INVENTORY MANAGEMENT FOR AUTOMOTIVE PARTS: AN APPLICATION DEVELOPMENT

CHANG HAO JIE

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

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

May 2023

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.

Signature : Jie

Name : CHANG HAO JIE

ID No. : 1904983

Date : 27/4/2023

APPROVAL FOR SUBMISSION

I certify that this project report entitled **"TITLE TO BE THE INVENTORY MANAGEMENT FOR AUTOMOTIVE PARTS: AN APPLICATION DEVELOPMENT"** was prepared by **CHANG HAO JIE** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Science (Honours) Software Engineering with Honours at Universiti Tunku Abdul Rahman.

Approved by,

Signature : kckhor

Supervisor : Khor Kok Chin

Date : 19/5/2023

Signature :

Co-Supervisor : Tham Mau Luen

Date : 28/4/2023

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor, Dr. Khor Kok Chin, for his invaluable guidance and support throughout the duration of this project. His expertise and advice have been instrumental in shaping the direction of this development project and in overcoming the various challenges that arose during the project.

I would also like to extend my appreciation to my co-supervisor, Dr. Tham Mau Luen, for his insightful feedback and valuable suggestions on this project.

Special thanks are due to Chua Chen Yang, a student from the Department of Mechanical and Materials Engineering (DMME), for his collaboration in designing the storage prototype system and for his valuable input and feedback throughout the project.

I would also like to acknowledge the contribution of Dr Kuan Seng How from the Department of Mechanical and Materials Engineering (DMME) for his support and advice.

I am deeply grateful to my family and friends for their unwavering support and encouragement throughout my academic journey.

Finally, I am also grateful to the Department of Internet Engineering and Computer Science (DIECS) for providing the resources and facilities necessary for carrying out this development project.

ABSTRACT

This collaborative project with the Department of Mechanical and Materials Engineering (DMME) aims to address the challenges faced by industries in managing their automotive parts inventory. The proposed solution for this project includes developing an inventory management system that includes a web and mobile application to store inventory records. The inventory management system will be integrated with the DMME-designed storage prototype system that is designed to store inventory items. The web application will handle CRUD (Create, Read, Update, Delete) operations to manage inventory data such as product information, stock levels, and reorder levels. The mobile application provides users with the ability to read product details, and update stock levels conveniently and includes a Quick Response (QR) code scanning feature for identifying inventory items and their associated details. As part of the evaluation process, five participants were selected to test the system that includes both the web application and the mobile application. The participants' responses were analyzed to calculate the System Usability Score (SUS), which was found to be 90%. This high score suggests that the inventory management system is highly usable and user-friendly, indicating that it has the potential to improve the management of automotive parts inventory in industries.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS / ABBREVIATIONS

API	Application Programming Interfaces
AWS	Amazon Web Service
CLI	Command Line Interface
CRUD	Create, read, update and delete
DOM	Document Object Model
ERD	Entity Relationship Diagram
FCM	Firebase Cloud Messaging
IDE	Integrated Development Environment
HTML	Hypertext Markup Language
JS	JavaScript
JSX	JavaScript Syntax Extension
JWT	JSON Web Token
QR code	Quick Response Code
SDK	Software Development Kit
SKU	Stock Keeping Unit
SUS	System Usability Scale
UI	User Interface
UID	Unique Identifier
UML	Unified Modelling Language
URL	Uniform Resource Locator
UX	User Experience
WBS	Work Breakdown Structure

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 General Introduction

Inventory management of automotive parts is a process of ordering, storing, using, and selling inventory (Hayes, 2022). This project focuses on managing and warehousing automotive parts, aiming to improve the inventory management processes for the automotive industry.

In automotive vehicles, there are a lot of functional automotive parts to consider, and these parts go into making an automobile become something that can use. As each of these parts is sophisticated, they each have the potential to work well or not. These parts will impact the overall performance of the cars.

Provided automotive parts are already complicated. It stands to reason that if the business is involved in automotive parts or automotive parts management, the business owners should be capable of maintaining the organisation of auto parts when a client requires a specific part for their broken vehicle. Owners should have an effective auto parts management system to track all the parts in a typical car, truck, or complex vehicle. Therefore, an inventory management system will ensure that customers always leave the auto shop with what they need and always have the components available when they need them for their vehicle.

The proper arrangement of the parts will make the business work smoothly and effectively between businesses and customers. However, using an auto parts management system is only the start.

Keeping track of the thousand different kinds of parts can be challenging. Implementing a QR code (quick response code) smart feature will help automotive part businesses manage inventory efficiently and precisely organised. Businesses would benefit from such a system since it would simplify handling physical inventories using QR scanning and digitally saving all the data (Vazquez, 2021). Scannable QR codes ensure employees get the correct item and automatically update the inventory at the register,

guaranteeing that the business stock level is updated immediately. QR code scanners make business inventory management simpler (Kamble, 2021).

Operating an auto parts store poses a set of challenges. Thousands of parts make up a car. There are numerous car brands and models, and things change from one model year to another. As a result, auto parts shops may stock lots of different parts. It might be challenging to manage such extensive inventories.

1.2 Problem Statement

The main problems in auto parts management are outlined below.

1.2.1 Lack of real-time inventory reporting due to manual information entry

Making wise decisions on inventory requires access to inventory reports. Adopting a manual data entry method, the inventory department cannot perform summarisation and reports based on real-time inventory data. It might be challenging to put reports together based on historical patterns fast. Making informed decisions about purchasing and inventory management becomes tricky when management cannot visualise inventory or trends. Hence, the lack of real-time inventory reporting immediately impacts the bottom line of businesses (Ethiraj, 2022).

1.2.2 Inefficient inventory management process

Manual inventory management is a time-consuming and challenging task. Even regular tasks seem to take longer than they should. The process gets slower and more inefficient as businesses grow. Adopting manual paper-based inventory management or non-dedicated inventory management system such as excel across several warehouse sites becomes much more complicated. The handling of inventories ineffectively slows down business operations. Customer dissatisfaction happens because of inventory management issues that result in late product shipment. Poor inventory management may manifest in several ways, including inefficiency and redundancy. The lack of digitalisation and the ineffective inventory management system only led to poor outcomes (Ethiraj, 2022).

1.2.3 Overstocking

Purchasing the same automotive part without first selling the existing one will seriously reduce the profitability of a business because money spent on inventory is locked in (Ethiraj, 2022). The financial health of a business depends on inventory management to stock the correct number of things. Besides, overstocking is the reason for the accumulation of obsolete products, where they are overstocked and no longer in demand by the market. In a manual system, this obsolete product could be overlooked or abandoned. Instead of utilising the existing inventory, the business will waste more funds to purchase more when demand picks up again (Dikov, 2020). If an inventory level review is not done promptly, the likelihood of having excessive inventory rises. Therefore, poor inventory control is a direct cause of excess inventory (Macas, Aguirre and Arcentales-Carrión, 2021).

1.2.4 Understocking

Another issue with inventory management is too few items in stock. In manual systems, inventory records depend on the inventory staff to indicate which inventory to reorder. If the staff makes a mistake, the business might not have enough inventory to satisfy the demand. Sales of automotive parts might be slowed or even stopped due to understocking. In addition, businesses that do not manage their inventory well and do not fully utilise the available storage space are wasting money or incur overhead costs in paying rent. (Weiss, n.d.). Inventory shortages make it more difficult for companies to maintain productivity, increase conflict between departments, and adversely affect customer satisfaction (Bachara, 2019). A regular replenishing schedule and policy are necessary. The precise and timely replenishment of products with customers is essential for the retailing industry to prevent the loss of sales opportunities (Macas, Aguirre and Arcentales-Carrión, 2021).

1.3 Objectives

The objectives of this project are:

a. To develop a web application for inventory management of automotive parts that allows CRUD operations.
b. To develop a mobile application for checking inventory details.

c. To include a Quick Response (QR) code scanning feature for reading inventory details and updating stocks in the mobile application.

1.4 Proposed Solution

Developing a web-based and Android-based mobile application for an automotive parts inventory management system is needed to address the issues described in the problem statement above. The web-based management system would serve as the primary management system allowing users to update the inventory of automotive parts. In contrast, the Android-based mobile application will serve as a companion app to check the inventory status of automotive parts. The mobile application will be built with a QR scanner to obtain the inventory details of automotive parts.

A web-based inventory management system will include most of the standard basic necessary functions seen in other existing inventory management systems. These features will streamline and automate an organisation's operational and inventory processes to ensure operational resiliency and efficiency. For example, current inventory is monitored so that the business can conduct frequent inventory audits to determine inventory levels and prevent over- or under-stocking. The system also streamlines inventory management by allowing users to use Create, Read, Update and Delete (CRUD) operations to inventory automotive parts. In addition, alerts about inventory details will be available as one of the system's unique features. For example, this feature will alert staff when inventory levels fall below the threshold the inventory manager sets.

The mobile application will combine smart features and QR code scanning to identify each item. As mentioned in the problem statement, trying to track thousands of automotive parts can be difficult. Implementing QR code scanning capabilities can help companies organise and manage inventory quickly and accurately. For example, the system makes physical inventory handling faster while maintaining all digitally recorded information. Another outstanding aspect of QR scanning is eliminating common human errors. Manual data entry can lead to mistakes, but scanning QR codes can save employees time and effort. Likewise, mobile applications can check inventory status and receive alerts like a web-based system.

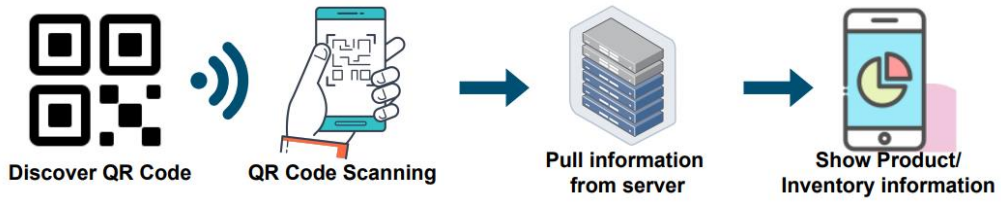


Figure 1.1: Operation flow of QR scanning for inventory management system

The inventory management system will assist businesses in effectively managing inventories and tracking them from entering the facility until they depart. In addition, an inventory management system will simplify the process and provide businesses with real-time data from the web-based system to the phone application, greatly enhancing inventory efficiency.

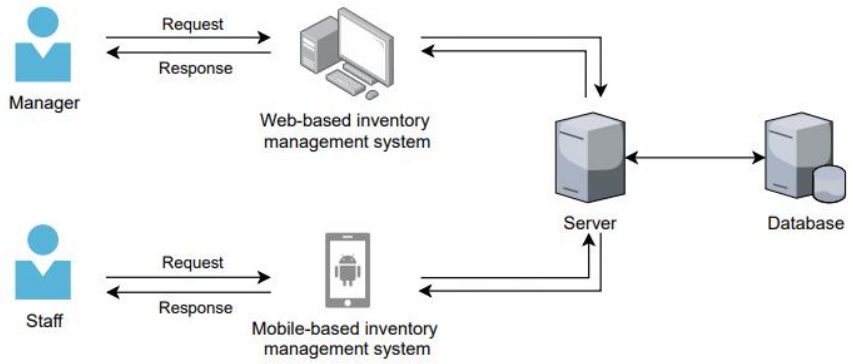


Figure 1.2: Overview of the Design Architecture

1.5 Proposed Approach

The decided software development methodology for this project is the evolutionary prototyping methodology. It is crucial to have a suitable prototyping model before full implementation. Many developers prefer evolutionary prototyping since it is one of the most popular prototyping approaches that has shown remarkable success. First, gather initial user requirements through the planning, analysis, design, and implementation phases. After gathering user requirements, a prototype with limited functionality is created, and the prototype will be presented to the user for the first evaluation process to gather feedback. Feedback will be processed, and the prototype model will be refined again. This approach enables better

knowledge of the requirements to ensure that the system created meets the demands of the users. This procedure continues until the creation of the final successful prototype. At this stage, the iteration of adjusting based on user requirements and feedback on the project will cease, and then the prototype will implement into an actual final product (Martinez, 2020).

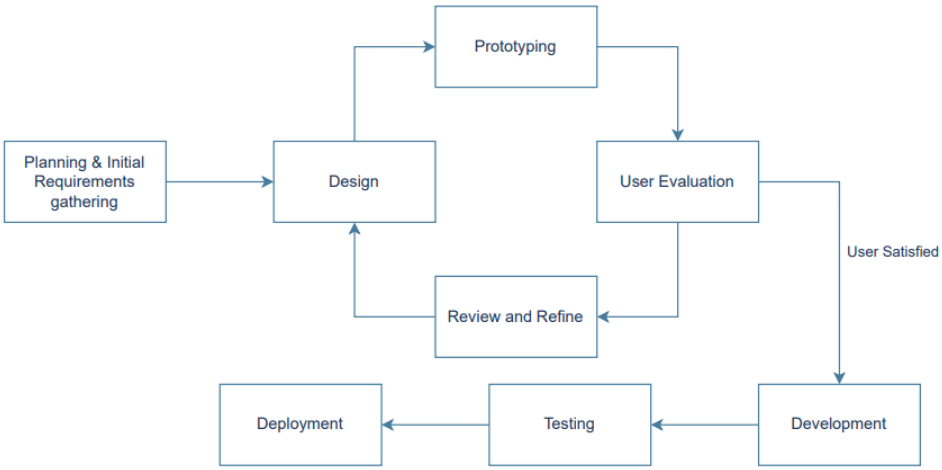


Figure 1.3: Overview of the Software Development Methodology

1.6 Project Scope

This project will cover the development of web-based inventory management and a mobile-based inventory management system for automotive parts. Both systems will integrate for communication and exchange of data and information.

1.6.1 Web-based inventory management

The inventory management system developed for automotive part retailers allows owners, managers, and staff to manage their inventory products easily. The system enables them to create new inventory products, view an overview of the inventory, update inventory information, and remove obsolete or unsellable inventory products. Additionally, the system notifies users via web and mobile applications when inventory falls below the reorder point, or the item is out of stock. Each inventory item comes with a QR code, generated automatically by the web application, which can be printed and pasted onto the item box. This feature enables the mobile application to scan and identify the item easily. The web application also includes user and role

management functionalities that allow for efficient employee management by owners and managers. The React framework provides a user-friendly interface for efficient management of automotive parts inventory.

1.6.2 Mobile-based inventory management system

The mobile application will be exclusively developed on the Android platform using the React Native framework. Auto parts retailers, owners, managers, or employees can access the inventory management mobile application through their Android-based phones, providing users with easy and convenient access to the system. With the mobile application, users can effortlessly check the status of their inventory, receive alerts for low-stock and out-of-stock items, and use their phone's camera to scan the QR codes of products for quick identification and retrieval of inventory details. The mobile-based inventory management system offers seamless integration with the web-based inventory management system, allowing for effective management of automotive parts inventory.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter discusses several existing web and mobile applications for inventory management in the market. The features and functionality of existing inventory management systems will be reviewed as some of them may be used in this project. Moreover, these applications' user interface (UI) and user experience (UX) will be observed to gather ideas for designing an inventory management system based on both mobile and web platforms. This review focuses on discovering similar systems on mobile and web platforms.

2.2 Research and Evaluation of the Existing Inventory Management System

This section discusses five existing inventory management systems: Zoho Inventory, inFlow Inventory, Katana Inventory, Megaventory, and Sortly.

2.2.1 Zoho Inventory

Zoho Inventory is an online inventory management software with free or subscription plans. It is cloud-based software that can be used on a computer, tablet, or cell phone. It can be accessed through a web browser, Android, and iOS apps. Zoho Inventory allows users to add items, create sales orders, create purchase orders, create customer/vendor lists, provide status updates on packages, and view item details from different platforms.

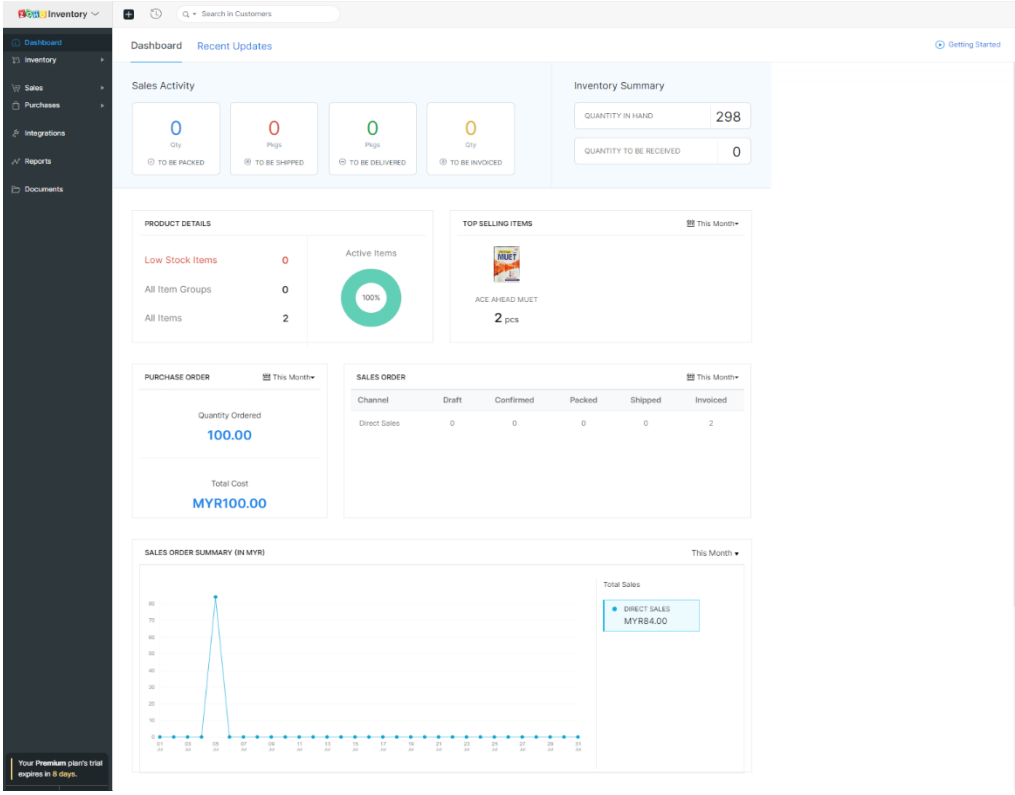


Figure 2.1: Web-based view of the dashboard

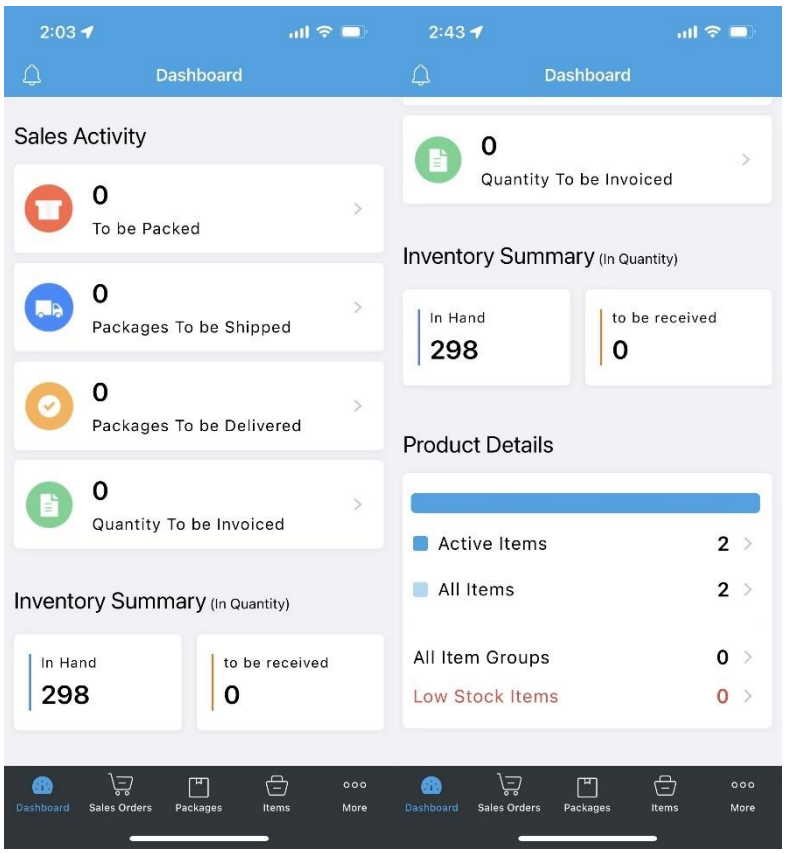


Figure 2.2: The mobile view of the dashboard

When entering the Zoho Inventory Management System web or mobile application, users first see the intuitive dashboard. In the dashboard, users can view sales activity, inventory summary, product details, top sellers, purchase orders, sales orders, and sales order summary, as shown in Figure 2.1 and 2.2.

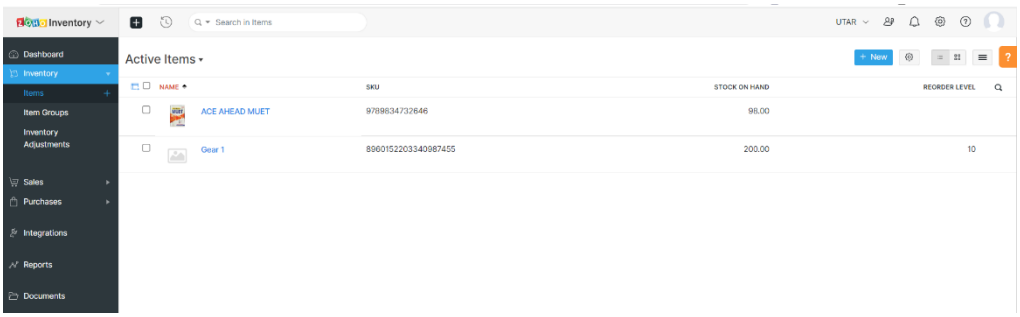


Figure 2.3: List of inventory items

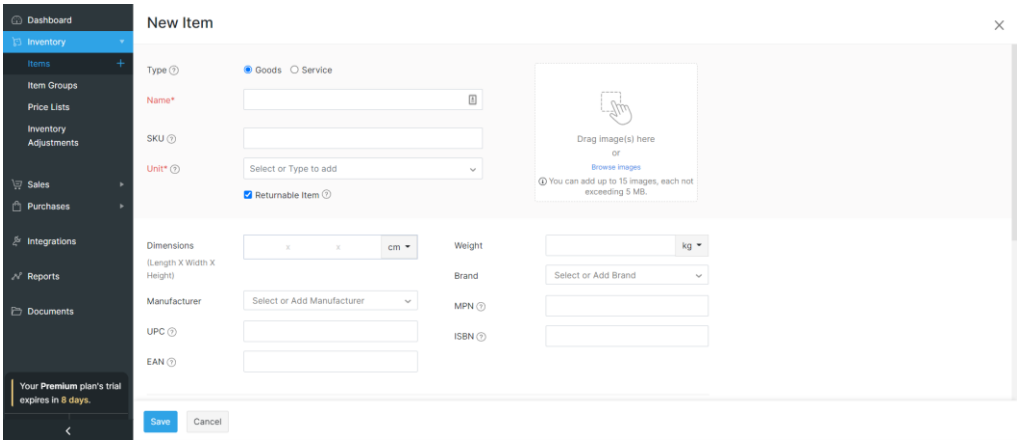


Figure 2.4: Form for adding a new inventory item

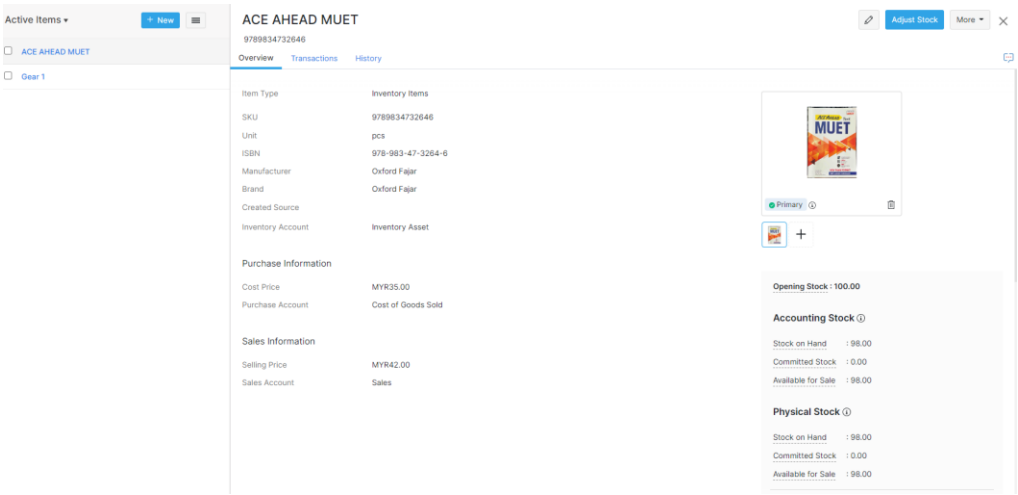


Figure 2.5: Detailed information about the selected item

The user can navigate to the Inventory section through the sidebar menu on the left of the web interface. Next, the user can create a new inventory item through the "+ New" blue button shown in Figure 2.3. Then, referring to Figure 2.4, to create a new item, the user must give a name, unit, selling price, and cost price and choose between Goods or Services for the item while other details like SKU, item image, dimensions, weight, manufacturer, brand, description, tax type, opening stock, opening stock rate, reorder point, preferred vendor and different type of product identifier (UPC, MPN, UPC, EAN, ISBN) are optional.

As shown in Figure 2.3 above, the user can view the list of created items and their information like stock-keeping unit (SKU), stock on hand, and the item reorder level. Besides, the user can click any items from the product list to get detailed information about the selected item, as shown in Figure 2.5 above. Besides, users can adjust the stock quantity through the blue button in the upper right corner in Figure 2.4. Furthermore, Zoho Inventory allows users to group similar items into the same categories.

