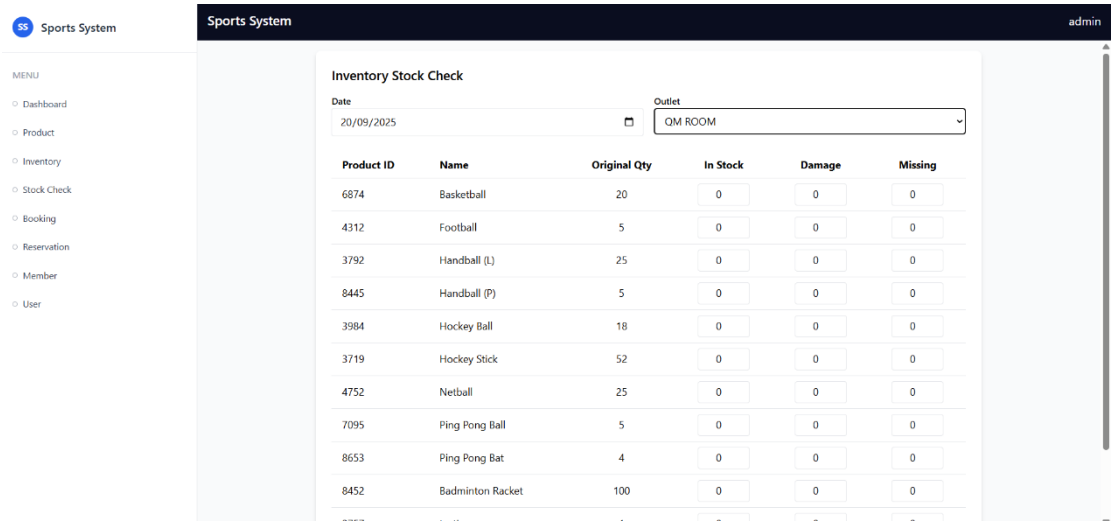


Figure 5.22: Add Stock Check - List

5.4.6 Booking Module

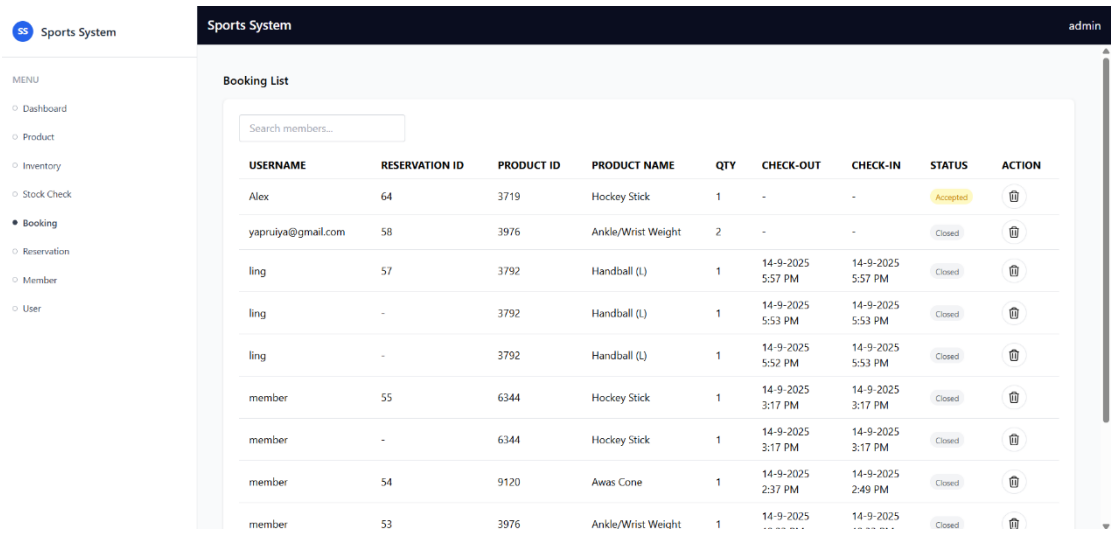


Figure 5.23: Booking List

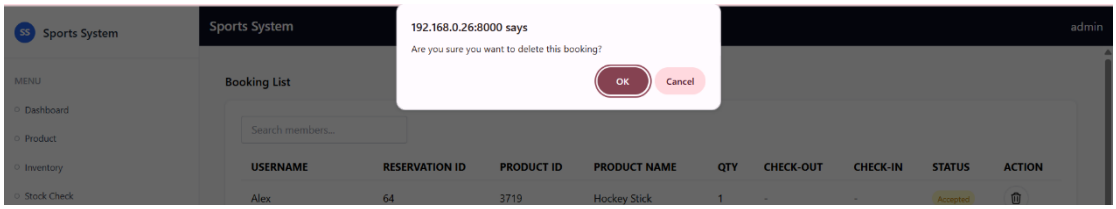


Figure 5.24: Booking List - Delete Confirmation

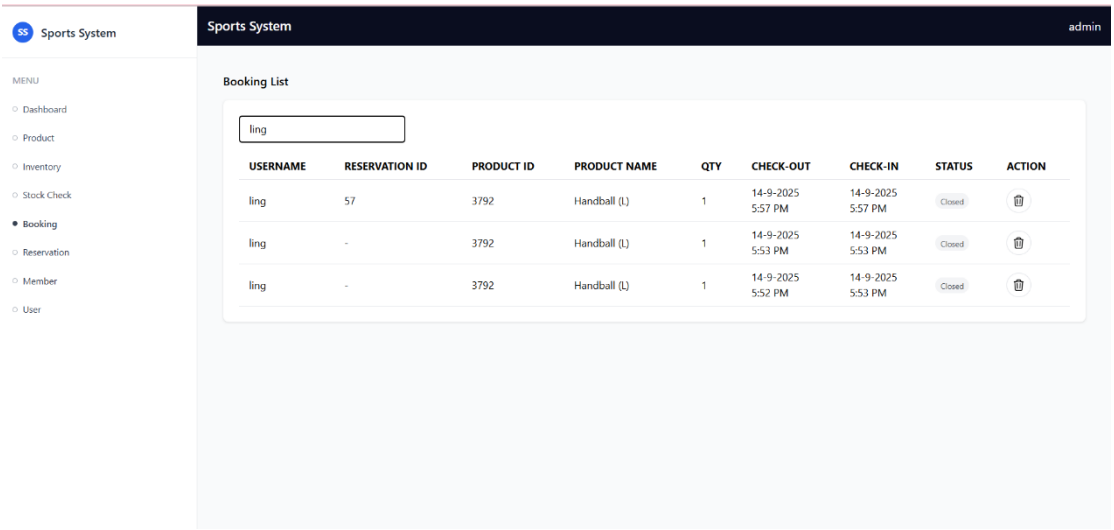


Figure 5.25: Booking List- Filter Function

5.4.7 Reservation Module

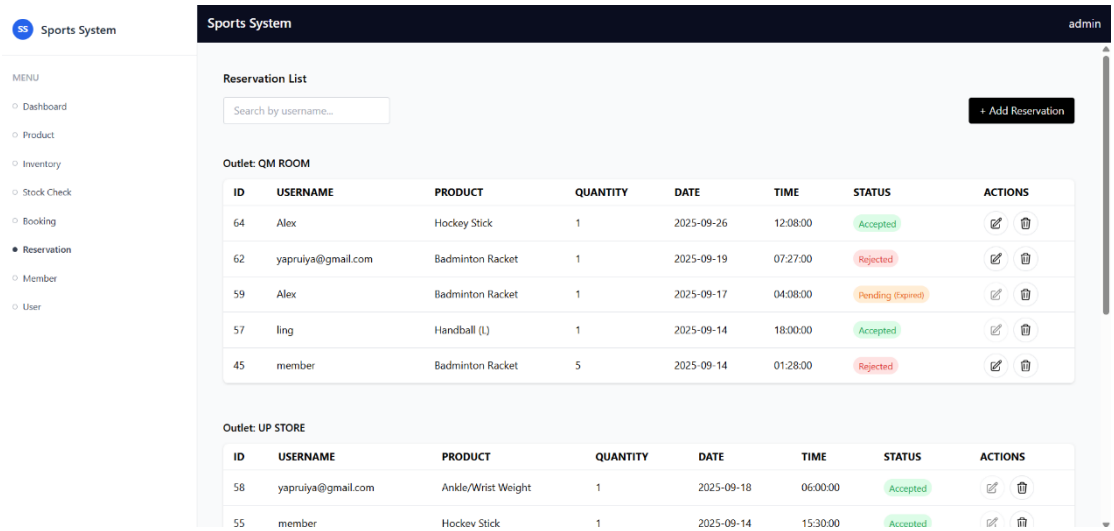


Figure 5.26: Reservation List

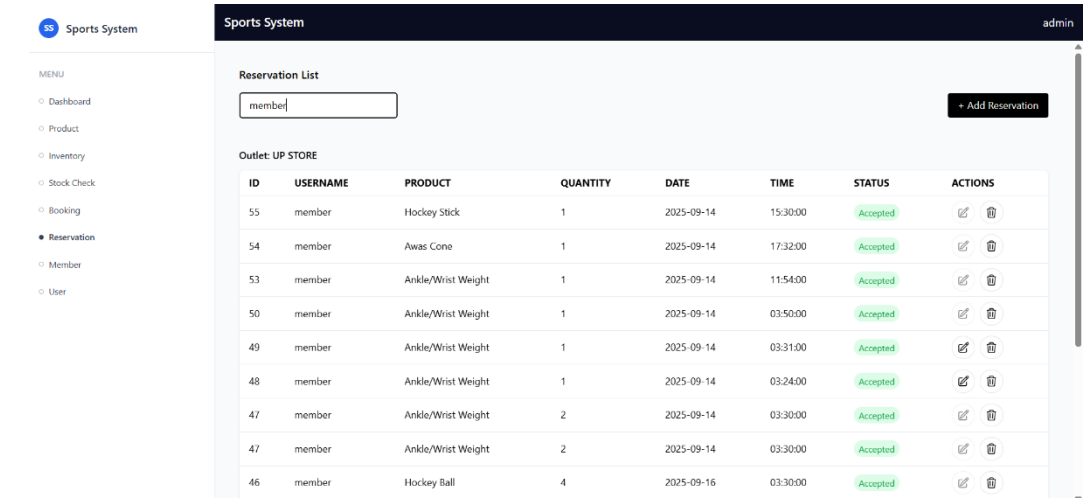


Figure 5.27: Reservation List- Filter Function

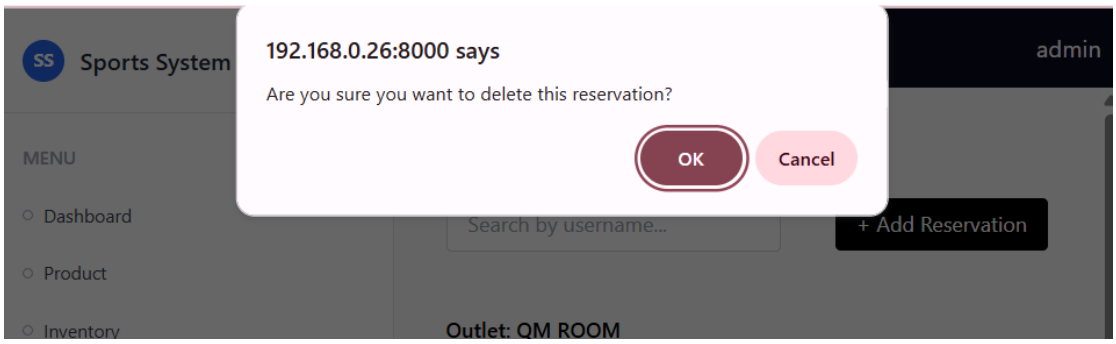


Figure 5.28: Reservation List - Delete Confirmation

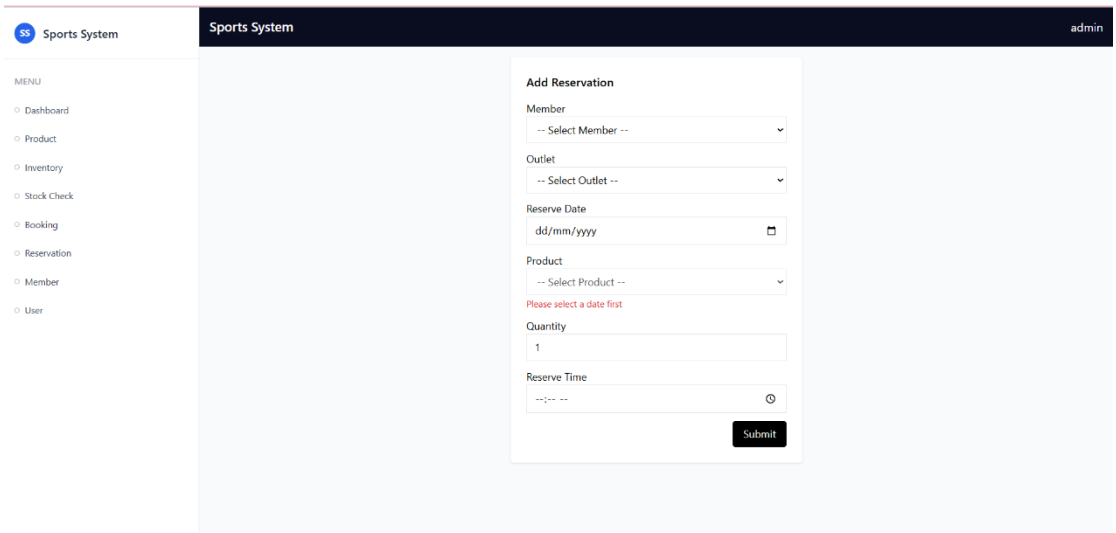


Figure 5.29: Reservation Add Page

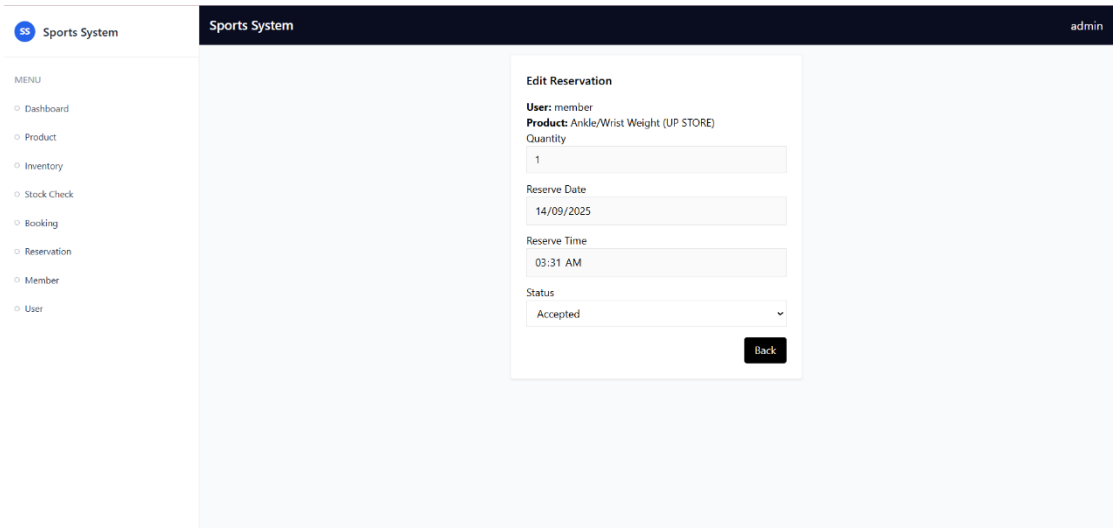


Figure 5.30: Reservation Edit Page

5.4.8 Member Module

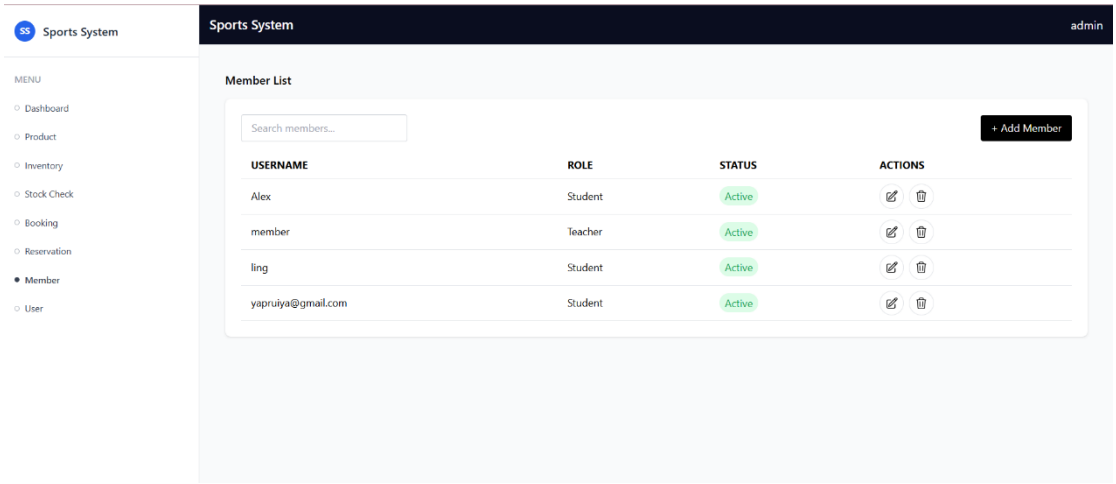


Figure 5.31: Member List Page

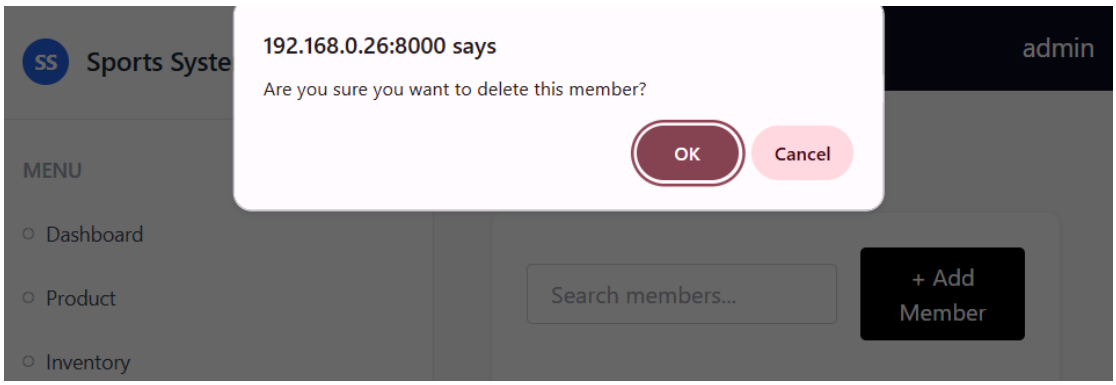


Figure 5.32: Member Page - Delete Confirmation

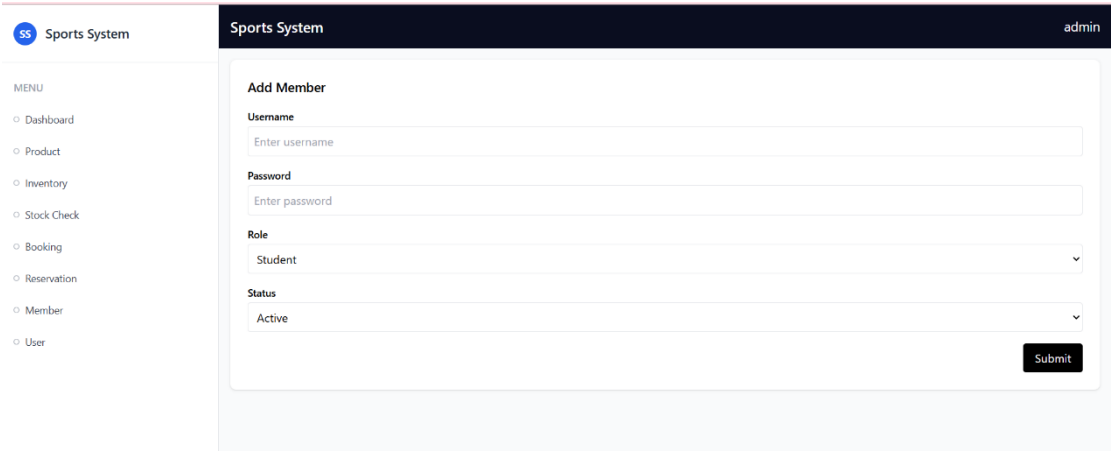


Figure 5.33: Member Add Page

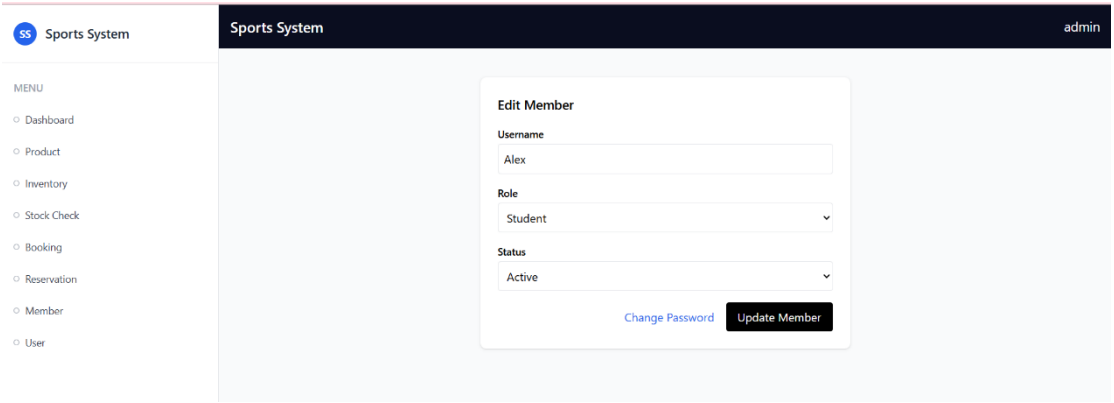


Figure 5.34: Member Edit Page

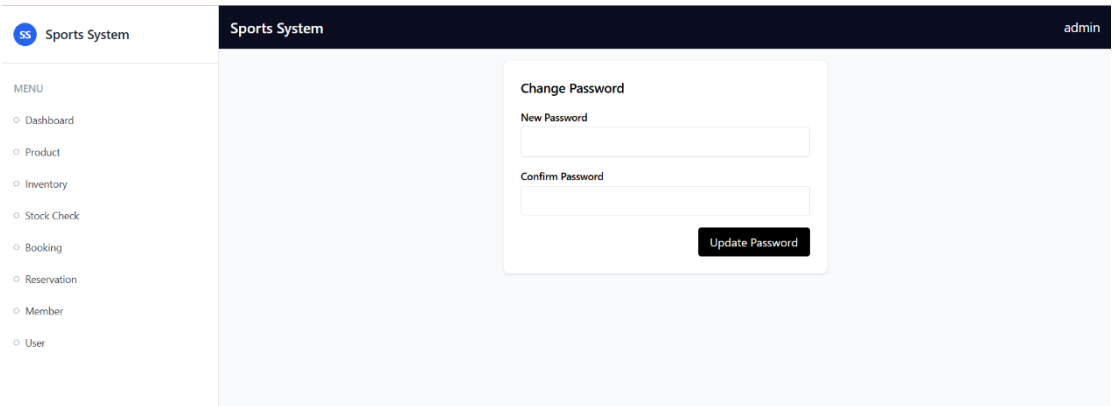


Figure 5.35: Member Change Password Page

5.4.9 User Module

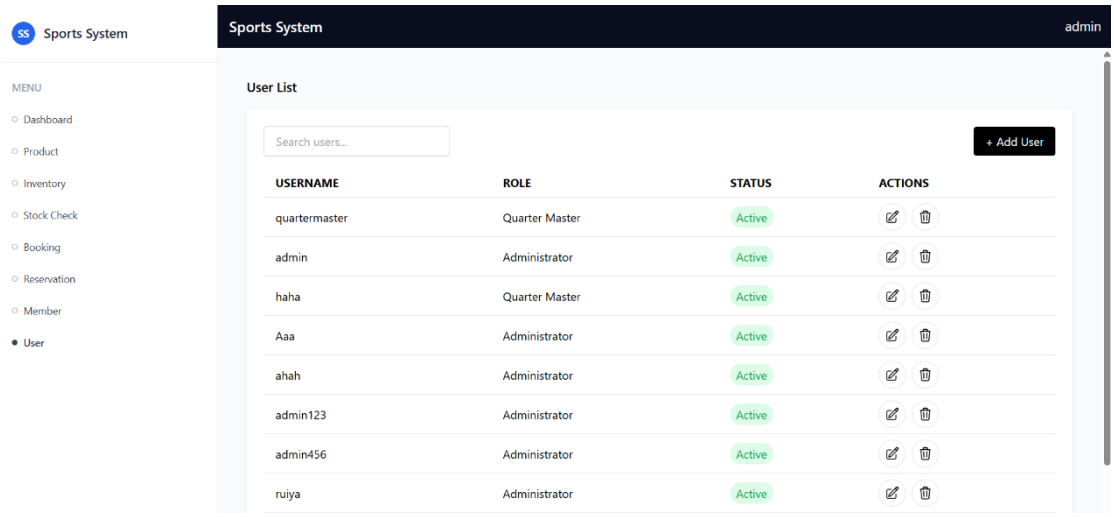


Figure 5.36: User List Page

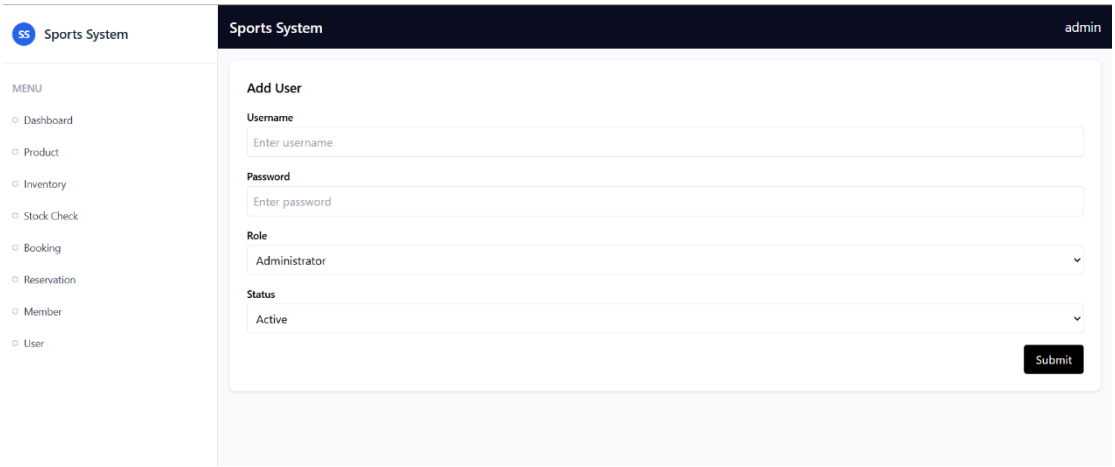


Figure 5.37: User Add Page

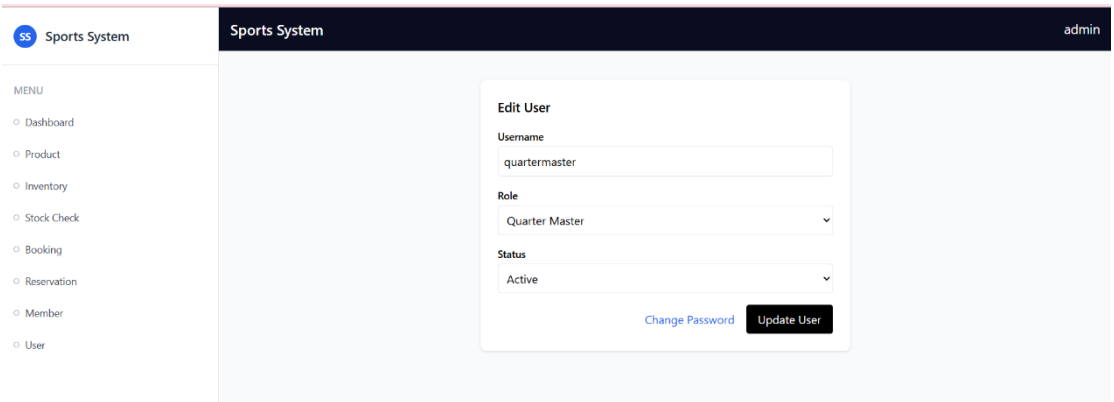


Figure 5.38: User Edit Page

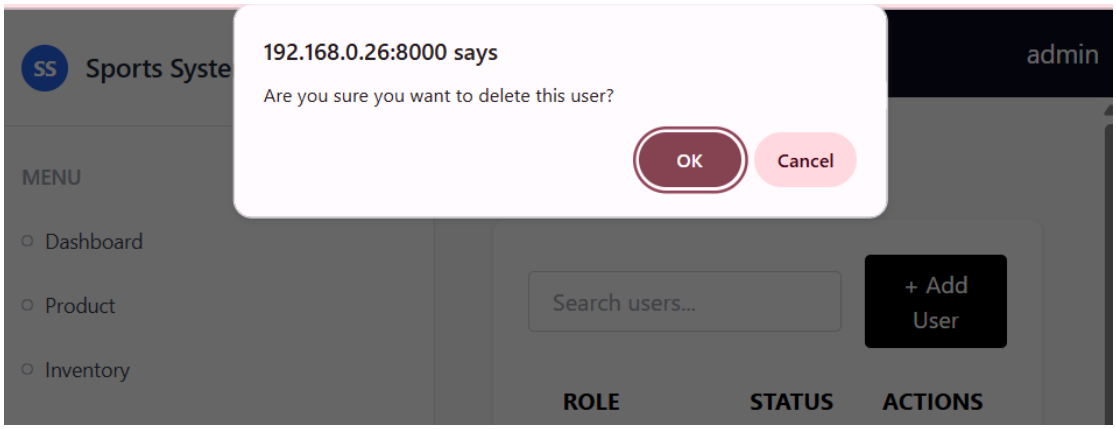


Figure 5.39: User Delete Confirmation

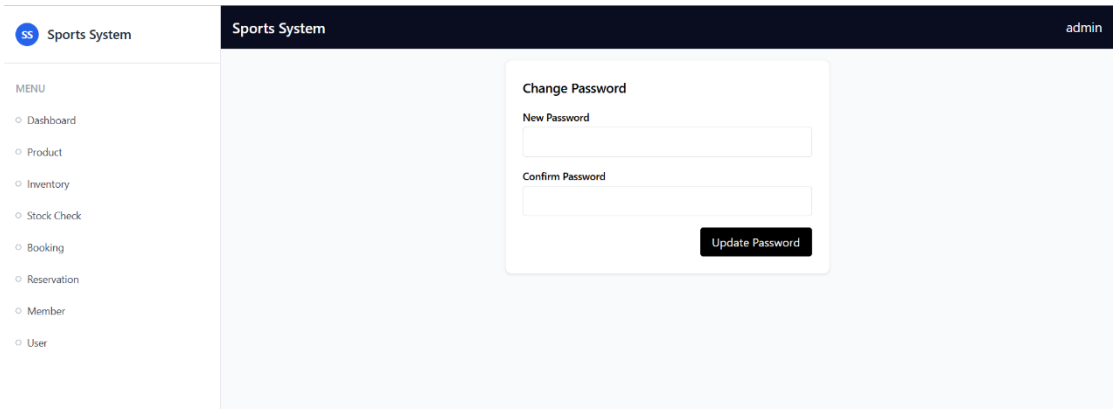


Figure 5.40: User Change Password Page

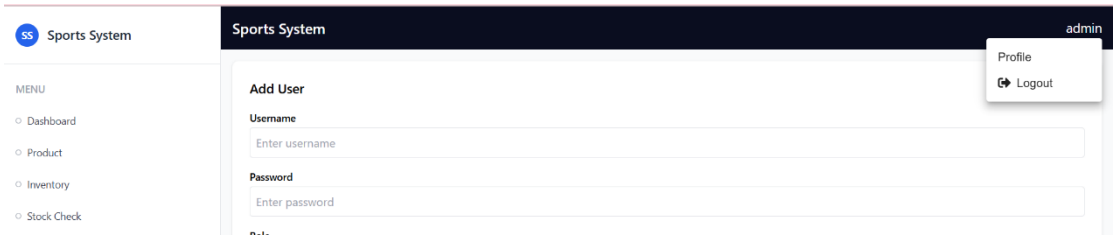


Figure 5.41: User Profile Page

CHAPTER 6

SYSTEM IMPLEMENTATION

6.1 Introduction

This chapter describes the process of implementing the proposed sports inventory management system. The implementation covers the setup of the development environment, configuration of the Laravel backend on WAMPServer, and integration of the ReactJS frontend. The chapter also explains the main features of the system and provides code snippets to illustrate how these features are implemented. By detailing the project setup and implementation, this chapter demonstrates how the design from Chapter 5 is translated into a working system.