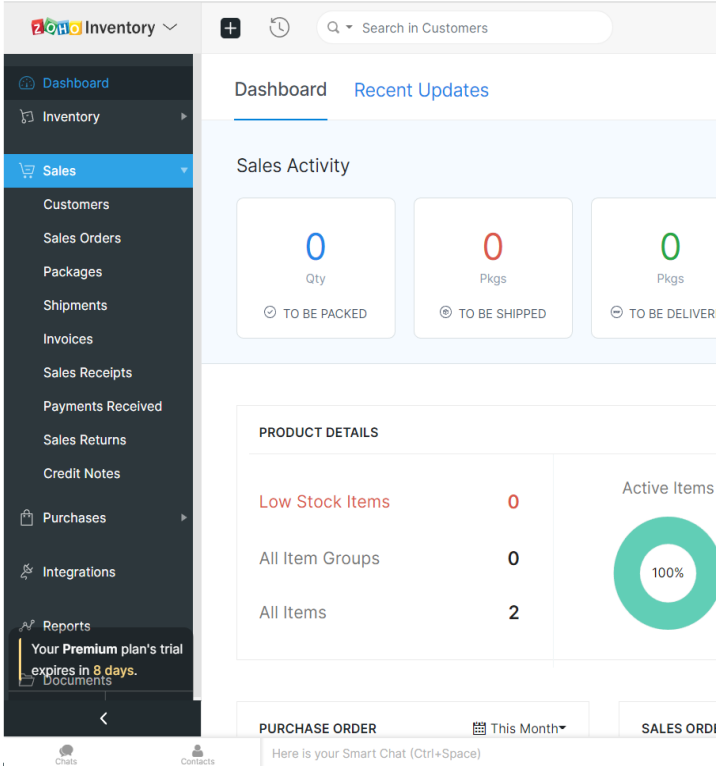


Figure 2.6: Overview of the Sales tab with all the sub-section

After an item is added to the inventory management system, the user can start creating sale orders. The user will find the "Sales" section under the sidebar menu on the left, as shown in Figure 2.6 above. Under the sales section, there are a few sub-sections: Customers, Sales Orders, Packages, Shipments, Invoices, Sales Receipts, Payments Received, Sales Returns, and Credit Notes. Some of the sub-section in the Sales tab will not be discussed since they might not be helpful in this project due to being out of scope, such as Sales Receipts and Credit Notes.

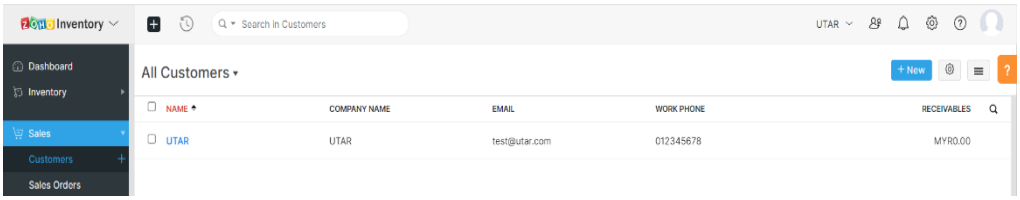


Figure 2.7: View of customer sub-section

The Customer sub-section shown above in Figure 2.7 is like an address book that stores information about the customer's name, company

name, email, work phone number and accounts receivable (outstanding amount). In addition, users can create as many customer contacts as they want.

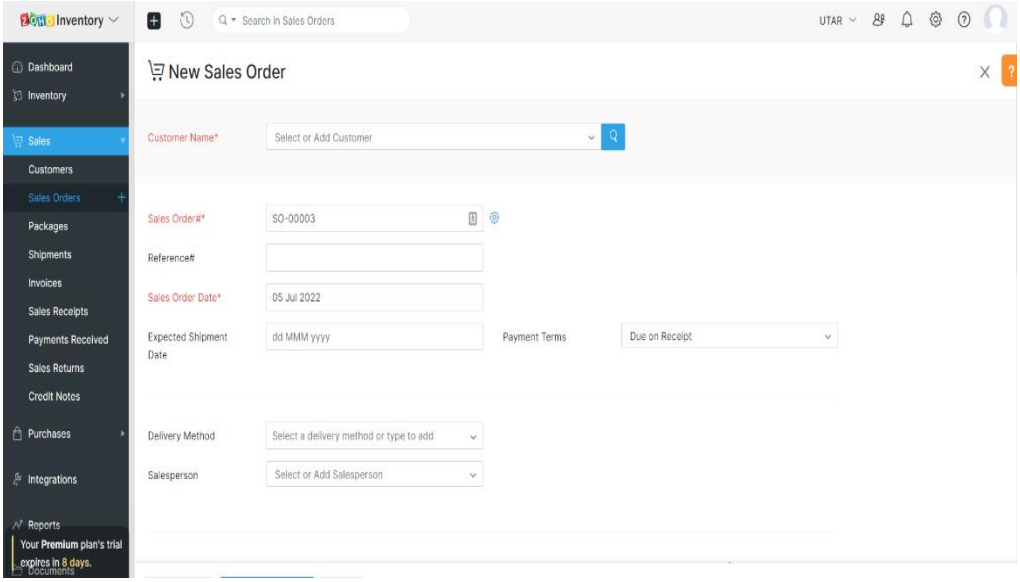


Figure 2.8: Form for creating new sales order (Part 1)

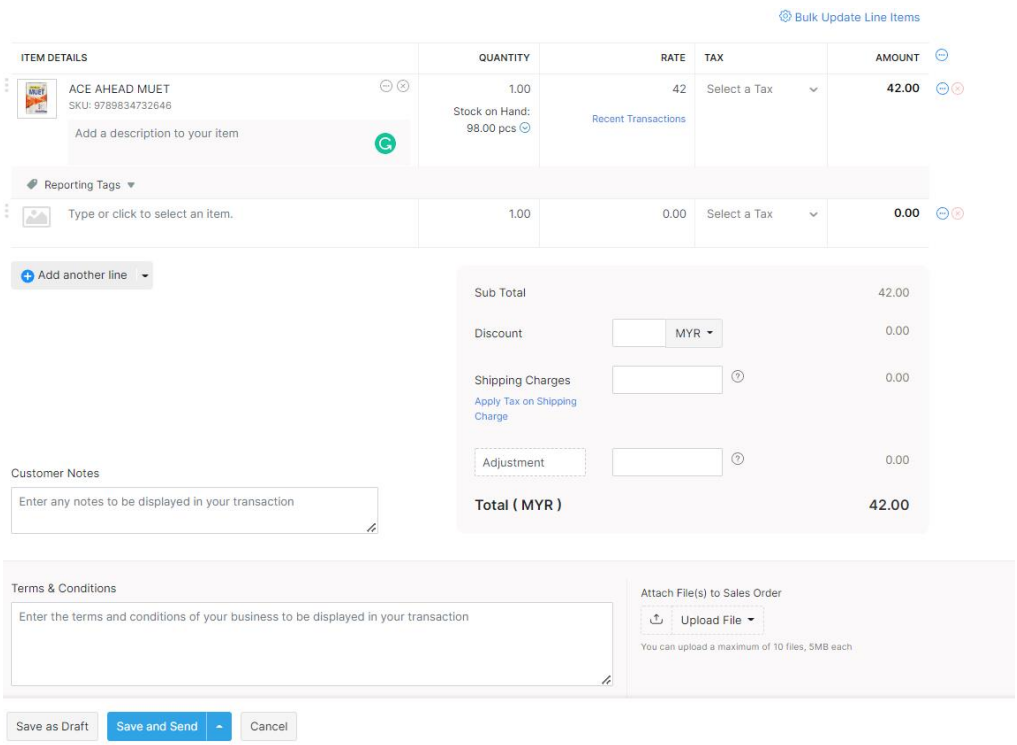


Figure 2.9: Form for creating new sales order (Part 2)

The "Sales Orders" sub-section allows users to create new sales orders. Next, the user will use this form to create new sales orders shown in Figure 2.8

& 2.9. Referring to figures, to create a new sales order, the user must give a customer name (selected contact or create a new one), sales order number (which can be generated automatically if the auto-generate mode is on), sales order date and select items for sales along with their quantity, rate, and tax. While other details like reference no, expected shipment data, payment terms, delivery method, salesperson, customer notes and terms & conditions are optional. In addition, the sales order note can be sent to the customer through the customer's email with additional attachments if required.

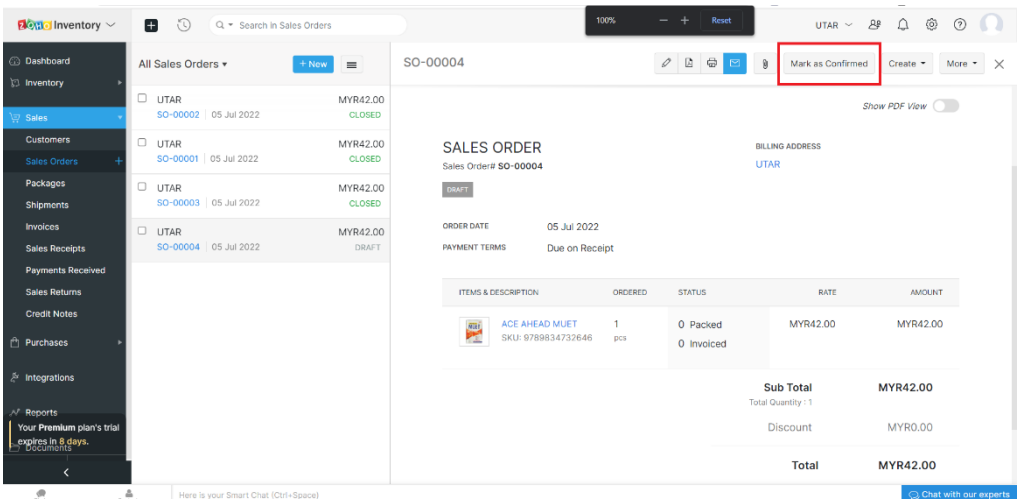


Figure 2.10: Confirmation of sales order

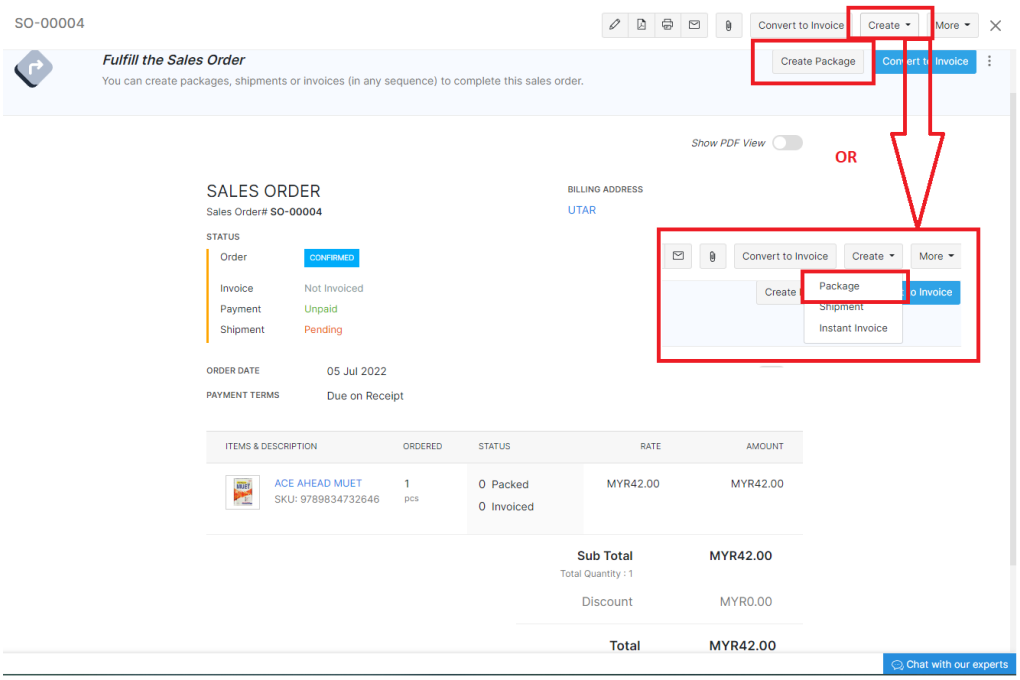


Figure 2.11: Create a package for sales order

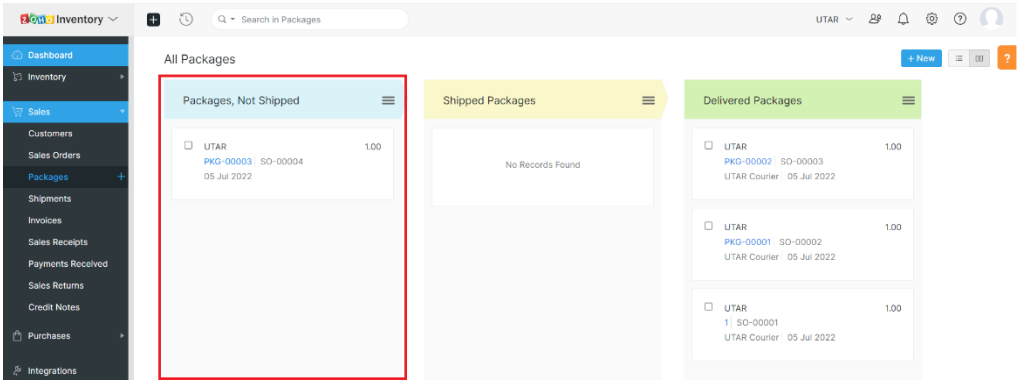


Figure 2.12: Created packages will be shown in packages sub-section

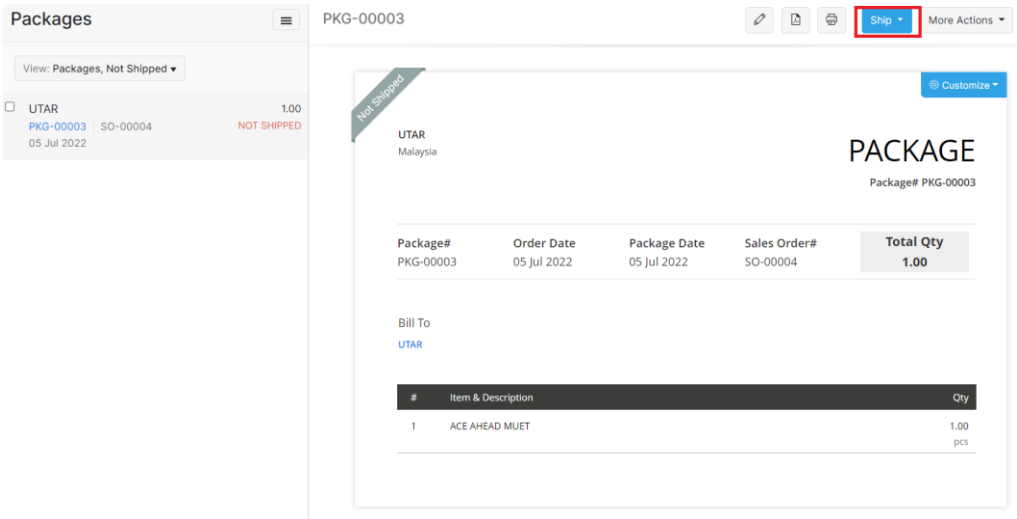


Figure 2.13: Created packages to be shipped

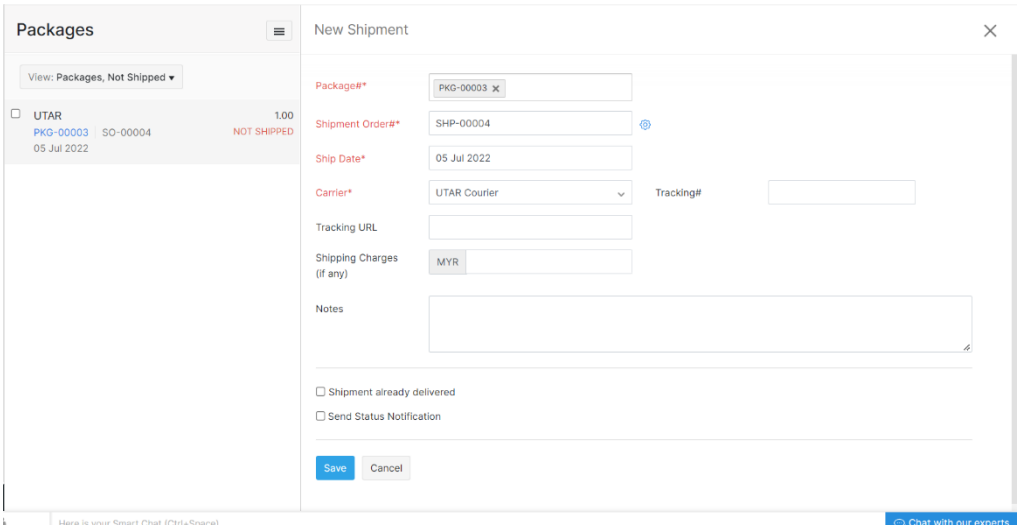


Figure 2.14: Form for creating a new shipment

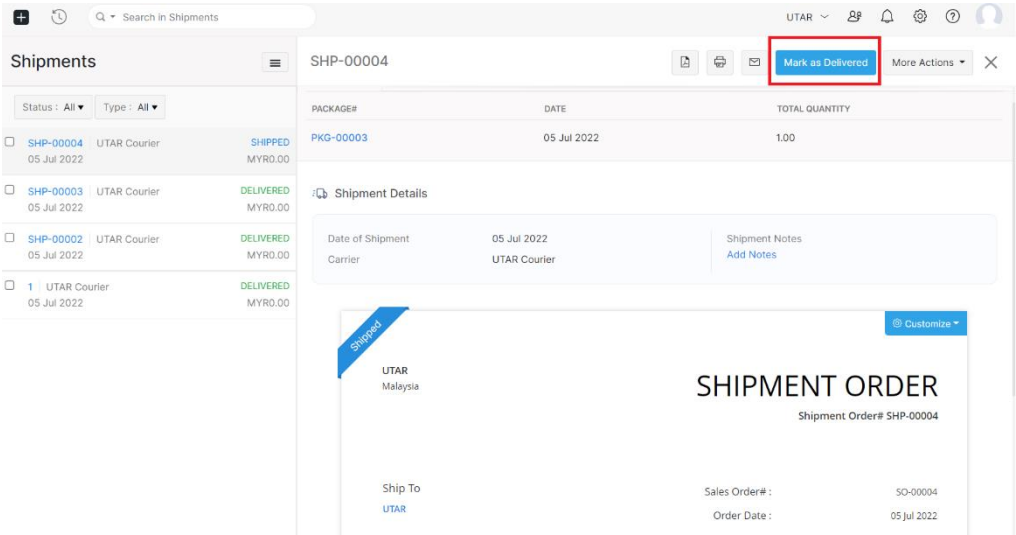


Figure 2.15: Mark shipment as delivered

After the sales order note is sent, the user must confirm the sales order > create a package for sales order > create shipment for sales order > mark the sales order as delivered. All processes are shown in all the figures above, from Figure 2.10 to 2.15. After this process, the quantity in hand will be deducted. In addition, after the customer has made the payment to the business, the business can update the invoice to "payment received" to record that the business has received the payment.

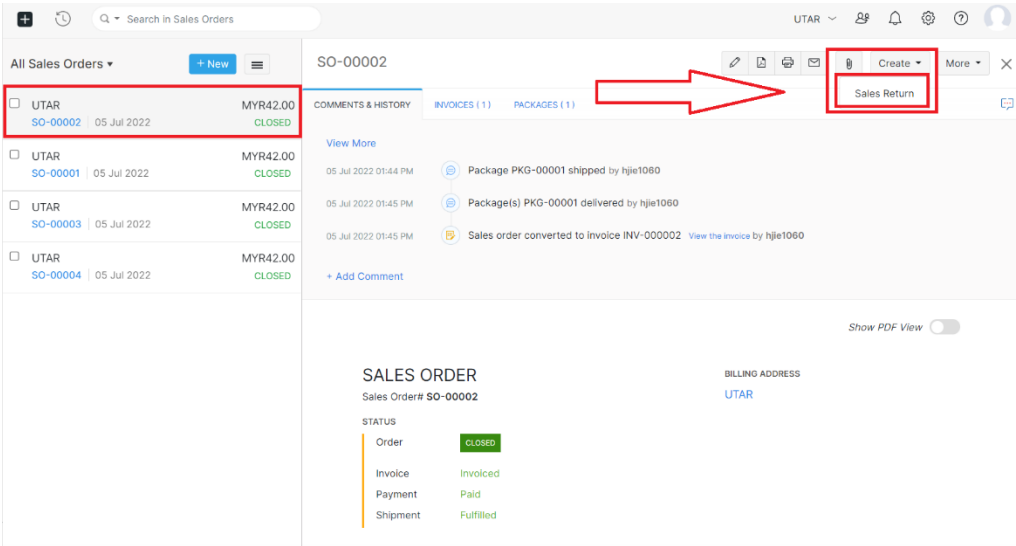


Figure 2.16: Creation of sales return

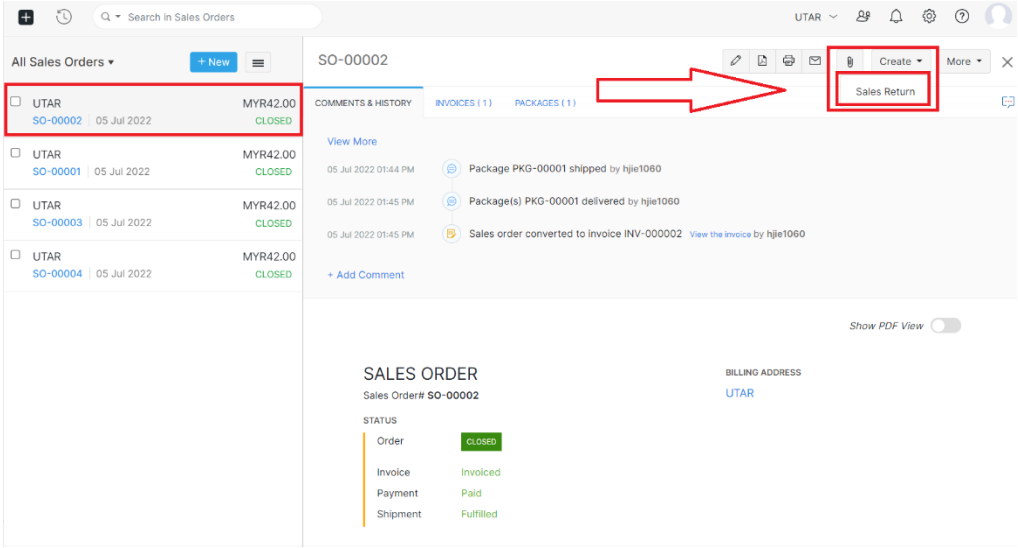


Figure 2.17: Sales return form

Zoho Inventory allows the business to process sales returns for reasons such as broken products or customers wanting to exchange items. The user can perform sales returns by selecting any sales order from the list of existing sales orders, as shown in Figure 2.16 & 2.17 above. In addition, businesses can decide to bring the product back into inventory or discard the item.

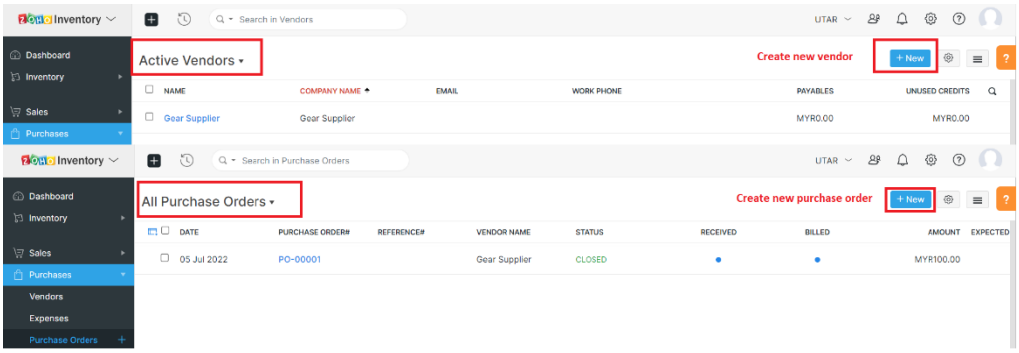


Figure 2.18: Overview of vendor list & purchase orders list

Zoho Inventory allows the business to create purchase orders of different products from their list of vendors, as shown in Figure 2.18 above. Besides, the business can add a new vendor to the list when they have a new supplier/vendor. In addition, the user can create a purchase order to restock. After the user clicks the "+New" blue button, the user will be brought to the form for creating purchase orders.

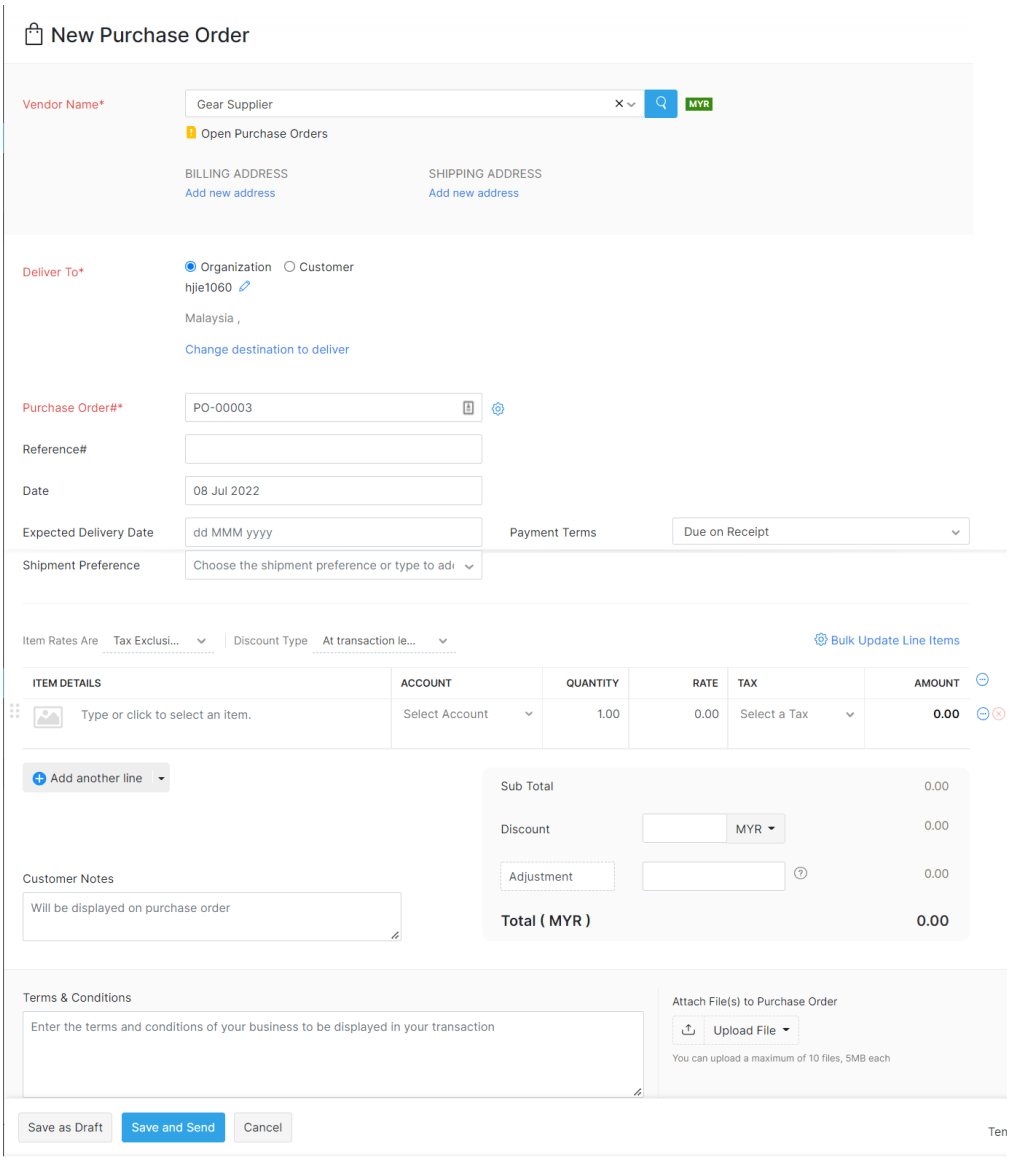


Figure 2.19: Form for creating a purchase order

In the purchase order form shown above, the user must give a vendor name, delivery to who (own organisation customers), purchase order number (generated automatically if the auto-generation mode is turned on) and select one or more than one item while other details like reference number, date, discount, and other details are optional. In addition, the purchase order note can be sent to the vendors/suppliers through the vendor's email and additional attachments if required.

After the purchase order note is sent, the user must confirm the warehouse has received the item > convert to the bill. After this process, the quantity in hand will be added.

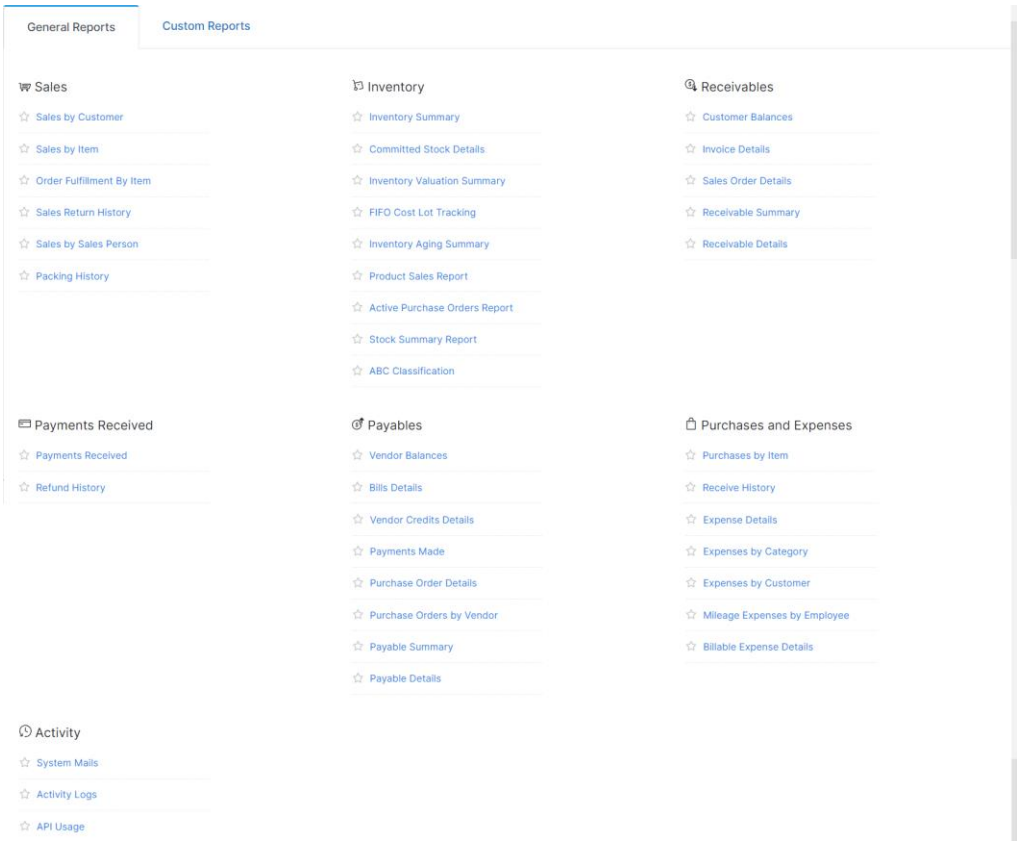


Figure 2.20: General reports and custom reports

Moreover, Zoho Inventory allows businesses to generate various reports such as sales by customer, inventory summary, purchases by item and more. Besides, businesses can also generate customised reports, as shown in Figure 2.20.

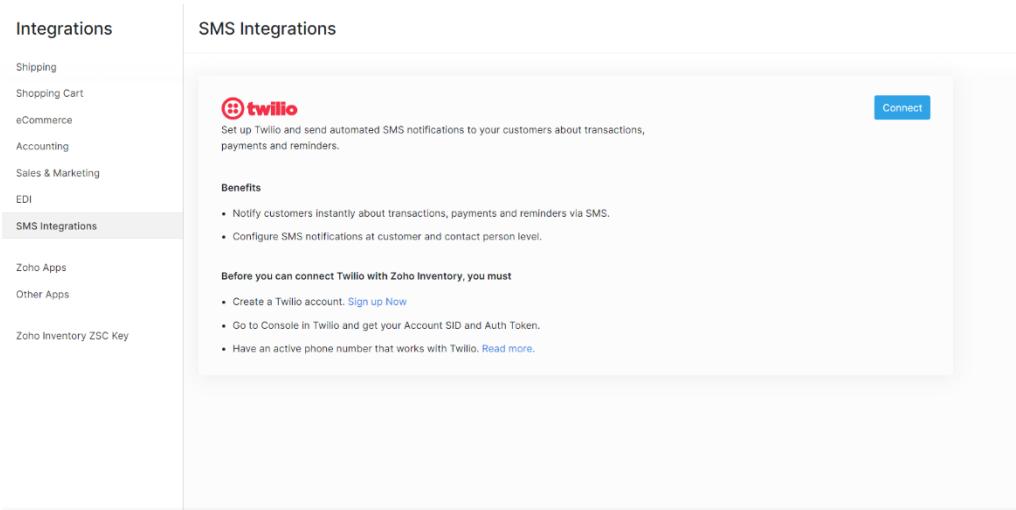


Figure 2.21: Zoho Inventory's API integration

Advanced users might find API integration useful in Zoho Inventory. A list of different APIs is shown in Figure 2.21. In addition, the inventory management system has a few API integrations, such as SMS integration which can help alert customers immediately about transactions, payments, and reminders through SMS notifications.

In addition, the system allows businesses to add different currencies and set up taxes, custom document templates, multi-user with different roles and multiple warehouses or branches with different locations.

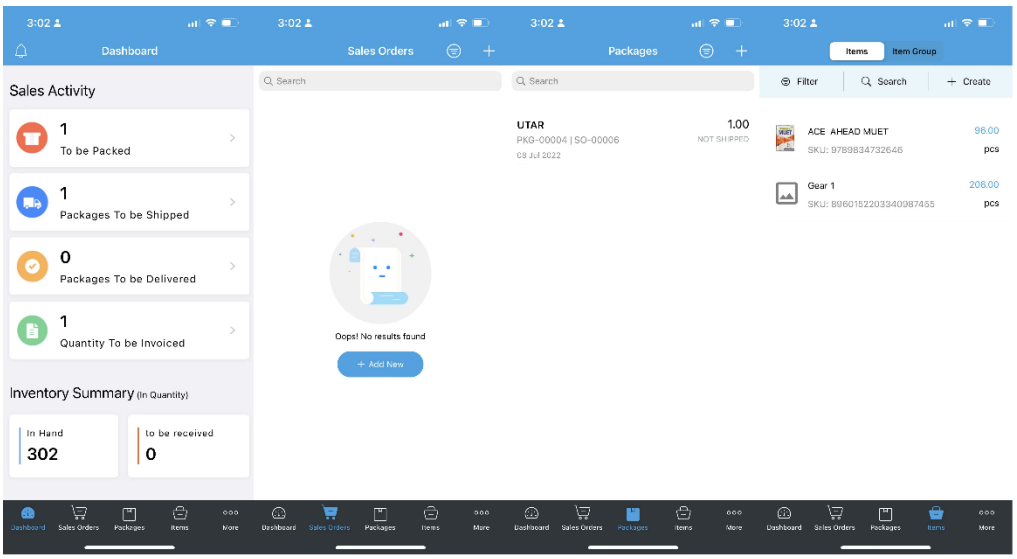


Figure 2.22: Overview of Zoho Inventory mobile application

Furthermore, the Zoho Inventory management system has a Zoho Inventory companion mobile app on iOS and Android. The mobile application helps businesses manage orders, track shipments, keep tabs on inventory, and much more, just like the web version of the inventory management system with essential features.

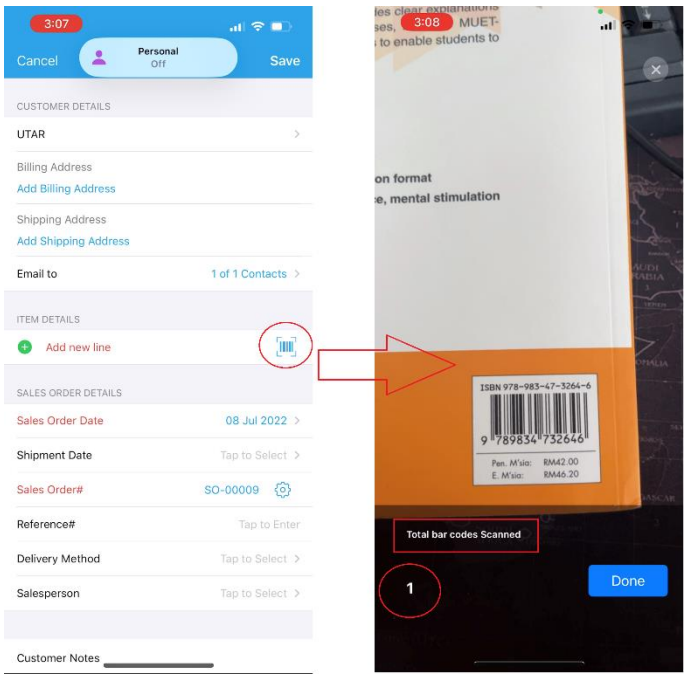


Figure 2.23: Overview of how the barcode scanning works

The mobile application features a smart function: the barcode scanning option that streamlines business order processing. This inventory tracker uses the camera of any mobile phone to scan barcodes on items to detect the item and display its details. In addition, the barcodes scanning smart feature allow the user to scan multiple barcodes. An overview of how the barcode scanning works on the iOS application is shown in Figure 2.23.

2.2.2 inFlow Inventory

inFlow Inventory is an inventory management system suitable for small and medium-sized enterprises (SMEs). inFlow inventory is used in manufacturing, warehousing, distribution, and retailing. It is a cloud-based service that stores user data in the cloud and works on computers, tablets, and phones. It is accessible through the web browser, client software (Only available on Desktop - Windows), Android and iOS applications. inFlow Inventory allows

business users to create purchase orders, sales orders, new products, vendor lists, and customer lists and transfer or fulfil stock in real-time. The mobile application is a cloud companion app that helps businesses to manage inventory from anywhere. Since this is a paid subscription, the trial version will be used to demonstrate the operation of the software.

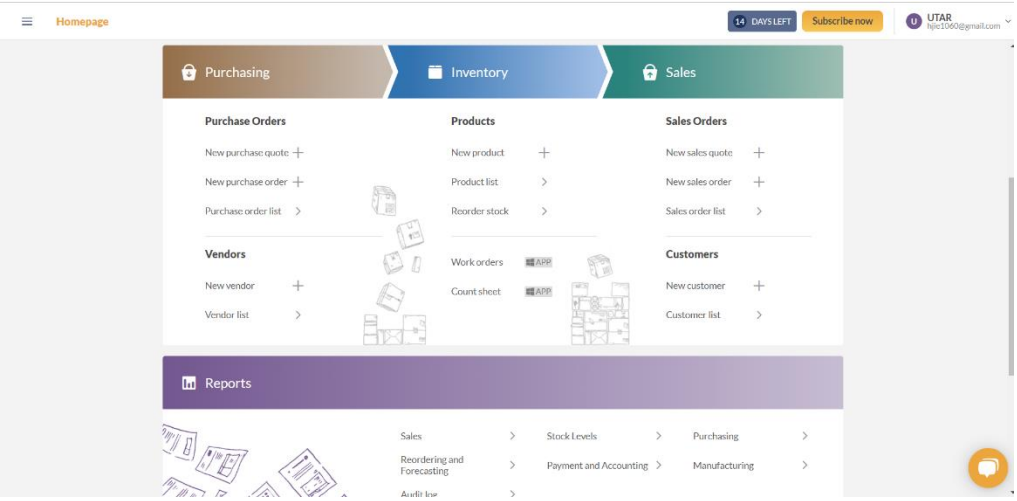


Figure 2.24: Overview of the homepage for inFlow inventory web application

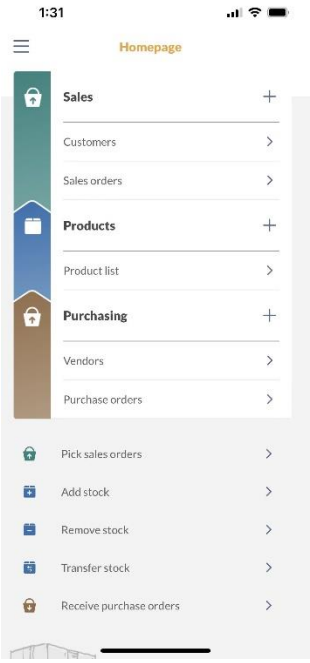


Figure 2.25: Overview of the homepage for inFlow inventory mobile application

When entering the inFlow Inventory Management System web or mobile application, the user will first see the home page. On the home page, the user will see navigation links to all the operational functions of inventory management in the home page, as shown in Figure 2.24 and 2.25.

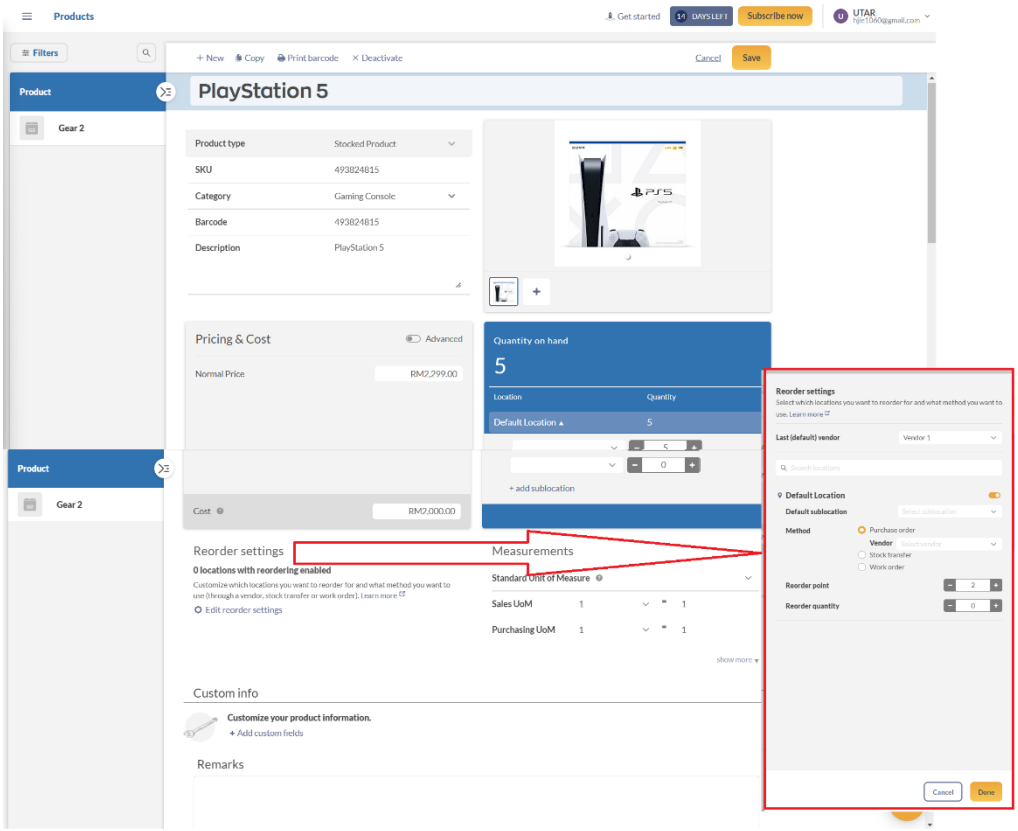


Figure 2.26: Form for adding new inventory product

On the homepage, the user can create a new product. After the user clicks the "New Product" link, the user will be brought to a form, as shown in Figure 2.26. In creating a new product, the user may fill in the product SKU, categories, barcode no., description, reorder point, pricing & cost, quantity on hand and remarks. In addition, the business can set up a reorder setting for this product so the business will get an alert on restocking based on the reorder point. The business is reminded to reorder from the respective vendor.

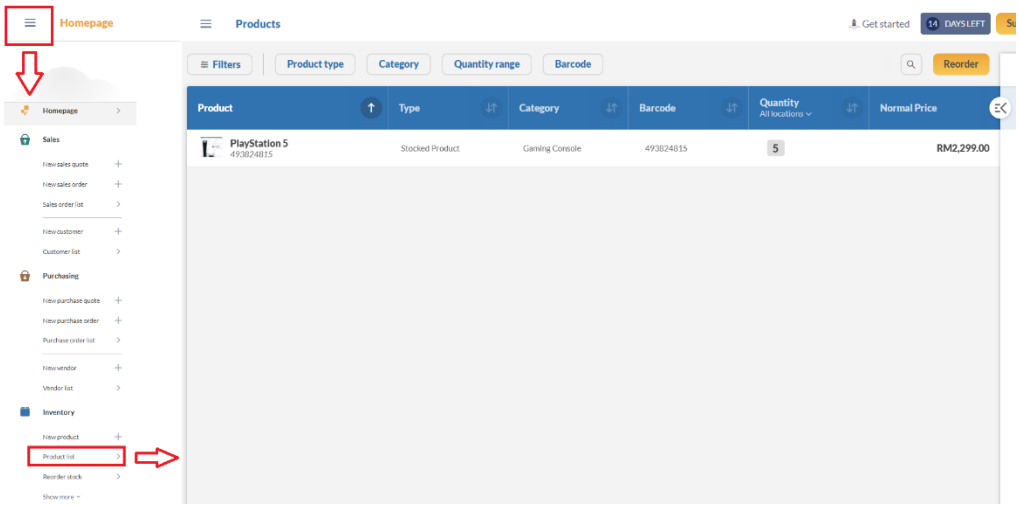


Figure 2.27: Form for adding new inventory product

As shown in Figure 2.27 above, the user can navigate to the product list through the navigation bar on the left to view the entire list of created products/items, along with the type, category, barcode, quantity, and normal price. Besides, the user can click any of the products in the list to get detailed information about the selected product. In addition, users can adjust the stock quantity, and change product name, SKU, barcode, and other details mentioned in creating product/item operation.

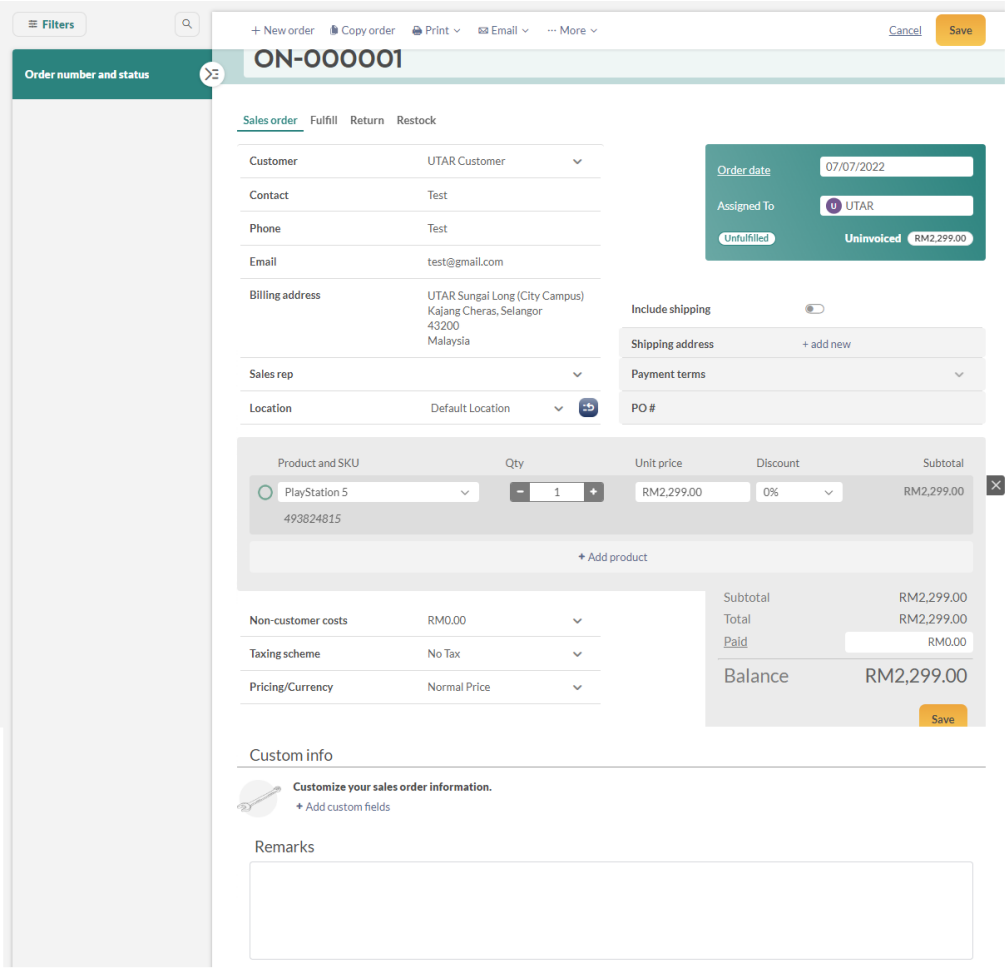


Figure 2.28: Form for creating sales orders

After the user has successfully created a new product, the user can now create sales orders. Next, the user will use this form to create new sales orders shown in Figure 2.28. Referring to figures, to create a new sales order, the user may fill in the order number, order data, customer (Select customer from the customer list), some customer details, taxes, remarks and lastly, add products to the sales order (Can be select from the product list).

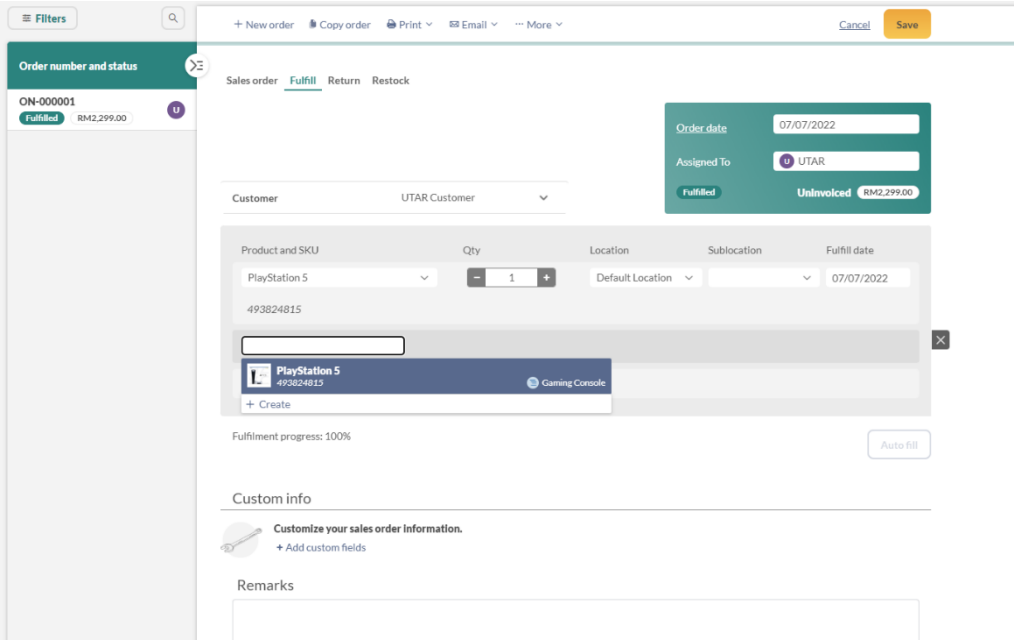


Figure 2.29: Form for fulfilling sales orders

After the user has created the sales order, the user can fulfil the order by filling up the fulfil form, as shown in Figure 2.29. The user may add more products to the fulfil form if the customer requests additional quantities or other types of products.

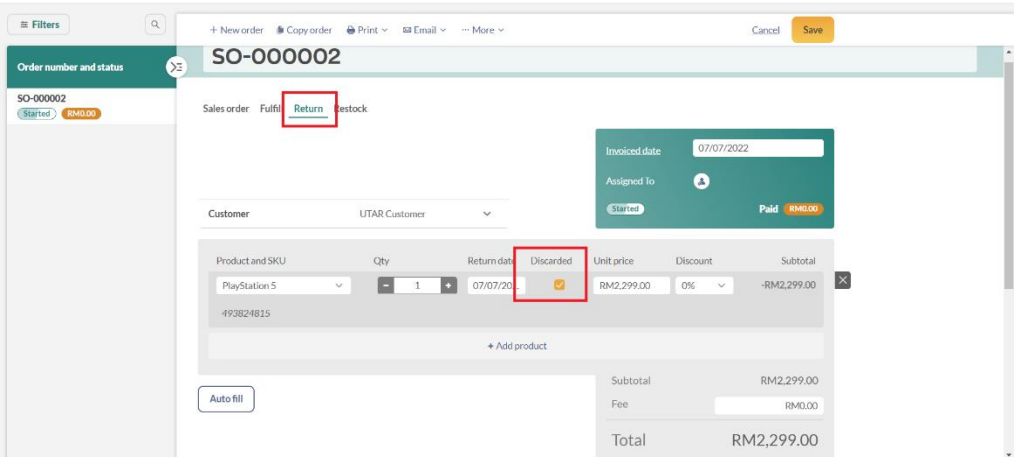


Figure 2.30: Form to process sales returns

In case the customer receives the wrong product, a broken product, or some other reason, inFlow inventory allows the business to process sales returns. The user can perform sales returns through the form shown in Figure 2.30 above. After the product is returned from the customer, the user can

decide to add the returned product back to the inventory (Known as "restock" in this system) or discard the broken product by clicking the checkbox "Discarded".