6.2 Project Setup

The project was developed on a Windows environment using WAMPServer as the local development server. WAMPServer provides an Apache web server, PHP, and MySQL database required to run Laravel. The frontend application was developed using ReactJS, while the backend was developed using Laravel. Firstly, download and install the latest version of

- WAMPServer (Apache + PHP + MySQL): https://www.wampserver.com/en/

Figure 6.1: Wampserver Official Website

- Composer (PHP dependency manager) from: https://getcomposer.org/.

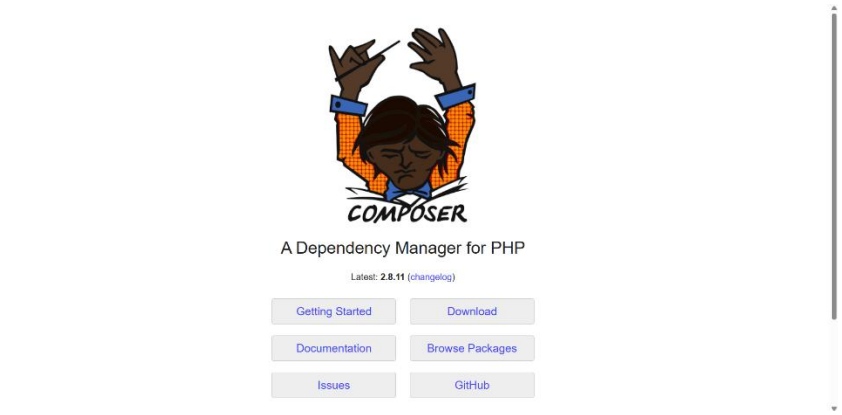


Figure 6.2: Composer Official Website

- Node.js + npm (for the frontend) from: https://nodejs.org/en/download.

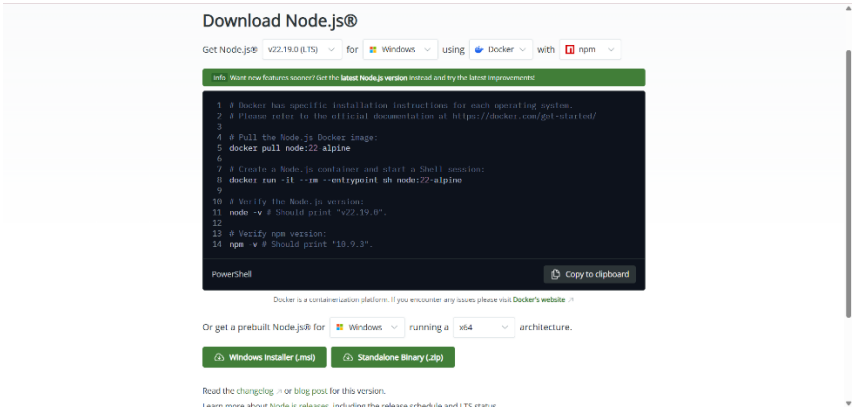


Figure 6.3: Node.js Official Website

After installation, start WAMP and confirm Apache and MySQL are running from the WAMP tray icon. Keep WAMP running while develop so Laravel can use the local Apache and MySQL services.

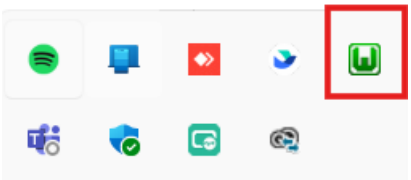


Figure 6.4: WampServer Running (Green)

Next, create a new Laravel project (this will be the main project folder). This will create a folder called my-app which will contain Laravel backend.

```
composer create-project laravel/Laravel:8.*  my-app
```

Inside the folder,install npm dependencies.This will install Vite, React, and build tools inside the Laravel project. This ensures that React components can be compiled and rendered smotthly within Laravel environment

```
cd sports-system
npm install
```

Next, run the following command to install React into Laravel project:

```
npm install react react-dom
```

Then, install the React Vite plugin:

```
npm install @vitejs/plugin-react
```

Open the file **vite.config.js** in Laravel project and add React support

```
vite.config.js > default > plugins
1   import { defineConfig } from 'vite';
2   import laravel from 'laravel-vite-plugin';
3   import react from '@vitejs/plugin-react';
4
5   export default defineConfig({
6       plugins: [
7           laravel({
8               input: ['resources/css/app.css', 'resources/js/app.jsx'],
9               refresh: true,
10          }),
11          react(),
12      ],
13    server: {
14     '/api': {
15          target: 'http://localhost:8000', // Laravel backend URL
16          changeOrigin: true,
17          secure: false,
18
19        }
20      },
21  });
22
```

Figure 6.5: vite.config.js

The React entry point is placed inside **resources/js/**, create a file called **app.jsx** which represents the main application component,

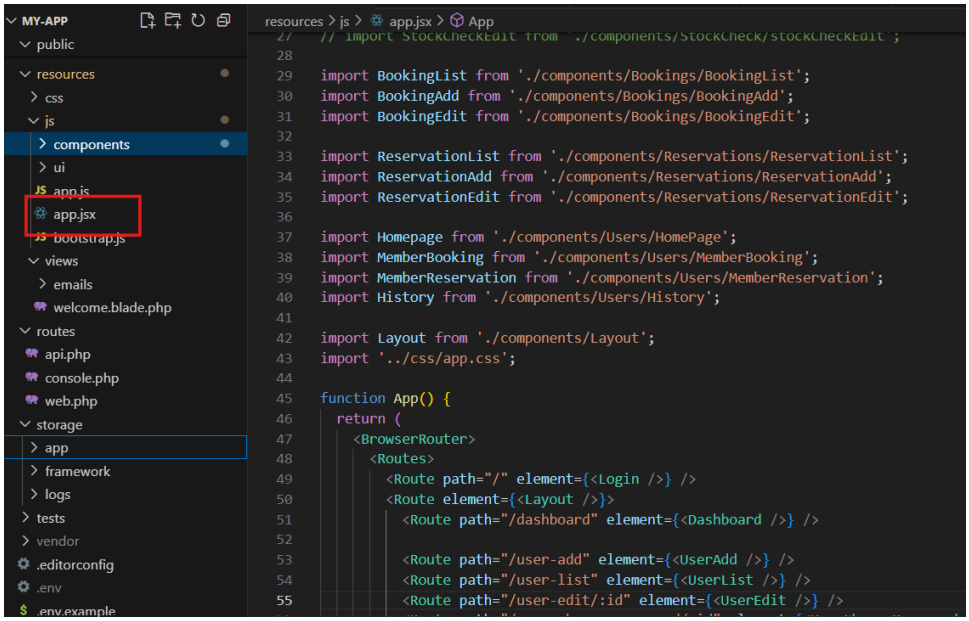


Figure 6.6: Code Snippet of app.jsx

Then, open **resources/views/welcome.blade.php.** This allows to load React app inside the Blade view. This is because blade template provides the HTML structure with root where the React app is installed. Laravel can integrate React into Blade view by using the @viteReactRefresh and @vite directives as it allows React components to render when the Laravel server is accessed.

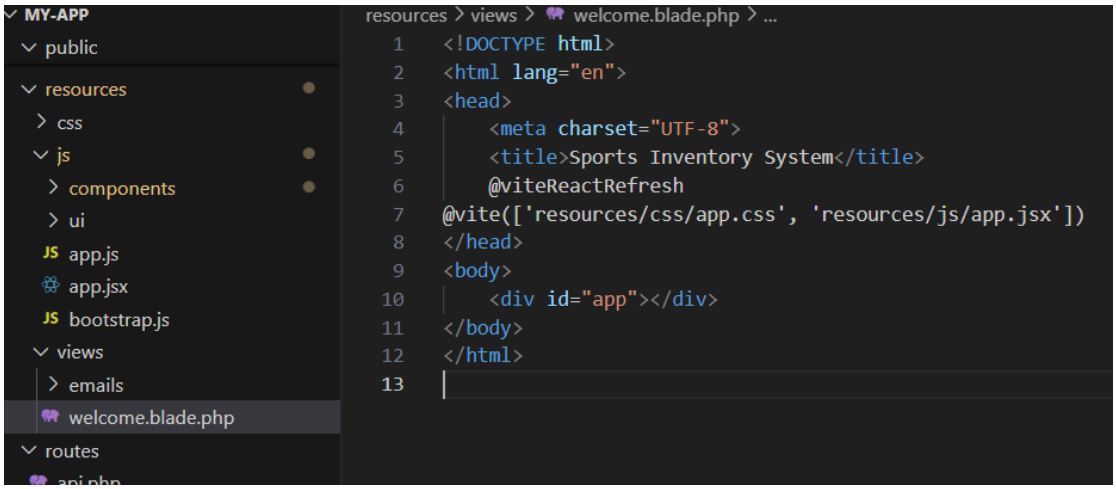


Figure 6.7: Code Snippet of welcome,blade.php

During development, the backend and frontend run in one environment. The Laravel server is started with php artisan serve. For React, it is served by Vite by using npm run dev command.When the application is accessed in the browser, Laravel

```
php artisan serve
npm run dev
```

6.2.1 Database Setup

The .env() need to be change according to each local or production environment else it will prevent the Laravel application from connecting to the database properly.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=sports
DB_USERNAME=root
DB_PASSWORD=
```

Figure 6.8: Database Connection Config

6.3 System Modules

The project consists of two integrated applications: a web-based application and a mobile application. Each of these applications is designed to meet the needs of system. The web-based application serves as a primary platform for administrators and quarter master to manage products, inventory, bookings, reservation, stock check and members. The mobile application allows students and teachers scan QR codes to booking the sport equipment. Both applications ensure an efficient management of sport equipment and facilities. At the same time, fulfil the needs of different user roles with the sports centre.

6.3.1 Modules for Web-based Application

6.3.1.1 Login Module

The login module is an important feature of the Sports Inventory Management System. It provides authentication to ensure that only registered users and members can access the system. It is built using React for the frontend and Axios for API communication with the backend.

```
const [username, setUsername] = useState('');
const [password, setPassword] = useState('');
```

Figure 6.9: useState Hook

When the user submits the login form, the entered username and password are captured using React's useState hook. These credentials are then sent to the backend API (/api/login) via an Axios POST request. The backend validates the credentials and returns either a user or member object if the login is successful.

```
const handleLogin = async (e) => {
  e.preventDefault();
  setError("");
  try {
    const res = await axios.post("/api/login", { username, password });

    if (res.status === 200) {
      if (res.data.member) {
        localStorage.setItem("member", JSON.stringify(res.data.member));
        navigate(redirect, { replace: true });
      } else if (res.data.user) {
        localStorage.setItem("user", JSON.stringify(res.data.user));
        navigate("/dashboard");
      }
    }
  } catch (err) {
    if (err.response) {
      if (err.response.status === 401) {
        setError("Invalid password. Please try again.");
      } else if (err.response.status === 404) {
        setError("Account not found. Please ask the administrator to create an account for you.");
      } else {
        setError("Server error. Please try again later.");
      }
    } else {
      setError("Network error. Please check your connection.");
    }
  }
};
```

Figure 6.10: Code Segment for Login Functionality

Based on above Figure 6.10, the login module is used the handleLogin function. If the login is successfully (status=200), the user data is stored in localStorage which allows the system to remain the session data even after refreshing the page. After login, the system will redirect the user or member to different pages (/dashboard or the redirect URL) based on the whether the login is user or member. If login fails, different error messages are displayed based on the backend response, such as 401 Unauthorized for wrong passwords or 404 Not Found if the account doesn't exist. Below figure show the error message displayed in different scenarios.

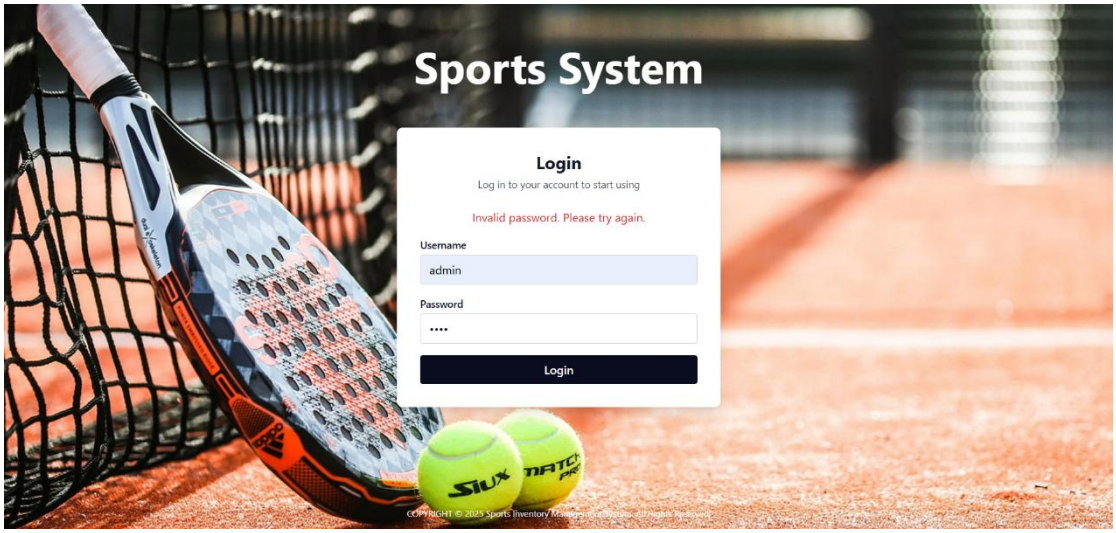


Figure 6.11: Login Page -Unauthorized for Wrong Passwords

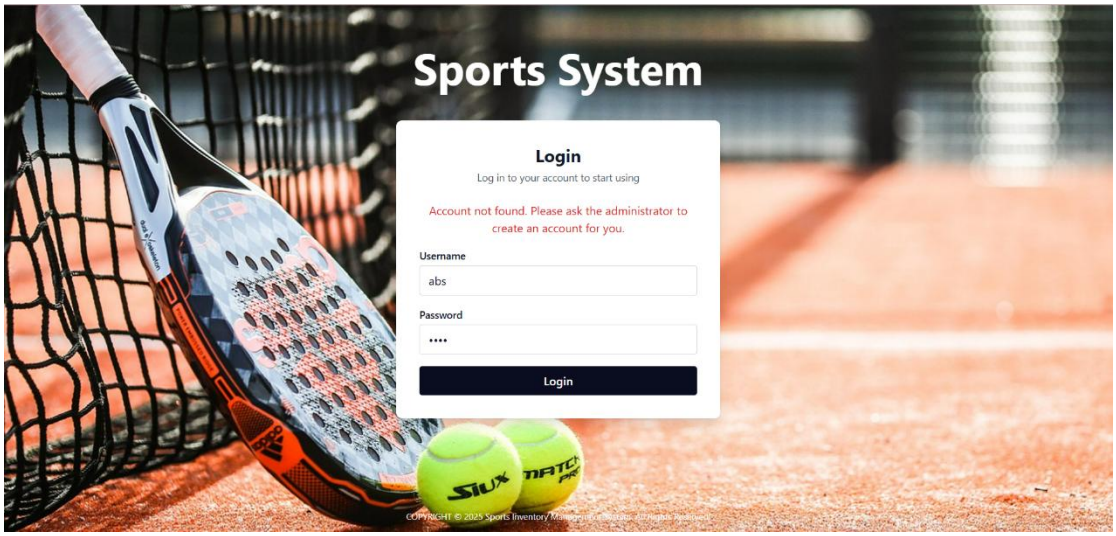


Figure 6.12: Login Page- Account Not Found

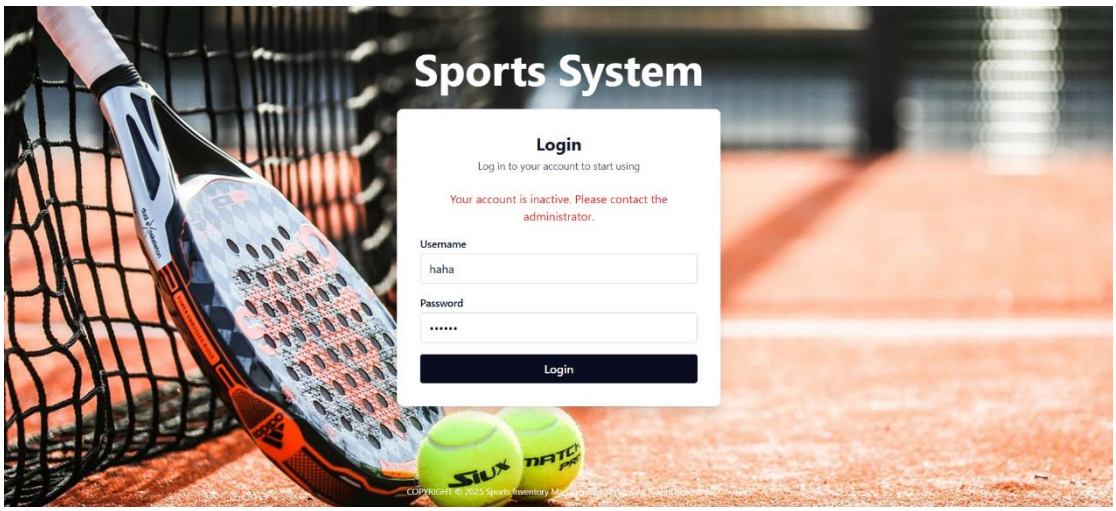


Figure 6.13: Login Page- Account Inactive

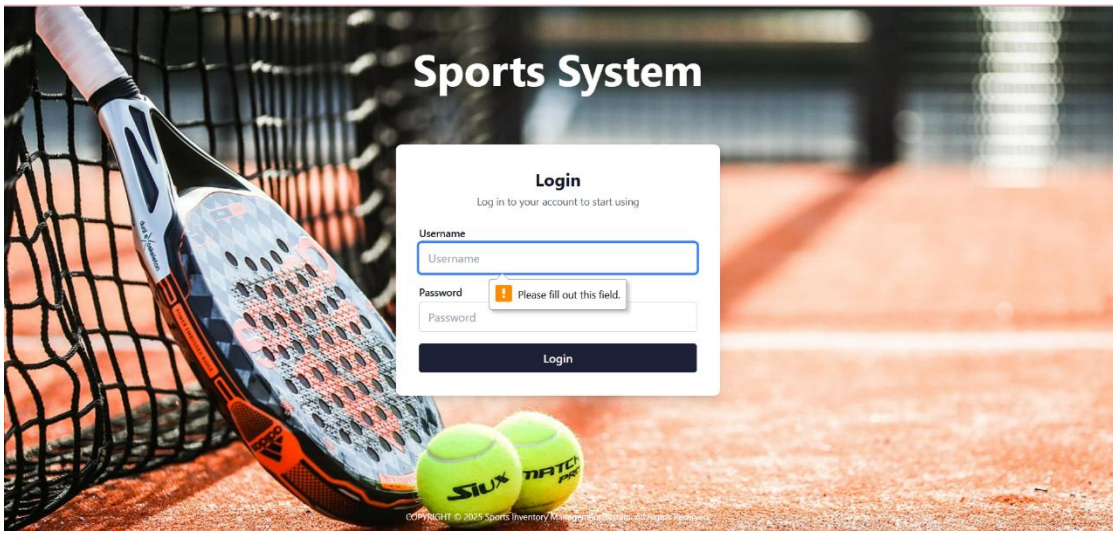


Figure 6.14: Login Page -Verify Empty Field

```
<form onSubmit={handleLogin} autoComplete="off">
  <div className="mb-4">
    <label htmlFor="username" className="block text-sm font-medium mb-1">Username</label>
    <input
      id="username"
      name="username"
      type="text"
      placeholder="Username"
      value={username}
      autoComplete="username"
      onChange={(e) => setUsername(e.target.value)}
      required
      className="w-full px-3 py-2 border border-gray-300 rounded focus:outline-none focus:ring focus:ring-blue-500"
    />
  </div>
  <div className="mb-4">
    <label htmlFor="password" className="block text-sm font-medium mb-1">Password</label>
    <input
      id="password"
      name="password"
      type="password"
      placeholder="Password"
      value={password}
      autoComplete="current-password"
      onChange={(e) => setPassword(e.target.value)}
      required
      className="w-full px-3 py-2 border border-gray-300 rounded focus:outline-none focus:ring focus:ring-blue-500"
    />
  </div>
```

Figure 6.15: Login Page - User Input Form

Based on above figure, it shows that these inputs ensure that the values are always in sync with the component's state. Then, submit button triggers the handleLogin function to complete the authentication process.

Backend Login Function- AuthController

```php
public function login(Request $request)
{
    $request->validate([
        'username' => 'required|string',
        'password' => 'required|string',
    ]);

    // Check Users table first (admins/quartermasters)
    $user = User::where('username', $request->username)->first();
    if ($user) {
        if (\Hash::check($request->password, $user->password)) {
            return response()->json(['user' => $user], 200);
        }
        return response()->json(['message' => 'Invalid password'], 401);
    }

    // Check Members table (teachers/students)
    $member = Member::where('username', $request->username)->first();
    if ($member) {
        if (\Hash::check($request->password, $member->password)) {
            return response()->json(['member' => $member], 200);
        }
        return response()->json(['message' => 'Invalid password'], 401);
    }

    // If username not found in both tables
    return response()->json([
        'message' => 'Account not found. Please ask the administrator to create an account for you.'
    ], 404);
}
```

Figure 6.16: Login Function

The backend login function in Laravel is responsible for authenticating both system users (administrators or quartermasters) and members (teachers or students). It ensures secure login by validating credentials against two different database tables which are users and members. The process starts with validating the request to make sure both username and password fields are provided. This is done using Laravel's built-in validation. After validation, the system first checks the Users table to ensure the entered username exists. If the username exists, the entered password is compared with the stored hashed password using Laravel's Hash::check() method. If the password matches, the function returns a JSON response with the authenticated user's data. Otherwise, it responds with a 401 Unauthorized error for an invalid password.

If user not found, the function then checks the Members table using the same logic. This allows both administrative staff and members to log in through the same endpoint but be authenticated based on their respective roles. If the provided username is not found in either table, the system responds with a 404 Not Found message, instructing the user to contact the administrator to create an account.

6.3.1.2 Dashboard

The Dashboard module serves as the central control panel for administrators to provide a quick and comprehensive overview of system operations to ensure that the users can monitor the overall performance of the sports equipment and facilities immediately.

```
useEffect(() => {
  axios.get('/api/dashboard-stats')
    .then(resp => setStats(resp.data))
    .catch(console.error);

  axios.get('/api/inventory-summary')
    .then(resp => setEquipmentStatus(resp.data))
    .catch(console.error);

  fetchBookingData(bookingRange);
  fetchReservationData(reservationRange);
  fetchTopProducts();
}, []);
```

Figure 6.17: Load Dashboard Data Function

Dashboard has Key Performance Indicators (KPIs) that display the total number of products, bookings, reservations and members. The KPIs are fetched from the backend (/api/dashboard-stats) to get the essential statistics such as total products and total booking, total members and total reservations from the Laravel backend and stored in React state. Then, the values are passed to a reusable KpiCard component to be displayed in styled cards. Therefore, this allows administrators to assess the system usage trend without navigating through multiple modules.

```
const doughnutData = {
  labels: ['Damage', 'In Stock', 'Missing'],
  datasets: [{
    data: [
      equipmentStatus.damage,
      equipmentStatus.instock,
      equipmentStatus.missing
    ],
    backgroundColor: ['#EF4444', '#10B981', '#F59E0B'],
  }],
};
```

Figure 6.18: Get Equipment Status Chart Data

The equipment status is displayed using a doughnut chart which highlights the distribution of items in stock, damaged, or missing to enable administrators to detect issues quickly and plan for maintenance or replacements. It retrieves the data from

/api/inventory-summary and used doughnutData function to map the data into datasets array so that it can display the output in doughnut chart with color codes.

```
const fetchTopProducts = async () => {
  try {
    const resp = await axios.get('/api/top-products');
    setTopProducts(resp.data);
  } catch (error) {
    console.error('Error fetching top products:', error);
  }
};
```

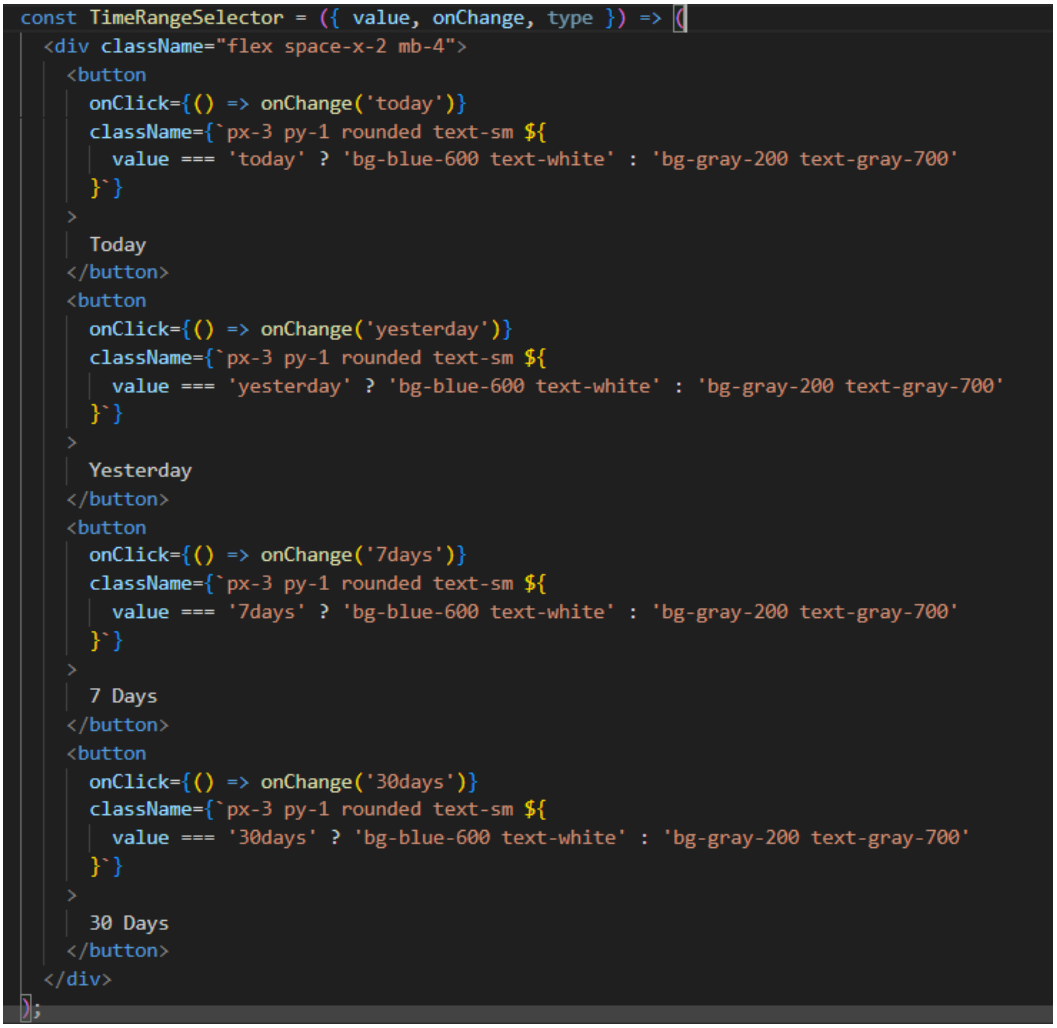
Figure 6.19: Get the Most Used Product Data

Besides, Figure 6.19 shows that it fetches the top five products which means it is the most frequently used product among members. It is fetched from /api/top-products directly in order to identify the high-demand items.