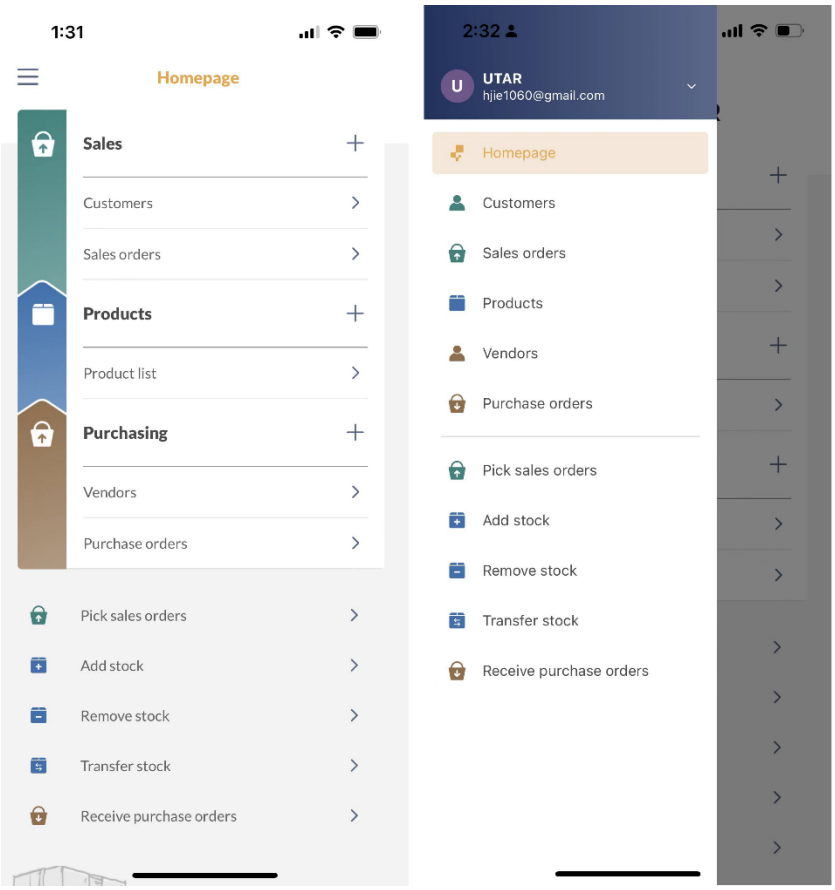


Figure 2.31: Overview of the homepage for the iOS application

In addition, the inFlow Cloud Inventory Management System has an inFlow Cloud Companion mobile app for iOS and Android that helps businesses stay productive and provide a more convenient solution from anywhere. The mobile app allows companies to manage operational functions without a computer, as long as the user has the app installed on their phone. The companion app works similarly to the inFlow web system, with the basic functionality and one additional smart feature, a barcode scanner. The home page overview of the iOS app is shown in Figure 2.31 above.

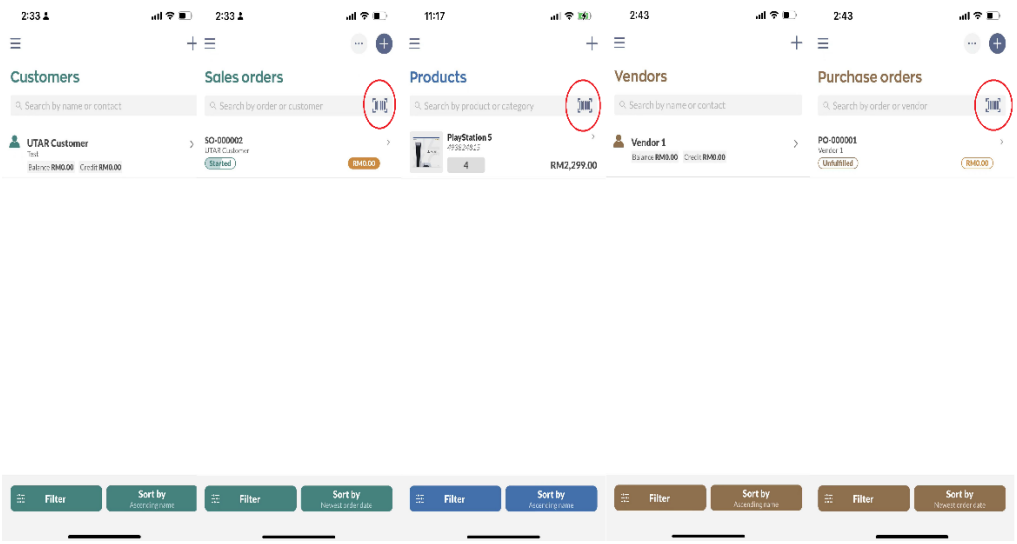


Figure 2.32: Overview of the operational functions

As shown above, the user can access each of these lists with their operational functions through the navigation sidebar menu displayed on the right in Figure 2.31, and all the list is shown in Figure 2.32. These lists are the customer list, sales order list, product list, vendor list and purchase order list, all similar essential operational functions in the web version.

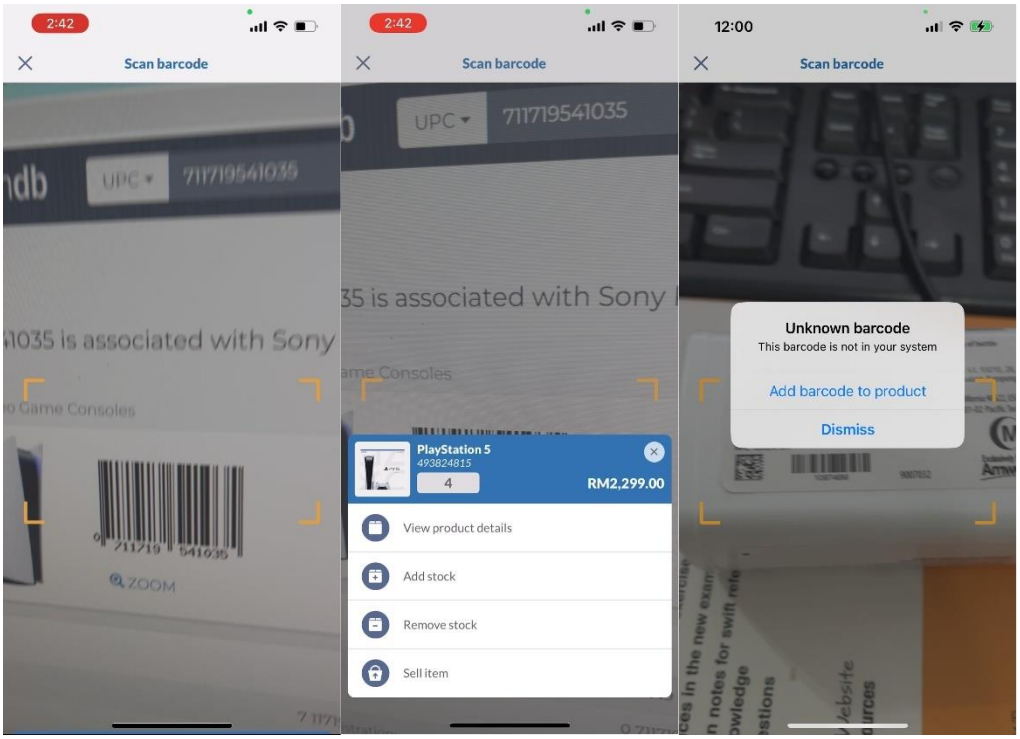


Figure 2.33: Demonstration of barcode scanning

The mobile application allows the user to utilise the built-in camera of their mobile phone as a barcode scanner to get information on the product, such as stock count and selling price. In addition, businesses can mark items as shipped by scanning barcodes without having additional hardware like a barcode reader. As shown in Figure 2.32, the barcode scanning feature can be accessed by clicking the barcode icon (Circled in red). The demonstration of barcode scanning is shown in Figure 2.33. After the barcode is scanned and matched in the system, the matched product's operational functions, such as viewing product details and adding stock, will be shown. If the barcode does not match any product in the product list, a prompt will ask the user to relate it to a product or dismiss the prompt to cancel it.

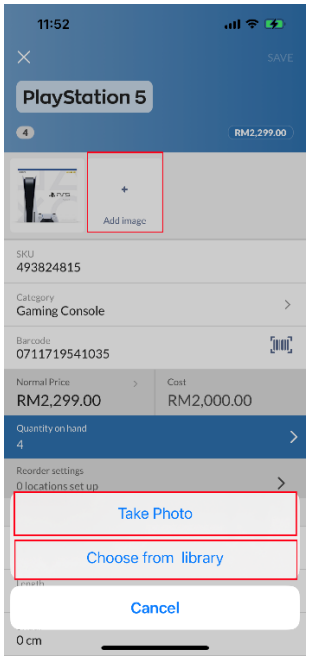


Figure 2.34: Adding product images using the mobile application

In addition, adding product images using the mobile application is more convenient than using the inFlow web, as shown in the figure above. The mobile application user can take photos through the built-in camera or select images from their library without moving product images from their camera device to their computer and uploading them to the inFlow cloud.

2.2.3 Megaventory

Megaventory is a business operation management tool that enables businesses to manage inventories, orders, and fulfilment while keeping track of the packaging or assembly process. Its simplicity makes it an ideal inventory management system for small and medium-sized manufacturing businesses. Megaventory is a cloud-based service that works well on any internet-connected browser. Since this is a paid subscription software, the trial version will be used to demonstrate the operation of the software. In addition, this inventory management system uses the First In, First Out (FIFO) method. FIFO is a method of accounting where assets bought or acquired are sold off first.

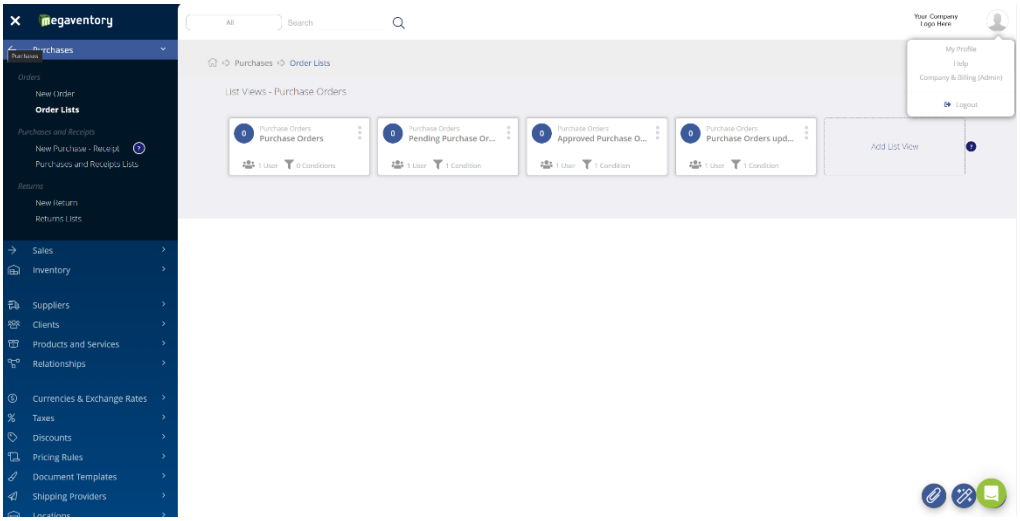


Figure 2.35: Overview of the Megaventory

When entering the web application of the Megaventory inventory management system, there is a navigation menu on the left and a view of the order list in the centre. In the navigation menu, the user can navigate to all other sections to access all the operational functions of the system. An overview of the Meganventory system is shown in Figure 2.35 above.

Meganventory inventory management system is like the Zoho Inventory and inFlow Inventory management system but offers different designs and slightly different features. In this section, only a brief discussion of the Megaventory system will be given due to the duplication of functionality with other systems, but unique features will be discussed further.

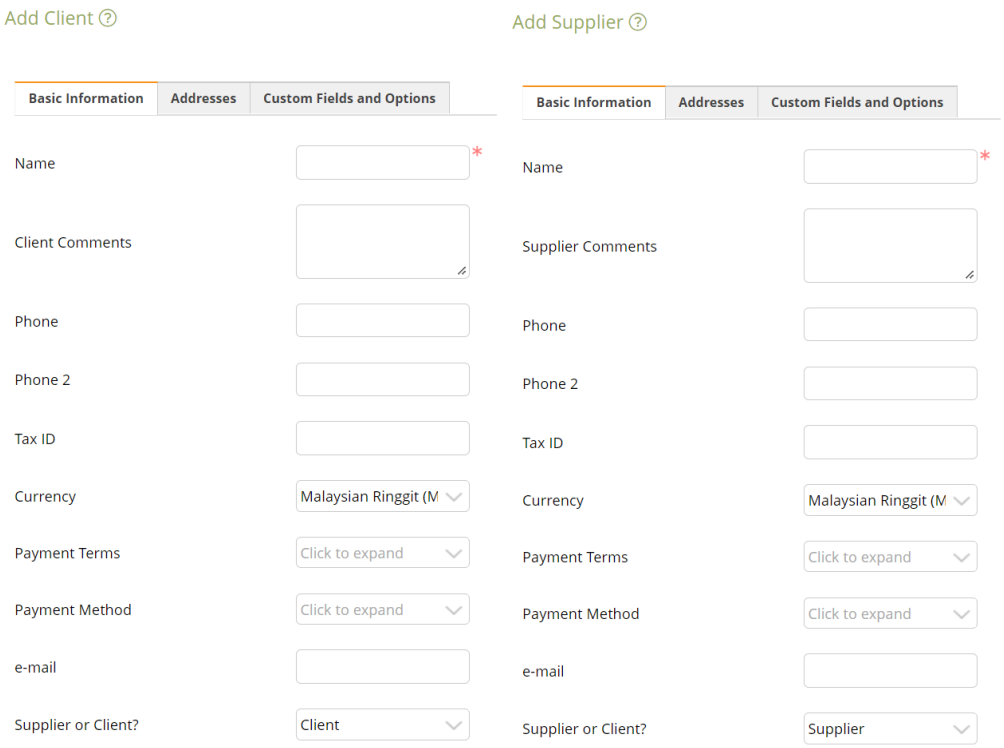


Figure 2.36: Forms for adding client/supplier entity-relationship

Before using the Megaventory inventory management system, the user must add supplier and client to create an entity relationship between the business and supplier or client. The business user can add clients and suppliers using the form shown in Figure 2.36 above. These entity relationships allow items/products to be transferred, sold, or purchased.

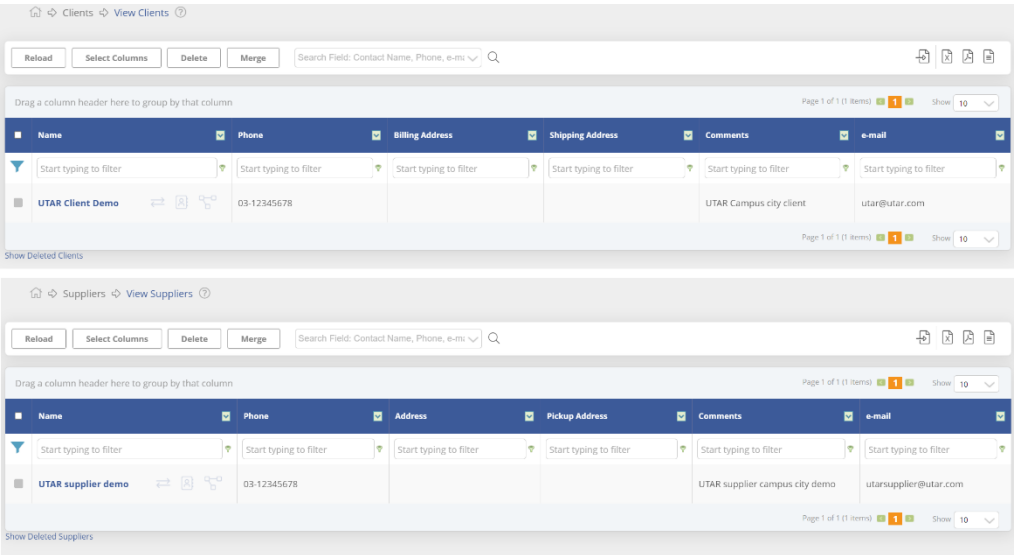


Figure 2.37: Client/Supplier list

After all customers or suppliers are added to the system, users can view the list of customers or suppliers, as shown in Figure 2.37 above. The list of suppliers can later be assigned to specific products to facilitate the purchase order process. Conversely, the customer list can be used later for sales orders. In addition, users can sort the list, import/export the list, search for specific customers/vendors, and some unique features that enable business users to recover deleted customers/vendors.

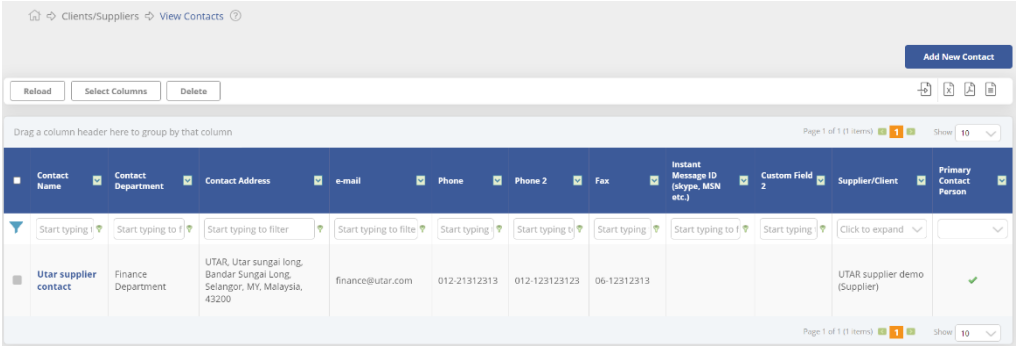


Figure 2.38: Contact list

Moreover, the system has a contact list for both clients and suppliers. A Contact in the contact list is a physical person responsible for other organisations' sales and purchase orders. They can be classified as a supplier or a customer. For example, if a supplier is a huge corporation or organisation, the contact person field assists in determining who (name, department, et cetera) issues the buy order or receives the sales order

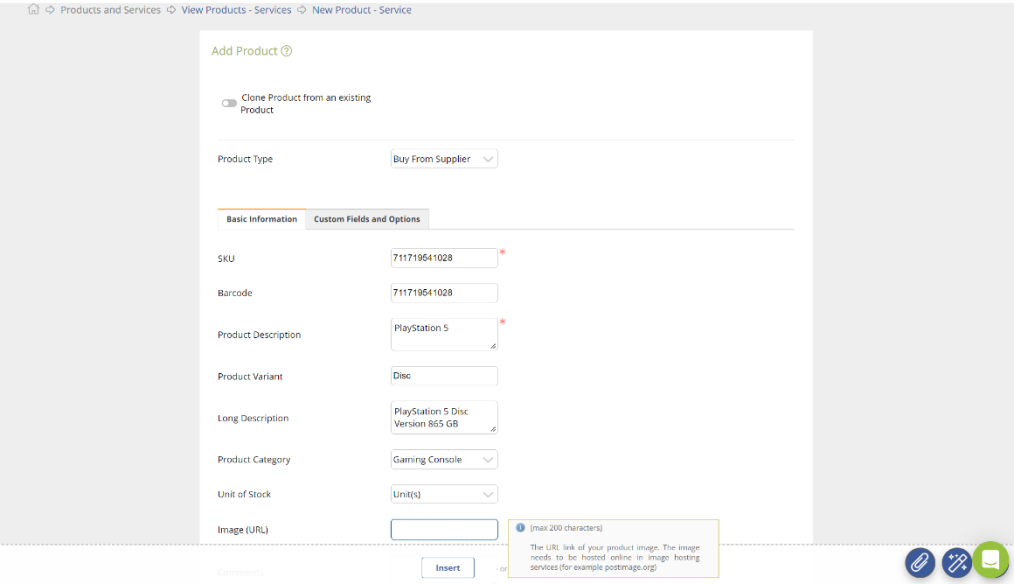


Figure 2.39: Add product/item form

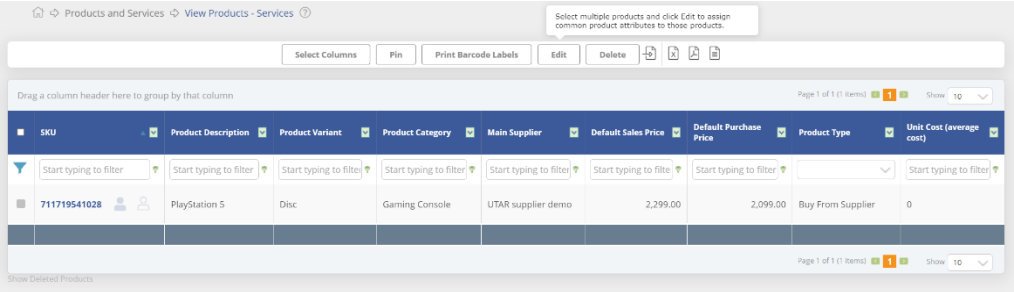


Figure 2.40: Product/item list

After the supplier is successfully added, the user can add products/items into the system. Figure 2.39 shows the required details, such as SKU, barcode, product description, main supplier, and other information. This system has a downside because the product image needs to be hosted online outside the system (3rd party image hosting service), which means the user needs to provide the image link to the system. Figure 2.40 shows the product added to the product list.

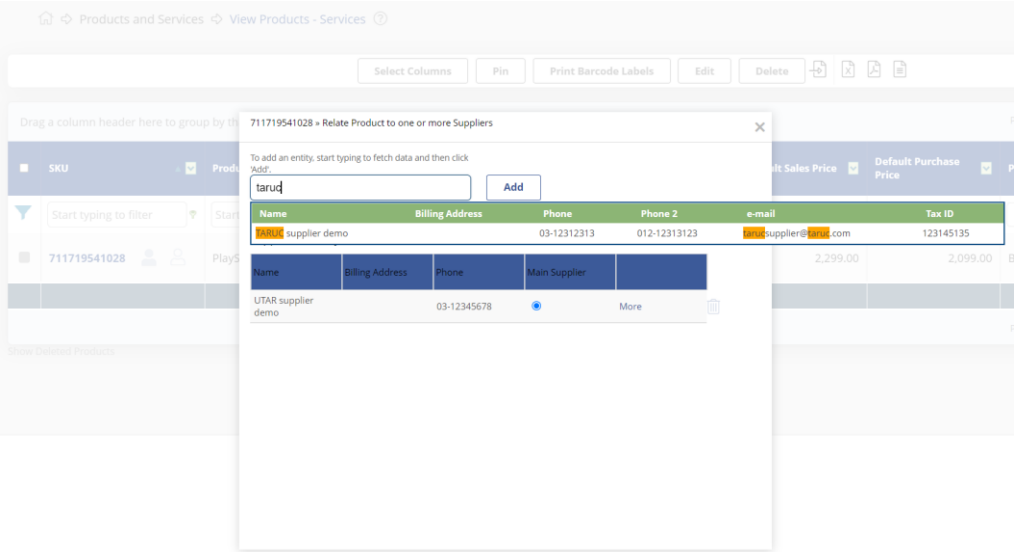


Figure 2.41: Relate product to one or more suppliers

In addition, the product item added can be related to one or more suppliers, as shown in Figure 2.41. Hence, the business can source the item/product from multiple suppliers.

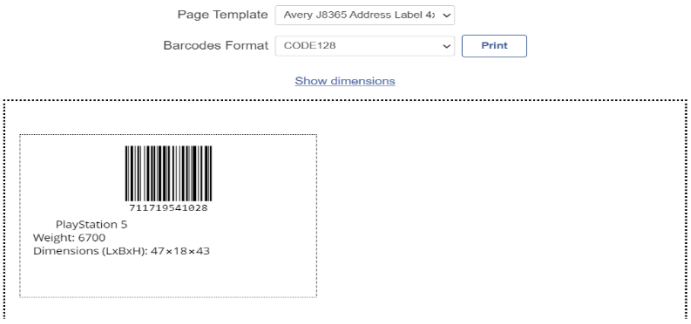


Figure 2.42: Printing barcode labels of product/item

Another unique feature is that the system can print barcode labels of the item/product using the product list.

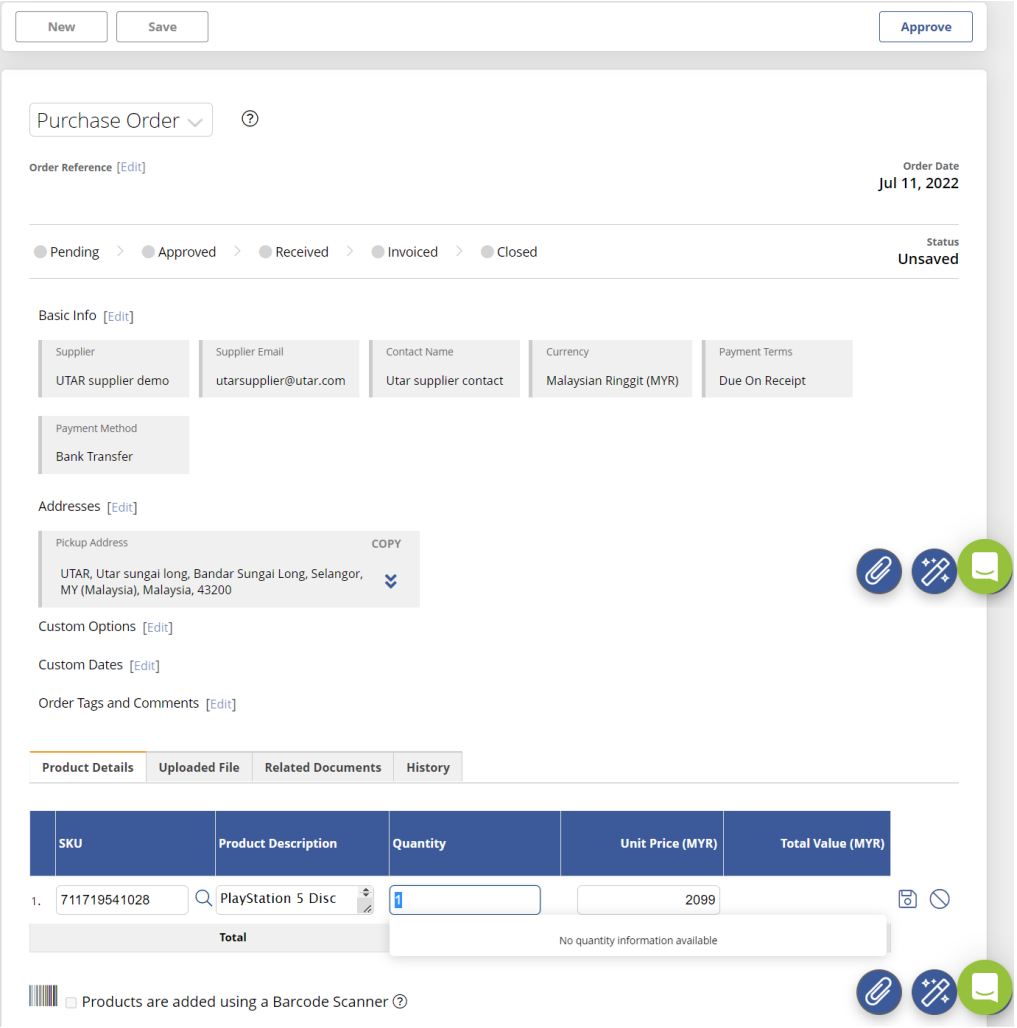


Figure 2.43: Purchase Order form

After all the clients, suppliers and products are successfully created, the business can start creating purchase orders using the form above in Figure 2.43. Like other inventory management systems, the system automatically fills out the supplier detail after the user selects a supplier from the supplier list by searching the supplier's name. Next, the user needs to add items/products from the product list to the purchase order form by searching for the product name, SKU, or barcode. Then, the user needs to fill in the number of products that need to be purchased from the supplier. If the checkbox "Products are added using Barcode Scanner" is checked. In that case, whenever a barcode is scanned using a barcode reader device, the pointer will automatically advance to the next row to add another product to save time. The codes recognised are the product SKU, the product barcode, and the supplier SKU of the product.