```
const fetchBookingData = async (range) => {
  try {
    const resp = await axios.get(`/api/weekly-bookings?range=${range}`);
    setBookingData({
      labels: resp.data.labels,
      data: resp.data.data,
      type: range === 'today' || range === 'yesterday' ? 'bar' : 'line'
    });
  } catch (error) {
    console.error('Error fetching booking data:', error);
  }
};

const fetchReservationData = async (range) => {
  try {
    const resp = await axios.get(`/api/weekly-reservations?range=${range}`);
    setReservationData({
      labels: resp.data.labels,
      data: resp.data.data,
      type: range === 'today' || range === 'yesterday' ? 'bar' : 'line'
    });
  } catch (error) {
    console.error('Error fetching reservation data:', error);
  }
};
```

Figure 6.20: Fetch Booking Data Function

The system uses Chart.js to implement the booking and reservation charts. It used fetchBookingData and fetchReservationData function to allow the chart dynamically to change between bar and line graphs depending on the selected time range which offers a clear insight into daily, weekly or monthly activity. The selected time range is handle using timeRangeSelector function which is shows in Figure 6.21.



```
const TimeRangeSelector = ({ value, onChange, type }) => (
  <div className="flex space-x-2 mb-4">
    <button
      onClick={() => onChange('today')}
      className={`px-3 py-1 rounded text-sm ${
        value === 'today' ? 'bg-blue-600 text-white' : 'bg-gray-200 text-gray-700'
      }`}
    >
      Today
    </button>
    <button
      onClick={() => onChange('yesterday')}
      className={`px-3 py-1 rounded text-sm ${
        value === 'yesterday' ? 'bg-blue-600 text-white' : 'bg-gray-200 text-gray-700'
      }`}
    >
      Yesterday
    </button>
    <button
      onClick={() => onChange('7days')}
      className={`px-3 py-1 rounded text-sm ${
        value === '7days' ? 'bg-blue-600 text-white' : 'bg-gray-200 text-gray-700'
      }`}
    >
      7 Days
    </button>
    <button
      onClick={() => onChange('30days')}
      className={`px-3 py-1 rounded text-sm ${
        value === '30days' ? 'bg-blue-600 text-white' : 'bg-gray-200 text-gray-700'
      }`}
    >
      30 Days
    </button>
  </div>
);
```

Figure 6.21: Time Range Selector Function

6.3.1.3 Product Management

The Product Management is a core feature of the Inventory Management system as it designed to display, manage and interact with a list of sports equipment products. Its main features include displaying products grouped by outlet, filtering products by name, paginating the product list, generating and displaying QR codes for equipment tracking, and performing CRUD operations (edit, delete, and add products) for administrators. These features are implemented using React.js, leveraging its component-based architecture, state management, and third-party libraries like axios for API calls, qrcode.react for QR code generation, and react-router-dom for navigation.

```
const [products, setProducts] = useState([]);

const fetchProducts = async () => {
  try {
    const res = await axios.get("/api/products");
    setProducts(Array.isArray(res.data) ? res.data : []);
  } catch (err) {
    console.error("Error fetching products", err);
  }
};

useEffect(() => {
  fetchProducts();
}, []);
```

Figure 6.22: Code Segment to Retrieve the List of Product Data

Based on above figure, it shows that the module uses fetchProducts to retrieve a list of product data from the backend database through an API call using Axios to make a GET request to the /api/products endpoint and displays the product retrieve in a table format.This ensures that the product list is dynamically updated and reflects the most recent changes made to the inventory. The retrieved data is stored within a state variable for rendering in the user interface.

Filter Function

```
const filteredProducts = products.filter((product) =>
  product.name.toLowerCase().includes(filterText.toLowerCase())
);
```

Figure 6.23: Code Segment to Filter Product by Name

The product management module implements a search feature that allows user to filter products by name. It will compare the user's search input against the product dataset to ensure that only relevant products are displayed. Below figure show the output result.

Add function

The ProductAdd is designed to allow administrators to add new sport equipment products to the system. In the product add page, it allows administrators to generate uique product ID and associated QR code, collect product details such as name, quantity,status,outlet and image.

```
  const generateId = () => {
  const newId = Math.floor(1000 + Math.random() * 9000).toString();
  setProductId(newId);
  const url = `${window.location.origin}/home/${newId}`;
  setQrUrl(url);
};

const getSvgString = () => {
  const svgElement = qrRef.current?.querySelector("svg");
  return svgElement ? svgElement.outerHTML : null;
};
```

Figure 6.24: Generate Product ID & QR Code

Based on Figure 6.24, it shows that a random 4-digit product ID is created using Math.random() and combined with the system base URL. The generate link is store in state and shows as a QR code using the react-qr-code library which need to run npm install react-qr-code first before using this library.

```
const handleChange = (e) => {
  setForm({ ...form, [e.target.name]: e.target.value });
};
```

Figure 6.25: Handle Form Input

The handleChange functions is used to captures the product information such as name, quantity, status (active/inactive), outlet (QM Room, UP Store, Down Store) and image

```
const handleImageChange = (e) => {
  if (e.target.files && e.target.files[0]) {
    setImage(e.target.files[0]);
  }
};
```

Figure 6.26: Image Upload with Preview

Figure 6.26 shows that the function is used to upload the image and a live preview is shown to confirm the correct image.

```
const handleSubmit = async (e) => {
  e.preventDefault();
  if (!productId) {
    setMessage("Please generate a QR code first.");
    return;
  }
  const svgString = getSvgString();
  try {
    const formData = new FormData();
    formData.append("productId", productId);
    formData.append("name", form.name);
    formData.append("quantity", form.quantity);
    formData.append("status", form.status);
    formData.append("outlet", form.outlet);
    formData.append("qrcode", svgString || "");
    if (image) {
      formData.append("image", image); // ✅ add image
    }

    await axios.post("/api/products", formData, {
      headers: { "Content-Type": "multipart/form-data" },
    });

    setMessage("Product added successfully.");
    setForm({ name: "", quantity: 1, status: "active", outlet: "QM ROOM" });
    setProductId("");
    setQrUrl("");
    setImage(null);
    if (onProductAdded) onProductAdded();

    navigate("/product-list");
  } catch (err) {
    if (err.response && err.response.status === 422) {
      const errors = err.response.data.errors;
      if (errors.name) {
        setMessage(errors.name[0]);
      } else {
        setMessage('Validation error. Please check the form.');
      }
    } else {
      setMessage('Failed to add product. Please try again.');
    }
  }
};
```

Figure 6.27: Submit the Product Data

Figure 6.27 shows that the handleSubmit function combined all data is into a FormData Object and sent to /api/products. If the process is validation, it will send a message to show that the Product Add successfully and navigate to product-list page. Else, it will display error message and the handleSubmit function fails.

Edit Function

The users allow to update existing product details in the system by using edit function. This component is implemented as a React functional component using hooks for state management and Axios for API communication. It utilizes useParams to retrieves the productid and executes an API call to fetch the current product data from the server. Below figure shows the edit button and it navigates to product edit page after clicking it.

```
useEffect(() => {
  const fetchProduct = async () => {
    try {
      const res = await axios.get(`/api/products/${productId}`);
      setFormData({
        product_id: res.data.product_id,
        name: res.data.name,
        quantity: res.data.quantity,
        status: res.data.status,
        outlet: res.data.outlet,
        image: null,
      });
      if (res.data.image) {
        setPreviewImage(`/storage/${res.data.image}`);
      }
      setLoading(false);
    } catch (err) {
      if (err.response && err.response.status === 422) {
          const errors = err.response.data.errors;
          if (errors.name) {
              setMessage(errors.name[0]);
          } else {
              setMessage("Validation error. Please check the form.");
          }
      } else {
          setMessage("Failed to edit product. Please try again.");
      }
      setLoading(false);
    }
  };
  fetchProduct();
}, [productId]);
```

Figure 6.28: Code Segment to Retrieve Product Data

Based on above figure, it shows that the retrieve data will fill in the input field form. Users can edit the product's name, quantity, status, outlet, and optionally upload a new image. The try-catch block handles error messages such as validation issues and updates the message state for user feedback. The loading state ensures a loading message is displayed until the data is fetched in order to enhanced user experience.

```
const handleChange = (e) => {
  const { name, value } = e.target;
  setFormData((prev) => ({ ...prev, [name]: value }));
};

const handleImageChange = (e) => {
  const file = e.target.files[0];
  if (file) {
    setFormData((prev) => ({ ...prev, image: file }));
    setPreviewImage(URL.createObjectURL(file));
  }
};
```

Figure 6.29: Code Segment to Handle Change - Part 1

```
{/* Name */}
<div>
  <label className="block text-sm font-medium">Product Name</label>
  <input
    type="text"
    name="name"
    value={formData.name}
    onChange={handleChange}
    required
    className="w-full border px-3 py-2 rounded"
  />
</div>

{/* Quantity */}
<div>
  <label className="block text-sm font-medium">Quantity</label>
  <input
    type="number"
    name="quantity"
    value={formData.quantity}
    onChange={handleChange}
    min="0"
    required
    className="w-full border px-3 py-2 rounded"
  />
</div>

{/* Outlet */}
<div>
  <label className="block text-sm font-medium">Outlet</label>
  <select
    name="outlet"
    value={formData.outlet}
    onChange={handleChange}
    className="w-full border rounded p-2"
  >
    <option value="QM ROOM">QM ROOM</option>
```

Figure 6.30: Code Segment to Handle Change - Part 2

The handleChange function updates formData using the spread operator to keep other fields and ensure the input remains controlled. The required attribute enforces client-side validation. The select element also provides predefined options for

outlets and status. This result that the enumeration constraints match with the database schema.

```
const handleImageChange = (e) => {
  const file = e.target.files[0];
  if (file) {
    setFormData((prev) => ({ ...prev, image: file }));
    setPreviewImage(URL.createObjectURL(file));
  }
};
```

Figure 6.31: Code segment to Handle Image Change

Based on above figure, handleImageChange can uploads product image and preview the image which provides device visual recognition ability. The handleImageChange function also capture the 'formData.image' and uses URL.createObjectURL to generate a temporary preview URL. The image will display through the storage/{image} path when images are retrieved. In order to link the storage in project need to run.

npm run storage:link.

Delete Function

```
103  // Delete product
104  const handleDelete = async (productId) => {
105    if (!window.confirm("Are you sure you want to delete this product?")) return;
106
107    try {
108      await axios.delete(`/api/products/${productId}`);
109      setProducts(products.filter((p) => p.product_id !== productId));
110    } catch (err) {
111      console.error("Delete failed", err);
112      alert("Failed to delete product.");
113    }
114  };
115
```

Figure 6.32: Code Segment to Handle Delete

Based on the figure above, the handleDelete function is to design to provide a secure way to remove products from the product list. It deletes a product by sending a request to server and there is a confirmation prompt to avoid accidental deletions. When user triggers the delete action, the functions will display a confirmation prompt using

window.confirm("Are you sure you want to delete this product?"). if user click the cancel button, the function exits immediately without making any changes. If the user clicks ok, the function sends an HTTP DELETE request to the server using Axios. Below figure shows the delete confirmation prompts.

QR code pop up model

The QR code popup modal implements in order to allow users to view the QR code in a larger version as it allows administrator and quarter master can easy testing the QR code to make booking and reservation without scanning wrong QR code. The pop-up model can be trigger by clicking the QR code.

```
{/* QR Popup Modal */}
{selectedQR && (
  <div
    className="fixed inset-0 flex items-center justify-center bg-black bg-opacity-70 z-50"
    onClick={() => setSelectedQR(null)}
  >
    <div
      className="bg-white p-6 rounded-lg shadow-lg relative"
      onClick={(e) => e.stopPropagation()}
    >
      <button
        className="absolute top-2 right-2 text-gray-600 hover:text-black"
        onClick={() => setSelectedQR(null)}
      >
        ✕
      </button>
      <h3 className="text-lg font-semibold mb-4">{selectedQR.name}</h3>
      <QRCodeSVG value={selectedQR.value} size={250} />
    </div>
  </div>
)}
```

Figure 6.33: Code segment to Handle QR Code Pop Up Model

Based on the figure above, it shows that the pop-up model only appears when the selectedQR state contains a value. The outer div uses fixed positioning combined with a semi-transparent black background to create an overlay effect that covers the entire viewport. The pop-up modal will close when clicking on this overlay sets selectedQR to null. e.stopPropagation() is used to prevent accidental closure from click inside the modal. There is a close button displayed in top right corner to allow user to close the pop-up model. The QRCodeSVG also renders the QR code to display at the larger size which is 250 based on the value stored in selectedQR.value.

QR Code Printing

To facilitate the sports equipment labelling, the system also provides a feature to print multiple QR codes for a single product.

```
const handlePrint = async (product) => {
  const copies = parseInt(
    prompt("Enter number of QR codes to print:", "1"),
    10
  );
  if (isNaN(copies) || copies < 1) return;
  try {
    // Generate QR code as Data URL
    const qrDataUrl = await QRCode.toDataURL(
      product.product_id.toString(),
      { width: 150 }
    );

    // Repeat based on quantity
    const qrHTML = Array.from({ length: copies })
      .map(
        () => `
      <div style="display:inline-block; margin:10px; text-align:center">
        <img src="${qrDataUrl}" width="150" height="150"/>
        <p style="font-size:12px; margin-top:4px">${product.name}</p>
      </div>`
      )
      .join("");

    const printWindow = window.open("", "_blank", "width=600,height=600");
    printWindow.document.write(`
      <html>
        <head>
          <title>Print QR Codes</title>
        </head>
        <body>
          <h3>QR Codes for ${product.name}</h3>
          ${qrHTML}
          <script>
            window.onload = function() {
              window.print();
              window.onafterprint = () => window.close();
            }
          </script>
        </body>
      </html>
    `);
    printWindow.document.close();
  } catch (err) {
    console.error("Failed to generate QR code", err);
  }
```

Figure 6.34: Code Segment for Handle Printing Function

The handlePrint Function is designed to generate and printing multiple QR codes for a selected product. It combines user input, QR code generation, dynamic HTML rendering and browser print functionality to achieve this. The function starts by asking the user to fill in the quantity QR code they want to print. It uses prompt() with a default value of 1. The result is parsed into an integer using parseInt. If user input is invalid which is not a number or quantity less than 1, the function will exist.

Next, it will generate QR Code as Data URL. The qrDataUrl function calls QRCode.toDataURL() and this generates a base64-encoded image string of the QR code. The QR code is created using product.product_id as its value and set to a width of 150px. This makes the QR code easy to implement because it's embedded directly in the HTML as an img src. After that, qrHTML creates HTML for multiple copies. The QR code HTML block is repeated based on the number of copies entered by the user.Array.from({ length: copies }) creates an array with the desired length, and .map() fills it with QR code HTML.This result that each QR block has QR image itself and product's name displayed below it. Then, printWindow is call and a new window is opened using window.open(). The function creates a complete HTML document into this window which has title, header showing product name, the dynamically generated QR code blocks and a script to automatically print and close the window afterward. The script inside ensures that the print dialog opens immediately and closes automatically after printing when the page loads (window.onload = window.print()). The result output is shown in figure xx.

Backend- Product Controller

```
    public function index(Request $request)
{
    $query = Product::query();

    if ($request->has('outlet') && $request->outlet !== '') {
        $query->where('outlet', $request->outlet);
    }

    $query->orderBy('outlet', 'desc');

    return $query->get();
}
```

Figure 6.35: Code Segment to Retrieve A List of Products

The index function retrieves a list of products from the database. It allows optional filtering by the outlet query parameter. The function will check if an outlet is specified and applies a where filter based on the request. Then, the results are sorted by outlet in descending order before returned as JSON. Therefore, this provides a flexible way to list product dynamically.

```php
public function store(Request $request)
{
    $validated = $request->validate([
        'productId' => 'required|string|unique:products,product_id',
        'name'      => ['required','string','max:255',
        function ($attribute, $value, $fail) use ($request) {
                $existingProduct = Product::where('name', $value)
                    ->where('outlet', $request->outlet)
                    ->first();

                if ($existingProduct) {
                    $fail("The product name '{$value}' already exists in the {$request->outlet} outlet.");
                }
            }
        ],
        'quantity'  => 'required|integer|min:0',
        'status'    => 'required|in:active,inactive',
        'outlet'    => 'required|in:QM ROOM,UP STORE,DOWN STORE',
        'qrcode'    => 'nullable|string',
        'image'     => 'nullable|image|mimes:jpg,jpeg,png,gif|max:2048',
    ]);

     $imagePath = null;
        if ($request->hasFile('image')) {
            $imagePath = $request->file('image')->store('products', 'public');
        }

    $product = Product::create([
        'product_id' => $validated['productId'],
        'name'       => $validated['name'],
        'quantity'   => $validated['quantity'],
        'status'     => $validated['status'],
        'outlet'     => $validated['outlet'],
        'qrcode'     => $validated['qrcode'] ?? null,
        'image'      => $imagePath,
    ]);

    \App\Models\Inventory::create([
        'product_id' => $product->product_id,
        'instock'    => $validated['quantity'] ?? 0,
        'damage'     => 0,
        'missing'    => 0,
        'status'     => $validated['status'],
    ]);

    return response()->json([
        'message' => 'Product created successfully',
        'product' => $product,
    ], 201);
```

Figure 6.36: Code Segment for Store Function

Based on the figure above, the store function is used to handle creating new products. Firstly, it validates the input fields such as product ID, name, quantity, status, store, and optional images. The validation also ensures there is no duplicate product name exist within the same store else when clicking submit button, it will pop up message as shown in figure below.

Figure 6.37: Error Message Display for Duplicate Product Name

Besides, the uploaded images are stored in the public/product directory. The function also handles once a new product is created, it will also create the corresponding inventory record and initialize the instock quantity and damage and missing quantity will at default 0. If the product is successfully created it will return a JSON response to show that the product is successfully created.

```
public function show($productId)
{
    return Product::where('product_id', $productId)->firstOrFail();
}
```

Figure 6.38: Code Segment to Display Product

Based on the figure above, the show function retrieves each of the products by using the product_id. It returns firstOrFail mechanism and returns a 404 response automatically if the product does not exist. This function is used to simple read operation for editing and display purposes.

```php
public function update(Request $request, $productId)
    $product = Product::where('product_id', $productId)->firstOrFail();
    $validated = $request->validate([
        'name'      => ['required','string','max:255',
            function ($attribute, $value, $fail) use ($request, $product) {
                $existingProduct = Product::where('name', $value)
                    ->where('outlet', $request->outlet)
                    ->where('product_id', '!=', $product->product_id)
                    ->first();
                if ($existingProduct) {
                    $fail("The product name '{$value}' already exists in the {$request->outlet} outlet.");
                }
            }
        ],
        'quantity' => 'required|integer|min:0',
        'status'   => 'required|in:active,inactive',
        'outlet'   => 'nullable|string|max:255',
        'image'    => 'nullable|image|mimes:jpg,jpeg,png,gif|max:2048',
    ]);

    if ($request->hasFile('image')) {
        if ($product->image) {
            Storage::disk('public')->delete($product->image);
        }
        $validated['image'] = $request->file('image')->store('products', 'public');
    } else {
        // keep existing image if none uploaded
        $validated['image'] = $product->image;
    }
    // Update the product row
    $product->update([
        'name'      => $validated['name'],
        'quantity' => $validated['quantity'],
        'status'   => $validated['status'],
        'outlet'   => $validated['outlet'],
        'image'    => $validated['image'],
    ]);

    $inventory = Inventory::where('product_id', $product->product_id)->first();
    if (! $inventory) {
        $inventory = Inventory::create([
            'product_id' => $product->product_id,
            'instock'    => max(0, $validated['quantity']),
            'damage'     => 0,
            'missing'    => 0,
            'status'     => $validated['status'],
            'reserved'   => 0,
            'rented'     => 0,
        ]);
    } else {
        $occupied = intval($inventory->reserved) + intval($inventory->rented)+ intval($inventory->missing) + intval($inventory->damage);
        $newTotal = intval($validated['quantity']);
        $newInstock = $newTotal - $occupied;
        $inventory->instock = max(0, $newInstock);
        $inventory->status = $validated['status'] ?? $inventory->status;
        $inventory->save();
    }
    return response()->json([
        'message'   => 'Product updated successfully',
        'product'   => $product,
        'inventory' => $inventory,
```

Figure 6.39: Code Segment for Update Purpose

Based on the figure above, the update function finds the product that needs to be updated by using product_id that provided in request. If the product does not exist, it throws 404 error through firstOrFail() to ensure the function always operate on a valid product. Before updating, the function validates the incoming request data. It checks required fields like name, quantity, status, and validates optional fields like outlet and image. It also set a constraint that same product names are not allowed to exist within the same outlet. The function then checks if new image has been uploaded. It deletes the old images from storage and stores a new one in the public/products directory. If no new image is provided, it keeps the old image. Once the validation and image handling

are completed, the product details are updated in the database. The fields updated include the product's name, quantity, status, outlet, and image. Then, the function updates the related inventory record to ensure it consistent with the product's updated details. If the inventory record is newly created, it will created damage, missing, reserved and rented column with default values 0. If the inventory already exists, it recalculates the instock value by decreasing the occupied quantity (reserved, rented, missing, damaged) from the total updated quantity. This prevents negative stock values and ensures accurate stock tracking. Lastly, the function return a JSON response message that the product and related inventory has created successfully.

```php
public function destroy($productId)
{
    $product = Product::where('product_id', $productId)->firstOrFail();

    if ($product->image) {
        Storage::disk('public')->delete($product->image);
    }
    // Delete related inventory via relationship
    $product->inventory()?->delete();

    $product->delete();

    return response()->json([
        'message' => 'Product and related inventory deleted successfully.',
    ]);
}
```

Figure 6.40: Code Segment for Destroy Function

Based on the figure above, the destroy function is used to delete a product and the inventory related to the product will also be deleted. Before deleting, the function checks if the product has an uploaded image and removes it from the storage to prevent orphaned files. After that, it deletes the related inventory records and remove the product itself. After deleting the product, it shows the message to notify user that the product already deleted successfully.

6.3.1.4 Inventory Management

The inventory management module is used to display and manage the inventory details based on different outlets. It allows user to view inventory items in a table format which have Product ID, Name, In Stock, Damage, Missing, Reserved, and Rented quantities columns.

```
const fetchInventories = async () => {
  try {
    const res = await axios.get("/api/inventories");
    setInventories(res.data);
  } catch (err) {
    console.error("Error fetching inventory", err);
  }
};

useEffect(() => {
  fetchInventories();
}, []);
```

Figure 6.41: Code Segment for Fetching Data from API

Based on above figure, fetchInventories function is used to retrieve and display the list of inventories from /api/inventories so that the system can always get the latest data without requiring manual refresh. The response data is stored by using setInventories() when the validation is successful else it will display error message for exception handling. This function allows administrator and member to view available products in real time.

Backend- Inventory Controller

```php
public function index()
    {
        // join with products to get product name
        $inventories = Inventory::join('products', 'inventories.product_id', '=', 'products.product_id')
            ->select(
                'inventories.id',
                'inventories.product_id',
                'products.name',
                'products.image',
                'products.outlet',
                'inventories.instock',
                'inventories.damage',
                'inventories.missing',
                'inventories.status',
                'inventories.reserved',
                'inventories.rented',
            )
            ->get();

        return response()->json($inventories);
    }
```

Figure 6.42: Code Segment to Retrieve to Inventory Data

Based on the figure above, the index function shows a list of all inventory records and the product details. It uses a join between the inventories and products tables so that both product-specific attributes (like name, image, and outlet) and inventory-related attributes (like instock, damage, missing, reserved, and rented) are returned together. This allows the frontend or API callers to display a complete inventory view without requiring multiple queries.

```php
    public function show($productId)
{
    $inventory = Inventory::join('products', 'inventories.product_id', '=', 'products.product_id')
        ->select(
            'inventories.id',
            'inventories.product_id',
            'products.name',
            'products.image',
            'products.outlet',
            'inventories.instock',
            'inventories.damage',
            'inventories.missing',
            'inventories.status',
            'inventories.reserved',
            'inventories.rented',
        )
        ->where('inventories.product_id', $productId)
        ->firstOrFail();

    return response()->json($inventory);
}
```

Figure 6.43: Code Segments to Display Inventory Data

Based on the Figure above, the show function retrieves the inventory details for a specific product based on its product_id. It also joins the inventories table with products

to return a combined dataset of product and inventory information. It focuses on a single record by filtering with the given product ID and ensures data integrity with firstOrFail(), which throws an error if the product inventory does not exist.

```php
public function summary()
{
    $subQuery = Inventory::where('status', 'active')
        ->select('product_id')
        ->selectRaw('MAX(updated_at) as latest_updated_at')
        ->groupBy('product_id');

    $totals = Inventory::where('status', 'active')
        ->joinSub($subQuery, 'latest_inventories', function ($join) {
            $join->on('inventories.product_id', '=', 'latest_inventories.product_id')
                 ->on('inventories.updated_at', '=', 'latest_inventories.latest_updated_at');
        })
        ->selectRaw('
            COALESCE(SUM(instock), 0) as total_instock,
            COALESCE(SUM(damage), 0) as total_damage,
            COALESCE(SUM(missing), 0) as total_missing,
            COALESCE(SUM(reserved), 0) as total_reserved,
            COALESCE(SUM(rented), 0) as total_rented
        ')
        ->first();

    return response()->json([
        'instock' => (int) $totals->total_instock,
        'damage'  => (int) $totals->total_damage,
        'missing' => (int) $totals->total_missing,
        'reserved' => (int) $totals->total_reserved,
        'rented' => (int) $totals->total_rented,
    ]);
}
```

Figure 6.44: Code Segment for Generating Inventory Overview

Based on the figure above, the summary function is used to generate an overview of all active product inventory statuses. First, a subquery ensures only the most recently updated inventory record for each product is considered to prevent the outdated data from skewing results. The SUM function combined with COALESCE handles null values and calculating the total quantity for each status (in stock, damaged, missing, reserved, rented). The final response is a JSON object containing these totals, making this function ideal for dashboard summaries or high-level reporting.

6.3.1.5 StockCheck Management

The stockCheck module is designed to help the administrator and quarter master maintain accurate equipment availability records by validating and updating the condition of each product in the inventory.

```
// Fetch products by outlet
const fetchProducts = async (selectedOutlet) => {
  try {
    const res = await axios.get(`/api/products?outlet=${selectedOutlet}`);
    if (Array.isArray(res.data)) {
      setProducts(res.data);

      // Initialize stockData
      const initialData = {};
      res.data.forEach((p) => {
        initialData[p.product_id] = {
          instock: 0,
          damage: 0,
          missing: 0,
        };
      });
      setStockData(initialData);
    }
  } catch (err) {
    console.error("Error fetching products", err);
  }
};
```

Figure 6.45: Code Segment Fetches Products by Outlets

Based on above figure, the stock check allows the user to select date and outlet. After select the date and outlet, it will dynamically fetch the products stored in the chosen outlet from fetchProducts function. Each product is displayed in a table together with its original quantity. Then, the user is required to input the current quantity for each item based on it condition such as in stock, damage, or missing.

```
const handleSubmit = async (e) => {
  e.preventDefault();

  // ✅ Validation: instock + damage + missing must equal product.quantity
  for (let p of products) {
    const entry = stockData[p.product_id];
    const total = (entry?.instock || 0) + (entry?.damage || 0) + (entry?.missing || 0);

    if (total !== p.quantity) {
      alert(
        `Quantity mismatch for product "${p.name}" (ID: ${p.product_id}).\n` +
        `Expected total: ${p.quantity}, but got: ${total}`
      );
      return; // stop saving
    }
  }

  try {
    const payload = {
      date,
      outlet,
      items: stockData,
    };
    await axios.post("/api/stockchecks", payload);
    alert("Stock check saved successfully!");
    navigate("/stockcheck-list");
  } catch (err) {
    console.error("Error saving stock check", err);
    alert("Failed to save stock check.");
  }
};
```

Figure 6.46: Handle Submit Function

The handleSubmit function ensures data accuracy by validating the total of these 3 categories matches the product's original quantity. If the values do not align with the original quantity, the system send the alert message to user and prevents submission. Once the data is confirmed, the system combine it into a payload that containing the date outlet and updated stock details and send it through POST request to Laravel backend. This feature is essential because stock checks serve as a systematic way to verify the actual physical quantity of items with the recorded inventory to ensures that the future reservations and booking have reliable inventory data.

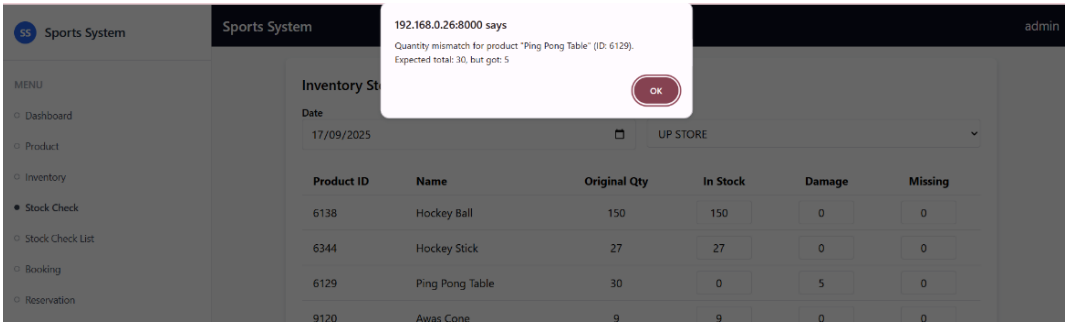


Figure 6.47: Stock Check Page - Mismatch Data

```
const fetchStockChecks = async () => {
    if (!date) {
        alert("Please select a date");
        return;
    }

    try {
        const params = { date };
        if (outlet) params.outlet = outlet;

        const res = await axios.get("/api/stockchecks", { params });
        setStockChecks(res.data);

        // Initialize current pages for each outlet
        const outlets = [...new Set(res.data.map((item) => item.outlet))];
        const initialPages = {};
        outlets.forEach((outlet) => {
            initialPages[outlet] = 1;
        });
        setCurrentPages(initialPages);
    } catch (err) {
        console.error("Error fetching stock checks", err);
    }
};
```

Figure 6.48: Fetch Stock Check Function

Based on the above figure, fetchStockChecks function used a GET request with query parameters to fetch the stock check records for the selected date and outlet. Once the data is retrieved, it is grouped by outlet and displayed in a table format. The "+ Stock Check" button at the top enables quick navigation to add new stock check page. Therefore, stock check list is easily accessible as it allows searching the stock check by date and outlet to narrow down the results.

Backend- StockCheck Controller

```
public function store(Request $request)
{
    $validated = $request->validate([
        'date' => 'required|date',
        'outlet' => 'required|string',
        'items' => 'required|array',
    ]);

    $stockCheck = StockCheck::create([
        'date' => $request->date,
        'outlet' => $request->outlet,
        'items' => $request->items,
    ]);

foreach ($request->items as $productId => $data) {
    $inventory = Inventory::where('product_id', $productId)->first();
    if ($inventory) {
        $inventory->update([
            'instock' => $data['instock'],
            'damage' => $data['damage'],
            'missing' => $data['missing'],
        ]);
    }
}

    return response()->json($stockCheck, 201);
}
```

Figure 6.49: Code Segment for Creating New Stock Check

Based on the figure above, the store function is used to create new stock check records. It validates the incoming request first to ensure the date, store, and product list are valid. Once the validation done, it creates a new entry in the StockCheck table to store the check date, outlet, and product details. The function also iterates through each submitted product item after submitting the request and it will update the in-stock quantity, damaged quantity, and missing quantity within the related inventory record.

```php
public function index(Request $request)
{
    $request->validate([
        'date' => 'required|date',
        'outlet' => 'nullable|string',
    ]);
    $query = StockCheck::query();
    // Required date filter
    $query->whereDate('date', $request->date);

    if ($request->filled('outlet')) {
        $query->where('outlet', $request->outlet);
    }

    $stockChecks = $query->get()->map(function ($check) {
        $items = [];
        foreach ($check->items as $productId => $data) {
            $product = Product::where('product_id', $productId)->first();
            $items[] = [
                'product_id' => $productId,
                'name' => $product ? $product->name : 'Unknown',
                'instock' => $data['instock'] ?? 0,
                'damage' => $data['damage'] ?? 0,
                'missing' => $data['missing'] ?? 0,
            ];
        }
        return [
            'id' => $check->id,
            'date' => $check->date,
            'outlet' => $check->outlet,
            'items' => $items,
        ];
    });
    return response()->json($stockChecks);
}
```

Figure 6.50: Index Function

Based on figure above, the index function is used to lists all stock checks filtered by date and by outlet. Firstly, it validates that the request contains a required date and an optional outlet field. The filter queries the StockCheck table and retrieves all matching records. For each stock check, the function generates the detailed product list by linking the product ID with the Product model to enable synchronised display the product names with inventory data. The response contains the stock check's ID, date, outlet, and detailed item information which is useful for generating weekly stock check reports.

```php
public function show($id)
{
    return StockCheck::findOrFail($id);
}
```

Figure 6.51: Show Function

Based on figure above, the show function is used to retrieve a single stock check record by its ID. It uses findOrFail to ensure that if the given ID does not match any existing record, the system will throw an error rather than returning an empty result.

6.3.1.6 Booking Management

The booking Management allows the administrator and quarter master to view the booking rather than adding new booking. This is because the booking process is targeted to be done directly by members through walk-in registration at sports center, where the equipment is scanned using its QR code for immediate check-out and check-in. By relying on the QR scanning process, the system ensures that the bookings are created in real time based on actual equipment usage and reduced the manual errors risk or duplicate records may occur. It also can reduce the work done by administrator and quarter master and make the process for booking become efficient and convenient.

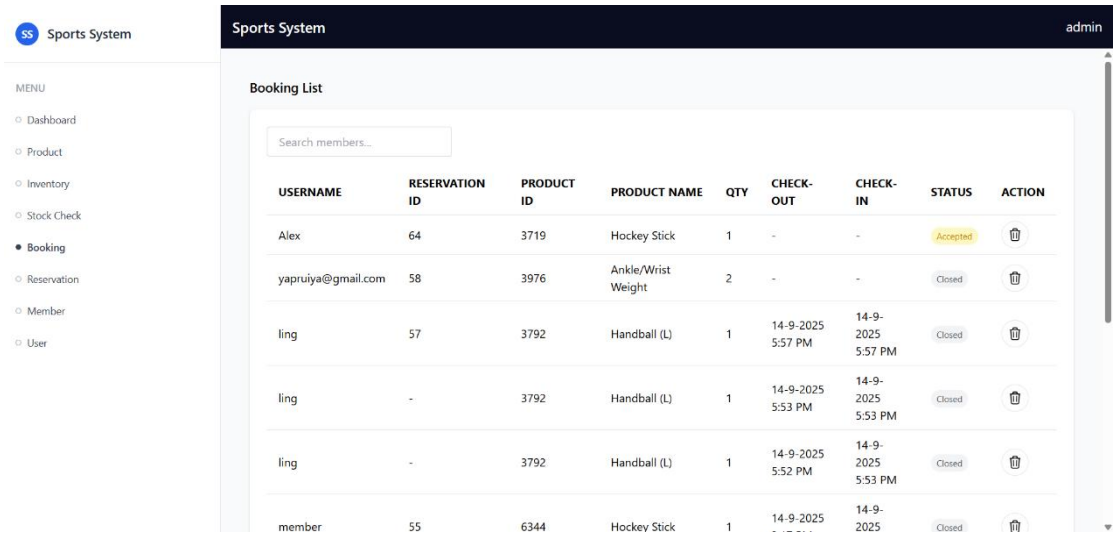


Figure 6.52: Booking List Page

Based on the image above, it shows that administrator and quarter master are not allowed to edit the booking records in order to protect the data integrity and ensure

accountability. Booking records must remain unaltered to provide an accurate historical record as it served as official log of equipment usage. If allowed users to edit the booking record, it could cause the inventory usage to become not accurate as the record can be edit. Therefore, the system ensures that the booking remains authentic, tamper-proof, and transparently auditable.

```
useEffect(() => {
    axios.get("/api/bookings").then((res) => {
        setBookings(res.data);
    });
}, []);

const filteredBookings = bookings.filter((b) =>
    b.username?.toLowerCase().includes(filterText.toLowerCase()),
);
```

Figure 6.53: Fetch Booking Data

The above figure shows how the bookings data is fetched, filtered and formatted. A request is sent to /api/booking and the response data is stored in the bookings state by using setBookings. The filteredBookings array applies the filter method to check whether the username has lowercase version of filterText input. This ensures that the bookings can search by using username.

```
const formatTime = (timeString) => {
    if (!timeString) return "";
    const date = new Date(timeString);

    const day = date.getDate();
    const month = date.getMonth() + 1;
    const year = date.getFullYear();

    const hours = date.getHours();
    const minutes = date.getMinutes();
    const ampm = hours >= 12 ? "PM" : "AM";
    const formattedHour = hours % 12 || 12;
    const formattedMinutes = minutes.toString().padStart(2, "0");

    return (
        <>
            {`${day}-${month}-${year}`}
            <br />
            {`${formattedHour}:${formattedMinutes} ${ampm}`}
        </>
    );
};
```

Figure 6.54: Format Time Function

The formatTime function is used to display date and time in a readable format. It used date-time string as input and concerts it into JavaScript Date object which result in the day, month,year,hours and minutes are extracted. The hours are converted to 12-hour format with AM/PM indicator and minutes are only allow 2 digit. Then it displays the date on 1 line and the formatted time on next line to provide a clean and user-friendly interface.

```
const handleDelete = async (bookingId) => {
    const confirmDelete = window.confirm(
        "Are you sure you want to delete this booking?",
    );
    if (!confirmDelete) return;

    try {
        await axios.delete(`/api/bookings/${bookingId}`);
        // Remove the deleted booking from the list
        setBookings((prevBookings) =>
            prevBookings.filter((b) => b.id !== bookingId),
        );
    } catch (error) {
        console.error("Failed to delete booking:", error);
        alert("An error occurred while deleting the booking.");
    }
};
```

Figure 6.55: Handle Delete Function

Based on the figure above, it shows that handleDelete function is used to delete each booking record through the confirmation process. When user clicks the delete button, a confirmation popup appears to prevent accidental deletions. If the user confirms the action, the system sends a DELETE request to the Laravel backend Api. Once the backend deleted the booking records from the database, the React frontend updates the state by filtering out the deleted booking without requiring full page reload.

Backend- Booking Controller

```
public function index()
{
    $bookings = Booking::latest()->get();
    return response()->json($bookings);
}
```

Figure 6.56: Index Function

Based on the figure above, the index function is used to retrieve a list of all bookings in descending order of creation time using latest(). It returns these bookings as JSON to make it easy for the frontend or administrators to view recent booking activity.

```php
public function show($id)
{
    $booking = Booking::findOrFail($id);
    return response()->json($booking);
}
```

Figure 6.57: Show Function

Based on the figure above, the show function is used to retrieve the details of a specific booking by its ID. It uses findOrFail to ensure that if no booking is found, the system will return an error.

```php
public function store(Request $request)
{
    // ✅ Validation rules
    $validated = $request->validate([
        'member_id'    => 'required|exists:members,id',
        'username'     => 'required|string|max:255',
        'product_id'   => 'required|string|max:255',
        'product_name' => 'required|string|max:255',
        'quantity'     => 'required|integer|min:1',
        'status'       => 'required|in:accepted,checkin,checkout,closed',
        'checkin_at'   => 'nullable|date',
        'checkout_at'  => 'nullable|date|after_or_equal:checkin_at',
        'outlet'       => 'nullable|string|max:255',
    ]);

    // ✅ Create booking
    $booking = Booking::create($validated);

    return response()->json([
        'message' => 'Booking created successfully!',
        'data'    => $booking,
    ], 201);
}
```

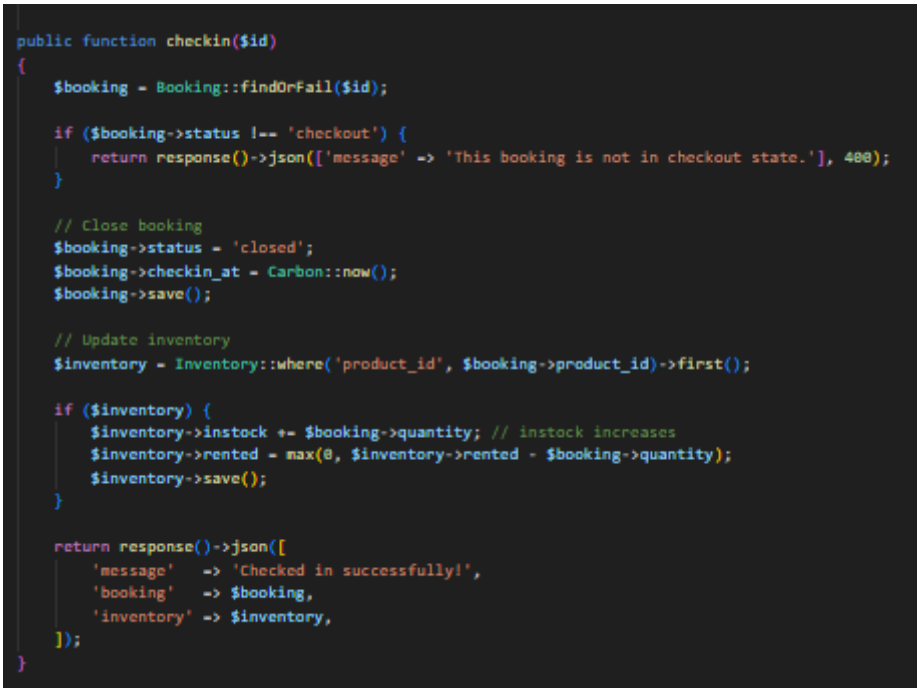
Figure 6.58: Store Function

Based on the figure above, the store function is used to handle the creation of a new booking. It validates incoming request data to ensure that required fields like member_id, product_id, quantity, and booking status are correct and consistent. Once validated, it creates a new booking record in the database and returns a success response. This function ensures that only valid bookings enter the system.

```php
public function checkout(Request $request)
{
    $validated = $request->validate([
        'member_id'      => 'required|exists:members,id',
        'username'       => 'required|string|max:255',
        'product_id'     => 'required|string|max:255',
        'product_name'   => 'required|string|max:255',
        'quantity'       => 'required|integer|min:1',
        'reservation_id' => 'nullable|exists:reservations,id',
    ]);

    if (!empty($validated['reservation_id'])) {
        $booking = Booking::where('reservation_id', $validated['reservation_id'])
            ->where('status', 'accepted')
            ->first();
        if ($booking) {
            $reservation = Reservation::find($validated['reservation_id']);
            if ($reservation) {
                $reserveDateTime = Carbon::parse("{$reservation->reserve_date} {$reservation->reserve_time}");
                $now = Carbon::now();

                // Allowed window: ±15 minutes
                if ($now->lt($reserveDateTime->subMinutes(15)) || $now->gt($reserveDateTime->addMinutes(30))) {
                    return response()->json([
                        'message' => 'You can only check out within 15 minutes before or after the reserved time.'
                    ], 400);
                }
            }
            // Update existing booking
            $booking->status = 'checkout';
            $booking->checkout_at = Carbon::now();
            $booking->save();
            // Update inventory
            $inventory = Inventory::where('product_id', $booking->product_id)->first();
            if ($inventory) {
                $inventory->instock = max(0, $inventory->instock - $booking->quantity);
                $inventory->reserved = max(0, $inventory->reserved - $booking->quantity);
                $inventory->rented += $booking->quantity;
                $inventory->save();
            }

            return response()->json([
                'message'   => 'Checked out successfully from reservation booking!',
                'booking'   => $booking,
                'inventory' => $inventory,
            ]);
        }
    }
    // Normal checkout (no reservation)
    $activeBooking = Booking::where('member_id', $validated['member_id'])
        ->where('product_id', $validated['product_id'])
        ->where('status', 'checkout')
        ->first();
    if ($activeBooking) {
        return response()->json([
            'message' => 'You already have this product checked out. Please check it in before checking out again.',
            'booking' => $activeBooking
        ], 400);
    }
    $booking = Booking::create([
        ...$validated,
        'status'      => 'checkout',
        'checkout_at' => Carbon::now(),
    ]);
    $inventory = Inventory::where('product_id', $validated['product_id'])->first();
    if ($inventory) {
        $inventory->instock = max(0, $inventory->instock - $validated['quantity']);
        $inventory->rented += $validated['quantity'];
        $inventory->save();
    }
    return response()->json([
        'message'   => 'Checked out successfully!',
        'booking'   => $booking,
        'inventory' => $inventory,
    ]);
}
```

Figure 6.59: Checkout Function

Based on the figure above, the checkout function is used to handle both reservation-based and normal checkouts. If the booking is based on reservation, it verifies the timing rules and only allowing checkout within a 15-minute window before or after the reserved time. If valid, it updates the booking status, decreases inventory instock and reserved quantity, and increases rented quantity. For non-reservation checkouts, it prevents duplicate active bookings, creates new booking and updates inventory for the same product and member to ensure the sport equipment distribution under control and prevent misuse.



```php
public function checkin($id)
{
    $booking = Booking::findOrFail($id);

    if ($booking->status !== 'checkout') {
        return response()->json(['message' => 'This booking is not in checkout state.'], 400);
    }

    // Close booking
    $booking->status = 'closed';
    $booking->checkin_at = Carbon::now();
    $booking->save();

    // Update inventory
    $inventory = Inventory::where('product_id', $booking->product_id)->first();

    if ($inventory) {
        $inventory->instock += $booking->quantity; // instock increases
        $inventory->rented = max(0, $inventory->rented - $booking->quantity);
        $inventory->save();
    }

    return response()->json([
        'message'   => 'Checked in successfully!',
        'booking'   => $booking,
        'inventory' => $inventory,
    ]);
}
```

Figure 6.60: Checkin Function

Based on the figure above, the checkin function is used to complete the booking after equipment or items have been returned. It verifies the booking is in a checkout status and when check in, it records the checkin time and the booking status marks as closed. When checkin, it also updates inventory by increasing instock and reducing rented quantity to ensure that returned products are available for future use.

```php
public function closeExpiredReservations()
{
    $now = Carbon::now();
    $expiredBookings = Booking::where('reservation_id', '!=', null)
        ->where('status', 'accepted')
        ->get()
        ->filter(function($booking) use ($now) {
            $reservation = Reservation::find($booking->reservation_id);
            if (!$reservation) return false;
            $reserveDateTime = Carbon::parse("{$reservation->reserve_date} {$reservation->reserve_time}");
            return $now->gt($reserveDateTime->addMinutes(15));  // more than 15 mins after
        });

    foreach ($expiredBookings as $b) {
        $b->status = 'closed';
        $b->checkin_at = $b->checkout_at ?? null;
        $b->save();

        // Optionally update inventory: remove reserved, etc
        $inventory = Inventory::where('product_id', $b->product_id)->first();
        if ($inventory) {
            $inventory->reserved = max(0, $inventory->reserved - $b->quantity);
            $inventory->save();
        }
    }
}
```

Figure 6.61: Close Expired Reservation Function

Based on the figure above, the closeExpiredReservations function is used to close bookings tied to reservations that have expired automatically. User must check out the sport equipment reserved within the reserve date and time else the status marks as closed. It also adjusts inventory by reducing reserved quantities, releasing the equipment for others to use and preventing the equipment from being locked long-term.

```php
public function destroy($id)
{
    $booking = Booking::findOrFail($id);
    // Fetch inventory
    $inventory = Inventory::where('product_id', $booking->product_id)->first();

    if ($inventory) {
        switch ($booking->status) {
            case 'checkout':
                // Return items to stock and reduce rented
                $inventory->instock += $booking->quantity;
                $inventory->rented = max(0, $inventory->rented - $booking->quantity);
                break;

            case 'accepted':
                // Remove from reserved
                $inventory->reserved = max(0, $inventory->reserved - $booking->quantity);
                break;

            case 'checkin':
                $inventory->instock += $booking->quantity;
                $inventory->rented = max(0, $inventory->rented - $booking->quantity);
                break;

        }

        $inventory->save();
    }

    $booking->delete();

    return response()->json([
        'message' => 'Booking deleted successfully and inventory updated!',
    ]);
}
```

Figure 6.62: Destroy Function

Based on the figure above, the destroy function is used to handle the deletion of a booking while ensuring that the inventory data remains accurate and consistent with actual item availability. findOrFail is used to find the booking records and it ensures an error is thrown if the booking does not exist. Once the booking is found, the system retrieves the related inventory record based on the product_id of the booking. Before deleting the booking, the function checks the booking's status and adjusts the inventory accordingly. If the status is checkout, the system increases the instock quantity and reduces the rented quantity as it assumed the items are returned. If the status is accepted which means that the item was reserved but not yet checked out, the reserved quantity is reduced since the reservation is being cancelled. For a checkin status, it is similar with the 'checkout' and it increases instock items and reducing the rented quantity. The inventory is then saved with these updates. Once the inventory is updated, the function deletes the booking record itself and returns a JSON response to confirm that the

booking has been deleted successfully, and the inventory has been updated. This ensures that every booking deletion not only removes the record but also keeps the stock levels accurate, preventing inconsistencies between bookings and inventory.

```php
public function myBookings(Request $request)
{
    $memberId = $request->query('member_id');

    if (!$memberId) {
        return response()->json(['message' => 'Member ID is required'], 400);
    }

    $bookings = Booking::leftJoin('products', 'bookings.product_id', '=', 'products.product_id')
        ->leftJoin('reservations', 'bookings.reservation_id', '=', 'reservations.id')
        ->select(
            'bookings.*',
            'products.image as product_image',
            'reservations.reserve_date',
            'reservations.reserve_time'
        )
        ->where('bookings.member_id', $memberId)
        ->orderBy('bookings.created_at', 'desc')
        ->get();

    return response()->json($bookings);
}
```

Figure 6.63: myBookings Function

Based on the figure above, the myBookings function is used to retrieves all bookings for a specific member. It requires a member_id and joins with both the products and reservations tables to include product images and reservation details in the results. This provides members with a detailed history of their bookings.

6.3.1.7 Reservation Management

```
const fetchReservations = async () => {
    try {
        const res = await axios.get("/api/reservations");
        setReservations(res.data);

        // Initialize current pages for each outlet
        const outlets = [
            ...new Set(res.data.map((item) => item.outlet)),
        ];
        const initialPages = {};
        outlets.forEach((outlet) => {
            initialPages[outlet] = 1;
        });
        setCurrentPages(initialPages);
    } catch (err) {
        console.error("Error fetching reservations", err);
    }
};
fetchReservations();
[]);
```

Figure 6.64: Fetch Reservation Function

```
const filtered = reservations.filter((res) =>
    res.username?.toLowerCase().includes(filterText.toLowerCase()),
);

// Group by outlet
const groupedByOutlet = filtered.reduce((acc, res) => {
    if (!acc[res.outlet]) acc[res.outlet] = [];
    acc[res.outlet].push(res);
    return acc;
}, {});
```

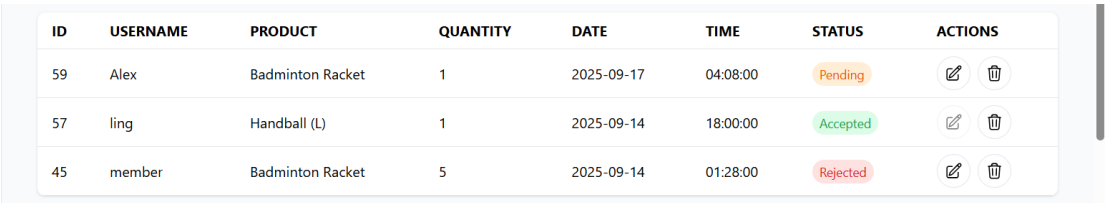
Figure 6.65: Search Function

Based on the figure above, the fetchReservations funcrion is used to retrieve the reservation data from the Laravel backend through API call using Axios and the data is display in structured and grouped by outlets. It also implements search function that allows the user to filter the reservation by username. Each outlet's reservations are separated and displayed in individual tables with pagination.

```
const getStatusClasses = (status) => {
    switch (status?.toLowerCase()) {
        case "pending":
            return "text-orange-600 bg-orange-100 px-2 py-1 rounded-full text-sm";
        case "accepted":
            return "text-green-600 bg-green-100 px-2 py-1 rounded-full text-sm";
        case "rejected":
            return "text-red-600 bg-red-100 px-2 py-1 rounded-full text-sm";
        default:
            return "text-gray-600 bg-gray-100 px-2 py-1 rounded-full text-sm";
    }
};
```

Figure 6.66: Get Status Classes Function

The status field is displayed using color-coded labels and each module status will have their own color-coded labels to improve readability and quick decision making. For example, pending in orange, accepted in green or rejected in red.



ID	USERNAME	PRODUCT	QUANTITY	DATE	TIME	STATUS	ACTIONS
59	Alex	Badminton Racket	1	2025-09-17	04:08:00	Pending	
57	ling	Handball (L)	1	2025-09-14	18:00:00	Accepted	
45	member	Badminton Racket	5	2025-09-14	01:28:00	Rejected	

Figure 6.67: Reservation Page - Different Status

It also provides actions button for editing and deleting reservations. Edit function is controlled by the validation rules which is when the reservation is past (handles by isReservationDatePast function), and reservation status is completed; the reservation cannot be modified to ensure data integrity and avoid accidental changes.

```
const isReservationDatePast = (reserveDate) => {
    if (!reserveDate) return false;

    const today = new Date();
    const reservationDate = new Date(reserveDate)

    // Reset time parts to compare only dates
    today.setHours(0, 0, 0, 0);
    reservationDate.setHours(0, 0, 0, 0);

    return reservationDate < today;
};
```

Figure 6.68: ReservationPast Function

```
<div className="relative group">
    <button
        disabled={
            isEditDisabled
        }
        className={`${
            isEditDisabled
                ? "opacity-50 cursor-not-allowed"
                : "hover:text-black-800"
        } text-black-600 border border-black-600 rounded-full p-2`}
        onClick={() =>
            navigate(
                `/reservation-edit/${res.id}`,
            )
        }
    >
        <svg
            xmlns="http://www.w3.org/2000/svg"
            fill="none"
            viewBox="0 0 24 24"
            strokeWidth="1.5"
            stroke="currentColor"
            className="w-5 h-5"
        >
            <path
                strokeLinecap="round"
                strokeLinejoin="round"
                d="m16.862 4.487 1.687-1.688a1.875 1.875 0 1 1 2.652 2.652L10.582 16.07a4.5 4.5 0 0 1-1.897 1.13L6 18l.8-2.68
            />
        </svg>
    </button>
```

```
{isEditDisabled && (
    <div className="absolute bottom-full left-1/2 transform -translate-x-1/2 mb-2 px-2 py-1 bg-gray-800 text-white text-xs roun
        {
            disabledReason
        }
        <div className="absolute top-full left-1/2 transform -translate-x-1/2 border-4 border-transparent border-t-gray-800"></
    </div>
)}
```

Figure 6.69: Disabled Buton

```
const handleDelete = async (id) => {
    const confirmDelete = window.confirm(
        "Are you sure you want to delete this reservation?",
    );
    if (!confirmDelete) return;

    try {
        await axios.delete(`/api/reservations/${id}`);
        // Remove deleted reservation from state
        setReservations((prev) => prev.filter((res) => res.id !== id));
    } catch (error) {
        console.error("Error deleting reservation:", error);
        alert("Failed to delete reservation. Please try again.");
    }
};
```

Figure 6.70: Handle Delete Function

For deletion, the system used the handleDelete function to prompt a confirmation popup by using window.confirm() before sending a DELETE request to Laravel. The reservation is removed from both the database and the frontend when the handleDelete function is successfully.

Add function

The reservation add is used to create new reservations by selecting the member, outlet, product, date, time and quantity,

```
useEffect(() => {
  const fetchData = async () => {
    try {
      const [membersRes, productsRes] = await Promise.all([
        axios.get("/api/members"),
        axios.get("/api/inventories"),
      ]);
      setMembers(membersRes.data);
      setProducts(productsRes.data);
    } catch (err) {
      console.error("Error fetching data", err);
    }
  };
  fetchData();
}, []);

// Filter products when outlet changes
useEffect(() => {
  if (outlet) {
    setFilteredProducts(products.filter((p) => p.outlet === outlet));
    setSelectedProduct(null);
  }
}, [outlet, products]);
```

Figure 6.71: FetchData Function

Based on the above code snippet, the fetchData function is used to fetch members and product data. When the user selects an outlet, the product list is automatically filtered and only show items that available in that outlet to ensure the product allocation in accurate.

```
useEffect(() => {
  const fetchAvailableQuantities = async () => {
    if (reserveDate && outlet) {
      try {
        const response = await axios.get(`/api/reservations/available-quantities`, {
          params: { date: reserveDate, outlet }
        });
        setAvailableQuantities(response.data);
      } catch (err) {
        console.error("Error fetching available quantities", err);
      }
    }
  };

  fetchAvailableQuantities();
}, [reserveDate, outlet]);
```

Figure 6.72: Fetch Available Quantity Function

The system sends a request to /api/reservations/available-quantities to retrieve updated availability to retrieve updated availability when the reservation date or outlet is change in order to prevent overbooking. When the reservation is added, the form is validated by using the handleSubmit function to verify that a member and product chosen, the quantity does not exceed the available stock, and the date is not in past. The selected details are then submitted through POST request to store the reservation in database and if the reservation is added successfully, the user will notify and it will redirect back to the reservation-list page.