Company Logo

Jul 11, 2022
Purchase Order #3

Inventory Location	Universiti Tunku Abdul Rahman
Inventory Location Address	UTAR, Jalan Sungai Long, Bandar Sungai Long, 43000 Kajang, Selangor, Bandar Sungai Long, Negeri Sembilan, MY, Malaysia, 43000
Payment Terms	Due On Receipt
Payment Method	Bank Transfer

Company	UTAR
e-mail	hjie30@1utar.my

Supplier	UTAR supplier demo
Pickup Address	UTAR, Utar sungai long, Bandar Sungai Long, Selangor, MY, Malaysia, 43200
Phone	03-12345678
Phone 2	012-3456789
e-mail	utarsupplier@utar.com
Tax ID	321
Supplier Comments	UTAR supplier campus city demo
Contact Name	Utar supplier contact

#	Product Description	Quantity	Unit Price (MYR)	Row Total (MYR)
1	PlayStation 5 Disc	5 Unit(s)	2,099.00	10,495.00
			Quantity (Total)	5
			SubTotal	10,495.00
			Net Total	10,495.00
			Total (MYR)	**10,495.00**

Order Comments:

Extra Comments:

Print this Page | e-mail | Print/E-mail Options | - or - | Cancel

Figure 2.44: Purchase bill

After purchase, the business can generate a purchase bill with all the details, as shown in Figure 2.44 above. In addition, the business can return the product to the supplier for reasons like receiving the wrong product. Besides, the system will prevent the user from returning more than the quantity purchased.

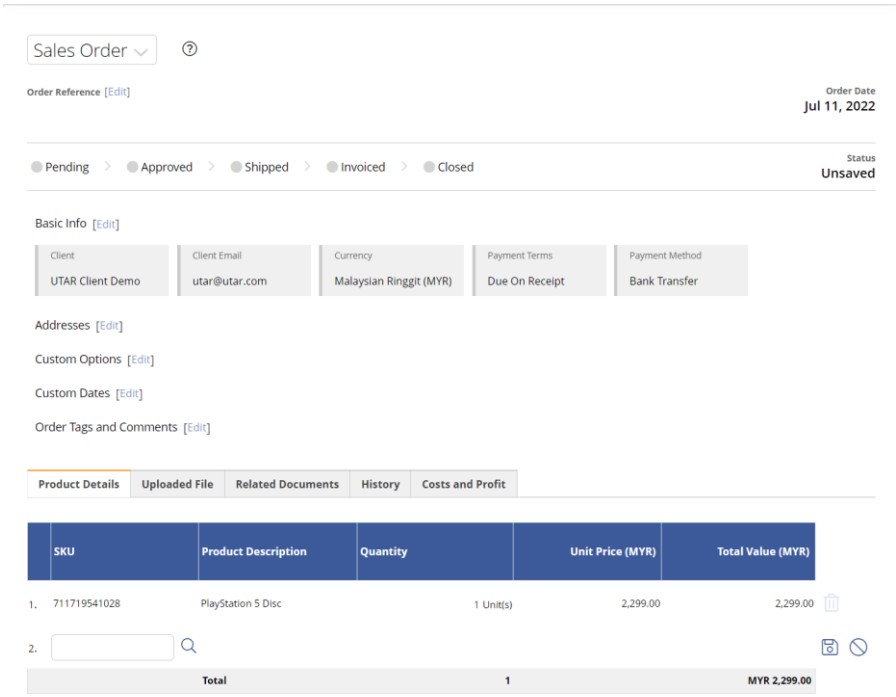


Figure 2.45: Sales Order form

Businesses can create sales orders using the form in Figure 2.45. After completing a sales order, the business can generate shipping consignment and sales invoices. In addition, if the quantity to be shipped exceeds the available quantity (quantity in hand), the system will prevent the business from completing the sales order.

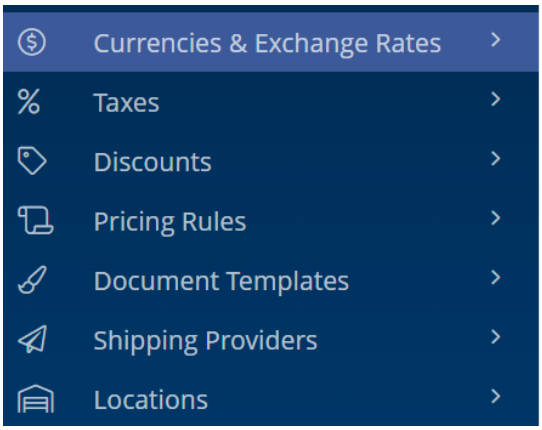


Figure 2.46: Minor features of the system

Furthermore, the system allows businesses to add different currencies and set up taxes, discounts, pricing rules, document templates, shipping

providers, multi-user with different permission and multiple warehouses with different locations.

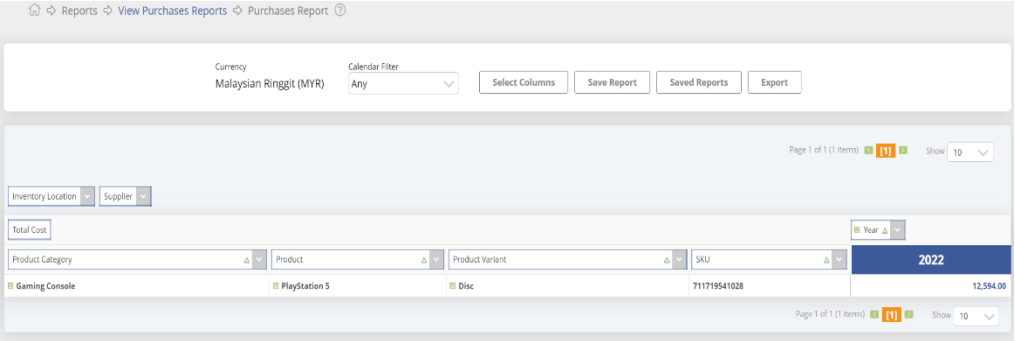


Figure 2.47: Purchases report

Megaventory systems can generate different types of reports: purchase reports, sales reports, product movement reports, availability levels reports, inventory value reports, inventory ageing reports, quantity tracking and unit cost tracking and production report for the manufacturing sector. An example of a purchase report is shown above.

Megaventory System has a significant drawback because it is only accessible on the web platform. Furthermore, unlike inFlow Inventory and Zoho Inventory, it does not offer a mobile application for Android and iOS.

2.2.4 Katana

Katana is an inventory management and production system for small and medium-sized businesses. Its graphical user interface and smart auto-booking engine enable business process automation, order prioritisation, and real-time monitoring of the availability of raw materials and completed products. Besides, the Katana software provides open API. An extensive range of native and third-party connections allow businesses or manufacturers to see the whole business from a single, centralised point of truth. In addition, the software has a mobile web application known as the Shop Floor Control App that provides information on floor-level operations. However, this web-based Shop Floor Control application will not be demonstrated in this section since it is unavailable in the trial version and requires a paid subscription.

Katana inventory management system is like the Zoho Inventory, inFlow Inventory and Megaventory management system but has slightly different designs and features. This system has fewer features than the other three systems. In this section, only a brief discussion of the Katana inventory system will be discussed due to duplicated features, but unique features will be further discussed.

Katana is similar to all the systems previously mentioned. Before using the Katana inventory management system, the user must add contacts for suppliers and clients to allow items/products to be transferred, sold, or purchased.

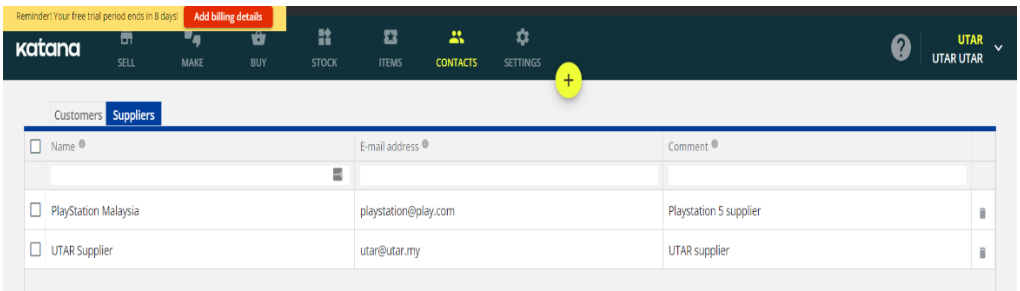


Figure 2.48: Supplier list

After all the clients or suppliers are added to the system, the user can view the list of clients or suppliers, as shown in the figure above. Two of these lists may later use for the specific product to easily make purchases and sales orders easily.

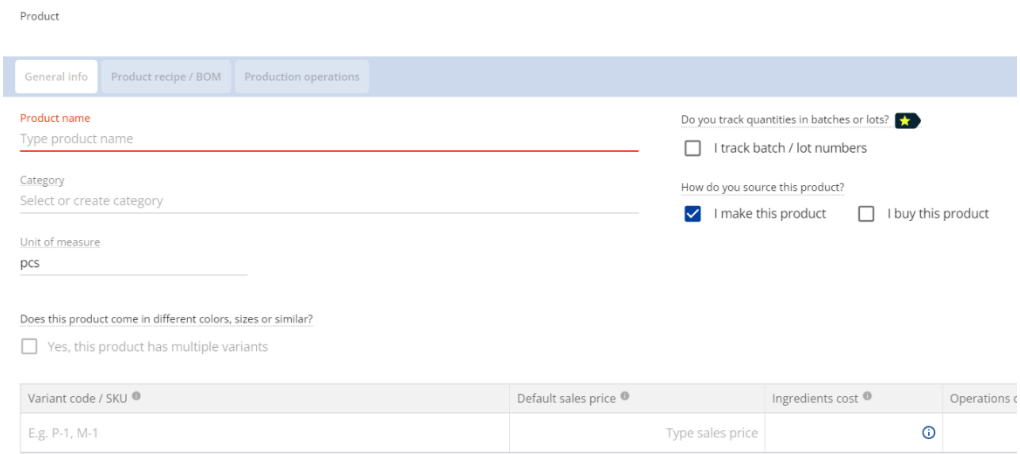


Figure 2.49: Form for adding products

After the suppliers or clients are successfully added, the business can add products/items into the system. Figure 2.49, shown above, is the form for adding a product, the required details are similar to other inventory management systems, but this form has an additional checkbox, "I make this product", which is a function of the manufacturing process. If the checkbox is clicked, the business can create new manufacturing orders and manage the manufacturing process.

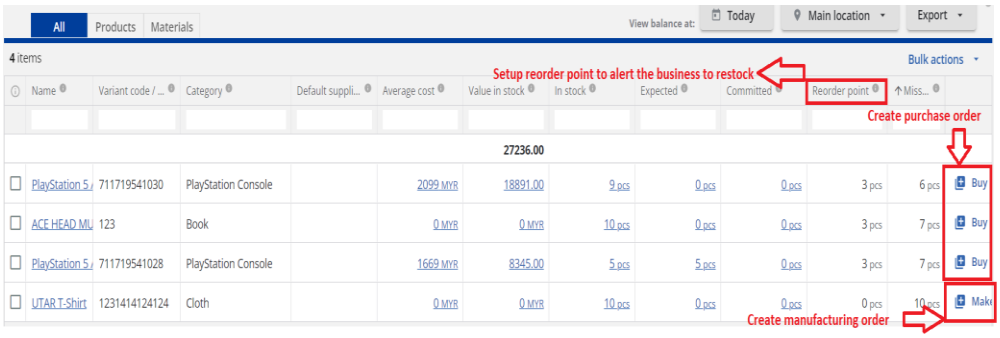


Figure 2.50: Product list

The products created will be added to the product list, as shown in Figure 2.50 above. Next, the business can create a purchase or manufacturing order for these products. As shown in the figure above, the "Buy" button creates the purchase order from the supplier, while the "Make" button creates manufacturing orders. In addition, the "Reorder point" is a specific level at which the stock is low and needs replenishment. The system will alert the business when the stock level is low.

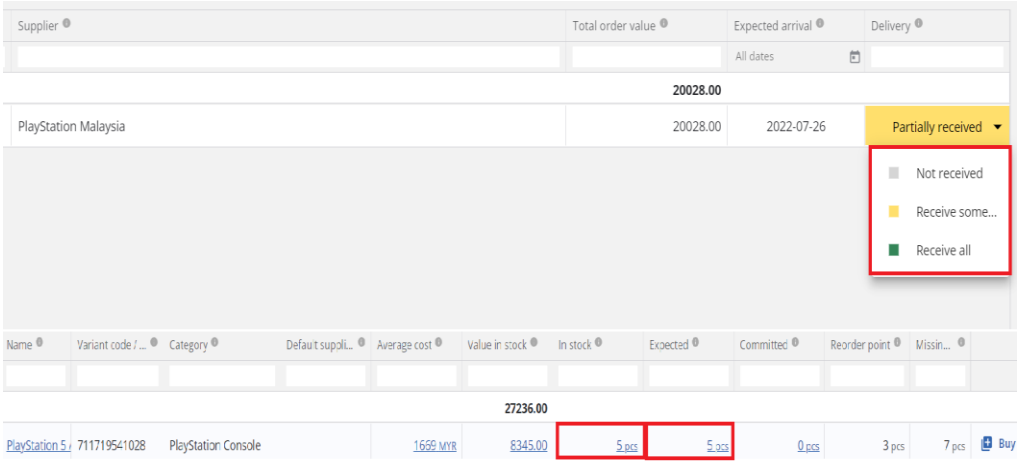


Figure 2.51: Example of receiving product partially

After the business user has created a purchase order, the business will wait for the supplier to deliver the item. The supplier may deliver only a portion of the purchase order but not all of it. In this case, the system allows businesses to record that the shipment is partially received, and the quantity in hand will also be updated. As shown in Figure 2.51 above, the business ordered ten units of a product, but the warehouse only received five units, and another five units are expected from the supplier.

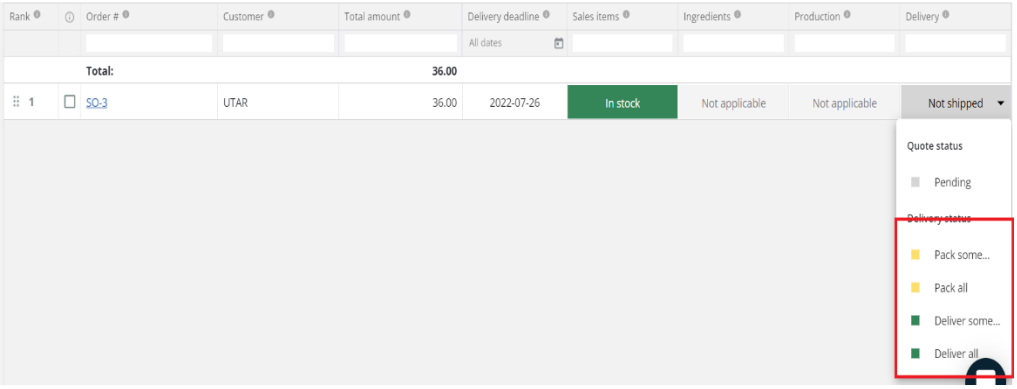


Figure 2.52: Delivery status of a sales order

When the business has all the products ready, the business can proceed to create sales orders. It is similar to the purchase order, but in contrast, this is a sale instead. Besides, the business can pack the item ordered by the client in a partial or complete package. Same for the delivery, the business can deliver some or all the products. In addition, the business can

view the Sales Item's availability ("In stock" green rectangle). In this case, it is shown as "In Stock, " meaning available.

SO-4	UTAR	1 pcs	2022-07-26	Not available
SO-4	UTAR	9 pcs	2022-07-26	In stock

Figure 2.53: Not available status for a sales order

When a business attempts to make a sales order that exceeds the available quantity, it is displayed as "Not available" in red. For example, as shown in the figure above, the business creates a sales order with ten product units (10 pieces). However, the warehouse only has nine pieces in stock, and one piece is not unavailable.

SO-4	UTAR	9 pcs	2022-07-26	In stock
SO-4	UTAR	1 pcs	2022-07-26	Expected 2022-07-26

Figure 2.54: Expected status for a sales order

When a business attempts to make a sales order that exceeds the available quantity, but more inventory is expected to arrive soon from the supplier, the status will be shown as "Expected" in yellow. As shown in the sample Figure 2.54 above, the business creates a sales order with ten product units, but the warehouse only has nine units in stock, and another unit is expected to arrive soon.

Sales order: SO-5

Created: 2022-07-12

Customer:
UTAR

Delivery deadline:
2022-07-26

Bill to:
UTAR , Universiti Tunku Abdul Rahman
Jalan Sungai Long, Bandar Sungai Long, 43000 Kajang, Selangor

Ph. no: 012-12312313

Ship to:
UTAR , Universiti Tunku Abdul Rahman
Jalan Sungai Long, Bandar Sungai Long, 43000 Kajang, Selangor

Ph. no: 012-12312313

	Item	Quantity	Price per unit	Total price	Tax
1	[711719541028] PlayStation 5 / Disc	1 pcs	1869.00 MYR	1869.00 MYR	20% - VAT [DEMO]

Total not shipped (with tax): 2242.80 MYR

Total units:	1 pcs
Subtotal:	1869.00 MYR
Plus tax:	373.80 MYR
Total:	**2242.80** MYR

Figure 2.55: Sales order invoice

The Katana system supports invoice management, allowing users to produce sales order invoices, packing lists and barcode labels. Besides, the business can customise or create a new print template for the sales order. An example of a sales order invoice generated by the system in the PDF version is shown in Figure 2.55 above.

Furthermore, the Katana system is similar to other systems that allow businesses to add different currencies, set up taxes, categories, pricing rules, document templates, multi-user, data import/export, API integrations, barcode scanning, and multiple warehouses with different locations.

2.2.5 Sortly

Sortly is a cloud-based inventory management software suitable for small and medium-sized enterprises (SMEs) in various sectors. The essential features of Sortly are activity monitoring, location tracking, inventory management, barcoding, and audit trails. Sortly users can add tags or labels to items, making it easier to search and track items based on their quantity, price or item details. Users of Sortly may also scan UPC, ISBN, and EAN barcodes to identify specific products and create customisable QR tags for their items. Some unique features are the ability for users to import CSV files and send reminders for expired items, return dates, or warranty expiration dates. The system has mobile applications for iOS and Android smartphones.

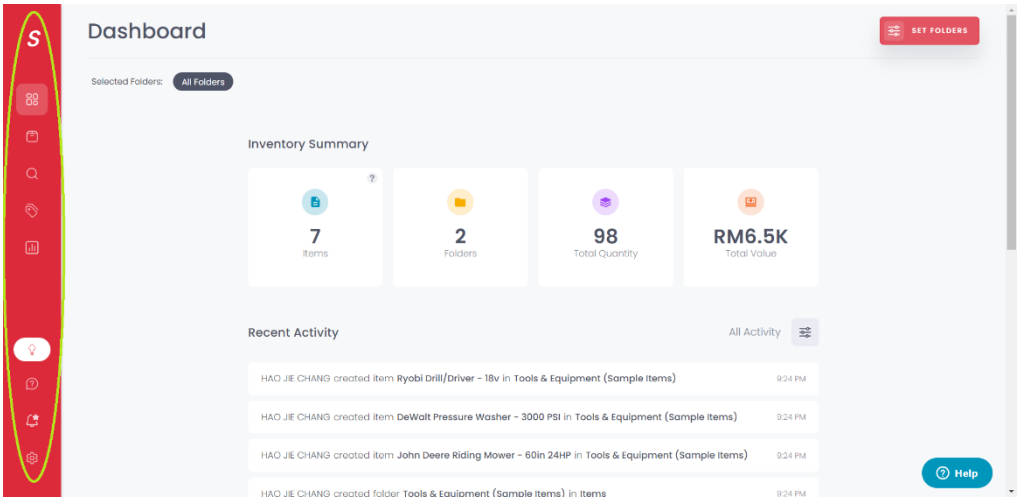


Figure 2.56: Home screen (Dashboard)

As the user navigates to the home screen, the user can view a dashboard that shows the inventory summary, recent activity, recent items and inventory levels. Besides, a vertical navigation bar (circled in blue) on the left of the dashboard enables users to navigate to different screens to access other operational functionality.

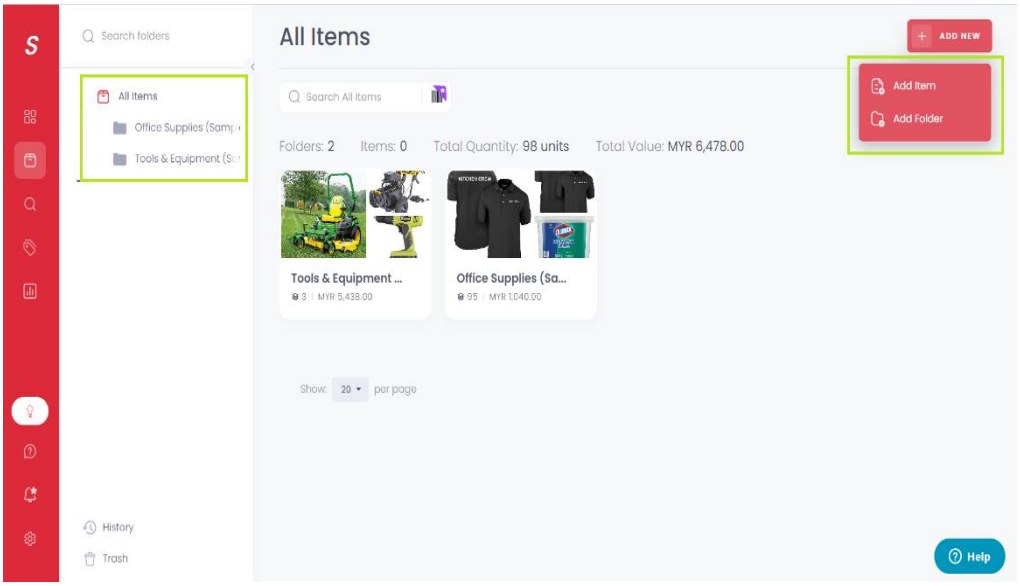


Figure 2.57: Item screen

Sortly allows users to add products/items, which is the essential function of any inventory management system similar to the other inventory management system such as Zoho Inventory and Megaventory. The second screen is the "item screen", allowing users to add items singly or add items to a

folder to group similar items or group them into the same category for easier management. When the user clicks on the "Add item" button, the user will be brought to the form for adding a product. In the form, the user may fill in the item quantity, minimum level of stock, price, tags, notes, QR code/barcode, and variants of the item/product and upload the product image. The user can specify the minimum stock level to detect items with low supply. When the quantity of an item is at or below the minimum threshold, it will be highlighted. Users will get an alert on low supply, but this alert feature is limited to paid advanced plan users. In addition, users can link existing QR codes or barcodes to the product using a scanner. When the product has no existing QR code or barcode, the user can create a unique QR code or barcode for the product.

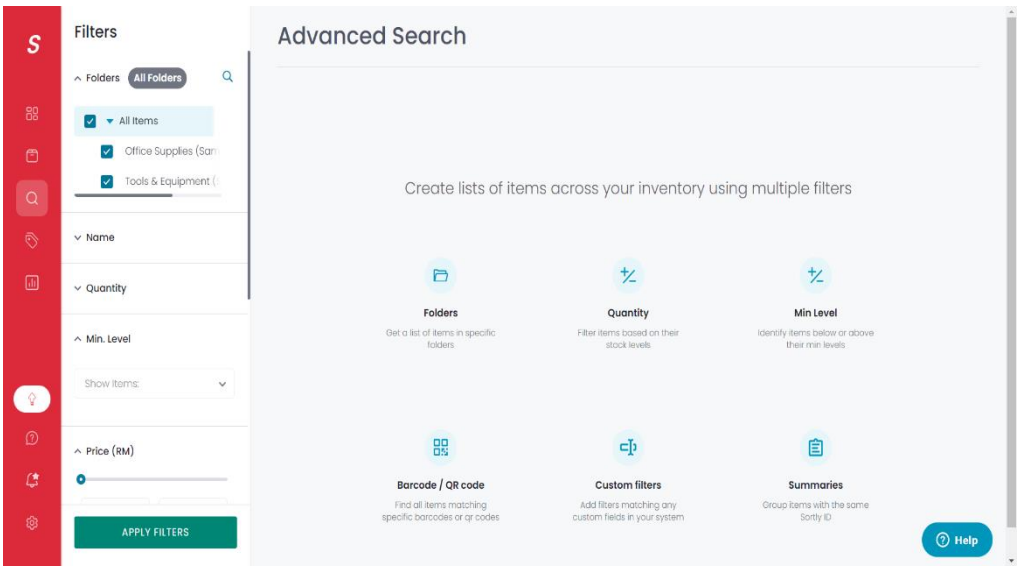


Figure 2.58: Advanced Search Engine

One unique feature of Sortly is the advanced search engine. Users can search for objects using the filters in the advanced search engine. The advanced search engine enables users to search for objects in a specified folder. In addition, users can search using filters: name, quantity, level of stock, price, quantity alerts, date alerts, tags, Sortly ID (SID), barcode/QR code and notes. These filters in the advanced search engine allow users to search for any object easily.

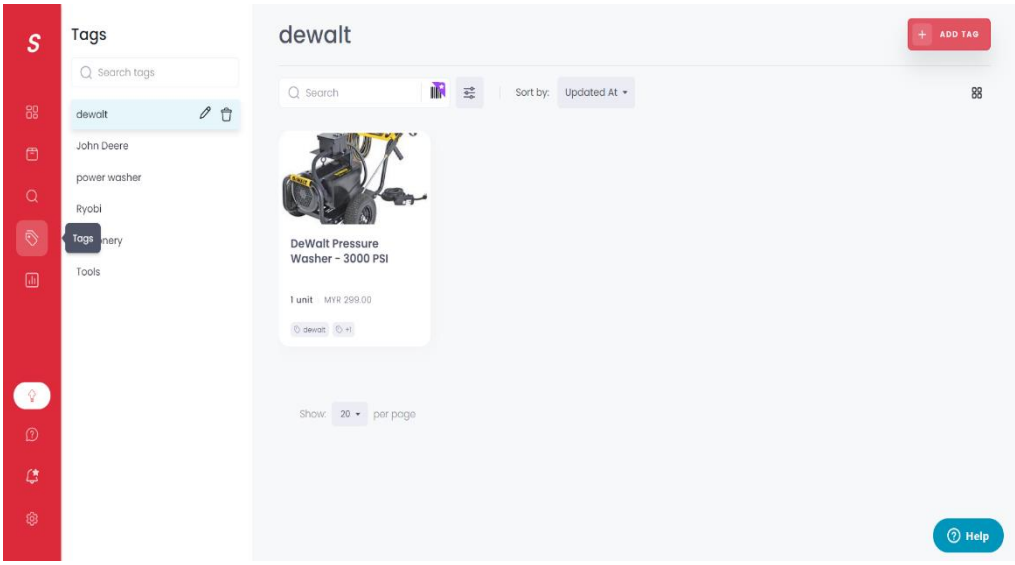


Figure 2.59: Tags screen

Another feature in Sortly is Tags. Users will have another method to classify the products using tags, making searching for products within the inventory simpler. For example, folders can be created based on categories to group items, but if a user wants to see items based on "brand" or "colour," they may add them as tags and see the items in the "Tags" section of the application.