```
const handleSubmit = async (e) => {
  e.preventDefault();

  if (!selectedMember) {
    alert("Please select a member!");
    return;
  }

  if (!selectedProduct) {
    alert("Please select a product!");
    return;
  }
  if (!reserveDate) {
  const availableQty = availableQuantities[selectedProduct.product_id] || selectedProduct.instock;
  if (quantity > availableQty) {
    alert(`Only ${availableQty} items available for ${selectedProduct.name} on ${reserveDate}`);
    return;
  }

  try {
    await axios.post("/api/reservations", {
      member_id: selectedMember.id,
      username: selectedMember.username,
      product_id: selectedProduct.product_id,
      product_name: selectedProduct.name,
      outlet,
      quantity,
      reserve_date: reserveDate,
      reserve_time: reserveTime,
    });

    alert("Reservation added successfully!");
    if (onSuccess) onSuccess();
    navigate("/reservation-list");
  } catch (err) {
    console.error("Add reservation failed", err);
  }
};
```

Figure 6.73: Handle Submit Function

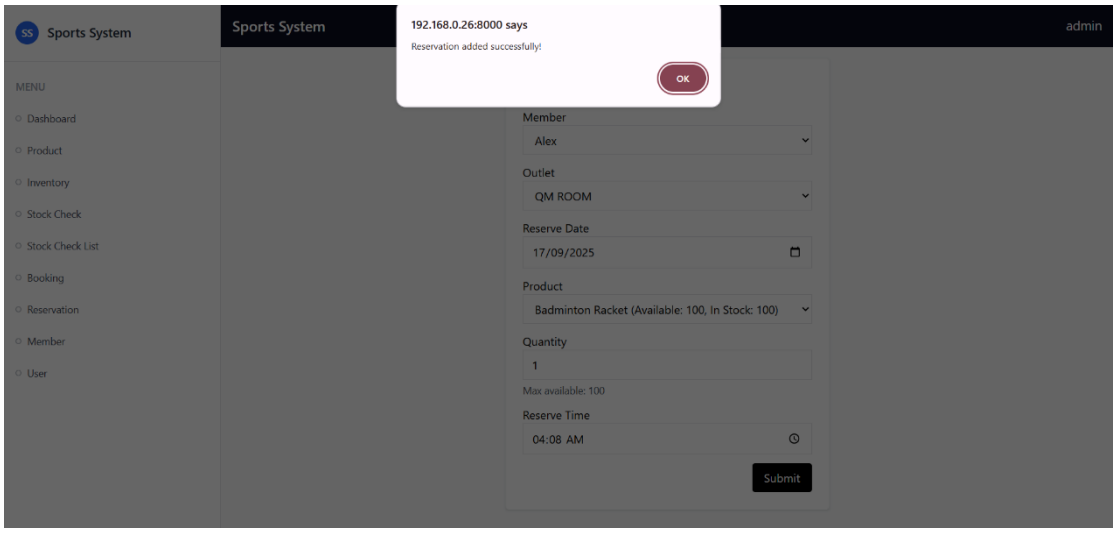


Figure 6.74: Reservation Page -Successful Notification

Email notification

The reservation notification is designed to auto send email to member when their reservation is rejected or accepted. It ensures that users are kept informed about the status of their reservation without requiring manual follow-up. The implementation follows three main steps: creating a Mailable class, designing an email template, and updating the controller method to send the email when a reservation is rejected or accepted.

```
php artisan make:mail ReservationAcceptedMail
```

Firstly, run above command in the terminal to create a mailable class to handle the email content preparation. It is implemented using a Laravel Mailable class combined with a Blade email template.

```php
namespace App\Mail;

use Illuminate\Bus\Queueable;
use Illuminate\Mail\Mailable;
use Illuminate\Queue\SerializesModels;

class ReservationAcceptedMail extends Mailable
{
    use Queueable, SerializesModels;

    public $reservation;
    public $booking;

    public function __construct($reservation, $booking)
    {
        $this->reservation = $reservation;
        $this->booking = $booking;
    }

    public function build()
    {
        return $this->subject('Your Reservation Has Been Accepted!')
                    ->view('emails.reservation_accepted')
                    ->with([
                        'reservation' => $this->reservation,
                        'booking' => $this->booking,
                    ]);
    }
}
```

Figure 6.75: Reservation Accepted Mail

Based on above figure, it shows that the reservation notification is implemented in Laravel using Mailable class which is ReservationAcceptedMail. When

the reservation is approved, the system passes both the reservation and booking details into mail class constructor. The email is configured with a subject line ("Your Reservation Has Been Accepted!") and linked to a Blade view (emails.reservation_accepted) that serves as the email template.

```blade
<!DOCTYPE html>
<html>
<head>
    <title>Reservation Accepted</title>
</head>
<body>
    <h2>Your Reservation Has Been Accepted!</h2>

    <p>Hello {{ $reservation->username }},</p>

    <p>We are pleased to inform you that your reservation has been accepted.</p>

    <h3>Reservation Details:</h3>
    <ul>
        <li><strong>Product:</strong> {{ $reservation->product_name }}</li>
        <li><strong>Quantity:</strong> {{ $reservation->quantity }}</li>
        <li><strong>Reservation Date:</strong> {{ $reservation->reserve_date }}</li>
        <li><strong>Reservation Time:</strong> {{ $reservation->reserve_time }}</li>
        <li><strong>Outlet:</strong> {{ $reservation->outlet }}</li>
        <li><strong>Booking ID:</strong> {{ $booking->id }}</li>
    </ul>

    <p>Please arrive on time to collect your items. If you have any questions, please contact us at 0123456789.</p>

    <p>Thank you for using our service!</p>

    <hr>
    <p><small>This is an automated message. Please do not reply to this email.</small></p>
</body>
</html>
```

Figure 6.76: ReservationAcceptedMail.blade.php

The blade template formats the notification by using HTML layout. It greets the member by username and displaying the needed reservation details as shown in above Figure to ensure that members is clear about the notification message. Therefore, the reservation notification ensures a better communication between system and its users. If the reservation is rejected, the members also will receive notification to notify them. Below is the output about the notification received format.

It also needs to add below information into .env().

```
MAIL_MAILER=smtp
MAIL_HOST=smtp.gmail.com
MAIL_PORT=587
MAIL_USERNAME=eugenetiang02@gmail.com
MAIL_PASSWORD="kykb cqwz mabx tgyh"
MAIL_ENCRYPTION=tls
MAIL_FROM_ADDRESS=eugenetiang02@gmail.com
MAIL_FROM_NAME="Sports Inventory System"
```

Figure 6.77: Mail Setup in .env()

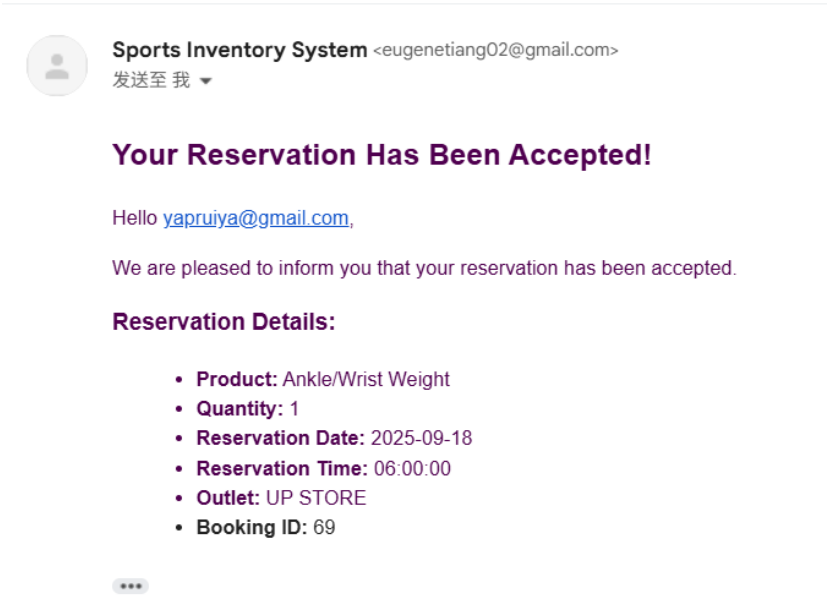


Figure 6.78: The Output Results of Notification

Backend -Reservation Controller

```
    public function index()
{
    $reservations = Reservation::leftJoin('bookings', 'reservations.id', '=', 'bookings.reservation_id')
        ->select(
            'reservations.*',
            'bookings.status as booking_status'
        )
        ->orderBy('reservations.created_at', 'desc')
        ->get();

    return response()->json($reservations);
}
```

Figure 6.79: Index Function

Based on the figure above, the index function is used to retrieve all reservations with linked booking status. It uses a left join between the reservations and bookings tables so that even reservations without a booking are included. The query selects all reservation details and adds the booking's status field, ordering results by the latest created date. The results are returned as a JSON response and giving administrators and quarter master an overview of reservation requests and current booking progress.

```php
public function store(Request $request)
{
    $validated = $request->validate([
        'member_id'     => 'required|integer',
        'username'      => 'required|string',
        'product_id'    => 'required|integer',
        'product_name'  => 'required|string',
        'outlet'        => 'nullable|string',
        'quantity'      => 'required|integer|min:1',
        'reserve_date'  => 'required|date',
        'reserve_time'  => 'required',
        'status'        => 'in:pending,accepted,rejected'
    ]);

    $reservation = Reservation::create($validated);

    return response()->json([
        'message' => 'Reservation created successfully',
        'reservation' => $reservation
    ], 201);
}
```

Figure 6.80: Store Function

Based on the figure above, the store function is used to create new reservations. It validates mandatory fields such as member ID, product details, reservation date/time, and quantity required. If the validation is successful, the function creates a new reservation record in the database and returns a success message and the stored reservation data. This ensures that every reservation request is accurately recorded and linked to the relevant member and product.

```php
public function availableQuantities(Request $request)
{
    $date = $request->query('date');
    $outlet = $request->query('outlet');
    $excludeReservationId = $request->query('excludeReservationId');

    if (!$date || !$outlet) {
        return response()->json(['message' => 'Date and outlet are required'], 400);
    }

    $acceptedReservations = Reservation::where('reserve_date', $date)
        ->where('outlet', $outlet)
        ->where('status', 'accepted');

    if ($excludeReservationId) {
        $acceptedReservations->where('id', '!=', $excludeReservationId);
    }

    $reservedQuantities = $acceptedReservations
        ->select('product_id', DB::raw('SUM(quantity) as total_reserved'))
        ->groupBy('product_id')
        ->pluck('total_reserved', 'product_id');

    $inventories = Inventory::where('status', 'active')->get();

    $availableQuantities = [];

    foreach ($inventories as $inventory) {
        $reserved = $reservedQuantities[$inventory->product_id] ?? 0;
        $availableQuantities[$inventory->product_id] = max(0, $inventory->instock - $reserved);
    }

    // print_r($availableQuantities);exit;

    return response()->json($availableQuantities);
}
```

Figure 6.81: Available Quantity Function

Based on the figure above, the availableQuantities function is used to calculate the remaining items available for reservation on a given date and outlet. It validates the provided date and outlet, then gathers all accepted reservations for that day while excluding an optional reservation ID (useful during edits). It summarizes reservation quantities per product by using database aggregation. Next, it fetches all active inventory items and reduces the reserved amounts from the in-stock value to calculate available quantities. The function ensures that only non-negative values are returned to provide real-time stock availability for reservation validation.

```php
public function update(Request $request, $id)
{
    $reservation = Reservation::findOrFail($id);

    // 🔒 If already accepted and linked booking is active → forbid editing
    if ($reservation->status === 'accepted') {
        $linkedBooking = Booking::where('reservation_id', $reservation->id)->first();

        if ($linkedBooking && in_array($linkedBooking->status, ['checkout', 'closed'])) {
            return response()->json([
                'message' => 'This reservation is locked because it has been used in a booking.'
            ], 403);
        }
    }

    $validated = $request->validate([
        'quantity'      => 'required|integer|min:1',
        'reserve_date'  => 'required|date',
        'reserve_time'  => 'required',
        'status'        => 'in:pending,accepted,rejected'
    ]);

    $reservation->update($validated);

    return response()->json([
        'message' => 'Reservation updated successfully',
        'reservation' => $reservation
    ]);
}
```

Figure 6.82: Update Function

Based on the figure above, the update function is used to modify an existing reservation to maintain integrity. If a reservation has already been accepted and is linked to a booking that is in checkout or closed status, the system blocks updates to prevent tampering with completed transactions and an error message is sent to notify users. Otherwise, users are allowed to edit the system. it validates the new inputs such as updated quantity, date, and status and updated the new data to the reservation record. The updated reservation is then returned as a successfully message.

```php
public function accept($id)
{
    $reservation = Reservation::findOrFail($id);
    $inventory = Inventory::where('product_id', $reservation->product_id)->first();

    if (!$inventory) {
        return response()->json(['message' => 'Inventory not found'], 404);
    }

    $reservedOnDate = Reservation::where('product_id', $reservation->product_id)
        ->where('reserve_date', $reservation->reserve_date)
        ->where('status', 'accepted')
        ->where('id', '!=', $reservation->id)
        ->sum('quantity');

    $available = $inventory->instock - $reservedOnDate;

    if ($reservation->quantity > $available) {
        return response()->json([
            'message' => 'Not enough inventory available. Only ' . $available . ' items left.'
        ], 400);
    }

    $reservation->status = 'accepted';
    $reservation->save();

    $inventory->reserved += $reservation->quantity;
    $inventory->save();

    $booking = Booking::create([
        'member_id'      => $reservation->member_id,
        'reservation_id' => $reservation->id,
        'username'       => $reservation->username,
        'product_id'     => $reservation->product_id,
        'product_name'   => $reservation->product_name,
        'quantity'       => $reservation->quantity,
        'status'         => 'accepted',
        'checkout_at'    => null,
    ]);

    try {
        $member = Member::find($reservation->member_id);
        if ($member && $member->username) {
            Mail::to($member->username)
                ->send(new ReservationAcceptedMail($reservation, $booking));
            $emailStatus = 'Email sent successfully';
        } else {
            $emailStatus = 'Member email not found';
        }
    } catch (\Exception $e) {
        $emailStatus = 'Email sending failed: ' . $e->getMessage();
    }
    return response()->json([
        'message'      => 'Reservation accepted, booking created, and inventory updated.',
        'email_status' => $emailStatus,
        'reservation' => $reservation,
        'booking'      => $booking,
        'inventory'    => $inventory,
    ]);
}
```

Figure 6.83: Accept Function

```php
public function reject($id)
{
    $reservation = Reservation::findOrFail($id);
    $reservation->status = 'rejected';
    $reservation->save();

    return response()->json([
        'message' => 'Reservation rejected',
        'reservation' => $reservation
    ]);
}
```

Figure 6.84: Reject Function

Based on the figure above, the reject function is used to reject the reservation request. It finds the reservation based on its ID and updated the status to ‘rejected’ and saves the change. A JSON response is returned with a confirmation message and the updated reservation. This function ensures members receive clear feedback when reservation request cannot be fulfilled.

```php
    public function destroy($id)
{
    $reservation = Reservation::find($id);

    if (!$reservation) {
        return response()->json(['message' => 'Reservation not found'], 404);
    }

    $reservation->delete();

    return response()->json(['message' => 'Reservation deleted successfully']);
}
```

Figure 6.85: Destroy Function

Based on the figure above, the destroy function is used to handle the deletion of reservations. It deletes the reservations from the database when the reservations exist and found. A success message is returned to confirm the removal. If the reservation is not found, a 404 response is sent. This ensures that only valid reservations are removed and that users are informed when attempting to delete non-existing records.

```php
public function myReservations(Request $request)
{
    $memberId = $request->query('member_id');

    if (!$memberId) {
        return response()->json(['message' => 'Member ID is required'], 400);
    }

    // Join with products to fetch image
    $reservations = Reservation::join('products', 'reservations.product_id', '=', 'products.product_id')
        ->select(
            'reservations.*',
            'products.image as product_image'
        )
        ->where('reservations.member_id', $memberId)
        ->whereIn('reservations.status', ['pending', 'rejected'])
        ->orderBy('reservations.created_at', 'desc')
        ->get();

    return response()->json($reservations);
}
```

Figure 6.86: myReservation Function

Based on the figure above, the myReservations function is used to retrieve a member's personal booking records. This function requires a member_id parameter and fetches all bookings associated with that member. It obtains product images by joining the products table to retrieve product images. It returns only reservations records with status of ‘Pending’ or ‘Rejected’ and it sorted in descending create time order.

```php
public function weeklyStats(Request $request)
{
    $range = $request->query('range', '7days');
    $today = Carbon::today();

    switch ($range) {
        case 'today':
            $timeSlots = [];
            for ($i = 0; $i < 24; $i++) {
                $timeSlots[] = $today->copy()->addHours($i)->format('Y-m-d H:00:00');
            }
            $startDate = $today;
            break;

        case 'yesterday':
            $yesterday = $today->copy()->subDay();
            $timeSlots = [];
            for ($i = 0; $i < 24; $i++) {
                $timeSlots[] = $yesterday->copy()->addHours($i)->format('Y-m-d H:00:00');
            }
            $startDate = $yesterday;
            break;

        case '30days':
            $timeSlots = [];
            for ($i = 29; $i >= 0; $i--) {
                $timeSlots[] = $today->copy()->subDays($i)->format('Y-m-d');
            }
            $startDate = $today->copy()->subDays(29);
            break;

        case '7days':
        default:
            $timeSlots = [];
            for ($i = 6; $i >= 0; $i--) {
                $timeSlots[] = $today->copy()->subDays($i)->format('Y-m-d');
            }
            $startDate = $today->copy()->subDays(6);
            break;
    }
```

Figure 6.87: Weekly Stas -Part1

```php
public function weeklyStats(Request $request)
    if ($range === 'today' || $range === 'yesterday') {
        $reservationData = Reservation::where('created_at', '>=', $startDate)
            ->where('created_at', '<', $startDate->copy()->addDay())
            ->select(DB::raw('DATE_FORMAT(created_at, "%Y-%m-%d %H:00:00") as time_slot'),
                    DB::raw('count(*) as count'))
            ->groupBy('time_slot')
            ->orderBy('time_slot')
            ->get()
            ->pluck('count', 'time_slot')
            ->toArray();
    } else {
        $reservationData = Reservation::where('created_at', '>=', $startDate)
            ->select(DB::raw('DATE(created_at) as date'),
                    DB::raw('count(*) as count'))
            ->groupBy('date')
            ->orderBy('date')
            ->get()
            ->pluck('count', 'date')
            ->toArray();
    }
    $dataset = [];
    $labels = [];
    foreach ($timeSlots as $slot) {
        $count = 0;

        if ($range === 'today' || $range === 'yesterday') {
            foreach ($reservationData as $time => $value) {
                if (strpos($time, substr($slot, 0, 13)) === 0) {
                    $count = $value;
                    break;
                }
            }
            $labels[] = Carbon::parse($slot)->format('H:i');
        } else {
            $count = $reservationData[$slot] ?? 0;

            if ($range === '7days') {
                $labels[] = Carbon::parse($slot)->format('m/d');
            } else {
                $labels[] = Carbon::parse($slot)->format('M d');
            }
        }

        $dataset[] = $count;
    }
    return response()->json([
        'labels' => $labels,
        'data'   => $dataset
    ]);
```

Figure 6.88: Weekly Stas -Part2

Based on the figure above, the weeklyStats function is used to provide an analytical feature that generates reservation statistics over different time periods in order to make it useful for dashboard. The main purpose of this function is to allow administrators to track reservation activity trends within specific timeframes such as today, yesterday, the past 7 days, or the past 30 days. This ensures that reservation data is presented in a structured format that can easily be visualized in charts or graphs.

It checks the range parameter from the request first, It sets a default to 7 days range. Based on the selected range, it dynamically builds time slots: hourly slots for today and yesterday, or daily slots for 7 days and 30 days. For example, if the range is today, it creates 24 hourly time slots representing each hour of the day, while for 7 days, it generates seven daily labels going back from today. Next, the function queries the reservations table to calculate the reservations were created within the chosen range. For hourly ranges (today and yesterday), it groups reservations by formatted hour (%Y-%m-%d %H:00:00), while for daily ranges (7 days and 30 days), it groups the data by date. The results are stored in an associative array to mapping each time slot or date to the number of reservations. After retrieving the raw data, the function iterates through the previously generated time slots to ensure that every slot has a corresponding count (defaulting to 0 if no reservations exist for that time). Labels are then formatted neatly — in H:i format for hours, m/d format for 7 days, and M d format for 30 days. These labels and values are compiled into arrays named labels and dataset. The function also returns the data as a JSON response with labels for chart axes and data for chart values. This implementation makes it easy to integrate the data into visualization libraries like Chart.js or Recharts to ensure administrators to monitor reservation activity patterns in real time.

6.3.1.8 User Management

The User Management Module is a core feature of the system that provides administrators with full control over handling user accounts through four main components: User Add, User Edit, User Change Password, and User List. The User Add component allows administrators to register new users by submitting details such as username, password, role, and status through a form. Upon submission, the data is sent via an Axios POST request to the backend API, and successful registration redirects the administrator back to the user list. The User Edit component retrieves the existing user's data using their unique ID, displays it in an editable form, and updates the details using an Axios PUT request. Additionally, it integrates a Change Password option that navigates to the User Change Password component. This component securely handles password updates by ensuring that the new password matches the confirmation before sending a PUT request to the backend to update the user's credentials. Finally, the User List component fetches all registered users from the

backend API, displays them in a paginated and searchable table, and provides action buttons for editing or deleting user accounts. Together, these components ensure a streamlined and secure workflow for managing user accounts effectively within the system.

```
const fetchUsers = async () => {
  try {
    const response = await axios.get("/api/users", {
      headers: {
        Accept: "application/json",
      },
    });
    console.log("Fetched users:", response);

    if (Array.isArray(response.data)) {
      setUsers(response.data);
    } else if (response.data.data && Array.isArray(response.data.data)) {
      setUsers(response.data.data);
    } else {
      setUsers([]);
    }
  } catch (err) {
    console.error(err);
    setError("Error fetching users");
  }
};

useEffect(() => {
  fetchUsers();
}, []);
```

Figure 6.89: FetchUser Function

Based on the above figure, it shows the how the data is fetched from the backend using Axios GET request. It uses Axios to send a GET request and specifies a request header with Accept: "application/json" to ensure the server responds in JSON format. Once the response is received, it checks whether the returned data is a plain array or wrapped in a data object.It also updates the users state with the retrieved list of users using the setUsers hook. If neither structure is valid, it sets an empty array to prevent errors. If the API call fails, the error is logged in the console and the setError hook updates the error state so it can be displayed to the user.

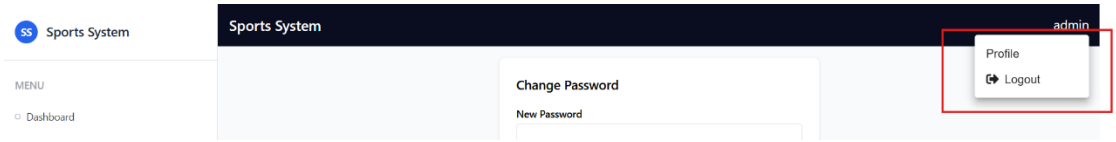


Figure 6.90: User Profile Page

After user click on edit button or Profile, it will navigate to user-edit page based on specific user that shown as below:

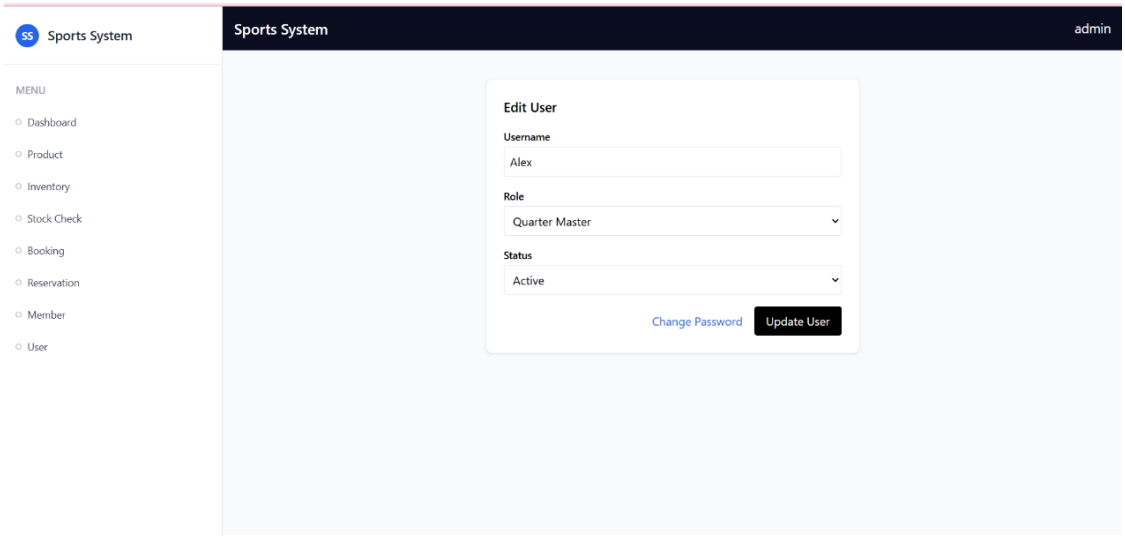


Figure 6.91: User Edit Page

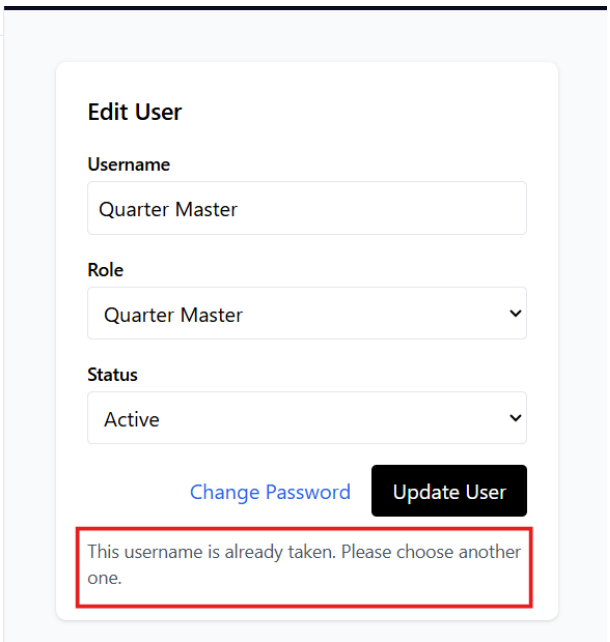


Figure 6.92: Error Message

The below code segment allows administrators to update user details in the system. The useEffect hook runs when the component is first loaded or when the id parameter changes, and it defines an asynchronous function fetchUser that sends a GET request to /api/users/${id} to retrieve the selected user's details. Once the data is fetched, the form state is updated with the user's username, role, and status, ensuring the form fields are pre-filled with the current information. If the request fails, an error message is displayed to the user. The handleChange function manages form updates by dynamically setting the input field values in the state based on the field being modified. It keeping the form controlled and synchronized with React state. The handleSubmit function handles form submission by sending a PUT request to /api/users/${id} with the updated form data. If the update is successful, it will be redirected to the user list page. If the validation errors such as a duplicate username or general failures are caught and displayed to guide the user. output that shows the validation is error.

```
// Fetch existing user data
useEffect(() => {
  const fetchUser = async () => {
    try {
      const response = await axios.get(`/api/users/${id}`);
            console.log("Fetched usersedit:", response);

      setForm({
        username: response.data.username,
        role: response.data.role,
        status: response.data.status,
      });
    } catch (err) {
      console.error("Error fetching user:", err);
      setMessage("Failed to load user data.");
    }
  };

  fetchUser();
}, [id]);

// Handle form changes
const handleChange = (e) => {
  setForm({ ...form, [e.target.name]: e.target.value });
};

// Handle form submission
const handleSubmit = async (e) => {
  e.preventDefault();
  try {
    await axios.put(`/api/users/${id}`, form);
    setMessage("User updated successfully.");
    navigate("/user-list"); // Redirect after success
  } catch (err) {
    if (err.response && err.response.status === 422) {
      const errors = err.response.data.errors;
      if (errors.username) {
        setMessage(errors.username[0]);
      } else {
        setMessage('Validation error. Please check the form.');
      }
    } else {
      setMessage('Failed to update user. Please try again.');
    }
  }
};
```

Figure 6.93: Fetch Data Function

Delete Function

```
const handleDelete = async (userId) => {
const confirmDelete = window.confirm("Are you sure you want to delete this user?");
if (!confirmDelete) return;
try {
  await axios.delete(`/api/users/${userId}`);
  // Remove the deleted user from the list
  setUsers((prevUsers) => prevUsers.filter((u) => u.id !== userId));
} catch (error) {
  console.error("Failed to delete user:", error);
  alert("An error occurred while deleting the user.");
}
};
```

Figure 6.94: Handle Delete Function

Based on above figure, the handleDelete function is used to remove a user from the system. It is trigger when administrator clicked the delete button. A confirmation dialogue will display through window.confirm() to ensure there is a clear user approval before any deletion occurs in order to prevent accidental deletion occurred. After the confirmation, the system initiates an HTTP DELETE request to the backend API endpoint /api/users/{userId} via Axios to target the specific user ID. The setUsers() then updated the local state to remove the deleted user from the active user list

Backend -User Controller

```
public function index()
{
    return response()->json(User::all());
}
```

Figure 6.95: Index Function

Based on the figure above, the index function is used to retrieve and display all user records from the system. It simply queries the users table using User::all() and returns the complete collection as a JSON response.

```php
    public function show($id)
{
    $user = User::find($id);

    if (!$user) {
        return response()->json(['message' => 'User not found'], 404);
    }

    return response()->json($user);
}
```

Figure 6.96: Show Function

Based on the figure above, the show function is used to retrieve a single user's details based on their ID. It uses User::find($id) to find the record, and if the user does not exist, it returns a 404 Not Found response. Otherwise, the user's data is returned in JSON format.

```php
public function update(Request $request, $id)
{
    $user = User::find($id);

    if (!$user) {
        return response()->json(['message' => 'User not found'], 404);
    }

    $request->validate([
        'username' => 'required|string|max:255|unique:users,username,' . $id,
        'role' => 'required|in:administrator,quarter master',
        'status' => 'required|in:active,inactive',
    ], [
        'username.unique' => 'This username is already taken. Please choose another one.',
    ]);

    $user->update($request->only('username', 'role', 'status'));

    return response()->json(['message' => 'User updated successfully', 'user' => $user]);
}
```

Figure 6.97: Update Function

Based on the figure above, the update function is used to modify user account information. It validates the request to ensure that the username is unique, the role is either "administrator" or "quarter master," and the status is set to "active" or "inactive." If validation passes, the user record is updated with the new details. The function then returns a success message with the updated user data to ensure that administrators can effectively manage user information.

```php
public function changePassword(Request $request, $id)
{
    $request->validate([
        'password' => 'required|string|min:6',
    ]);

    $user = User::find($id);

    if (!$user) {
        return response()->json(['message' => 'User not found'], 404);
    }

    $user->password = bcrypt($request->password);
    $user->save();

    return response()->json(['message' => 'Password updated successfully.']);
}
```

Figure 6.98: Change Password Function

Based on the figure above, the changePassword function is used to update a user's password when the user forgets the password. It validates that a new password is provided and meets the minimum length requirement of six characters. Once validation passes, the function finds the user by ID, hashes the new password using bcrypt, and saves it to the database. A confirmation message is then returned to ensure a secure and proper handling of password updates.

```php
public function destroy($id)
{
    $user = User::find($id);
    if (!$user) {
        return response()->json(['message' => 'User not found'], 404);
    }
    $user->delete();
    return response()->json(['message' => 'User deleted successfully.']);
}
```

Figure 6.99: Destroy Function

Based on the figure above, the destroy function is used to delete a user account. It finds the user by ID and it returns a 404-error message when user not found. If the user exits, the record is deleted from the database, and a success message is returned. This feature

ensures that administrators can remove inactive or unnecessary user accounts from the system while providing clear feedback.

6.3.1.9 Member Management

The MemberController and UserController are highly similar in structure and functionality. Both of them are design to manage account related operations such as creating, retrieving, updating, deleting and changing passwords. The main difference lies in the type of accounts being managed. For example, the UserController handles system users such as administrators and quarter masters,while the MemberController specifically manages members like students and teachers. Both controllers follow the same RESTful approach and validation rules to ensure consistency in implementation and maintaining a standardized workflow for handling different types of users within the system.

6.3.1.10 Member Booking

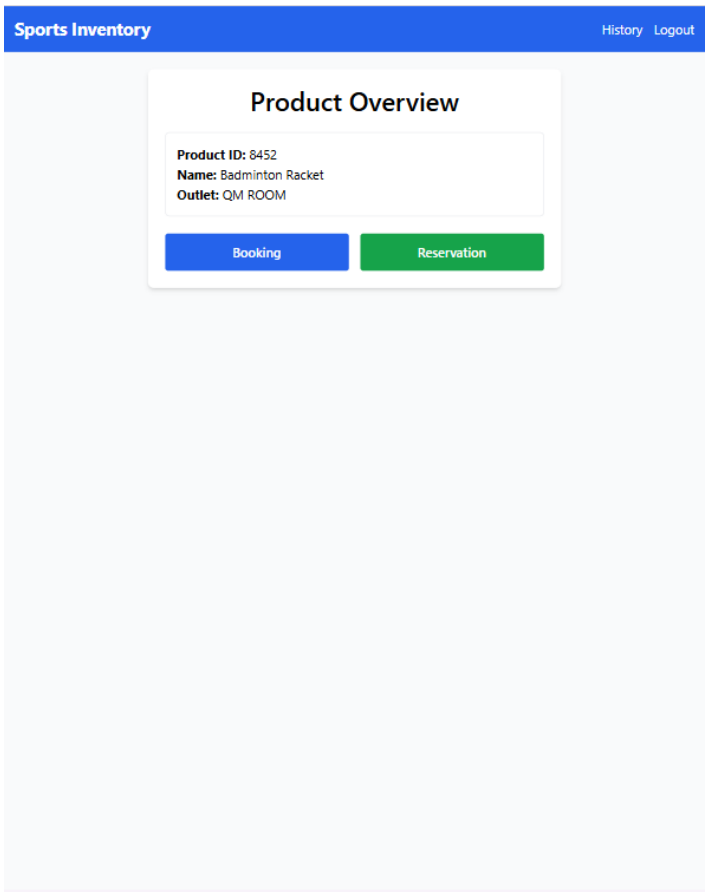


Figure 6.100: Home Page (Member)

```
useEffect(() => {
    const member = localStorage.getItem("member");

    if (!member) {
        navigate(`/?redirect=/home/${productId}`);
        return;
    }

    const fetchProduct = async () => {
        try {
            const res = await axios.get(`/api/inventories/${productId}`);
            setProduct(res.data);
        } catch (err) {
            console.error("Error fetching product", err);
        }
    };

    if (productId) fetchProduct();
}, [productId, navigate]);
```

Figure 6.101: Fetch Product Function

The homepage ensures that only authenticated members can access product details and perform actions such as booking or reservation. Firstly, the system checks the presence of stored member session in localStorage. If member not found, the user is automatically redirected to the login page with a redirect query string (/?redirect=/home/${productId}) to notify them that the authentication is required. If a valid member is found, the system retrieves the specific product details through GET request to /api/inventories/{productId} and displays the information such as product ID, name, and outlet.

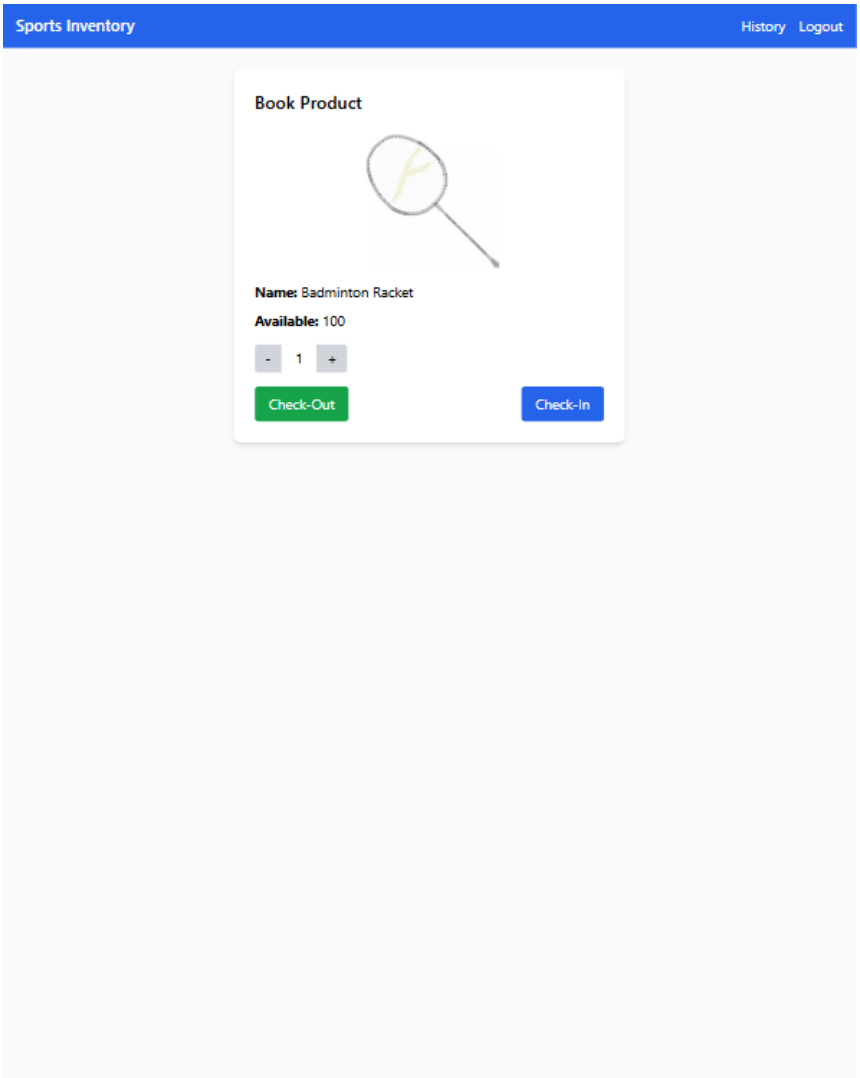


Figure 6.102: Booking Page Function

```
const member = JSON.parse(localStorage.getItem("member"));

//  Load product details
useEffect(() => {
  const fetchProduct = async () => {
    try {
      const res = await axios.get(`/api/inventories/${productId}`);
      setProduct(res.data);
    } catch (err) {
      console.error("Error fetching product", err);
    }
  };

  if (productId) fetchProduct();
}, [productId]);

//  Load booking from localStorage on page load
useEffect(() => {
  const saved = localStorage.getItem("currentBooking");
  if (saved) {
    const booking = JSON.parse(saved);
    if (booking.status === "checkout" && booking.product_id === productId) {
      setCurrentBooking(booking);
    } else {
      localStorage.removeItem("currentBooking");
    }
  }
}, [productId]);
```

Figure 6.103: Fetch Product and Booking Function

The member booking is used to perform check out and check in sport equipment in a structured and controlled manner. The system retrieves both the product details and logged in member information from localStorage. When member selects a product, the details are fetched and displayed as Figure shown. It allows member to choose the quantity without exceeding the available stock.

```
const handleCheckout = async () => {
  if (!member) {
    alert("You must be logged in as a member!");
    return;
  }

  try {
    const res = await axios.post("/api/bookings/checkout", {
      member_id: member.id,
      username: member.username,
      role: member.role,
      product_id: product.product_id,
      product_name: product.name,
      product_image: product.image,
      quantity,
    });

    const booking = res.data.booking;
    setCurrentBooking(booking);
    localStorage.setItem("currentBooking", JSON.stringify(booking)); // ✅ Save

    alert("Checked out successfully!");
  } catch (err) {
    if (err.response && err.response.status === 400) {
      alert(err.response.data.message);
      if (err.response.data.booking) {
        // Optionally restore active booking
        setCurrentBooking(err.response.data.booking);
        localStorage.setItem("currentBooking", JSON.stringify(err.response.data.booking));
      }
    } else {
      console.error("Check-out failed", err);
      alert("Checkout failed!");
    }
  }
```

Figure 6.104: Handle Checkout Function

The check out process is handled by handleCheckout function by calling /api/bookings/checkout API that records the checkout transaction and save the booking in state and in localStorage. It notified the user with a success message when the equipment is checkout. If the member already have active booking, the system will alert the user and restores the active booking data.

```
const handleCheckin = async () => {
  if (!member) {
    alert("You must be logged in as a member!");
    navigate("/");
    return;
  }

  if (!currentBooking) {
    alert("No active checkout found! Please checkout first.");
    return;
  }

  try {
    const res = await axios.put(`/api/bookings/${currentBooking.id}/checkin`);
    setCurrentBooking(null); // Clear booking state
    localStorage.removeItem("currentBooking"); // Clean up

    alert("Checked in successfully!");
  } catch (err) {
    console.error("Check-in failed", err);
    alert("Check-in failed!");
  }
};
```

Figure 6.105: Handle Check in Function

The handleCheckin function is used to updates the booking status by using PUT request to clear the localStorage and resets the booking state to ensure that the product is marked as checkin and receive alert messages that is about “check-in successfully”.

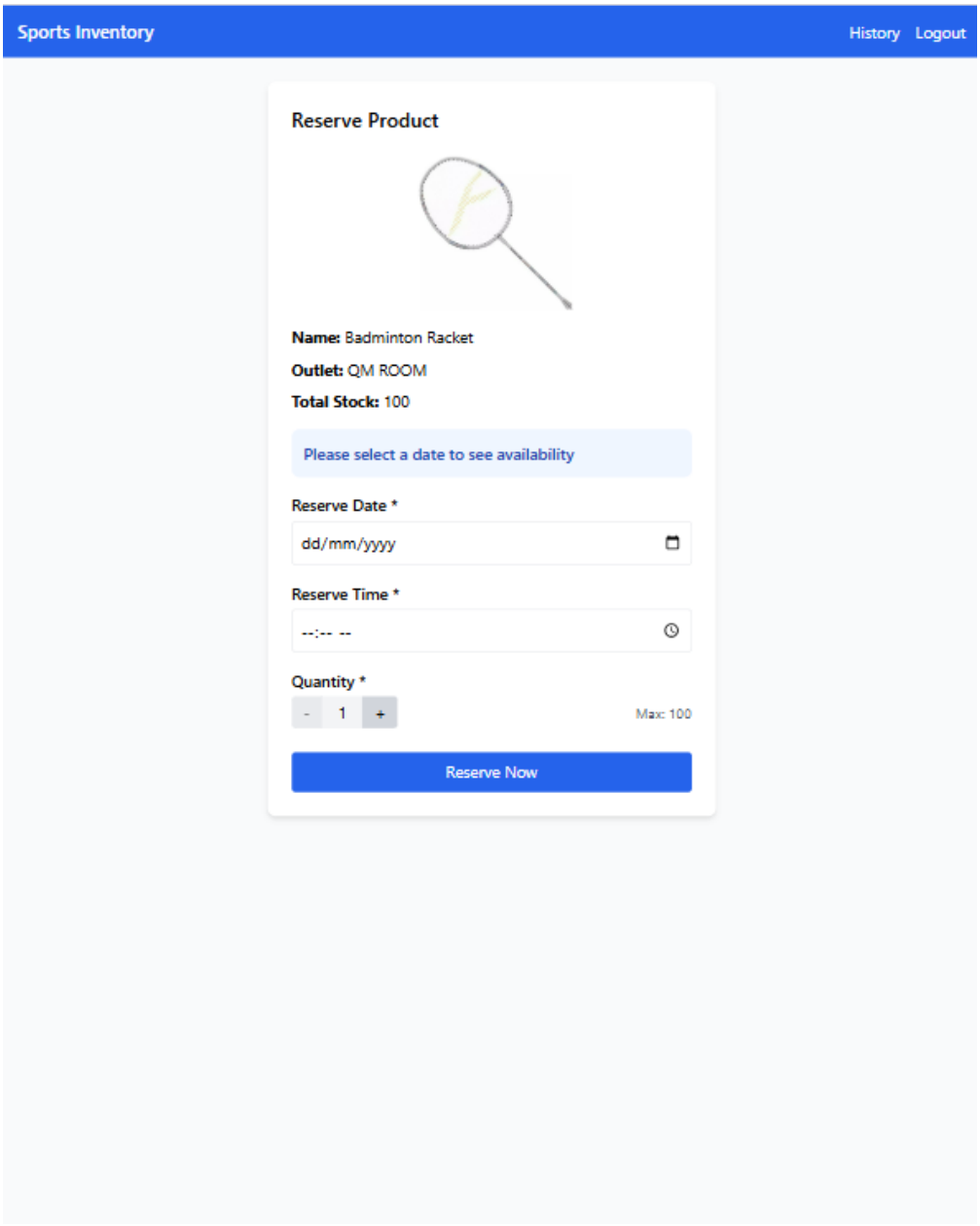


Figure 6.106: Reserve Booking Page

```
useEffect(() => {
  const fetchProduct = async () => {
    try {
      const res = await axios.get(`/api/inventories/${productId}`);
      setProduct(res.data);
      setAvailableQuantity(res.data.instock);
    } catch (err) {
      console.error("Error fetching product", err);
    }
  };
  if (productId) fetchProduct();
}, [productId]);
```

Figure 6.107: FetchBooking Function

```
const handleReserve = async () => {
  if (!member) {
    alert("You must be logged in as a member!");
    return;
  }
  if (!reserveDate || !reserveTime) {
    alert("Please select date and time!");
    return;
  }

  if (quantity > availableQuantity) {
    alert(`Only ${availableQuantity} items available for ${product.name} on ${reserveDate}`);
    return;
  }

  try {
    await axios.post("/api/reservations", {
      member_id: member.id,
      username: member.username,
      product_id: product.product_id,
      product_name: product.name,
      outlet: product.outlet,
      quantity,
      reserve_date: reserveDate,
      reserve_time: reserveTime,
      status: "pending",
    });
    alert("Reservation submitted successfully!");
    navigate("/history");
  } catch (err) {
    console.error("Reservation failed", err);
    alert("Reservation failed. Please try again.");
  }
};
```

Figure 6.108: Handle Reserve Function

```
useEffect(() => {
  const fetchAvailableQuantity = async () => {
    if (reserveDate && product) {
      setLoading(true);
      try {
        const response = await axios.get(`/api/reservations/available-quantities`, {
          params: {
            date: reserveDate,
            outlet: product.outlet,
            product_id: product.product_id // 只获取当前产品的可用数量
          }
        });

        const available = response.data[product.product_id] || product.instock;
        setAvailableQuantity(available);

        // 如果当前选择的数量超过可用数量，自动调整
        if (quantity > available) {
          setQuantity(Math.max(1, available));
        }
      } catch (err) {
        console.error("Error fetching available quantity", err);
        setAvailableQuantity(product.instock);
      } finally {
        setLoading(false);
      }
    } else if (product) {
      setAvailableQuantity(product.instock);
    }
  };

  fetchAvailableQuantity();
}, [reserveDate, product, quantity]);
```

Figure 6.109: Fetch Availability Quantity

```
const handleQuantityChange = (newQuanti any) {
  const maxQuantity = Math.min(product.instock, availableQuantity);
  setQuantity(Math.max(1, Math.min(newQuantity, maxQuantity)));
};

if (!product) {
  return (
    <div className="p-6 text-center">
      <p>Loading reservation data...</p>
    </div>
  );
}

const maxSelectable = Math.min(product.instock, availableQuantity);
```

Figure 6.110: Handle Quantity Change

The member reservation allows members to reserve the sports equipment. It retrieves the product information using the product ID from GET request. The instock is sets as initial available quantity. When the member selects a reservation date, another API is called to fetch the available quantity for that date and ensure that the members cannot reserve the items more than available quantity. The useEffect hook is used to listen the changes of reserveDate and product. The handleQuantityChange function is used to ensure that the selected quantity does not exceed the available quantity. This reservation forms include date and time pickers and quantity selector. The quantity selector allows the user to increase or decrease the number of items. After member fill in all the required field, member clicks the “Reserve now button” to call handleReserve function verify the input which will check the login status, selected date and time, and quantity limits before sending POST requests to API. When member reserve the equipment successfully, it will send an alert message and navigate to history page to see the reservation status.

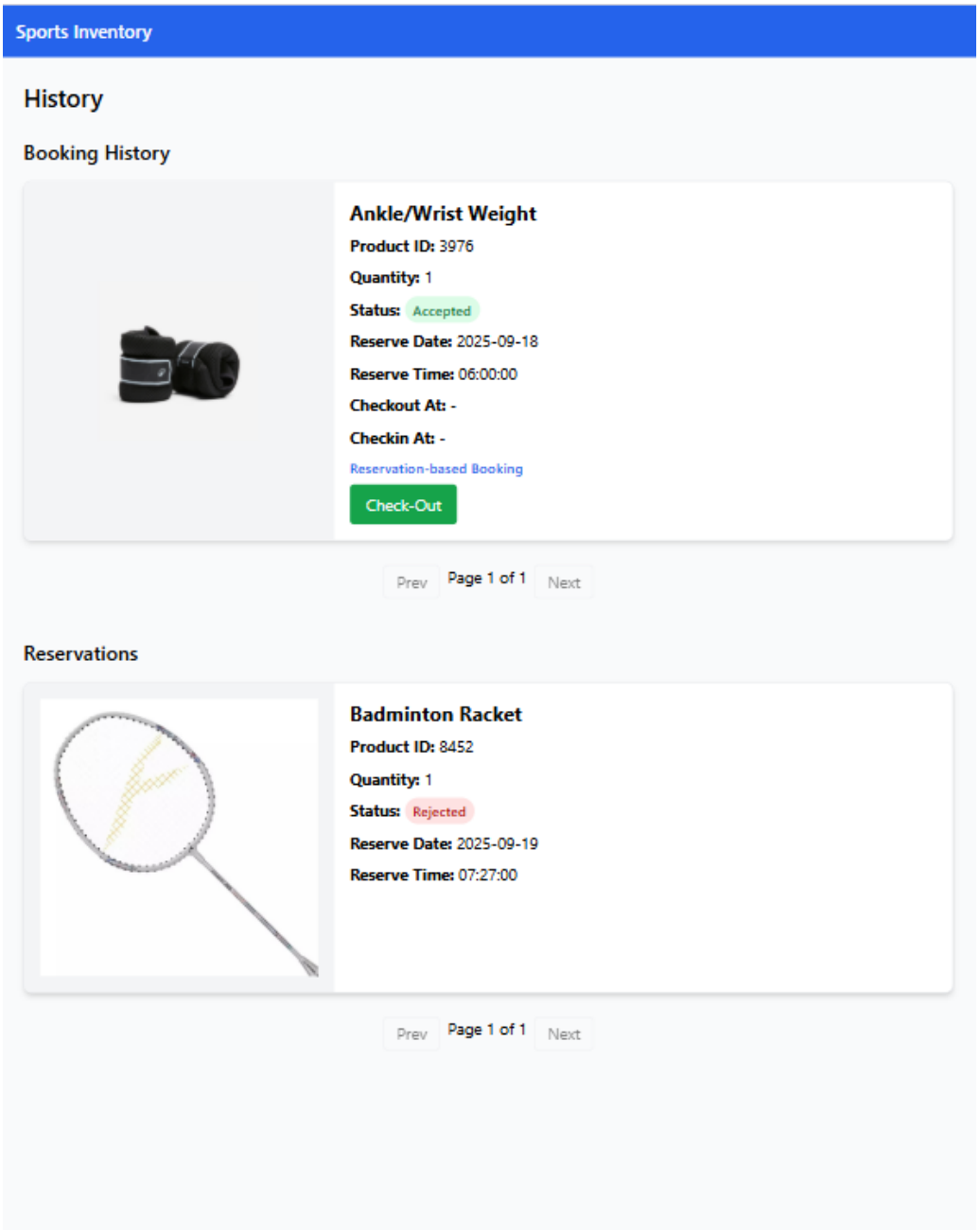


Figure 6.111: History Page

```
const fetchData = async () => {
    try {
        const bookingsRes = await axios.get(
            `/api/my-bookings?member_id=${member.id}`,
        );
        setBookings(bookingsRes.data);

        const reservationsRes = await axios.get(
            `/api/my-reservations?member_id=${member.id}`,
        );
        setReservations(reservationsRes.data);
    } catch (err) {
        console.error("Error fetching data", err);
    }
};
```

Figure 6.112: Fetch Data Function

Based on the figure above, the fetchData function is used to retrieve the both the bookings and reservations associated with a specific member. It sends GET request to 2 API endpoints which is /api/my-bookings and /api/my-reservations, passing the member's ID as a query parameter. Once the responses are received, the results are stored in the component state using setBookings and setReservations to ensure that the frontend displays the latest booking and reservation data for the logged-in member. The function is wrapped inside a useEffect hook, so it runs automatically when the component mounts or when the member.id changes to keep the displayed data always up to date.

```
const handleCheckout = async (booking) => {
    try {
        await axios.post("/api/bookings/checkout", {
            member_id: booking.member_id,
            username: booking.username,
            product_id: booking.product_id,
            product_name: booking.product_name,
            quantity: booking.quantity,
            reservation_id: booking.reservation_id,
        });

        alert("Checked out successfully!");

        // Refresh
        const bookingsRes = await axios.get(
            `/api/my-bookings?member_id=${member.id}`,
        );
        setBookings(bookingsRes.data);

        const reservationsRes = await axios.get(
            `/api/my-reservations?member_id=${member.id}`,
        );
        setReservations(reservationsRes.data);

    } catch (err) {
        alert("Checkout failed");
        console.error(err);
    }
};

const handleCheckin = async (bookingId) => {
    try {
        await axios.put(`/api/bookings/${bookingId}/checkin`);
        alert("Checked in successfully!");

        // Refresh
        const bookingsRes = await axios.get(
            `/api/my-bookings?member_id=${member.id}`,
        );
        setBookings(bookingsRes.data);
    } catch (err) {
        alert("Check-in failed");
        console.error(err);
    }
};
```

Figure 6.113: Handle Check Out Function

Based on the above figure, the handleCheckout and handleCheckIn function allows the member to check out or check in the equipments they reserved when the reservation status is accepted. It sends a POST request to the /api/bookings/checkout endpoint with all relevant booking details such as member_id, username, product_id, product_name,

quantity, and reservation_id. However, for the check in function it sends a PUT request to the /api/bookings/{bookingId}/checkin endpoint, where {bookingId} identifies the booking being checked in. It will send an alert to notify the member that the check out or check in is successful and updates the data. If the operation fails, the function handles the error by showing a failure alert and logging details to the console.