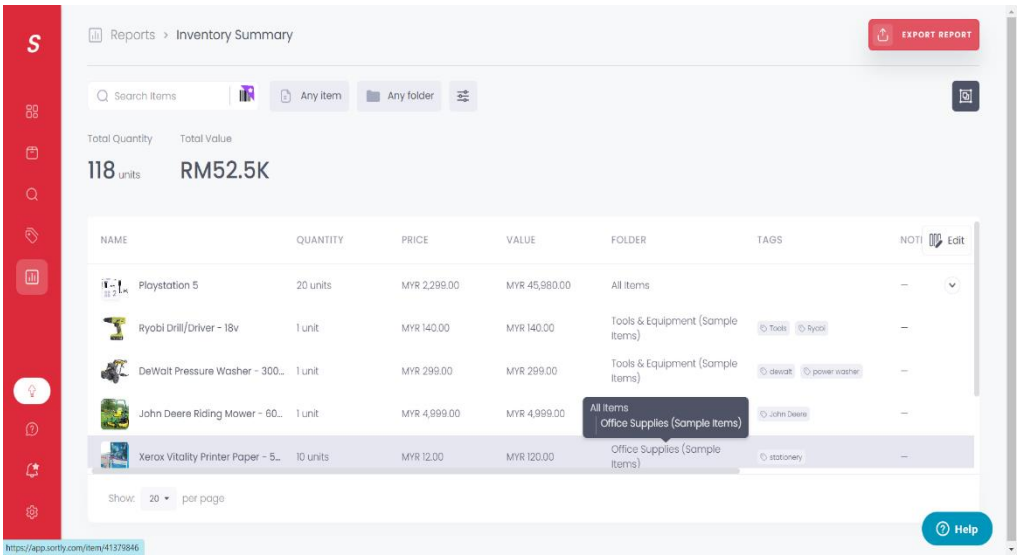


Figure 2.60: Inventory summary report

Sortly allows users to generate reports based on inventory summary, low stock, quantity changes by item, move summary and transaction. For example, the inventory summary report is shown above.

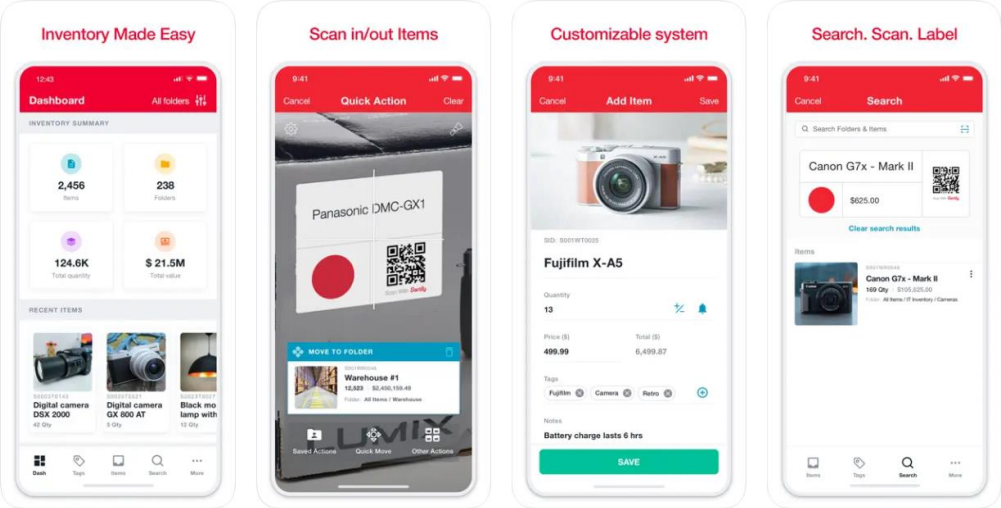


Figure 2.61: Mobile application of Sortly System on the iOS platform

The Sortly system has a mobile application supported on iOS and Android platforms. All the features of mobile applications are similar to the web platform.

2.2.6 Comparison of Features between Similar Inventory Management Systems

Table 2.1: Comparison of features between Similar Inventory Management System

	Zoho Inventory	inFlow Inventory	Megaventory	Katana Inventory	Sortly Inventory
Platform supported	Mobile (Android & iOS) and web	Mobile (Android & iOS), Windows App and Web	Web	Web, mobile web-based app (Required subscription)	Web and Mobile (Android & iOS)
Activity Dashboard	✓				✓
Barcode Recognition	✓	✓	✓	✓ (Required subscription)	✓
Multi-Currency	✓	✓	✓	✓	✓
Warehouse Management	✓	✓	✓	✓	
Supplier Management	✓	✓	✓	✓	
Customer/Client Management	✓	✓	✓	✓	
Shipping Management	✓	✓	✓		
Return Management	✓	✓	✓		
Item Management	✓	✓	✓	✓	✓
Invoice Management	✓	✓	✓	✓	
Real-time Updates	✓	✓	✓	✓	✓
Third-Party Integrations (API)	✓	✓	✓	✓	✓
Alerts/Notifications	✓	✓	✓	✓	✓
Reporting/Analytics	✓	✓	✓		✓
Multi-user	✓	✓	✓	✓	✓
Data Import/Export	✓	✓	✓	✓	✓
Tax Management	✓	✓	✓	✓	

2.2.7 Limitation of Existing Inventory Management System

a) Zoho Inventory

The onboarding process of Zoho Inventory is notoriously lacking compared to other inventory software brands, making the Zoho Inventory harder to use. Besides, the inventory management process of the Zoho Inventory is rigid. Zoho assumes that everyone operates in the same manner. Therefore, the process can be more customisable to suit different business processes.

b) inFlow Inventory

The report features in the system are adequate to present information on various aspects of the business, but there is not much customisation available for reports. Reporting is critical in assisting management in decision-making, so it is vital to have a customisable report that can be used for different purposes. In addition, since inFlow primarily focuses on the retail side of inventories, it provides relatively few supply chain management functions. Therefore, businesses have less control over how their goods are transported, which some business owners find intolerable. However, these limitations are a little nit-picky since it is hard to identify anything substantially wrong with the platform, even though these inFlow weaknesses can be a deal breaker for some companies.

c) Megaventory

The limitation of Megaventory is the lack of mobile applications for Android and iOS. Sometimes it is troublesome to update the inventory status when the user does not have access to the web application or a computer. In addition, the search filter of the system lacked a partial match search feature, making it harder for users to search for what they were looking for, and they had to type in the whole spelling to find it. Besides, for the user with a smaller display is challenging to navigate through the dashboard.

d) Katana

The Katana system has several disadvantages because it is only accessible on the web and is not mobile-friendly. For example, unlike inFlow

Inventory and Zoho Inventory, it does not offer a mobile application for Android and iOS. Besides, the Katana system does not offer return or shipping management and cannot generate reports and perform analytics.

e) Sortly

Compared with other systems, the Sortly system lacks many major functions, such as warehouse management, supplier management, customer management, transportation management, return management and tax management. Since the system does not include some basic operational features, it is difficult for business users to use Sortly to help facilitate business processes and improve organisational efficiency.

2.3 Review of Software Development Methodologies

Most software development projects aim to produce software in the least amount of time and with the highest quality. Achieving this goal requires proper planning and control of the development process using the appropriate methodology. Many software development methodologies offer various approaches for getting the desired outcome quickly. These methodologies set up the framework for planning, controlling, designing, and developing information systems (Oladele, n.d.). Therefore, selecting a correct software development methodology and implementing it systematically throughout the project is crucial. Besides, there is no perfect development methodology since each methodology has pros and cons. Choosing the optimal methodology for a given project relies on the composition of the team, the objectives, and the requirements. It is also feasible to apply different software development methodologies to different projects (Nikolaieva, n.d.)

2.3.1 Agile Development Methodology

Agile development is one of the most widely used methods in the IT industry (Oladele, n.d.). Agile places less emphasis on paperwork and less rigid procedures but more on satisfying consumers (Nikolaieva, n.d.). The Agile software development methodology focuses primarily on a final product with teamwork (Oladele, n.d.). It involves a short-term software development cycle known as iterations. Every iteration is adequately planned out and represents a

small software project that takes about 1 to 4 weeks to complete. Tasks that include adding new functionality, planning and analysis, designing, coding, testing, and documentation, are included in the iterations. The development team evaluates the project and rearranges the backlog as the number of iterations rises. At the closing of each iteration, it is a good technique that helps to assure product quality and releases with a better version. The agile approach has benefits, but there are also drawbacks. Therefore, it is not a perfect approach for each project (Oladele, n.d.)

2.3.1.1 Advantage of Agile Development Methodology

- Agile produces high-quality products and minimal defects due to the effort of small iterations that involve testing and maintenance.
- This methodology enables innovative alterations and upgrades when developing software with little impact.
- Developers get a chance to explore different code tweaks.
- During the production process, the customer, developers, and other stack holders often interact and communicate to clarify topics transparent, promoting positive working relationships.

2.3.1.2 Disadvantage of Agile Development Methodology

- Overwhelming modification requests might sometimes cause the team to lose focus.
- Agile does not emphasise documentation, which might cause issues later in development.
- Agile demands skilled developers who can work independently because of its unstructured methodology.
- When there are any changes in specifications and needs, it is hard to have a precisely estimated deadline for the project completion resulting in no rigid or fixed deadlines.
- It is challenging to estimate the costs and resources due to unpredictable changes.

2.3.2 Waterfall Development Methodology

The waterfall methodology, introduced in 1970 by Dr Winston W. Royce, is the earliest one used in the IT sector. It is a well-known and traditional software engineering system development life cycle method. For each stage of development, the objectives are already specified (Oladele, n.d.). The waterfall methodology remains applicable in certain projects, even though it was initially utilised decades ago. It is a straightforward, linear process in which development steps are structured into sequential, cascading processes. The waterfall development process is quite well for teams with no prior design experience since it is simple to understand (Nikolaieva, n.d.) However, as the name given Waterfall, which only flows in one direction. The Waterfall is rigid because it limits making reversible adjustments based on new requirements after a stage is finished (Oladele, n.d.).

2.3.2.1 Advantage of Waterfall Development Methodology

- The stages of the waterfall model make it simple to understand, especially for novice developers, because it is straightforward, requiring little or no prior knowledge.
- Since all requirements and deliverables are laid down before development begins, it is a good strategy for small projects.
- The waterfall approach eliminates the possibility of miscommunicating information since each step is explicitly defined.

2.3.2.2 Disadvantage of Waterfall Development Methodology

- Early client feedback is not considered, which raises the possibility of the project deviating from its original course.
- Since testing is only carried out once development is complete, specific issues like bugs become more difficult to fix as the project progresses.
- The team may spend too much time documenting rather than providing solutions that address the issues of the users.
- Not suitable for projects that often change as they go.

- It is ideal for short- and medium-term software projects but not long-term or R&D projects.
- Once a project enters the testing phase, there can be no further revisions or alterations.
- It functions best when only clearly specified requirements are known in advance.

2.3.3 Prototyping Development Methodology

This prototype methodology enables developers to work on the prototype version of the final product rather than creating fully functional software. After that, the prototype is made accessible to the client for testing, evaluation, and feedback (Nikolaieva, n.d.). The advantage of the prototype technique is that it covers all software engineering process issues (Oladele, n.d.).

The prototype is refined through multiple iterations until the client is satisfied based on the feedback obtained. The benefit of the prototype methodology is its thorough analysis, which identifies potential problems before the actual development starts.

The effectiveness of this strategy depends not only on the development team or individuals but also on how effectively they interact with the test participants. It is also important to note that the developers often bear the expense of creating the prototype (Nikolaieva, n.d.)

2.3.3.1 Advantage of Prototyping Development Methodology

- This methodology allows the early phases of development to identify risks potential problems and remedy mistakes to significantly lower the likelihood of product failure.
- Ensures the user is satisfied with the prototype before the commencement of actual development work.
- Working on a prototype allows developers and testers to quickly adapt it to the client's expectations, check that they are on target, and make the necessary adjustment.
- The frequent communication that results from this methodology strengthens the link between the client and developer.

- In the absence of the document, obtaining and analysing requirements become simpler.

2.3.3.2 Disadvantage of Prototyping Development Methodology

- The development schedule may be slowed down by excessive back-and-forth prototype testing with the client.
- The prototype may not match the expectations of the client for the finished product.
- No initial product is ready for the market unless the final product is developed.

2.3.4 Comparison of Software Development Methodologies

Table 2.2: Comparison of Software Development Methodologies

Criteria / Methodologies	Agile	Waterfall	Prototype
Client involvement	Throughout the project	Only involved at the beginning	Throughout the project
Requirement specification	Keep on changing	Only specified in the first phase of the lifecycle.	Identified throughout a separate process by using prototypes
Development Process	Phases follow in the cycle	works in a sequential method	Linear: one phase after another
Development Pace	Fast	Slow	Slow
Documentation dependence	Low	High	Moderate
System failure risk	Low	Comparatively high	Relatively low
Developer experience	High	High	Medium
Flexibility to change	High	Low	High
Testing	Performed concurrently with the development	Performed after the build phase	Performed after the final system is developed
Project Timeline	Adapts to the progression of a project	Fixed timeline	No fixed timeline
Cost	Low	Moderate	High

In short, after reviewing different methodologies, the Prototyping Development Methodology is chosen as the methodology to be used in this project. The prototype is chosen because it is suitable for one personal development project. The prototype is the most effective method of introducing and demonstrating the software product to the user or client. After prototype demonstration, the developer can gather valuable feedback from the user, client, or testers at the early stage of the development cycle since issues become more challenging to fix as the project development starts. The waterfall is not chosen because risk and uncertainty are high and flexibility to change is low. Agile methodology is not selected because the development pace is fast and requires an experienced developer who can work independently.

2.3.5 Conclusion of Software Development Methodologies

Every project must use the appropriate software development methodologies. Development methodologies guide how to create software and applications. In addition, software development methodologies may speed up the development process and provide the best outcomes. All software development methodologies prove to be more effective for specific project types. Depending on the chosen methodology, different complexity degrees and expert skills will be needed. Developers must evaluate all the advantages and disadvantages of methodologies since no method is entirely perfect. The project cost, scope, availability of resources, time frame, objective and preferences are important factors to consider (Oladele, n.d.).

CHAPTER 3

METHODOLOGY AND WORK PLAN

3.1 Introduction

The chapter discusses the software development methodology and project planning for this project. The chapter will include the work breakdown structure (WBS), project plan, Gantt chart and discussion on the development tools used for developing the system.

3.2 Software Development Methodology

After comparing several popular software development methodologies in the literature review, evolutionary prototyping was selected as the software development methodology for this project because it is the most suitable methodology for developing this project.

3.2.1 Evolutionary Prototyping

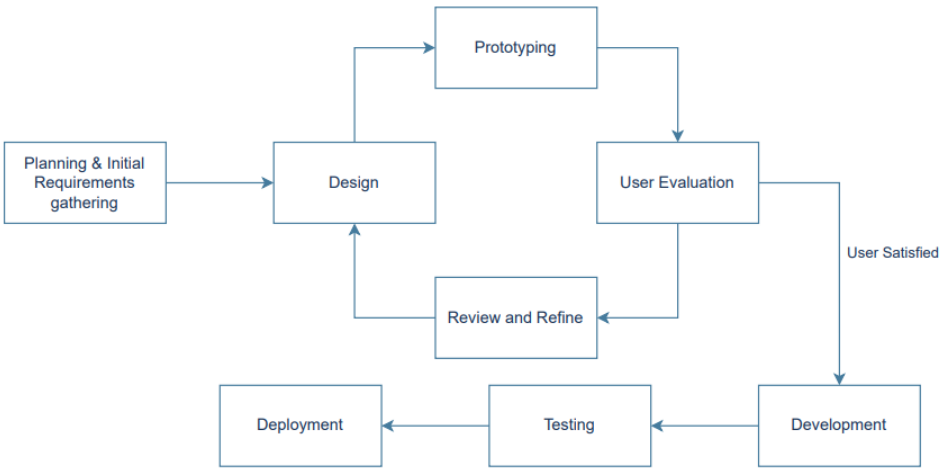


Figure 3.1: Overview of the Software Development Methodology

Evolutionary prototyping is a software development methodology that involves planning and initial requirement gatherings to build the prototype. The first prototype will be sent to the project supervisor for evaluation to obtain initial feedback from the supervisor to confirm that the system's requirements and functions to be developed are correct and on track. The

development of subsequent prototypes, each with refinement or added functionality, and sent for evaluation again for further refinement. The system is continually refined and rebuilt until the requirement stated is met and the supervisor is satisfied with the system. For example, users interacting with the system discover opportunities for additional features and submit requests for those features to the developer. When developers get these requests for improvements, they modify the prototype design and update the software requirements specification.

Evolutionary prototyping is appropriate for projects where all requirements are unclear and builds those already clear first. As a result, developers can focus on certain parts of the system that they are familiar with in each iteration rather than focusing on a complete system. In addition, evolutionary prototyping minimises project risk because developers do not implement features that are not understood. In addition, this approach allows developers to add features or make changes that could not have been anticipated during the requirements and design phases.

A. Planning and Initial Requirement Gathering

The first step in evolutionary prototyping methodology is planning and gathering initial requirements. The purpose of planning is to determine the project objectives, project scope, project schedule, and project constraints. Project planning begins with the initial requirements gathering, determining how the software will be developed. Initial requirements are gathered by reviewing similar existing inventory management systems and from the project. The review of similar existing systems is a useful technique in requirement gathering. The essential feature of the existing system will be identified. In addition, any additional features not provided in the existing system are also obtained through the project manager. The next step is to develop a project plan outlining the tasks needed to finish the project. Work breakdown structure (WBS) and Gantt chart will aid project planning. The primary goal of WBS is to break down complex tasks into a collection of smaller sub-tasks. The Gantt chart provides a common format for illustrating information about project schedules using graphical depiction by listing

project activities (activities in WBS) and their corresponding start and end dates in a calendar format.

B. Design

In the design phase, the Unified Modelling Language (UML) diagram is created to enable the project supervisor to get a visualisation of the design of the inventory management system. The use case diagram is created to visualise the entire system, and the use case descriptions are created for each function in the system. The UML diagram was created using the gathered requirements in the first phase.

C. Prototyping

In the prototyping phase, the prototype of the inventory management system will be developed based on the initial gathered requirements. The purpose of a prototype software model with limited functionality is to enable the user to understand the functionalities that will be included in the system. Besides, the user could evaluate the prototype system instead of evaluating the design of the system based on just word descriptions. In addition, the prototype enables the user to interact with the system to get an overview of how the system should work and what the interface should look like.

D. User Evaluation

The prototype created in the previous phase will be presented to users for evaluation. After the evaluation, the feedback and comment obtained from the user will be recorded. User feedback and comments are valuable information that can be helpful to developers in improving the prototype since feedback may reveal requirements that are not met. Potential problems in the prototype will also be identified during the evaluation phase, especially business domain.

E. Review and Refine

The feedback will be analysed to refine and improve the prototype through multiple iterations. Iteration will be repeated until the user is satisfied with the prototype and all the system requirements are met. After the user accepts the prototype design, the design will be developed into the complete system.

F. Development

The actual system will be developed using the design from the prototyping.

G. Testing

The testing will be conducted after the development of the system is completed. Types of testing that will be conducted are unit testing, integration testing, system testing and user acceptance testing. Software testing aims to verify the system is developed with minimal bugs, that it satisfies the technical specifications as determined by its design and development, and that it effectively and efficiently satisfies the user's requirement

H. Deployment

The system is prepared for deployment after completing and passing every test performed during the testing process.

3.3 Project Plan

3.3.1 Work Breakdown Structure (WBS)

0.0 Inventory Management System for Automotive Parts

1.0 Project Planning & Requirements gathering

- 1.1. Preliminary planning
  - 1.1.1. Study background of the project problem
  - 1.1.2. Define problem statements
  - 1.1.3. Define project objectives
  - 1.1.4. Define project proposed solution
  - 1.1.5. Define project proposed approach
  - 1.1.6. Define project scope
- 1.2. Literature review
  - 1.2.1. Review existing similar inventory management system
  - 1.2.2. Review software development methodologies
- 1.3. Methodology and work plan
  - 1.3.1. Identify suitable software development methodology

1.3.2. Determine work plan

1.3.2.1. Create a work breakdown structure

1.3.2.2. Create a Gantt Chart

1.3.3. Select development tools

1.4. User Design

1.4.1. Requirement specification

1.4.1.1. Identify functional requirement

1.4.1.2. Identify non-functional requirement

1.4.2. Create UML diagram

1.4.2.1. Develop a use case diagram

1.4.2.2. Define use case descriptions

2.0 Development

2.1. First Iteration

2.1.1. Design user interface

2.1.2. Build low-fidelity prototype

2.1.2.1. Build a prototype for the web-based management system

2.1.2.2. Develop a prototype for the mobile-based management system

2.1.3. Evaluation and gathering feedback

2.1.4. Refine prototype

2.2. Second Iteration

2.2.1. Design

2.2.1.1. System architecture design

2.2.1.2. Database design

2.2.2. Prototyping

2.2.2.1. Develop CRUD operations

2.2.2.2. Develop essential features

2.2.2.3. Create database

2.2.3. Evaluation and gathering feedback

2.2.4. Refine prototype

2.3. Third Iteration

2.3.1. Functionality design

2.3.2. Web application prototyping

2.3.2.1. User Authentication and Authorisation

2.3.2.2. Profile management

2.3.2.3. User management

2.3.2.4. Role/Access management

2.3.2.5. Page access management

2.3.2.6. Inventory item management

2.3.2.7. Dashboard

2.3.3. Mobile application prototyping

2.3.3.1. User Authentication and Authorisation

2.3.3.2. Profile viewing

2.3.3.3. Inventory item viewing

2.3.3.4. Inventory item searching

2.3.3.5. QR code scanning

2.3.3.6. Dashboard

2.3.4. Evaluation and gathering feedback

2.3.5. Refine prototype

3.0 Testing

3.1. Unit testing

3.2. System Usability Testing

4.0 Deployment

4.1. System deployment

3.3.2 Gantt Chart

Inventory Management System for Automotive Parts

Project Start: Monday, 13 June, 2022

Display Week: 1

TASK	Duration(Days)	START	END
Project Planning & Requirements Gathering			
Preliminary planning	21	13/06/22	03/07/22
Study background of the project problem	4	13/6/2022	16/06/22
Define problem statements	5	17/06/22	21/06/22
Define project objectives	3	22/06/22	24/06/22
Define project proposed solution	3	25/06/22	27/06/22
Define project proposed approach	3	28/06/22	30/06/22
Define project scope	3	01/07/22	03/07/22
Literature review	21	04/07/22	24/07/22
Review existing similar inventory management system	14	04/07/22	17/07/22
Review software development methodologies	7	18/07/22	24/07/22
Methodology and work plan	14	25/07/22	07/08/22
Identify suitable software development methodology	4	25/07/22	28/07/22
Create work breakdown structure	4	29/07/22	01/08/22
Create Gantt Chart	4	02/08/22	05/08/22
Select development tools	2	06/08/22	07/08/22
Project and design specification	14	08/08/22	21/08/22
Identify functional requirement	3	08/08/22	10/08/22
Identify non-functional requirement	3	11/08/22	13/08/22
Develop use case diagram	3	14/08/22	16/08/22
Define use case descriptions	5	17/08/22	21/08/22

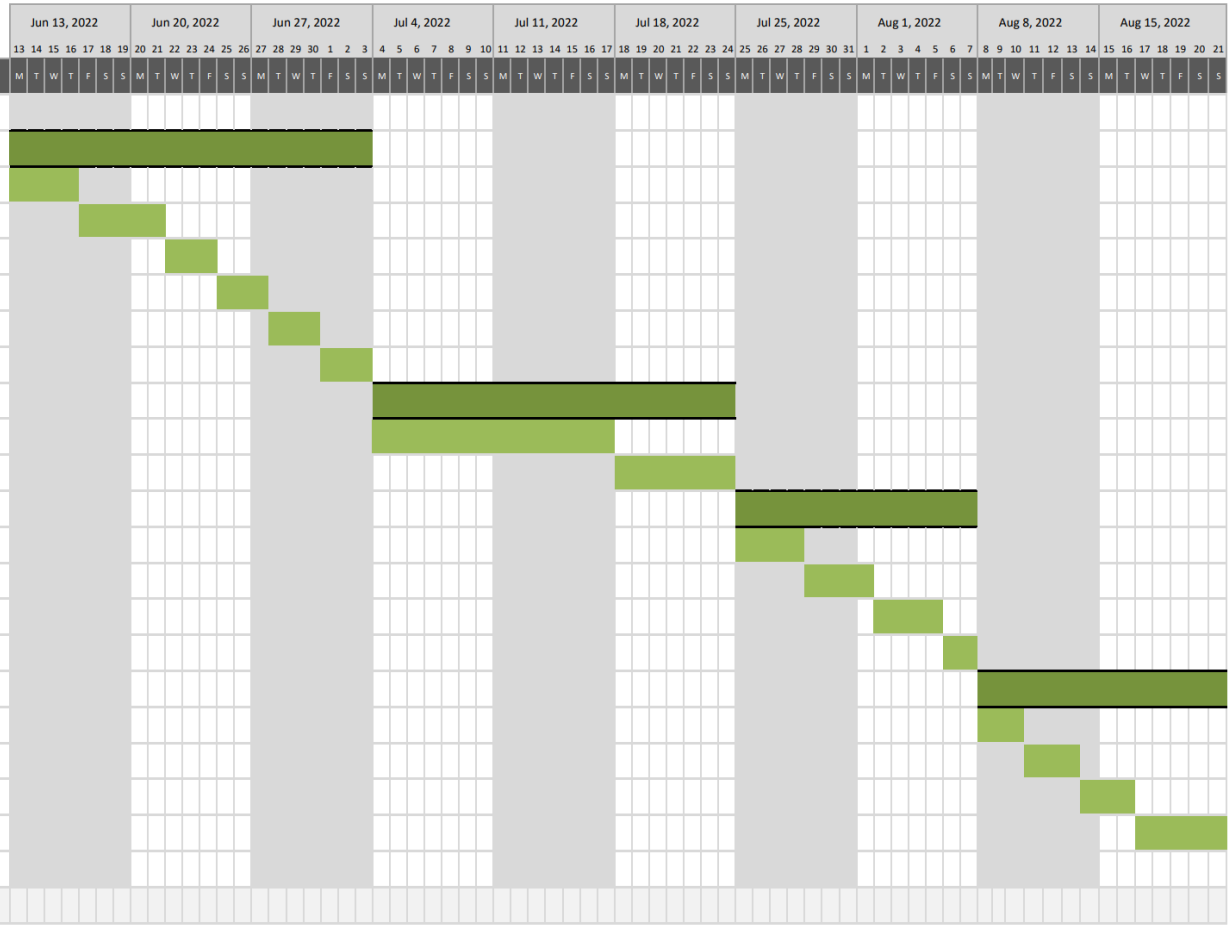


Figure 3.2: Gantt Chart for Project Planning & Requirement Gathering

Display Week: 1

TASK	Duration(Days)	START	END
Development			
First iteration	14	30/01/23	12/02/23
Design user interface	3	30/1/2023	01/02/23
Build low-fidelity prototype for web app and mobile app	7	02/02/23	08/02/23
Evaluation and gathering feedback	1	09/02/23	09/02/23
Refine prototype	3	10/02/23	12/02/23
Second iteration	21	13/02/23	05/03/23
System architecture design	3	13/02/23	15/02/23
Database design	2	16/02/23	17/02/23
Develop CRUD operations	4	18/02/23	21/02/23
Develop essetial features	4	22/02/23	25/02/23
Create database	3	26/02/23	28/02/23
Evaluation and gathering feedback	1	01/03/23	01/03/23
Refine prototype	3	02/03/23	04/03/23
Third iteration	36	05/03/23	09/04/23
Functionality design	3	05/03/23	07/03/23
Web application prototyping	17	08/03/23	24/03/23
User Authentication and Authorisation	2	08/03/23	09/03/23
Profile management	2	10/03/23	11/03/23
User management	2	12/03/23	13/03/23
Role/Access management	2	14/03/23	15/03/23
Page Access management	2	16/03/23	17/03/23
Inventory item management	5	18/03/23	22/03/23
Dashboard	2	23/03/23	24/03/23
Mobile application prototyping	11	25/03/23	04/04/23
User Authentication and Authorisation	2	25/03/23	26/03/23
Profile viewing	1	27/03/23	27/03/23
Inventory item viewing	2	28/03/23	29/03/23
Inventory item searching	2	30/03/23	31/03/23
QR code scanning	2	01/04/23	02/04/23
Dashboard	2	03/04/23	04/04/23
Evaluation and gathering feedback	2	05/04/23	06/04/23
Refine prototype	3	07/04/23	09/04/23