```
const isReservationDatePassed = (reserveDate) => {
    if (!reserveDate) return false;

    const today = new Date();
    const reservationDate = new Date(reserveDate);

    // Reset time parts to compare only dates
    today.setHours(0, 0, 0, 0);
    reservationDate.setHours(0, 0, 0, 0);

    return reservationDate < today;
};
```

Figure 6.114: Reservation Date Pass Function

The isReservationDatePassed is used to check whether a reservation date has already passed compared to current date. If the reserveDate is valid, it returns false if no date is provided. Then, it creates Date objects which is one for the current date and another for reservation date. The function then compares 2 dates and returns true if the reservation date is earlier than today to show date the reservationdate has passed. This results in the checkout button is disabled or a warning is display when the reservation is no longer valid.

CHAPTER 7

SYSTEM TESTING

7.1 Introduction

System testing is an important phase in the cycle of software development life as it ensures that the developed system functions correctly, meets the requirements, and provides a smooth user experience. For this project, testing was conducted at different levels such as unit testing, integration testing, and user acceptance testing (UAT). Unit and integration testing focused on technical verification, while UAT interaction with the actual users (administrators, quartermasters, students and teachers) to validate whether the system meets the intended objectives.

7.2 Unit Testing

According to Koomen and Pol, unit testing is performed by developers in a controlled environment to verify whether software meets to the behavioural definitions specified in its design specifications. Whitaker also mentioned that unit testing is a process of conducting isolated examinations of independent components or groups of components. He emphasised that testers should focus on defining the input space relevant to these units without considering the broader system environment (Runeson, 2006).

In this project, unit testing was implemented to the proper functioning of each independent module within the Sports Centre Inventory Management System. The primary goal was to identify and resolve errors at the earliest stage of testing by focusing on small, testable components such as reservation management, booking, notification handling, inventory control, and user authentication. Each test case was designed to verify both standard operations and system's exception handling to ensure that the modules were functioned as expected when valid or invalid inputs were received and when they encountered out of bounds scenarios. By utilizing this testing approach, this project was able to achieve higher code reliability, reduce integration risks, and improve the overall quality of the system before moving to more complicated testing such as integration and user acceptance testing.

Table 7.1: Unit testing of User Login

Test Module	Authentication – Login Page		**Test Title**	Verifying user login functionality	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-001	Login with valid User (Administrator/Quarter Master)	1. Navigate to login page 2. Enter correct username & password 3. Click Login	username: admin password: 123456	User logged in successfully and redirected to Dashboard	Pass
UT-002	Login with invalid password for User	1. Navigate to login page 2. Enter valid username but wrong password 3. Click Login	username: admin password: 1234567	System displays error message: "Invalid password. Please try again."	Pass
UT-003	Login with inactive User	1. Navigate to login page 2. Enter valid username & password for a User with status = inactive 3. Click Login	username: admin2 password: 123456	System displays error message: "Your account is inactive. Please contact the administrator."	Pass
UT-004	Login with non-existent username	1. Navigate to login page 2. Enter username not in users or members table 3. Click Login	username: member password: 123456	System displays error: "Account not found. Please ask the administrator to create an account for you."	Pass
UT-005	Empty input fields	1. Navigate to login page 2.Leave username & password blank 3. Click login button	username: password:	System shows validation error: "The username field is required."	Pass

Table 7.2: Unit testing of Member Login

Test Module	Authentication – Login Page		**Test Title**	Verifying Member login functionality	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-006	Login with valid Member (Teacher/Student)	1. Navigate to login page 2. Enter correct username & password 3. Click Login	username: member password: 123456	User logged in successfully and redirected to Homepage	Pass
UT-007	Login with invalid password for Member	1. Navigate to login page 2. Enter valid username but wrong password 3. Click Login	username: member password: 1234567	System displays error message: "Invalid password. Please try again."	Pass
UT-008	Login with inactive Member	1. Navigate to login page 2. Enter valid username & password for a Member with status = inactive 3. Click Login	username: member2 password: 123456	System displays error message: "Your account is inactive. Please contact the administrator."	Pass
UT-009	Login with non-existent username	1. Navigate to login page 2. Enter username not in members table 3. Click Login	username: admin password: 123456	System displays error: "Account not found. Please ask the administrator to create an account for you."	Pass
UT-010	Empty input fields	1. Navigate to login page 2.Leave username & password blank 3. Click login button	username: password:	System shows validation error: "The username field is required."	Pass

Table 7.3: Unit testing of Add User

Test Module	User Management – CRUD & Password Update		**Test Title**	Add user to user list	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-011	Add new user (valid input)	1. Navigate to user page 2. Click "+ User" Button to navigate to user add page 3. Fill in username, password, role, and status 4. Submit form	username: admin password: 123456	User logged in successfully and redirected to **Dashboard**	Pass
UT-012	Add new user with duplicate username	1. Navigate to user page 2. Click "+ User" Button to navigate to user add page 3. Fill in username, password, role, and status 4. Submit form	username: admin password: 1234567	API validation fails with message "This username is already taken. Please choose another one.", status code 422	Pass

Table 7.4: Unit testing of Edit User

Test Module	User Management – CRUD & Password Update		**Test Title**	Edit user account from user list	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-013	Edit user details (valid input)	1. Navigate to user page 2. Click edit icon Button for user that need to be edited 3. Modify role or status 4. Submit form	username: user01 role: quarter master status: inactive	User created successfully and API returns JSON with user object, redirected to user list	Pass
UT-014	Edit user with duplicate username	1. Navigate to user page 2. Click edit icon Button for user that need to be edited 3. Change username to an existing one 4. Submit form	username: admin1 (already exists)	API validation fails with message "This username is already taken. Please choose another one.", status code 422	Pass

Table 7.5: Unit testing of Delete User

Test Module	User Management – CRUD & Password Update		**Test Title**	Delete user account from user list	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-015	Delete user (valid user)	1. Navigate to user page 2. Click delete Button for user that need to be edited 3. Confirm delete	-	User deleted successfully, API returns success message, user removed from list and delete in database	Pass

Table 7.6: Unit testing of Change Password (User)

Test Module	User Management – CRUD & Password Update		**Test Title**	Change user password	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-016	Change user password (valid input)	1. Navigate to Change Password page 2. Enter new password and confirm password 3. Submit form	password: newpass123 confirm Password: newpass123	Password updated successfully, API returns success message	Pass
UT-017	Change password with mismatch	1. Navigate to Change Password page 2. Enter different password and confirm password 3. Submit form	password: newpass123 confirm Password: newpass12356	System displays “Passwords do not match”.	Pass
UT-018	Change password shorter than 6 characters	1. Navigate to Change Password page 2. Enter a new password with less than 6 characters 3. Submit form	password: 123 confirm Password: 123	System displays “The password must be at least 6 characters.”	Pass

Table 7.7: Unit testing of List User

Test Module	User Management – CRUD & Password Update		**Test Title**	Display a list of user	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-019	Display user list	1. Navigate to User List page	-	System displays a paginated list of users with username, role, and status	Pass

Table 7.8: Unit testing of Search User

Test Module	User Management – Search		**Test Title**	Search user	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-020	Search user in list	1. Enter keyword in search bar 2. System filters users	Keyword: user	User list displays matching usernames result only	Pass

Table 7.9: Unit testing of Add Member

Test Module	Member Management – CRUD & Password Update		**Test Title**	Add member to member list	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-021	Add new member (valid input)	1. Navigate to member page 2. Click "+ Member" Button to navigate to member add page 3. Fill in username, password, role, and status 4. Submit form	username: member password: 123456	Member logged in successfully and redirected to member list page	Pass
UT-022	Add new member with duplicate username	1. Navigate to user page 2. Click "+ Member" button to navigate to member add page 3. Fill in username, password, role, and status 4. Submit form	username: member password: 1234567	API validation fails with message "This username is already taken. Please choose another one.".	Pass

Table 7.10: Unit testing of Edit Member

Test Module	Member Management – CRUD & Password Update		**Test Title**	Edit member account from member list	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-023	Edit member details (valid input)	1. Navigate to member page 2. Click edit icon Button for member that need to be edited 3. Modify role or status 4. Submit form	username: member1 role: student status: inactive	Member created successfully and redirected to member list page.	Pass
UT-024	Edit member with duplicate username	1. Navigate to member page 2. Click edit icon Button for member that need to be edited 3. Change username to an existing one 4. Submit form	username: member1 (already exists)	API validation fails with message "This username is already taken. Please choose another one.".	Pass

Table 7.11: Unit testing of Delete Member

Test Module	Member Management – CRUD & Password Update		**Test Title**	Delete member account from member list	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-025	Delete member (valid member)	1. Navigate to member page 2. Click delete Button for member that need to be edited 3. Confirm delete	-	Member deleted successfully, API returns success message, member removed from list and delete in database	Pass

Table 7.12: Unit testing of Change Password (Member)

Test Module	Member Management – CRUD & Password Update		**Test Title**	Change member password	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-026	Change member password (valid input)	1. Navigate to Change Password page 2. Enter new password and confirm password 3. Submit form	password: 1234567 confirm Password: 1234567	Password updated successfully.	Pass
UT-027	Change password with mismatch	1. Navigate to Change Password page 2. Enter different password and confirm password 3. Submit form	password: 1234567 confirm Password: 123456	System displays “Passwords do not match”.	Pass
UT-028	Change password shorter than 6 characters	1. Navigate to Change Password page 2. Enter a new password with less than 6 characters 3. Submit form	password: 123 confirm Password: 123	System displays “The password must be at least 6 characters.”	Pass

Table 7.13: Unit testing of List Member

Test Module	Member Management – CRUD & Password Update		**Test Title**	Display a list of members	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-029	Display member list	1. Navigate to Member List page	-	System displays a paginated list of members with username, role, and status	Pass

Table 7.14: Unit testing of Search Member

Test Module	Member Management – CRUD & Password Update		**Test Title**	Search member in list	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-030	Search member in list	1. Enter keyword in search bar 2. System filters members	Keyword: member	Member list displays matching usernames result only	Pass

Table 7.15: Unit testing of List Inventory

Test Module	Inventory Management – Inventory Listing & Summary		**Test Title**	Verifying inventory listing, grouping, and summary	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-031	Display inventory list (valid data)	1. Navigate to Inventory page	-	System displays inventory list with product ID, name, stock, damage, missing, reserved, rented	Pass
UT-032	Display grouped inventory by outlet	1. Navigate to Inventory page	-	System displays the inventory based on outlets	Pass

Table 7.16: Unit testing of Add Product

Test Module	Product Management – CRUD		**Test Title**	Add product to product list	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-033	Add new product (valid input)	1. Navigate to Product Add page 2. Fill in product details 3. Upload image 4. Generate QR code 5. Click Submit	Name: “Basketball” Quantity: 10 Status: Active Outlet: QM ROOM Image: basketball.png	Product successfully added QR code generated Product listed in Product List Product also added and displayed in inventory List.	Pass
UT-034	Add new product without generating QR code.	1. Navigate to Product Add page 2. Fill in product details 3. Skip QR code generation 4. Click Submit	Name: “Volleyball” Quantity: 5 Status: Active Outlet: UP STORE	System displays error message: “Please generate a QR code first.”	Pass
UT-035	Add duplicate product name in same outlet	1. Navigate to Product Add page 2. Enter product name already existing in the outlet 3. Generate QR code 4. Submit	Name: “Basketball” Outlet: QM ROOM	System rejects submission with error: “The product name 'Basketball' already exists in the QM ROOM outlet.”	Pass

Table 7.17: Unit testing of Delete Product

Test Module	Product Management – CRUD		**Test Title**	Delete product from product list	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-036	Delete existing product	1. Go to Product List 2. Click Delete button on a product 3. Confirm delete.	-	Product deleted successfully, no longer visible in Product List, inventory record also deleted	Pass

Table 7.18: Unit testing of Edit Product

Test Module	Product Management – CRUD		**Test Title**	Edit product from product list	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-037	Edit existing product details (valid input)	1. Go to Product List 2. Click Edit button for a product 3. Modify quantity and upload new image 4. Click Update	Product: "Basketball" New Quantity: 20 New Image: basketball2.png	Product updated successfully, new quantity and image reflected in Product List	Pass
UT-038	Edit existing product with duplicate name in outlet	1. Go to Product List 2. Change name to another existing product in same outlet 3. Click Update	Name: "Football" (already exists in QM ROOM)	System shows error: "The product name 'Football' already exists in the QM ROOM outlet."	Pass

Table 7.19: Unit testing of List Product

Test Module	Product Management – CRUD		**Test Title**	Display a list of products	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-039	Display product list	1. Navigate to Product List page	-	System displays a paginated list of products with productID, name, quantity, QR code, status and action	Pass

Table 7.20: Unit testing of Search Product

Test Module	Product Management – CRUD		**Test Title**	Search product in list	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-040	Search product in list	1. Enter keyword in search bar 2. System filters product	Keyword: Ball	Product list displays matching product name result only	Pass

Table 7.21: Unit testing of Print QR Code

Test Module	Product Management – Print QR code		**Test Title**	Print QR codes	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-041	Print QR codes	1. Go to Product List 2. Select a product 3. Click Print QR button 4. Enter number of copies	Product: "Basketball" Copies: 2	New window opens with QR codes generated for the product name, ready for printing	Pass

Table 7.22: Unit testing of Stock Check

Test Module	Inventory Management – Stock Check		**Test Title**	Verifying Stock Check Creation, Listing, and Validation	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-042	Add new stock check (valid data)	1. Navigate to Stock Check Add page 2. Select date and outlet 3. Enter In Stock, Damage, and Missing for each product ensuring totals match original quantity 4. Submit form	Date: 2025-09-20 Outlet: QM ROOM Product: Basketball (Qty: 10) → In Stock: 8, Damage: 1, Missing: 1	Stock check saved successfully, API returns JSON with created stock check and inventory updated	Pass
UT-043	Validation – totals not matching product quantity	1. Navigate to Stock Check Add page 2. Select date and outlet 3. Enter incorrect totals (e.g., In Stock + Damage + Missing ≠ Product Qty) 4. Submit form	Name: “Volleyball” Quantity: 5 Status: Active Outlet: UP STORE	Alert displayed: “Quantity mismatch for product Volleyball. Expected 12, but got 11.”, record not saved	Pass
UT-044	Save stock check without selecting outlet or date	1. Navigate to Stock Check Add page 2.Either enter date or select outlet 3. Submit form	Date: 2025-09-20 Outlet: empty	Validation error occurs: “Outlet is required.” or “please select a date.”, stock check not saved	Pass
UT-045	List stock checks by date and outlet	1. Navigate to Stock Check List page 2. Leave date empty 3. Click Search	Date: 2025-09-20 Outlet: UP STORE	Table displays stock checks for selected date and outlet.	Pass
UT-046	List stock checks (all outlets)	1. Navigate to Stock Check List page 2. Leave date empty 3. Click Search	Date: 2025-09-20 Outlet: empty	Table displays all stock checks data grouped by outlet	Pass
UT-047	Validation – fetch stock checks without date	1. Navigate to Stock Check List page 2. Leave date empty 3. Click Search	-	Alert displayed: “Please select a date”.	Pass

Table 7.23: Unit testing of Member Booking

Test Module	Member Booking Management		**Test Title**	Member booking the product by walk in	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-048	Checkout product	1. Scan QR code for specific product 2. Navigate to home page 3.Click Booking Button 4. Select quantity 5. Click Check out button	-	New booking inserted into DB. Inventory - instock decreases, - rented increases booking status become checkout.	Pass
UT-049	Checkout same product twice	1. Scan QR code for specific product 2. Navigate to home page 3.Click Booking Button 4. Select quantity 5. Click Check out button twice	-	System displays message that "You already have this product checked out.".	Pass
UT-050	Checkin after checkout	1. Scan QR code for specific product 2. Navigate to home page 3.Click Booking Button 4. Select quantity 5. Click Check in button.	-	Booking status change to closed, Inventory - instock increases, - rented decreases	Pass

Table 7.24: Unit testing of Search Booking

Test Module	Booking Management – Search		**Test Title**	Search booking in list	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-051	Search booking in list	1. Enter keyword in search bar 2. System filters username	Keyword: Ling	Booking list displays matching username result only	Pass

Table 7.25: Unit testing of Add Reservation

Test Module	Reservation Management – CRUD		**Test Title**	Add reservation to reservation list	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-052	Add new reservation (user)	1. Navigate to Reservation Add page 2. Select Member from dropdown 3. Select Outlet 4. Select Date and Time 5. Select Product (ensure available quantity > 0) 6. Enter Quantity within available stock 7. Click Submit	Member: john Product: Basketball Outlet: QM ROOM Quantity: 2 Date: 2025-09-20 Time: 10:00	Reservation successfully created Reservation listed in Reservation List Reservation status in “pending”.	Pass
UT-053	Add new reservation based on member	1. Scan QR code for specific product 2. Navigate to home page 3.Click Reservation Button 4. Select quantity, date, time 5. Click Check in button.	Member: ali Product: Basketball Outlet: QM ROOM Quantity: 2 Date: 2025-09-20 Time: 10:00	Reservation successfully created Reservation listed in Reservation List Reservation status in “pending”.	Pass
UT-054	Add reservation exceeding available quantity	1. Navigate to Reservation Add page 2. Select Member, Outlet and Date 3. Select Product with available quantity 4. Enter Quantity 5. Click Submit	Member: adam Product: Volleyball Outlet: UP STORE Quantity: 5 Date: 2025-09-21 Time: 11:00 (available: 3)	Error message occurs. Fail to make Reservation.	Pass
UT-055	Add reservation without selecting member	1. Navigate to Reservation Add page 2. Leave Member field empty 3. Fill in other details 4. Click Submit	Name: “Basketball” Outlet: QM ROOM	System displays error message to fill in the required field. Fail to make Reservation.	Pass

Table 7.26: Unit testing of Edit Reservation Management

Test Module	Reservation Management – CRUD		**Test Title**	Edit reservation to reservation list	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-056	Edit reservation (status: pending)	1. Go to Reservation List 2. Click Edit button for selected reservation. 3. Modify the reservation details 4. Click Update	Member: john Product: Basketball Outlet: QM ROOM New Quantity: 3 Date: 2025-09-20 Time: 10:00	Reservation updated successfully Updated details reflected in Reservation List	Pass
UT-057	Edit reservation (locked – already accepted, rejected & used in booking)	1. Go to Reservation List 2. Click Edit button for selected reservation. 3. Modify the reservation details 4. Click Update	-	Reservation is not allowed to edit.	Pass

Table 7.27: Unit testing of List Reservation

Test Module	Reservation Management – CRUD		**Test Title**	List reservation to reservation list	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-058	Display reservation list	1. Navigate to Reservation List page	-	System displays a paginated list of reservation with ID, username, product, quantity, date,time,status and action	Pass

Table 7.28: Unit testing of Search Reservation

Test Module	Reservation Management – Search		**Test Title**	Search reservation in list	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-059	Search reservation in list	1. Enter keyword in search bar 2. System filters username	Keyword: Ling	Reservation list displays matching username result only	Pass

Table 7.29: Unit testing of Delete Reservation Management

Test Module	Reservation Management – Delete		**Test Title**	Delete reservation in list	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-060	Delete existing reservation	1. Go to Reservation List 2. Click Delete button on a product 3. Confirm delete.	-	Reservation deleted successfully, no longer visible in Reservation List.	Pass

Table 7.30: Unit testing of View Dashboard

Test Module	Dashboard Management		**Test Title**	View Dashboard	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-061	Display Dashboard	1. Navigate to Dashboard page	-	System displays dashboard output	Pass

Table 7.31: Unit testing of View Member Booking

Test Module	Member Booking		**Test Title**	View member booking page	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-062	Display member booking	1. Navigate to member-booking page	-	System displays product image, quantity available and input field that allows member to check-out or check-in.	Pass

Table 7.32: Unit testing of View Member Reservation

Test Module	Member Reservation		**Test Title**	View member reservation page	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-063	Display member reservation	1. Navigate to member-reservation page	-	System displays product image, quantity available and input field that allows member to make reservation.	Pass

Table 7.33: Unit testing of View Home Page

Test Module	Home Page		**Test Title**	View Home Page	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-064	Display home page	1. Navigate to home page	-	System displays product overview that allows member to make booking or reservation.	Pass

Table 7.34: Unit testing of View History

Test Module	History		**Test Title**	View History Page	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-065	Display history page	1. Navigate to history page	-	System displays booking history.	Pass

Table 7.35: Unit testing of Notification

Test Module	Reservation Management – Notification		**Test Title**	Notification	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
UT-066	Send notification on reservation acceptance	1. Navigate to Reservation List 2. Select a pending reservation 3. Edit status to Accept 4. System triggers notification service	Reservation status "pending" to "accepted"	Email sent to member: "Your reservation has been accepted"	Pass
UT-067	Send notification on reservation rejection	1. Navigate to Reservation List 2. Select a pending reservation 3. Edit status to Rejected 4. System triggers notification service	Reservation status "pending" to "rejected"	Email sent to member: "Your reservation has been rejected"	Pass

7.2.1 Conclusion of Unit Testing

The results of the unit testing showed that all the core modules of the system were functioning according to their specifications. Some of the functions such as adding and editing reservations, updating inventory, managing bookings, and sending notifications all worked as expected. The error-handling methods also successfully prevented invalid operations such as duplicate entries or exceeding stock limits. The consistent outcomes across multiple test cases provided confidence that the individual components were stable, reliable, and ready for integration testing. Overall, the successful completion of unit testing showed that the system's foundation architecture is stable and able to prevent the risk of defects spreading into later stages of development.

7.3 Integration Testing

Integration testing was conducted after unit testing to validate whether different modules in the Sports Centre Inventory Management System work together correctly. This level of testing emphasizes interactions between modules and their interfaces, rather than internal logic as in unit testing (Leung and White, 1990). The goal of this testing was to ensure seamless data flow between modules such as authentication, user/member management, booking, reservation, inventory, and notifications. Each integration test case checks multiples modules behaved from beginning to end. This is because there are approximately 40% of software errors can be traced back to component interaction issues identified during the integration process (Leung and White, 1990).

Table 7.36: Testing Integration 1

Test Module	Reservation Management + Notification		**Test Title**	Accept/Rejected reservation and send notification	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
IT-001	Reservation acceptance triggers notification	1. Admin navigates to Reservation List 2. Accept a pending reservation 3. Check member email inbox	Reservation status “pending” to “accepted”	Reservation status changes to accepted Notification email sent to the member	Pass
IT-002	Reservation rejection triggers notification	1. Admin navigates to Reservation List 2. Accept a pending reservation 3. Check member email inbox	Reservation status “pending” to “rejected”	Reservation status changes to rejected Notification email sent to the member	Pass

Table 7.37: Testing Integration 2

Test Module	Booking (Checkout) + Inventory		**Test Title**	Member checkout updates inventory	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
IT-003	Checkout product reduces inventory in stock quantity and increases rented quantity	1. Member scans QR code 2. Click Booking Button 3. Select Quantity 4. Click Checkout Button 5. Check inventory page	Product: Football Checkout Quantity: 2	Booking created with status = checkout Inventory: in-stock decreases by 1, rented increases by 1	Pass

Table 7.38: Testing Integration 3

Test Module	Booking (Checkin)+ Inventory		**Test Title**	Member checkin updates inventory	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
IT-004	Checkin product increases inventory in stock quantity and reduces rented quantity	1. Member scans QR code 2. Click Booking Button 3. Select Quantity 4. Click Checkin Button 5. User check inventory page	Product: Football Checkin Quantity: 2	Booking status changes to closed Inventory: in-stock increases by 1, rented decreases by 1	Pass

Table 7.39: Testing Integration 4

Test Module	Reservation + Inventory +Booking		**Test Title**	Add reservation and check inventory update	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
IT-005	Reservation make by member.	1. Member scans QR code 2. Click Reservation Button 3. Select quantity, date and time 4. Click Submit Button 5. User checks the reservation management and adjust the reservation status make by member to “Accept”. 6. Check inventory page	Member: john Product: Basketball Outlet: QM ROOM Quantity: 2 Date: 2025-09-20 Time: 10:00	Inventory reserved quantity increases by 2. In-stock remains unchanged, but it holds the quantity for that member until the reservation date is past.	Pass
IT-006	Member checkout the reservation product.	1. Member scans QR code 2. Click History Button. 3. The reservation in History from “pending” change to “accept”. 4. Click Checkout Button shows in reservation record. 5. User check inventory page	Member: john Product: Basketball Outlet: QM ROOM Quantity: 2 Date: 2025-09-20 Time: 10:00 Checkout at: 2025-09-20	Inventory reserved quantity decrease by 2, in-stock decreases and rented quantity increase.	Pass