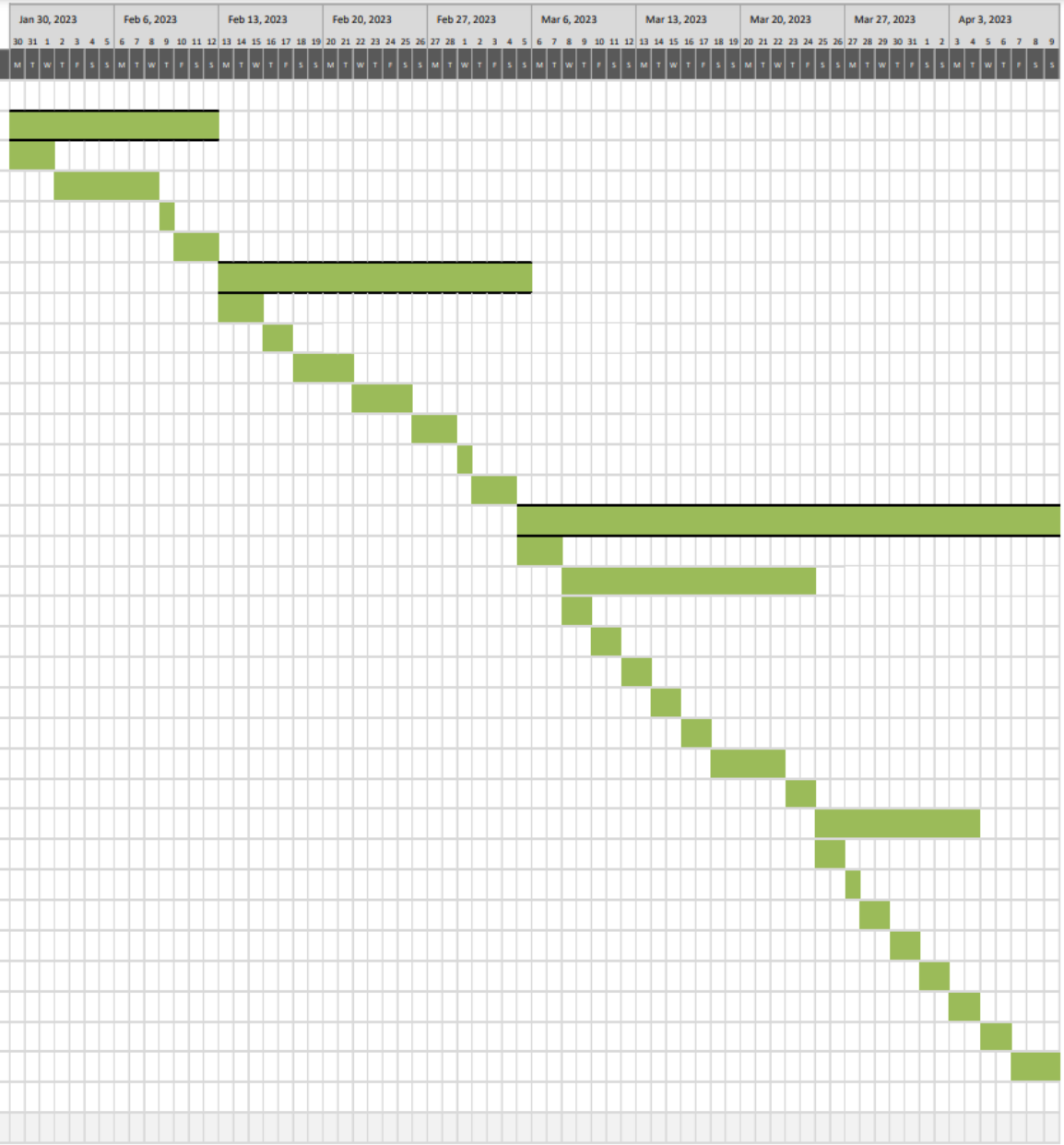


Figure 3.3: Gantt Chart for Development

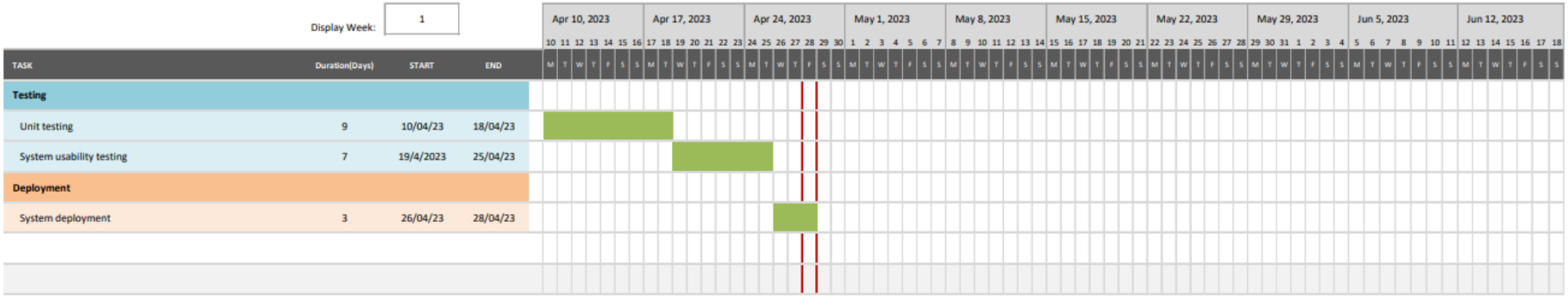


Figure 3.4: Gantt Chart for Testing & Deployment

3.4 Development Tool

The proposed inventory management system has a web application for users to manage inventory and a mobile application to check inventory status and details. Therefore, mobile and web development tools are used in this project to aid the development process. The development tools involved are the framework, development platform, integrated development environment (IDE), and emulator.

3.4.1 React Native

React Native is an open-source user interface (UI) framework for creating inventory management mobile applications for this project on the Android platform. React Native combines parts of native development with React. Besides, React Native includes JavaScript library for building user interfaces, and JavaScript serves as the foundational language for React Native. React Native has a high adoption rate and a sizable community due to the popularity of JavaScript. Additionally, React Native support a wide range of plugins and packages from third parties to expand the functionality of mobile applications, such as open-source JavaScript libraries that can be obtained on NPM. React Native has a fast refresh functionality that lets developers get instant feedback for changes made to the code, such as UI changes.

3.4.2 Visual Studio Code

Visual Studio Code is a source code editor for writing source code for React Native and React framework. In the project, visual studio code is used to create the source code for the system.

3.4.3 Android Studio (Emulator)

The emulator in the Android studio will work with React-Native to stimulate the mobile application on the computer. The Android studio emulator enables the developer to install and run the React Native application on various Android devices and Android API versions.

3.4.4 React JS

React JS is an open-source JavaScript library that creates user interfaces based on UI components. It is supported by Meta (formerly Facebook) and a community of independent developers and organisations. React is an effective, declarative, and extensible framework that can be used to create web frontends web applications that are simple, quick, and scalable (Pisuwala, 2022).

3.4.5 Firebase

Firebase is a platform with cloud computing and development tools that help develop web and mobile applications. It is supported and developed by Google. Firebase is a development platform with many APIs that allow developers to build, improve, and grow applications. Firestore is one product from Firebase that provides developers with a real-time document-oriented database that makes synchronisation of application data across mobile applications and web applications easier, such as iOS, Android, and devices with web browsers. In addition, Firebase also included Firebase Authentication, which enables the developer to develop their application with secure sign-in and sign-up, so the user of the applications can perform secure sign-in (Stevenson, 2018). Besides, It provides developers with a suite of services to help build and manage their applications, including cloud storage, real-time database, authentication, hosting, and more. Firebase allows developers to focus on building the front end of their applications while the platform handles the back-end infrastructure, scaling, and maintenance. At the time of this writing, Firebase has 17 products, some of which will be used to aid with the development process.

3.4.6 Algolia

Algolia is a search-as-a-service platform that offers a comprehensive search experience for websites and mobile applications, including React-based web application and React Native mobile applications. It enables developers to integrate advanced search functionality into their products with minimal effort, as Algolia handles the underlying infrastructure. Algolia's hosted search API provides real-time search results to users with features such as typo-tolerance, faceting, and autocomplete (Algolia, 2023).

Regarding Firebase, Algolia offers integrations with Firebase to enhance the search capabilities of Firebase-powered applications. Developers can use Algolia's indexing and search features to optimize their Firebase database search experience, making it faster and more user-friendly (Algolia, n.d.).

CHAPTER 4

PROJECT SPECIFICATION

4.1 Introduction

The chapter mainly focuses on the preliminary specification of the project that includes specifying the specification of the system in terms of both non-functional requirements & functional requirements and the use case diagram along with each use case description for the Inventory Management system for Automotive Parts. The use case diagram is a type of diagram used to describe the high-level functionality and scope of the system in the form of a graphical depiction. The use case diagram also depicts the interactions between the system and its actors (end-user). The use case description will describe how the actors (end-user) perform tasks on the system in words.

4.2 Requirement specification

The requirement for the proposed system is gathered and identified through reviewing existing systems similar to the proposed inventory management system mentioned in Chapter 1. In addition, feedback was collected from the project supervisor to refine and improve the requirements for the proposed system. The requirement specification is separated into two types which are non-functional requirements and functional requirements. The functional requirement will be specified for each platform: web-based and mobile applications. The non-functional requirement will be listed for the whole system as well.

4.2.1 Functional requirements

Functional requirements of a web-based application for an Inventory Management System for Automotive Parts are outlined in Table 4.1 below:

Table 4.1: Functional requirements for web-based application

Function	Functional Requirement
Login	The web application shall allow all types of users to log in to the web application using a username and password.
User Management	The web application shall allow the super administrator and inventory manager to view a list of users.
	The web application shall allow the super administrator and inventory manager to create a new user.
	The web application shall allow the super administrator and inventory manager to edit the details of a user.
	The web application shall allow the super administrator and inventory manager to delete a user.
	The web application shall allow the super administrator and inventory manager to search for a user.
	The web application shall allow the super administrator and inventory manager to turn the user status on or off.
Role/Access Management	The web application shall allow the super administrator to view a list of roles.
	The web application shall allow the super administrator to change access assignments for each role, limiting access to specified pages.
Page Access Management	The web application shall allow the super administrator to view a list of page access.
	The web application shall allow the super administrator to configure the page permission for all users, limiting every type of user from accessing the page.
Inventory Item Management	The web application shall display a list of inventory items.
	The web application shall allow users to add new items, including item information and inventory image.

	The web application shall allow users to remove items.
	The web application shall allow users to view the information for each item.
	The web application shall allow users to search for items using the item name.
	The web application shall allow users to edit the information associated with each item, including the inventory image.
	The web application shall allow users to download or print the QR code corresponding to an inventory item.
	The web application shall display a list of inventory categories.
	The web application shall allow users to add a new category.
	The web application shall allow users to change the name of the category.
	The web application shall allow users to search for categories using the category name.
	The web application shall allow users to delete a category.
Profile	The web application shall allow users to edit their details.
	The web application shall allow users to change their login credentials, including username and password.
	The web application shall allow users to log out from the web application.
Dashboard	The web application shall alert users with a list of items below reorder points and out of stock.
	The web application shall allow users to access the item in the reorder points list and out-of-stock list, allowing them to redirect to the corresponding item details page directly.

Functional requirements for mobile-based applications for the Inventory Management system for Automotive Parts are outlined in Table 4.2 below:

Table 4.2: Functional requirements for mobile-based application

Function	Functional Requirement
Login	The mobile application shall allow users to log in to the mobile application using a username and password.
Inventory Item	The mobile application shall display a list of items.
	The mobile application shall allow users to view the information for each item.
	The mobile application shall allow users to search for items using the item name.
	The mobile application shall allow users to edit the stock quantity associated with each item.
Profile	The mobile application shall allow users to view their profile page.
QR scanning	The mobile application shall allow users to identify an item by scanning QR barcodes on the item using the built-in camera of a mobile phone.
	The mobile application shall allow users to view an item's information after scanning that item's QR code.
Dashboard	The mobile application shall alert users with a list of items below reorder points and out of stock.
	The mobile application shall allow users to access the item in the reorder points list and out-of-stock list, allowing them to redirect to the corresponding item details page directly.

4.2.2 Non-functional requirements

- The system's response time should be responsive when the user interacts with the system.
- The system shall validate user input and prevent incorrect input format by prompting error messages to the user.

- The system's interface is easy to use, easy to navigate, simple, consistent, and easy to understand by the end user.
- The mobile application shall be able to gain access to the mobile camera.
- The system should be able to be accessed by multiple users simultaneously.
- The mobile application shall be compatible with the Android mobile device with an Android version of 4.1 (API 16) or newer.

4.2.3 Use case diagram

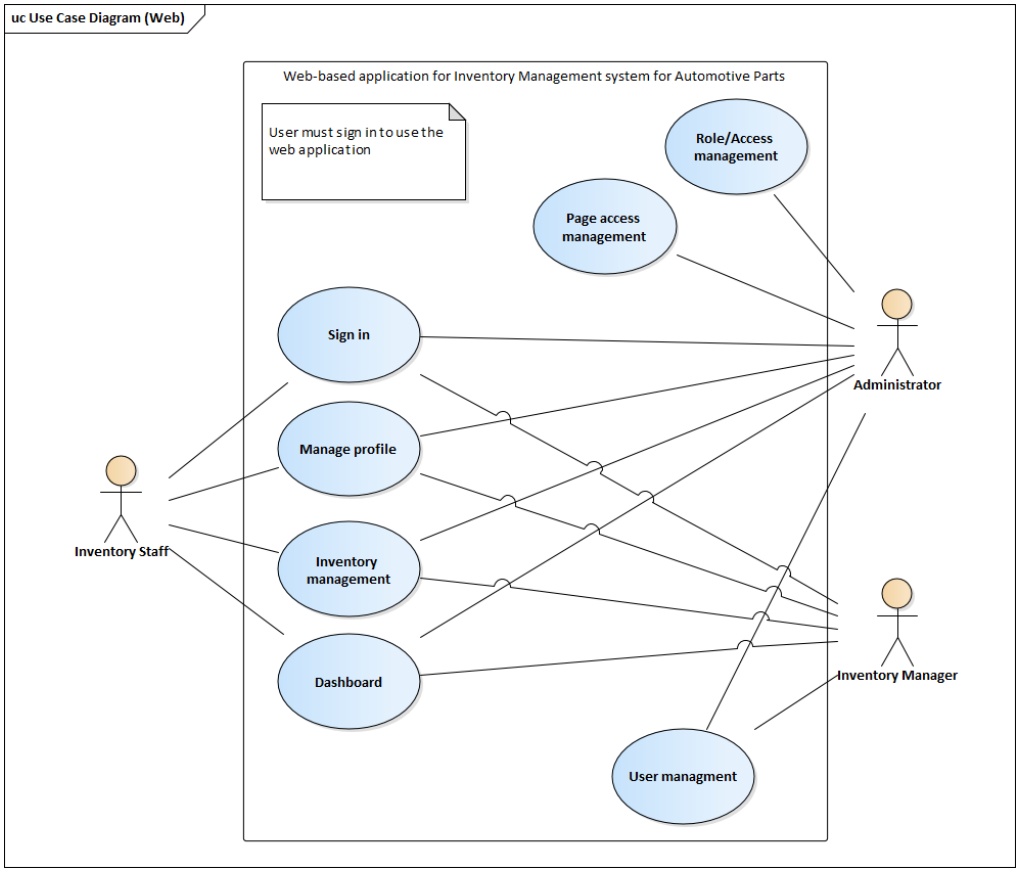


Figure 4.1: Use case diagram for web application

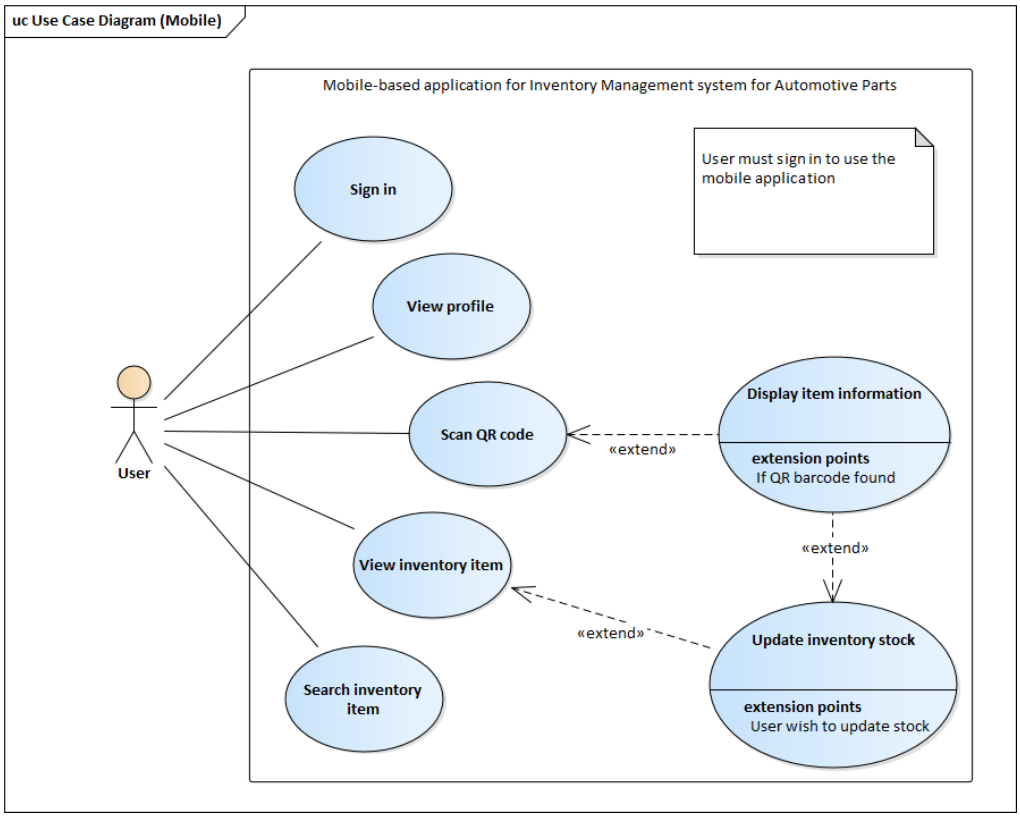


Figure 4.2: Use case diagram for mobile application

4.2.4 Use case description

Several use cases have been consolidated into a single form that can be used for both web and mobile applications.

Table 4.3: Use case description for sign-in

Use Case Name: Sign-in	ID: 1	Importance Level: High
Primary Actor: All types of user	Use Case Type: Detailed, Essential	
Stakeholders and Interests: User – Users already have an existing account and want to access the web-based or mobile-based inventory management system.		
Brief Description: This use case describes how the user can access the web-based or mobile-based inventory management system by filling up the username and password to sign in.		
Trigger: A user wants to access the application by signing into their account.		
Relationships: Association: Super Administrator, Inventory Staff, Inventory manager Include: N/A Extend: N/A Generalisation: N/A		
Normal Flow of Events: 1. The user tries to access the web application or mobile application. 2. The user fills in sign-in credentials, including username and password, on the sign-in form. 3. The user clicks the sign-in button under the sign-in form. 4. The system verifies that the login credentials are found and match the database. 5. The user is allowed to access the web application or mobile application.		
Sub-flows: 2.1 The system will validate whether the username exists or not 4.2 The system will search the database for the username and password filled in by the user. 5.2 The system retrieves the user records from the database.		
Alternate/Exceptional Flows: 4.1 If the username provided by the user is found in the database, the system will proceed to password matching. 4.2 If the username is not found in the database, the system will indicate that the user account with the username does not exist, and the user must re-enter the correct username that exists. 4.3 If the user provides a username and password that matches the database, access to the web and mobile applications will be granted to the user. 4.4 If the password does not match the username, the system will prompt		

the user with a password mismatch message, and the user must retype the password again until the password is matched. 4.5 If the user leaves out any input field as empty, the system will prompt the user with a missing input message.

Table 4.4: Use case description for user management

Use Case Name: User Management	ID: 2	Importance Level: High
Primary Actor: Super Administrator, Inventory Manager	Use Case Type: Detailed, Essential	
Stakeholders and Interests: Super administrator, inventory manager – Users who want to manage other users with lower access power.		
Brief Description: This use case describes how the user conducts user management.		
Trigger: The user wants to manage other users		
Relationships: Association: Super Administrator, Inventory Manager Include: N/A Extend: N/A Generalisation: N/A		
Normal Flow of Events: **Add user** 1. The user accesses the user management page through the navigation menu bar. 2. The user clicks the "add user" button. 3. The system will prompt a form to add a user. 4. The user fills in the necessary details, including username, password, region and role. 5. The user clicks the add button under the add item form. 6. The system verifies that all inputs are in the correct format. 7. The system will save the changes made to the item. **View user** 1. The user accesses the user management page through the navigation menu bar. 2. The system will display a list of the created user. **Edit user** 1. The user accesses the user management page through the navigation menu bar 2. The user selects one of the users from the list for editing. 3. The system will display an edit form with filled details corresponding to the item. 4. The user could change details about the user by replacing the current details through the input field. 5. The user confirms the information changes made to the user after clicking the confirm button.		

6. The system verifies that all inputs are in the correct format
7. The system will save the changes made to the item.

Delete user

1. The user accesses the user management page through the navigation menu bar.
2. The user selects a user from the list to be deleted.
3. The system will prompt a confirmation dialogue to confirm the deletion of the corresponding user.

Search User

1. The user accesses the user management page through the navigation menu bar.
2. The user enters a search query in the search text box
3. The system will display a list of users that match the search query.

Sub-flows:

Add user

7.1 The system will save the details of the user filled in by the user to the database after validation is completed.

View user

1.1 The system retrieves all the users' data from the database.

Edit user

7.1 The system will save the user details edited by the user to the database.

Delete user

3.1 The system will remove the corresponding user and its details from the database.

Alternate/Exceptional Flows:

Add user

4.1 The user must fill in all the details to create a user.

6.1 If the username provided by the user is found duplicated within the database, the user will be prompted to use another username,

6.2 If the user left any mandatory input field empty, the system would prompt the user with a missing input message.

Edit user

6.1 If the username provided by the user after editing is found duplicated within the database, the user will be prompted to use an alternative username.

6.2 If the user left out any mandatory input field as empty after editing, the system would prompt the user with a missing input message.

Search user

2.1 If the search query text provided by the user does not match any user in the user list, no data is found message will be displayed.

Table 4.5: Use case description for role/access management

Use Case Name: Role/Access management	ID: 3	Importance Level: High
Primary Actor: Super Administrator	Use Case Type: Detailed, Essential	
Stakeholders and Interests: Super administrator – want to change the access power of a user role		
Brief Description: This use case describes how the super administrator changes the access power of a user role.		
Trigger: The super administrator wants to manage the access power of a user role.		
Relationships: Association: Super Administrator Include: N/A Extend: N/A Generalisation: N/A		
Normal Flow of Events: **View roles and their access** 1. The super administrator accesses the role/access management page through the navigation menu bar. 2. The system will display a list of the roles. 3. The super administrator clicks on the edit button and a modal with checkboxes showing all the access power of the corresponding roles. **Edit the access power of the role.** 1. The super administrator accesses the role/access management page through the navigation menu bar. 2. The super administrator selects one of the roles from the list for editing. 3. The system will display an edit form with filled access power corresponding to the roles. 4. The super administrator could change the access power of the roles by checking or unchecking those checkboxes. 5. The super administrator confirms the access power changes to the corresponding role by clicking the confirm update button. 6. The system will save the access power changes made to the roles.		
Sub-flows:		
Alternate/Exceptional Flows:		

Table 4.6: Use case description for page access management

Use Case Name: Page Access Management	ID: 4	Importance Level: High
Primary Actor: Super Administrator	Use Case Type: Detailed, Essential	
Stakeholders and Interests: Super administrator – want to configure the page permission of a specified page		
Brief Description: This use case involves the actions of the super administrator to control the page permission settings for a specified page, allowing them to enable or disable access as needed.		
Trigger: The super administrator wants to configure the page permission of a specified page.		
Relationships: Association: Super Administrator Include: N/A Extend: N/A Generalisation: N/A		
Normal Flow of Events: **View page access** 1. The super administrator accesses the page access management page through the navigation menu bar. 2. The system will display a list of the page access. 3. The super administrator can configure page permission settings by clicking the edit button, displaying a small pop-up with toggle options. **Edit the access power of the role.** 1. The super administrator accesses the page access management page through the navigation menu bar. 2. The super administrator selects one of the page access from the list for editing. 3. The super administrator can configure page permission settings by clicking the edit button, displaying a small pop-up with toggle options to turn the page permission on and off. 4. The system will save the page permission setting made to the page access.		
Sub-flows:		
Alternate/Exceptional Flows:		

Table 4.7: Use case description to manage profile

Use Case Name: Manage profile	ID: 5	Importance Level: High
Primary Actor: Administrator, inventory manager, inventory staff	Use Case Type: Detailed, Essential	
Stakeholders and Interests: User – Users who want to manage their user profile that includes login credentials		
Brief Description: This use case describes how users can change their profile details, including login credentials.		
Trigger: A user wants to change login credential and its profile details		
Relationships: Association: Administrator, inventory manager, inventory staff Include: N/A Extend: N/A Generalisation: N/A		
Normal Flow of Events: **View Profile for web appliation** 1. The user accesses their profile through the profile icon. 2. The user can view their username, password, region and role. **View Profile for mobile application** 1. The user accesses their profile through the navigation menu bar. 2. The user will be able to view their username, region and role. **Edit Profile for web application** 1. The user accesses their profile through the profile icon. 2. The user could change their username, password, region and role by replacing the existing username, password, region and role in the input field. 3. The user confirms the changes made after clicking the save user details button. 4. The system verifies that all inputs are in the correct format and that usernames are not duplicated. 5. The system will save the changes made to the user profile.		
Sub-flows: 2.1 Users can change the specified details they want to change without changing all of them.		
Alternate/Exceptional Flows: 4.1 If the username provided by the user is found duplicated within the database, the user will be prompted to use alternative username.		