IT-007	Member checkin the reservation product.	1. Member scans QR code 2. Click History Button. 3. The reservation in History change to “checkin” 4. Click Checkin Button shows in reservation record. 5. User check inventory page	Member: john Product: Basketball Outlet: QM ROOM Quantity: 2 Date: 2025-09-20 Time: 10:00 Checkout at: 2025-09-20 Checkout at: 2025-09-20	Inventory in-stock increase and rented quantity decrease.	Pass

Table 7.40: Testing Integration 5

Test Module	User Management + Authentication		**Test Title**	Admin creates user account and login with new credentials	
TC ID	**Test Case Name**	**Test Steps**	**Test Data**	**Expected Result**	**Status**
IT-008	New user Login with the role “administrator” or “quarter master”	1. Administrator adds new user in User Management. 3. Create a new user with administrator role or quarter master role 4. Log out 5. Log in with new credentials	Username: newadmin Password: password123 **or** Username: newQM Password: 123456	User added successfully New user (administrator / quarter master) can login and access role-based dashboard	Pass
IT-009	New member Login with the role “student” or “teacher”	1. Administrator adds new user in Member Management. 3. Create a new member with student role or teacher role. 4. Log out 5. Log in with new credentials	Username: newmember Password: pass123	Memver added successfully New member (student / teacher) can access home page	Pass

7.3.1 Conclusion of integration testing

The result of the integration testing showed that there was a seamless collaboration of all modules within the Sports Center Inventory Management System. This is because the functions such as notifications triggered by appointment acceptance or rejection, real-time inventory updates during booking operations, and check-in/check-out processes accurately reflecting stock levels were all successfully verified which resulted in a comprehensive and fully functional workflow. Besides, the integration of user management, member management and authentication ensured that the role-based access control work correctly to handle administrators, quartermasters, teachers, and student's role. All designed integration test cases produced expected the outcomes as expected which demonstrated a reliable and consistent interactions between modules. In conclusion, the successful completion of integration testing shows that system components not only function independently but also collaborate efficiently to support end-to-end business processes.

7.4 User Acceptance Testing

User Acceptance Testing (UAT) was conducted to ensure that the system meets the requirements and expectations of targeted users. The purpose of UAT is to validate that the system's workflow is align with actual usage scenarios within the sport centre environment and ensured that all functionalities are intuitive, reliable, and efficient. Test cases involved simulating typical user scenarios, including system login, reservation management, booking, product checkout/in procedures and the others. The UAT is used to invite the actual and targeted user to involved in the testing as it provided invaluable feedback on system usability, functionality and overall satisfaction. Therefore, User Acceptance Testing is important as serves as the final validation stage that ensure the system is fully ready for operational use.

User Acceptance Testing				
UAT Test ID: UAT-User Type - 1		Tester Type: Administrator		
Test Date: 21/8/2025		Tester name: EugeneTiang		
Test Case ID	Module Under Tested	Test Scenario	Results	Feedback/Comments from participants
UAT-001	Authentication	Login	PASS	—
UAT-002	Inventory Management	Manage Inventory (create, delete, update,edit)	PASS	
UAT-003	User Management	Manage User (create, delete, update,edit)	PASS	
UAT-004	Member Management	Manage member (create, delete, update,edit)	PASS	
UAT-005	Product Management	Manage products (create, delete, update,edit)	PASS	
UAT-006	Dashboard Management	View Dashboard	PASS	
UAT-007	Reservation Management	Manage Reservation (create, delete, update,edit)	PASS	
UAT-008	Stock Check Management	Stock Check (missing, damaged, instock)	PASS	
UAT-009	Booking Management	Manage Bookings (delete, update,edit)	PASS	
UAT-010	Product Management	Generate QR Code	PASS	
UAT-011	QR Code Integration	Scan QR Code	PASS	
UAT-012	Product Management	Print QR Code	PASS	
UAT-013	Authentication	Logout	PASS	

User Acceptance Testing				
UAT Test ID: UAT-User Type - 2			Tester Type: Administrator	
Test Date: 21/8/2025			Tester name: Lim Zi Quan	
Test Case ID	Module Under Tested	Test Scenario	Results	Feedback/Comments from participants
UAT-001	Authentication	Login	PASS	—
UAT-002	Inventory Management	Manage Inventory (create, delete, update,edit)	PASS	
UAT-003	User Management	Manage User (create, delete, update,edit)	PASS	
UAT-004	Member Management	Manage member (create, delete, update,edit)	PASS	
UAT-005	Product Management	Manage products (create, delete, update,edit)	PASS	
UAT-006	Dashboard Management	View Dashboard	PASS	
UAT-007	Reservation Management	Manage Reservation (create, delete, update,edit)	PASS	
UAT-008	Stock Check Management	Stock Check (missing, damaged, instock)	PASS	
UAT-009	Booking Management	Manage Bookings (delete, update,edit)	PASS	
UAT-010	Product Management	Generate QR Code	PASS	
UAT-011	QR Code Integration	Scan QR Code	PASS	
UAT-012	Product Management	Print QR Code	PASS	
UAT-013	Authentication	Logout	PASS	

User Acceptance Testing				
UAT Test ID: UAT-User Type - 3		Tester Type: Administrator		
Test Date: 21/8/2025		Tester name: Soh Yi Jye		
Test Case ID	Module Under Tested	Test Scenario	Results	Feedback/Comments from participants
UAT-001	Authentication	Login	PASS	—
UAT-002	Inventory Management	Manage Inventory (create, delete, update,edit)	PASS	
UAT-003	User Management	Manage User (create, delete, update,edit)	PASS	
UAT-004	Member Management	Manage member (create, delete, update,edit)	PASS	
UAT-005	Product Management	Manage products (create, delete, update,edit)	PASS	
UAT-006	Dashboard Management	View Dashboard	PASS	
UAT-007	Reservation Management	Manage Reservation (create, delete, update,edit)	PASS	
UAT-008	Stock Check Management	Stock Check (missing, damaged, instock)	PASS	
UAT-009	Booking Management	Manage Bookings	PASS	
UAT-010	Product Management	Generate QR Code	PASS	
UAT-011	QR Code Integration	Scan QR Code	PASS	
UAT-012	Product Management	Print QR Code	PASS	
UAT-013	Authentication	Logout	PASS	

User Acceptance Testing				
UAT Test ID: UAT-User Type - 4			Tester Type: Quarter Master	
Test Date: 21/8/2025			Tester name: Tan Ke Ting	
Test Case ID	Module Under Tested	Test Scenario	Results	Feedback/Comments from participants
UAT-001	Authentication	Login	PASS	
UAT-002	Inventory Management	Update Inventory	PASS	
UAT-003	Reservation Management	Manage Reservation	PASS	
UAT-004	Stock Check Management	Stock Check (missing, damaged, instock)	PASS	
UAT-005	Booking Management	Manage Bookings	PASS	
UAT-006	QR Code Integration	Scan QR Code	PASS	
UAT-007	Product Management	Print QR Code	PASS	
UAT-008	Authentication	Logout	PASS	

User Acceptance Testing				
UAT Test ID: UAT-User Type - 5			Tester Type: Quarter Master	
Test Date: 21/8/2025			Tester name: Chow Zong Xian	
Test Case ID	Module Under Tested	Test Scenario	Results	Feedback/Comments from participants
UAT-001	Authentication	Login	PASS	
UAT-002	Inventory Management	Update Inventory	PASS	
UAT-003	Reservation Management	Manage Reservation	PASS	
UAT-004	Stock Check Management	Stock Check (missing, damaged, instock)	PASS	
UAT-005	Booking Management	Manage Bookings	PASS	
UAT-006	QR Code Integration	Scan QR Code	PASS	
UAT-007	Product Management	Print QR Code	PASS	
UAT-008	Authentication	Logout	PASS	

User Acceptance Testing				
UAT Test ID: UAT-User Type - 6			Tester Type: Quarter Master	
Test Date: 21/8/2025			Tester name: Vincy Lim	
Test Case ID	Module Under Tested	Test Scenario	Results	Feedback/Comments from participants
UAT-001	Authentication	Login	PASS	
UAT-002	Inventory Management	Update Inventory	PASS	
UAT-003	Reservation Management	Manage Reservation	PASS	
UAT-004	Stock Check Management	Stock Check (missing, damaged, instock)	PASS	
UAT-005	Booking Management	Manage Bookings	PASS	
UAT-006	QR Code Integration	Scan QR Code	PASS	
UAT-007	Product Management	Print QR Code	PASS	
UAT-008	Authentication	Logout	PASS	

User Acceptance Testing				
UAT Test ID: UAT-User Type - 7			Tester Type: Quarter Master	
Test Date: 21/8/2025			Tester name: Vinky Tan Zi Yi	
Test Case ID	Module Under Tested	Test Scenario	Results	Feedback/Comments from participants
UAT-001	Authentication	Login	PASS	
UAT-002	Inventory Management	Update Inventory	PASS	
UAT-003	Reservation Management	Manage Reservation	PASS	
UAT-004	Stock Check Management	Stock Check (missing, damaged, instock)	PASS	
UAT-005	Booking Management	Manage Bookings	PASS	
UAT-006	QR Code Integration	Scan QR Code	PASS	
UAT-007	Product Management	Print QR Code	PASS	
UAT-008	Authentication	Logout	PASS	

User Acceptance Testing				
UAT Test ID: UAT-User Type – 8		Tester Type: Student		
Test Date: 21/8/2025		Tester name: Chong Min Yew		
Test Case ID	Module Under Tested	Test Scenario	Results	Feedback/Comments from participants
UAT-001	Authentication	Login	PASS	
UAT-002	Booking Management	Book Equipment (Check In/Check Out)	PASS	
UAT-003	Reservation Management	Make Reservation	PASS	
UAT-004	Reservation Management	Check Reservation Status	PASS	
UAT-005	Booking Management	View History	PASS	
UAT-006	QR Code Integration	Scan QR Code	PASS	
UAT-007	Notification	Receive Notifications through email.	PASS	
UAT-008	Authentication	Logout	PASS	

User Acceptance Testing				
UAT Test ID: UAT-User Type – 9		Tester Type: Student		
Test Date: 21/8/2025		Tester name: Kee Sherru		
Test Case ID	Module Under Tested	Test Scenario	Results	Feedback/Comments from participants
UAT-001	Authentication	Login	PASS	
UAT-002	Booking Management	Book Equipment (Check In/Check Out)	PASS	
UAT-003	Reservation Management	Make Reservation	PASS	
UAT-004	Reservation Management	Check Reservation Status	PASS	
UAT-005	Booking Management	View History	PASS	
UAT-006	QR Code Integration	Scan QR Code	PASS	
UAT-007	Notification	Receive Notifications through email.	PASS	
UAT-008	Authentication	Logout	PASS	

User Acceptance Testing				
UAT Test ID: UAT-User Type – 10		Tester Type: Student		
Test Date: 21/8/2025		Tester name: Ng Jing Hong		
Test Case ID	Module Under Tested	Test Scenario	Results	Feedback/Comments from participants
UAT-001	Authentication	Login	PASS	
UAT-002	Booking Management	Book Equipment (Check In/Check Out)	PASS	
UAT-003	Reservation Management	Make Reservation	PASS	
UAT-004	Reservation Management	Check Reservation Status	PASS	
UAT-005	Booking Management	View History	PASS	
UAT-006	QR Code Integration	Scan QR Code	PASS	
UAT-007	Notification	Receive Notifications through email.	PASS	
UAT-008	Authentication	Logout	PASS	

User Acceptance Testing				
UAT Test ID: UAT-User Type – 11		Tester Type: Student		
Test Date: 21/8/2025		Tester name: Lau Jing Xuan		
Test Case ID	Module Under Tested	Test Scenario	Results	Feedback/Comments from participants
UAT-001	Authentication	Login	PASS	
UAT-002	Booking Management	Book Equipment (Check In/Check Out)	PASS	
UAT-003	Reservation Management	Make Reservation	PASS	
UAT-004	Reservation Management	Check Reservation Status	PASS	
UAT-005	Booking Management	View History	PASS	
UAT-006	QR Code Integration	Scan QR Code	PASS	
UAT-007	Notification	Receive Notifications through email.	PASS	
UAT-008	Authentication	Logout	PASS	

User Acceptance Testing				
UAT Test ID: UAT-User Type – 12			Tester Type: Teacher	
Test Date: 21/8/2025			Tester name: Tan Heng Jie	
Test Case ID	Module Under Tested	Test Scenario	Results	Feedback/Comments from participants
UAT-001	Authentication	Login	PASS	
UAT-002	Booking Management	Book Equipment (Check In/Check Out)	PASS	
UAT-003	Reservation Management	Make Reservation	PASS	
UAT-004	Reservation Management	Check Reservation Status	PASS	
UAT-005	Booking Management	View History	PASS	
UAT-006	QR Code Integration	Scan QR Code	PASS	
UAT-007	Notification	Receive Notification through email	PASS	
UAT-008	Authentication	Logout	PASS	

User Acceptance Testing				
UAT Test ID: UAT-User Type – 13			Tester Type: Teacher	
Test Date: 21/8/2025			Tester name: Kriss Tee	
Test Case ID	Module Under Tested	Test Scenario	Results	Feedback/Comments from participants
UAT-001	Authentication	Login	PASS	
UAT-002	Booking Management	Book Equipment (Check In/Check Out)	PASS	
UAT-003	Reservation Management	Make Reservation	PASS	
UAT-004	Reservation Management	Check Reservation Status	PASS	
UAT-005	Booking Management	View History	PASS	
UAT-006	QR Code Integration	Scan QR Code	PASS	
UAT-007	Notification	Receive Notification through email	PASS	
UAT-008	Authentication	Logout	PASS	

User Acceptance Testing				
UAT Test ID: UAT-User Type – 14			Tester Type: Teacher	
Test Date: 21/8/2025			Tester name: Tee Kuik Qun	
Test Case ID	Module Under Tested	Test Scenario	Results	Feedback/Comments from participants
UAT-001	Authentication	Login	PASS	
UAT-002	Booking Management	Book Equipment (Check In/Check Out)	PASS	
UAT-003	Reservation Management	Make Reservation	PASS	
UAT-004	Reservation Management	Check Reservation Status	PASS	
UAT-005	Booking Management	View History	PASS	
UAT-006	QR Code Integration	Scan QR Code	PASS	
UAT-007	Notification	Receive Notification through email	PASS	
UAT-008	Authentication	Logout	PASS	

User Acceptance Testing				
UAT Test ID: UAT-User Type – 15			Tester Type: Teacher	
Test Date: 21/8/2025			Tester name: Yee Jia Xuan	
Test Case ID	Module Under Tested	Test Scenario	Results	Feedback/Comments from participants
UAT-001	Authentication	Login	PASS	
UAT-002	Booking Management	Book Equipment (Check In/Check Out)	PASS	
UAT-003	Reservation Management	Make Reservation	PASS	
UAT-004	Reservation Management	Check Reservation Status	PASS	
UAT-005	Booking Management	View History	PASS	
UAT-006	QR Code Integration	Scan QR Code	PASS	
UAT-007	Notification	Receive Notification through email	PASS	
UAT-008	Authentication	Logout	PASS	

7.4.1 Findings

The User Acceptance Testing (UAT) was conducted to evaluate system's overall functionality, usability and deployment readiness. There are 15 testers representing different user roles participated in the testing. Each tester performed role-specific tasks to ensure the system met operational requirements within the school sports centre environment.

All test cases demonstrated a 100% pass rate. This result shows that the inventory management system had a reliable operation across various user scenarios. Every core system module has successfully met the respective acceptance criteria. For example, administrator can manage all the tasks efficiently, while quarter master can do the stock check, printed QR codes, and managed booking and reservations and others without errors. Teachers and students seamlessly logged into the system to book equipment by scanning the QR codes for check out and check in procedures, view booking histories, and receive email notifications. There are no major issues, system vulnerabilities or usability barriers were reported by any participants. Test results also confirmed the proper functioning of role-based access control to ensure users could only access functional modules relevant to their assigned roles.

7.4.2 Achievements

The successful completion of user acceptance testing indicated a significant milestone in the system development lifecycle. A 100% pass rate confirms that the inventory management dashboard has achieved its intended objectives which is provide a stable, user-friendly and efficient solution for sports equipment and facility management. The system meets all acceptance criteria, and this demonstrate that it compliances with user expectations and organisational requirements.

User acceptance test results validate:

- The system's functionality is complete and all essential operations functioning without fault.
- The user interface design is intuitive which enabling effortless operation by all users.
- QR code integration effectively supports real-time tracking and equipment management.

- The notification system ensures users receive timely updates on booking and reservation statuses.
- Role-based access control successfully implements security and operational boundary management.

In conclusion, the user acceptance test results show that the system is ready for deployment. The successful testing outcome highlights the project's achievement in delivering a robust, reliable, and scalable inventory management solution. This solution will enhance efficiency, reduce manual workload, and provide data-driven decision support for the school sports centre.

CHAPTER 8

CONCLUSION AND FUTURE WORK

8.1 Conclusion

This project successfully developed a web-based Inventory Management Dashboard and mobile app for scanning QR code to streamline the management of sports equipment and facilities at a secondary school's sports centre. The system was designed to address inefficiencies of manual record-keeping and lack of visibility in inventory movement. By integrating React for the web application and mobile application, Laravel as the backend framework, the system provides a modern, scalable, and user-friendly platform. The implementation of QR code technology further enhances reliability by streamlining equipment check-in and check-out processes, reducing errors, and improving accountability. In conclusion, the project demonstrates how technology can significantly improve the management and monitoring of sports resources in a school environment.

8.2 Achieved Objectives

The project has achieved the following objectives:

1. Successfully developed a web application with a dashboard for tracking the movement and inventory of sports equipment and facilities.
2. Implemented equipment tracking and maintenance function with QR code generation which allows user to manage equipment efficiently.
3. Designed and developed a mobile application that enables QR code scanning for real-time updates when the equipment is checked out or in of the sports center.
4. Conducted User Acceptance Testing (UAT) with selected users to demonstrate the system's effectiveness and usability in improving sports centre operations.

8.3 Limitations and Future Work

Although the system achieved its targeted objectives, some limitations remain that can be addressed in future work:

Table 8.1: Limitation and Future Work

Limitation	Future work
Notifications are limited to email only.	Integrate SMS and mobile push notifications for better accessibility.
System is primarily designed for local school network use.	Deploy on a cloud-based infrastructure to allow remote access anytime, anywhere.
Filtering and search functions are limited (e.g., by username only)	Improve filter options to allow advanced search (by product type, member, booking status, time range, etc.).
The system is web-based with no dedicated mobile application.	Build a mobile app for Android/iOS to improve convenience for teachers and students.
Dashboard Reporting features are basic and providing only essential information without advanced analytics or data visualization.	Integrate AI-driven analytics to provide predictive insights, usage patterns, and recommendations
Manual QR code scanning is required to track equipment.	Explore integration of IoT-based sensors for automated equipment tracking without manual scanning.
Outlets are fixed and cannot be dynamically added.	Enhance the system to allow administrators to create and manage new outlets.

REFERENCES

Okirie, A.J., Barnabas, M. and Adagbon, J.E, 2024 'Maintenance Management Optimization: Evaluating Manual and Automated Methods of Tracking Uptime Hours for Offshore Equipment', *American Journal of IR 4 0 and Beyond*, 3(1), pp. 15–27. Available at: https://doi.org/10.54536/ajirb.v3i1.3516.

Al-Saqqa, S., Sawalha, S. and AbdelNabi, H., 2020. Agile software development: Methodologies and trends. International Journal of Interactive Mobile Technologies, 14(11).

Aman, M.S., Ponnusamy, V., Elumalai, G., Mohamed, M.N.A., Kamalden, T.F.T. and Yahya, S., 2020. Trends and usage of sports facilities among Malaysians. *International Journal of Physiotherapy*, 7(6), pp.252-255. Available at: https://doi.org/10.15621/ijphy/2020/v7i6/840

Apke, L., 2016. Agile values: Working software over comprehensive documentation. [online] Agile Doctor. Available at: https://www.agile-doctor.com/2016/08/16/agile-values-working-software-documentation/

Asset Panda, 2022. How Asset Tracking Helps Educational Facilities Save Money. [online] Available at: https://www.assetpanda.com/resource-center/white-papers/asset-tracking-aids-educational-institutions/

Bassil, Y., 2012. A simulation model for the waterfall software development life cycle. arXiv preprint arXiv:1205.6904.

Bell, T.E. and Thayer, T.A., 1976. Software requirements: Are they really a problem? In: Proceedings of the 2nd International Conference on Software Engineering, pp.61-68.

Camburn, B.A., Viswanathan, V.K., Linsey, J.S., Jensen, D.D., Crawford, R.H., Otto, K. and Wood, K.L., 2017. Design prototyping methods: state of the art in strategies, techniques, and guidelines. Design Science, 3, e13. [online] Cambridge University Press. Available at: https://doi.org/10.1017/dsj.2017.10

Chemuturi, M., 2013. Requirements Engineering and Management for Software Development Projects. Springer Science & Business Media.

Daka, E. and Fraser, G., 2014. A survey on unit testing practices and problems. In: 2014 IEEE 25th International Symposium on Software Reliability Engineering, pp.201-211.

Davis, F.D. and Venkatesh, V., 2004. Toward preprototype user acceptance testing of new information systems: implications for software project management. IEEE Transactions on Engineering Management, 51(1), pp.31-46.

Discipline Infotech (2023) *Laravel Security Features*, *Disciplineinfotech.com*. Discipline Infotech. Available at: https://www.disciplineinfotech.com/blog/laravel-security-features

Dudley, M., 2023. 6 Surprising Stats About Tool Tracking | Link Labs | Blog. [online] Link-labs.com. Available at: https://www.link-labs.com/blog/6-surprising-stats-about-tool-tracking/

Erickson, J., 2024. What is MySQL? [online] Oracle. Available at: https://www.oracle.com/mysql/what-is-mysql/

Epifany Bojanowska, 2018. Naturaily: Web and Mobile Development Company from Poland. [online] Naturaily.com. Available at: https://naturaily.com/blog/pros-cons-vue-js/

Firebase, 2025. Firebase Realtime Database. [online] Available at: https://firebase.google.com/docs/database#store_other_types_of_data

Gackenheimer, C., 2015. Introduction to React. Apress.

GeeksforGeeks, 2019a. Unit Testing | Software Testing. [online] Available at: https://www.geeksforgeeks.org/unit-testing-software-testing/

GeeksforGeeks, 2019b. System Testing - GeeksforGeeks. [online] Available at: https://www.geeksforgeeks.org/system-testing/

GeeksforGeeks (2023) *Laravel Features*, *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/php/laravel-features/.

Gurung, G., et al., 2020. Software Development Life Cycle Models – A Comparative Study. International Journal of Scientific Research in Computer Science and Engineering, 6(4), pp.30–37.

Hawkes, L. (2025) *Facility Maintenance Plan*, *Click Maint CMMS*. Click Maint Inc. Available at: https://www.clickmaint.com/blog/facility-maintenance-plan

Horváthová, N. and Voštinár, M., 2018. Mistake as a source of feedback. In: International Scientific Conference on Distance Learning in Applied Informatics (DIVAI 2018). Štúrovo, Slovakia, 2–4 May. Wolters Kluwer, pp.40–45.

Jastrow, F., Greve, S., Thumel, M. *et al.* Digital technology in physical education: a systematic review of research from 2009 to 2020. *Ger J Exerc Sport Res* **52**, 504–528 (2022). https://doi.org/10.1007/s12662-022-00848-5

Jindal, S., Gulati, P. and Rohilla, P., 2015. Various Software Development Life Cycle Models. IJRDO - Journal of Computer Science Engineering, 1(4), pp.162–167.

Johnson, P., 2023a. 10 Reasons Why Vue.js Is Best for App Development [+ Benefits]. [online] Digital Marketing Agency in USA. Available at: https://foreignerds.com/why-vue-js/

Johnson, P., 2023b. The Good and the Bad of Vue.js Framework Programming. [online] Digital Marketing Agency in USA. Available at: https://foreignerds.com/the-good-and-the-bad-of-vue-js-framework-programming/

Jorgensen, P.C. and Erickson, C., 1994. Object-oriented integration testing. Communications of the ACM, 37(9), pp.30–38.

Ko, H.S., Azambuja, M. and Lee, H.F., 2016. Cloud-based materials tracking system prototype integrated with radio frequency identification tagging technology. *Automation in Construction*, *63*, pp.144-154.

Krahenbuhl, J.H., 2015. Learning Axure RP Interactive Prototypes. Packt Publishing Ltd.

Kute, S.S. and Thorat, S.D., 2014. A Review on Various Software Development Life Cycle (SDLC) Models. International Journal of Research in Computer and Communication Technology, 3(7), pp.776–781.

Leed Software Development, 2024. Advantages and Challenges of React Native App Development. [online] Medium. Available at: https://leeddev.medium.com/advantages-and-challenges-of-react-native-app-development-56c18d8fc834

Leung, H.K.N. and White, L. (1990) *A study of integration testing and software regression at the integration level*, *IEEE Xplore*. Available at: https://doi.org/10.1109/ICSM.1990.131377.

Menariya, N., 2022. Laravel: advantages and disadvantages. [online] Medium. Available at: https://mystorywigs.medium.com/laravel-advantages-and-disadvantages-fee90e40a41f

Morrow, D. (2018) *Sports Equipment: The Importance of Athletic Equipment Inspections | Recreation Management*, *recmanagement.com*. Available at: https://recmanagement.com/articles/152405/sports-equipment-importance-athletic-equipment-inspections.

React Native, 2024. React Native – A framework for building native apps using React. [online] Available at: https://reactnative.dev/

Royce, W.W., 1970. Managing the development of large software systems. In: Proceedings of IEEE WESCON, pp.1–9.

Royce, W.W., 1987. Managing the development of large software systems: concepts and techniques. In: Proceedings of the 9th International Conference on Software Engineering, pp.328–338.

Runeson, P. (2006) 'A survey of unit testing practices', *IEEE Software*, 23(4), pp. 22–29. Available at: https://doi.org/10.1109/ms.2006.91.

Sadiq, A., Khaskheli, N.A., Laghari, A.A., Sikandar, N., Alia, Rashid, N.K. and Siraj, M.J., 2023. Availability and utilization of sports facilities at high schools of district Faisalabad, Pakistan. *Elementary Education Online*, 20(2), pp.1559-1565. Available at: https://ilkogretim-online.org/index.php/pub/article/view/2274

Sayyed, M. (2015) *Artificial neural network approach for condition-based maintenance*, *arXiv.org*. Available at: https://arxiv.org/abs/1601.03809 (Accessed: 7 October 2025).

Senarath, U.S., 2021. Waterfall methodology, prototyping and agile development. Tech. Rep., pp.1–16.

Shetty, M.Y., et al., 2023. Software Development Life Cycle (SDLC) in Software Engineering – A Brief Review. Journal of Computer Science and System Software, 1(1), pp.5–9.

Singh, R.P., 2023. Pros and Cons of React Native – A Comprehensive Guide. [online] RichestSoft. Available at: https://richestsoft.com/blog/pros-and-cons-of-react-native/

Song, I.Y., Evans, M. and Park, E.K., 1995. A comparative analysis of entity-relationship diagrams. Journal of Computer and Software Engineering, 3(4), pp.427–459.

Srivastava, A., Bhardwaj, S. and Saraswat, S., 2017. SCRUM model for agile methodology. In: 2017 International Conference on Computing, Communication and Automation (ICCCA), pp.864–869. IEEE.

TestFyra, 2023. Integrating Testing Across the Software Development Life Cycle (SDLC). [online] Medium. Available at: https://testfyrablog.medium.com/integrating-testing-across-the-software-development-life-cycle-sdlc-33c770fefba6

Uhrowczik, P.P., 1973. Data dictionary/directories. *IBM Systems Journal*, *12*(4), pp.332-350.

Visual Paradigm, 2019. What is Unified Modeling Language (UML)? [online] Available at: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/

Vue.js, no date. vuejs.org. [online] Available at: https://vuejs.org/guide/introduction

Watanabe, N.M., Shapiro, S. and Drayer, J. (2021) 'Big Data and Analytics in Sport Management', *Journal of Sport Management*, 35(3), pp. 197–202. Available at: https://doi.org/10.1123/jsm.2021-0067.

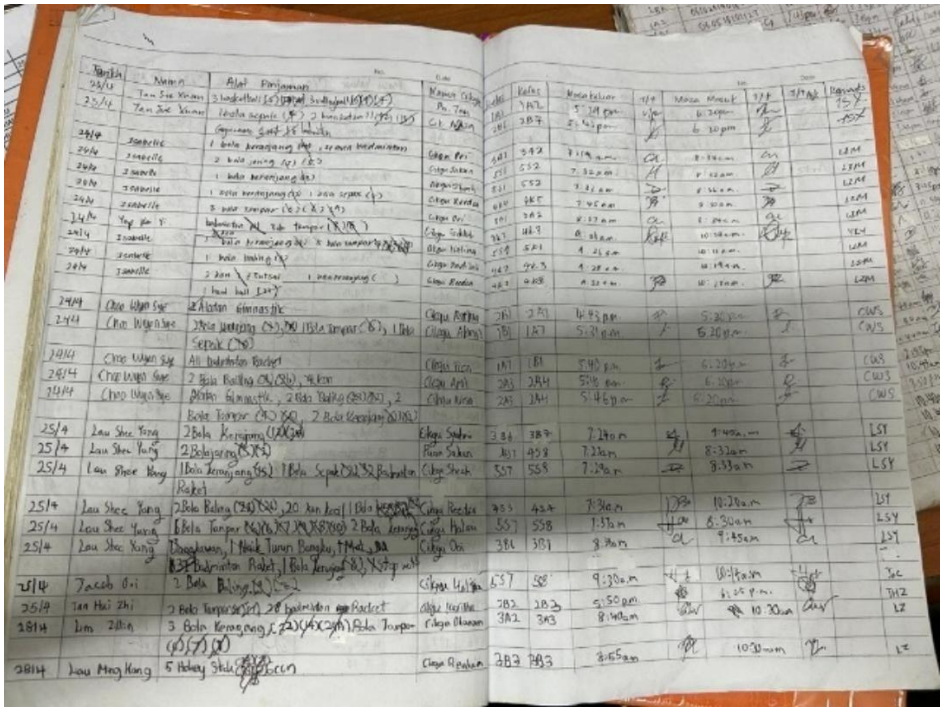
7 Ways Data Can Drive Better Facilities Management Decisions (2025) *Cbre.com.my*. Available at: https://www.cbre.com.my/insights/articles/7-ways-data-can-drive-better-facilities-management-decisions

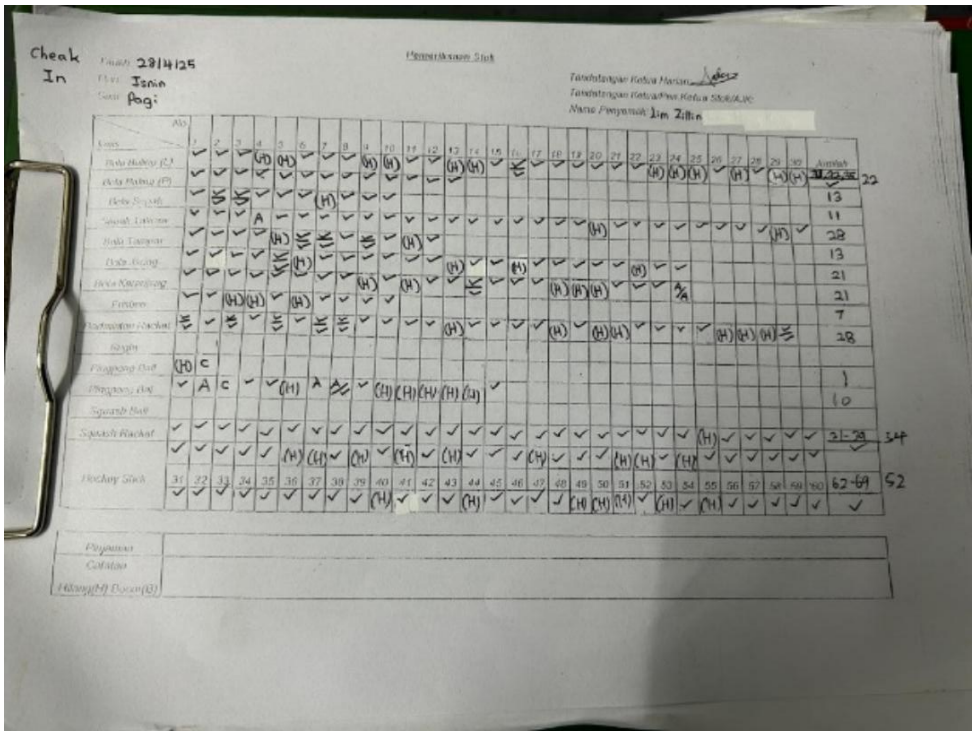
Wu, X., Lu, Y. and Ma, C., 2025. An evaluation method for safety applied to public sports facilities in urban communities. *MethodsX*, 14, p.103256. Available at: https://doi.org/10.1016/j.mex.2025.103256

Y, E. (2018) *What Is WAMP – a Friendly Guide for Beginners*, *Hostinger Tutorials*. Available at: https://www.hostinger.com/my/tutorials/what-is-wamp

APPENDICES

Appendix A: Hardcopy records and Manual Entry

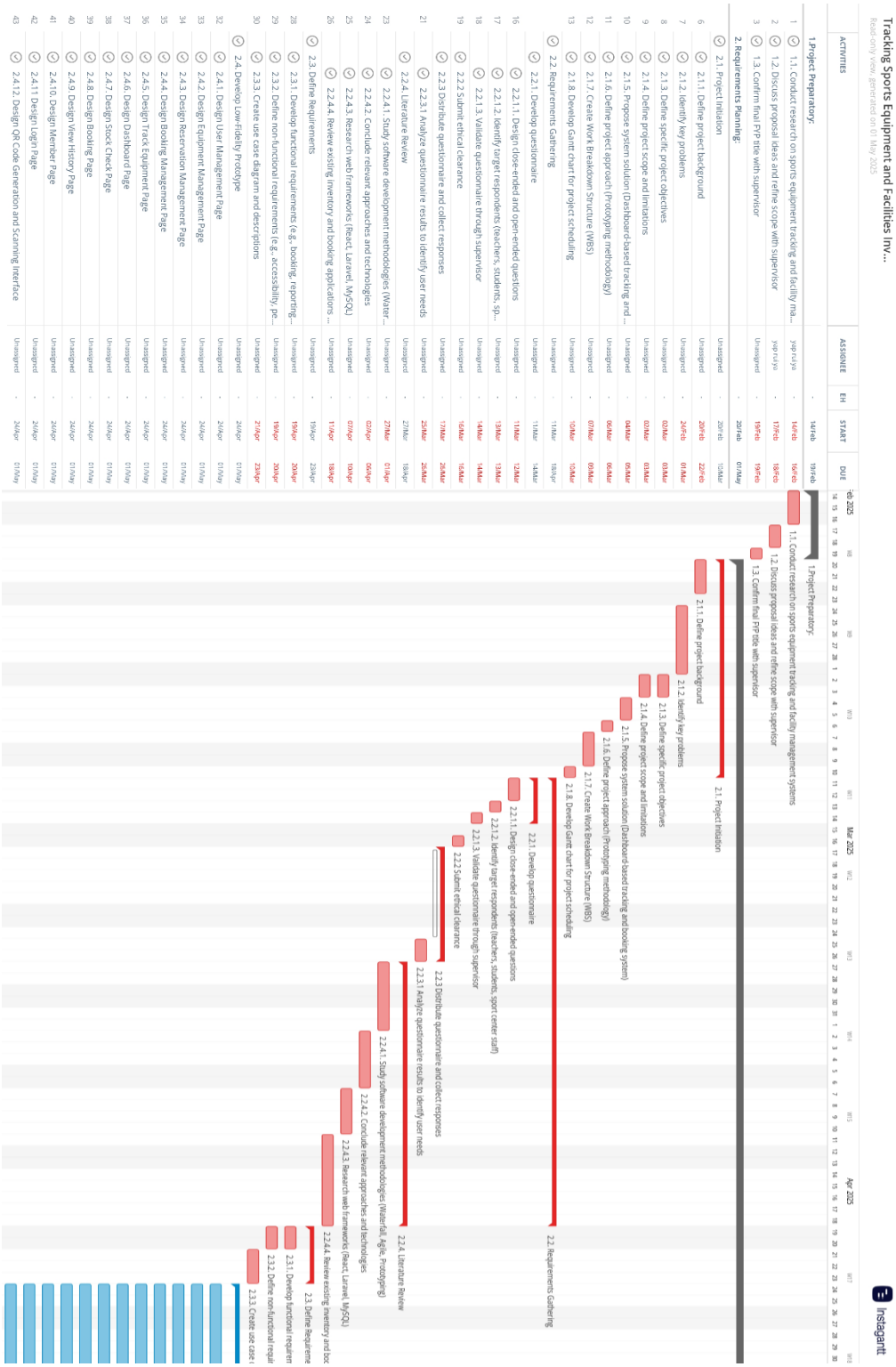




	A	B	C	D	E	F	G
1	**QM ROOM**						
2	BIL	BARANGAN	ADA	ROSAK			
3	1	Badminton racket	40	6			
4	2	Basketball	17				
5	3	Football	5				
6	4	Frisbee	8				
7	5	Handball (L)	24				
8	6	Handball (P)	4	1			
9	7	Hockey ball	18				
10	8	Hockey stick	52				
11	9	Medicine ball	0				
12	10	Netball	22				
13	11	Ping pong ball	5				
14	12	Ping pong bat	4				
15	13	Rugby ball	3				
16	14	Sepak takraw	19				
17	15	Shuttlecock	2				
18	16	Skipping rope	21	9			
19	17	Small cone	18	1			
20	18	Small disc cone	13				
21	19	Squash racket	5				
22	20	Volleyball	8				
23							
24	**UP STORE**						
25	BIL	BARANGAN	ADA	ROSAK			
26	1	Ankle/wrist weight	11				
27	2	Awas cone	9				
28	3	Badminton racket	36				
29	4	Baseball bat	23				

Appendix B : WBS Gantt Chart

FYP1- Gantt Chart



FYP2- Gantt Chart