Table 4.8: Use case description for inventory management

Use Case Name: Inventory Management	ID: 6	Importance Level: High
Primary Actor: Administrator, inventory manager, inventory staff	Use Case Type: Detailed, Essential	
Stakeholders and Interests: User – Users who want to manage inventory items		
Brief Description: This use case describes how users can add, view, update and delete the inventory item or CRUD operation. The user will also use the search function to search for inventory items.		
Trigger: A user wants to perform CRUD operations on the inventory item.		
Relationships: Association: Administrator, inventory manager, inventory staff Include: N/A Extend: N/A Generalisation: N/A		
Normal Flow of Events: **Add an item through the web application** 1. The user accesses the create inventory page through the navigation menu bar. 2. The user fills in add item form that includes the item name, item category, opening stock, reorder point, storage box no, sub-compartment, weight, dimensions, brand, model, vendor, description, note and inventory image. 3. The user clicks the save button under the create inventory form. 4. The system verifies that all inputs are in the correct format. **View items** 1. The user accesses the inventory list page through the navigation menu bar. 2. The system will display a list of created items, including the item's name, image, category, stock quantity, re-order point, and storage box no. **Edit items through the web application** 1. The user accesses the inventory list page through the navigation menu bar. 2. The user selects one of the items from the list for editing. 3. The system will display an edit form with filled information corresponding to the item. 4. The user could change information about an item by replacing the current information through the input field. 5. The user confirms the information changes made to the item after clicking the confirm button. 6. The system verifies that all inputs are in the correct format 7. The system will save the changes made to the item **Edit items stock through the mobile application**		

1. The user accesses the inventory list page through the navigation menu bar.
2. The user selects one of the items from the list for editing.
3. The mobile application will display all the item details and an input field to update the inventory stock.
4. The user modified the stock quantity
5. The user clicks the update stock button.

Delete item for web application

1. The user accesses the inventory list page through the navigation menu bar.
2. The user selects an item from the list to be deleted.
3. The system will prompt a confirmation dialogue to confirm the deletion of the corresponding item.
4. The user confirms to delete the item.

Search for an inventory item

1. The user accesses the inventory page through the navigation menu bar.
2. The user moves the cursor to the search box and clicks it.
3. The user enters the name of the item that needs to be search
4. The system will display the search result that matches the name of any item the user enters.
5. The system will display one or more search results, including information on the item's name, image, stock quantity, reorder point and storage box no.

Sub-flows:

Add an item through the web application

4.1 After verification, the system will save the information of the item filled in by the user to the database.

View items

2.1 The system retrieves all the information of all the items from the database.

2.2 The user can click on any item in the list to view the full details of the corresponding item.

Edit an item through the web application

7.1 The system will save the information edited by the user to the database.

Edit an item through the web application

4.1 The stock quantity is updated in the database

Delete an item through the web application

4.1 The system will remove the corresponding item and its information from the database.

Search for an inventory item

4.1 The system refreshes the search result on each text change the user makes.

4.2 The system retrieves the information of items matching the search name from the database.

Alternate/Exceptional Flows:

Add an item through the web application

3.1 The user does not have to fill in all the information, but the item name, category, opening stock, reorder point, storage box no., weight and dimensions are mandatory.

3.2 If the user left any mandatory input field empty, the system would prompt the user with a missing input message. **Edit an item through the web application** 6.1 If the user left out any mandatory input field as empty after editing, the system would prompt the user with a missing input message. **Search for an inventory item** 4.1 If the user enters an item name that does not exist in the database, no data message will be shown.

Table 4.9: Use case description for Dashboard

Use Case Name: Dashboard	ID: 7	Importance Level: High
Primary Actor: Administrator, inventory manager, inventory staff	Use Case Type: Detailed, Essential	
Stakeholders and Interests: User – Users want to check the inventory summary in the dashboard		
Brief Description: This use case describes how a user can access the inventory summary in the dashboard.		
Trigger: Users want to access the inventory summary in the dashboard.		
Relationships: Association: Administrator, inventory manager, inventory staff Include: N/A Extend: N/A Generalisation: N/A		
Normal Flow of Events: 1. The user accesses the dashboard through the navigation menu bar (After the user logs in, the user will be automatically redirected to the dashboard). 2. The system retrieves all the information needed for the inventory summary in the dashboard. 3. The system will display the Number of Items, Number of Total Stock, Low Stock Items count, out-of-stock item count, list of items below reorder point and list of out-of-stock items.		
Sub-flows:		
Alternate/Exceptional Flows:		

Table 4.10: Use case description for scan QR code

Use Case Name: Scan QR code	ID: 8	Importance Level: High
Primary Actor: User	Use Case Type: Detailed, Essential	
Stakeholders and Interests: User – Users want to scan a QR code on an item with the phone camera to get information about the item.		
Brief Description: This use case describes how a user can use the phone camera to scan a QR code on an item.		
Trigger: Users want to access an item's information through the mobile application on the phone.		
Relationships: Association: User Include: N/A Extend: display item information (Refer to use case ID: 6: view item) Generalisation: N/A		
Normal Flow of Events: 4. The user access the QR scanning feature through the navigation bar of the mobile application. 5. The mobile application will turn on the camera of the mobile phone. 6. The user points the camera at the QR code on the item. 7. The mobile application will scan the QR code and decode it into readable data. 8. The system will query the database and look for items with the inventory ID matching the decoded data. 9. The system will display product information that matches the QR code, including the item's name, quantity, images, etc. (Similar to the web application)		
Sub-flows:		
Alternate/Exceptional Flows: 2.1 If the mobile application does not have access to the camera, the application will request permission to access the camera. 5.2 If no inventory id matches the decoded data of the QR code, the system will prompt the user with the message "No items match the QR code".		

4.3 Low-fidelity prototypes

Low-fidelity prototypes with the User interface (UI) representations are created for mobile and web applications. This low-fidelity is created using the React, React Native framework and minor usage of Firebase. The use of Firebase in this low-fidelity prototype is to show how the data is stored and communicated between the web application and mobile application. The low-fidelity prototype of the web application was created for functions such as logging in, registering, managing profiles, managing inventory items and searching for inventory items. The low-fidelity prototype of a mobile application is created for the logging in, registering and firebase workflow. The purpose of creating a low-fidelity prototype was to provide ideas for visualizing an automotive parts inventory management system and illustrate what it would look like and how it would work. In addition, feedback will be recorded after the low-fidelity prototype is presented to the project supervisor. The low-fidelity prototype is shown in **Appendix A** on the last page of this report.

CHAPTER 5

SYSTEM DESIGN

5.1 Introduction

This chapter provides an overview of the system architecture and database design used in this project. The system architecture design includes a detailed explanation of the ReactJS and React Native frameworks and the cloud services used to develop the system. On the other hand, the database design is presented using an Entity-Relationship Diagram (ERD) to illustrate the relationships between entities and their attributes. Additionally, entities description and a data dictionary offer a comprehensive understanding of the database design. By presenting a clear and well-structured system architecture and database design, this chapter lays the foundation for the development and implementation of the system.

5.2 System Architecture Design

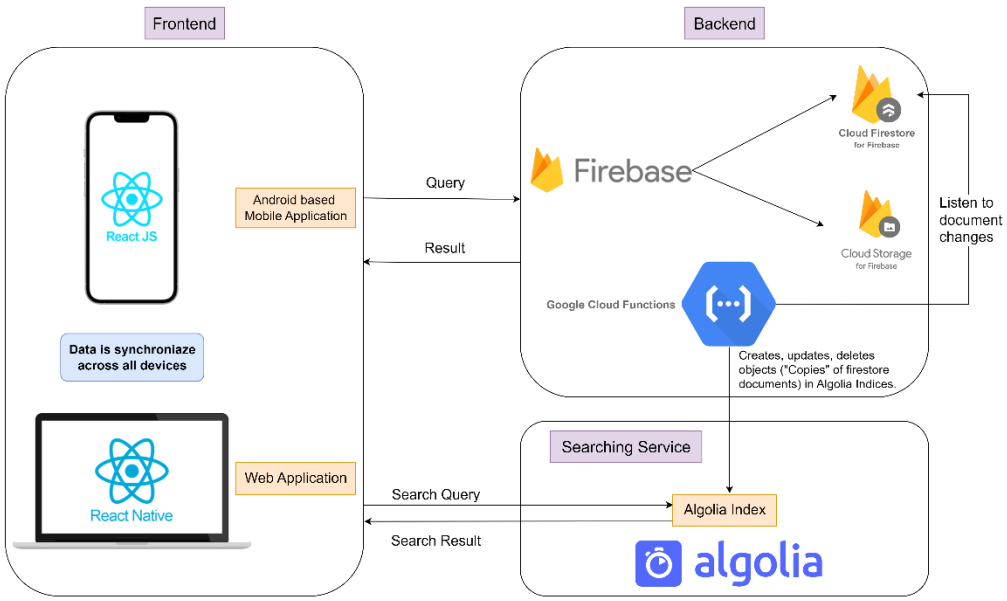


Figure 5.1: Overview of the System Architecture Design

Figure 5.1 shows an overview of the implemented system's architecture design. The system consists of two applications: An Android-based mobile application and a web application. The web application is developed using React.js, whereas the mobile application is developed using React Native. The

applications will send query requests to the Firebase's Cloud Firestore using Firebase SDK to fetch or send data to the Firebase. Next, the Firestore will return data to the front end. Similarly, the process is the same for Cloud Storage, but Cloud Storage will only store files such as images.

Algolia is a third-party service that provides a search-as-a-service platform that can be integrated with Firebase to enable full-text search capabilities for both web and mobile applications. Algolia searching service can be enabled by installing the extension in the Firebase console. Whenever a creates, updates, deletes object action in the Firebase, the extension will trigger Firebase/Google Cloud functions to trigger an indexing process in the Algolia. In layman's terms, Algolia will make a copy of those documents (data) into Algolia's server. Once those documents (data) are indexed, users of applications in the front end can perform search queries on the data by sending a request to Algolia's servers. The search result will then return to the front end (Farias, 2019).

5.2.1 ReactJS architecture

The ReactJS library is built on a solid foundation. It is easy to use, adaptable, and expandable. React is a library for building user interfaces in a web application. React enable programmers to construct user interfaces entirely in JavaScript.

Traditional web development frameworks or libraries often introduce their own templating language for developing user interfaces which developers need to learn the new template language. React, on the other hand, focuses on the concept of having different "components", which are essentially reusable pieces of UI that can be combined to create a complete web application. React breaks the UI into smaller, more manageable pieces to make building and maintaining large, complex user interfaces easy.

React introduces three basic concepts: **React elements**, **JSX** and **React Component.**

In React, a **React element** is a plain JavaScript object representing an HTML DOM element or a component. An element is the smallest building

block in React applications and can be considered a lightweight representation of a DOM node. React offers the React API to build a React element using the createElement() function.

JSX is an XML-based JavaScript extension for designing user interfaces that allows you to write HTML syntax code with slight modifications in the JavaScript files. JSX facilitates the creation and manipulation of React components and helps maintain the written code's clarity and organisation.

The fundamental building block of React application is **React components**. To create its user interface, it uses JSX and React Elements. React components are reusable UI components that may be combined to build complex user interfaces. React Component is either a pure JavaScript function or a JavaScript class that extends the React Component class. Components in React feature life cycles, event handlers, state management, and properties. React components are capable of performing both basic and complex logic.

The React library is only a UI framework as it does not provide any special patterns for developing a complex application. Developers are free to employ whatever design pattern they like if it is appropriate to the project.

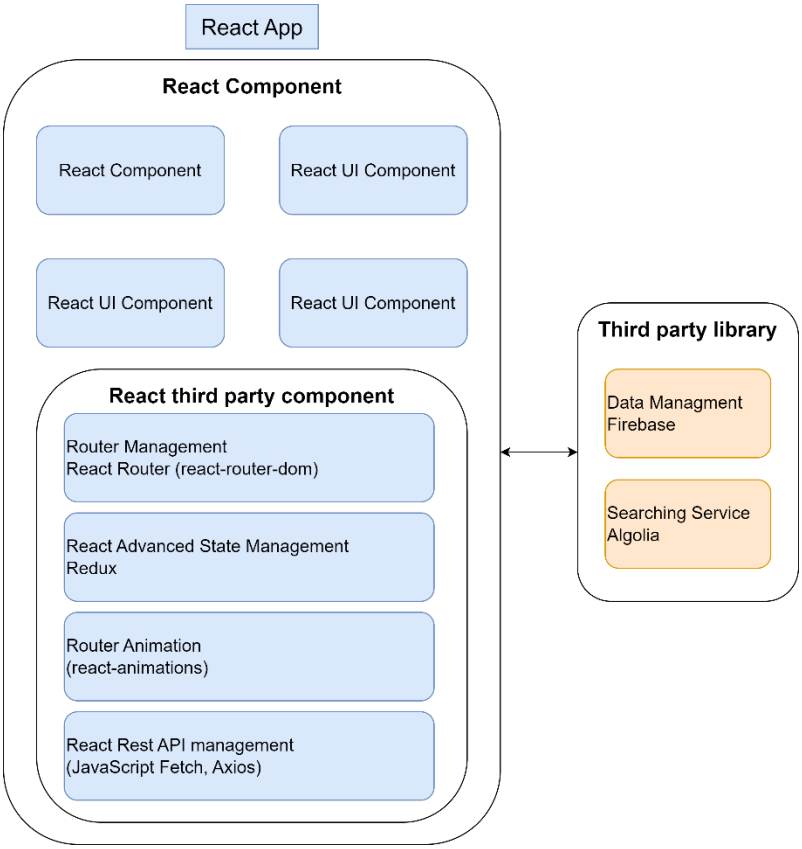


Figure 5.2: Overview of React Application Architecture

According to the Figure above, React app starts with a single root component responsible for rendering the entire application. This component may have one or more child components, each of which may be nested with any other component at any level. This establishes a hierarchy of components that cooperate to provide the overall user interface. In most situations, the components in a React application offer user interface features like buttons, forms, and menus. However, third-party components can offer extra features like routing management, animation management, or advanced state management (Javatpoint, n.d.).

For example, Ant design is used as the UI library for developing the web application in this project. Ant design is built using React, intended to function well with React architecture. Installing the Ant Design package and importing the required components into the React components will enable Ant Design to work in the React application.

5.2.2 React Native architecture

React Native is also built on top of React, and React Native use the same principle as React, such as having components, props and state. The architecture and syntax of React Native and React are similar. However, they were designed to work on different platforms. React is used to develop web applications, whereas React Native is used to create native mobile apps for iOS and Android. In this project, React Native will be used to develop the Android mobile application.

There are two versions of architecture for React Native. One is the older architecture version, and another is new with improved performance and efficiency. In this project, we have implemented React Native v0.69.5 using the new architecture. By using the latest version of React Native, the mobile application can be created using the advantage of its modern features and capabilities.

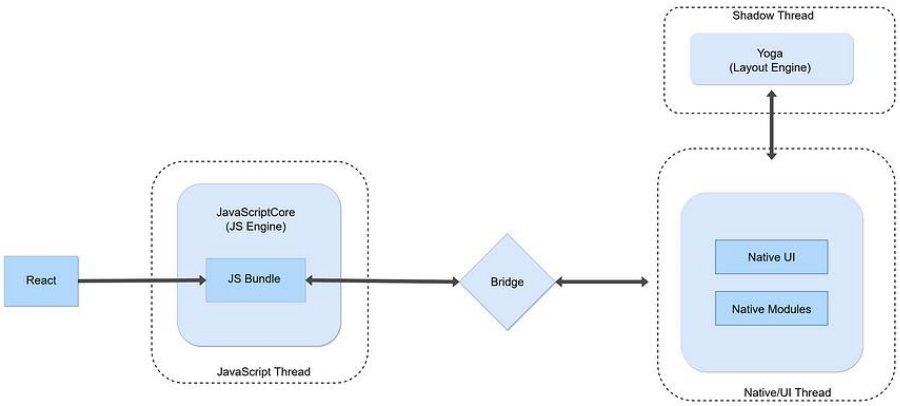


Figure 5.3: Overview of the React Native Architecture

The React Architecture operates as shown in the figure above. When running a React Native app, the React Architecture separates the Native Code from the JavaScript code, with the latter being bundled together into a package known as the JS Bundle. The execution of React Native apps involves three threads that will work together. The first thread is the JavaScript thread, which is executed by the JS engine to run the JS Bundle. The first thread is the

JavaScript thread, which is executed by the JS engine to run the JS Bundle. The second thread, known as the Native/UI thread, handling operations such as Native Modules and managing tasks like UI rendering and user gesture events. The third thread is Shadow thread, which is used to calculate the Layout of Elements (Positions of UI elements) before they are rendered on to the host screen. The bridge is an entity that facilitates communication between Native and JS Threads. Data must be batched and serialized as JSON to pass it via the bridge. However, the bridge entity can only support asynchronous communication. Additionally, a drawback of using Bridge is that any received data on the other end must also be decoded which may slow down the performance.

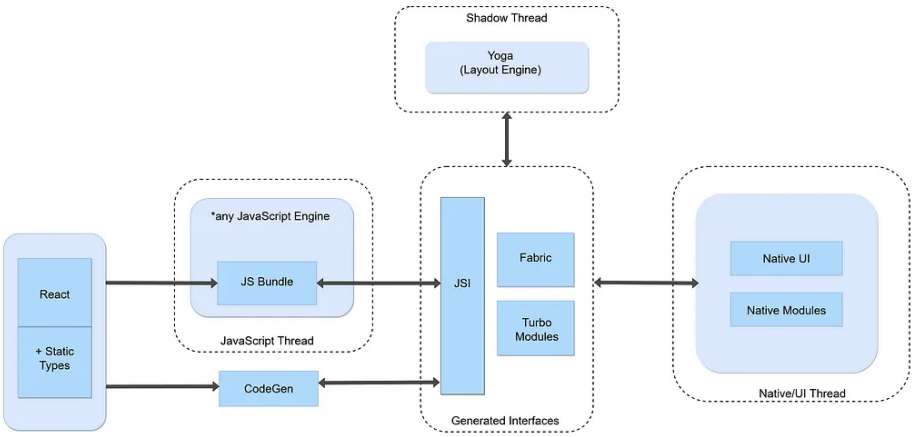


Figure 5.4: Overview of the new React Native Architecture

The issues associated with the old architecture have been resolved with the new React Native Architecture, illustrated in the figure above. The bridge will be replaced in the New Architecture by a module called JavaScript Interface, a lightweight, general-purpose layer written in C++ that the JavaScript engine may utilize to invoke/call functions in the native domain directly.

The old architecture utilizes the JavaScriptCore Engine, and the bridge is only compatible with this specific engine. However, the new **JSI** architecture is designed to be decoupled from the engine, enabling the use of other JavaScript engines like Chakra, v8, Hermes, and others. Using alternative JavaScript engines will be made possible through JSI, and the

threads will be completely interoperable. JavaScript code can communicate with the native side directly from the JS thread. This will resolve the congestion and asynchronous problems on the bridge and remove the requirement to serialise JSON data.

Fabric is React Native's new rendering technology that has taken the role of the existing UI Manager. It aims to improve performance and reduce memory consumption by synchronizing the JS and UI threads and prioritizing user interactions to be executed synchronously while other tasks, such as API requests, are executed asynchronously.

Turbo Modules are an enhancement over the old Native Modules in the old architecture used by JavaScript in React Native. With Turbo Modules, JavaScript can hold a reference to these modules, allowing modules to be loaded when required, significantly improving start-up time for React Native apps.

CodeGen is a static type checker included in the new React Native architecture to ensure smooth communication between JavaScript and C++. JavasSript is a dynamically typed language, whereas C++ is a statically typed language. CodeGen will define interface components utilised by Fabric and Turbo Modules by utilising typed JavaScript as the source of truth. Additionally, it will produce more native code during build time rather than run time (Patil, 2022).

5.3 System Database Design

This section will focus on discussing the database design. To visually represent the relationships between entities in a database system, two types of entity relationship diagrams (ERDs) are drawn, physical ERD and logical ERD. An ERD serves as a valuable tool in defining and comprehending the structure and organization of a database. It accomplishes this by visually illustrating the entities involved, along with their associated attributes and the relationships that connect them. Through the use of an ERD, one can gain a deeper understanding of the database, ensuring effective management and organization of data.

Furthermore, a table displaying the entities description and data dictionary is presented to provide a more comprehensive illustration of the system's database design.

5.3.1 Physical Entity Relationship Diagram

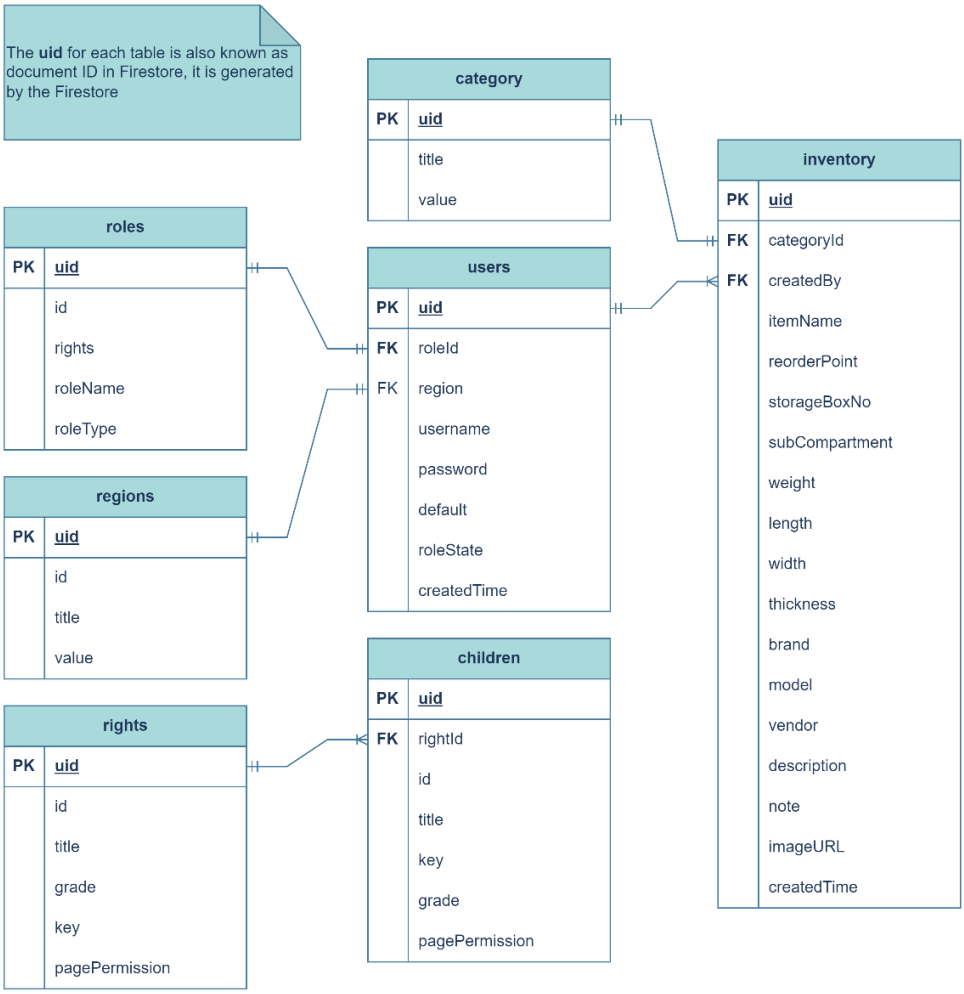


Figure 5.5: Physical Entity Relationship Diagram

5.3.2 Logical Entity Relationship Diagram

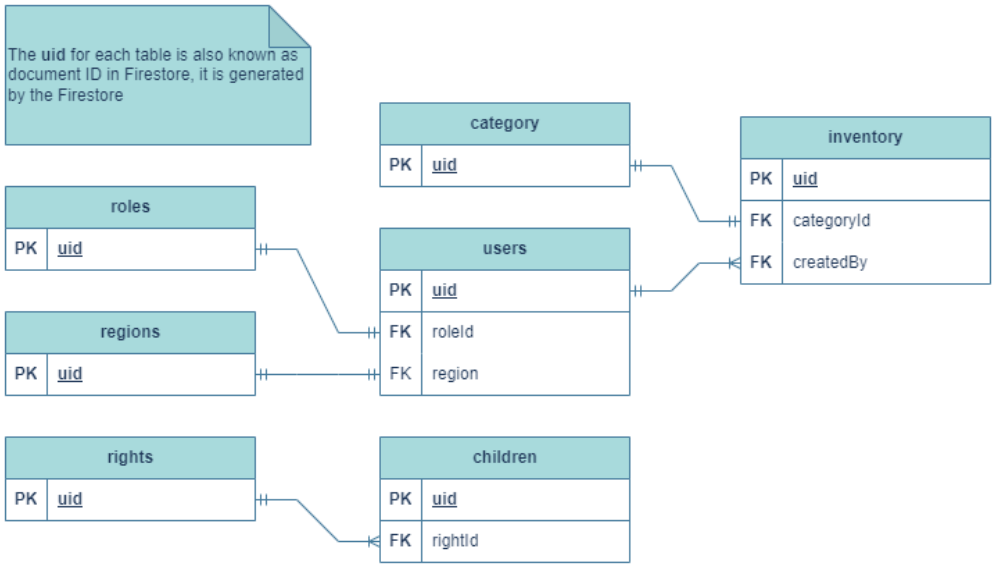


Figure 5.6: Logical Entity Relationship Diagram

5.3.3 Entities Description

The function of entity description is to provide clear definitions and descriptions of each entity in a database. It helps understand the purpose, characteristics, and relationships of entities, promoting a common understanding of the database structure

Table 5.1: Entities Description Table

Entity	Description
inventory	Stores all the inventory details
users	Stores all the user details, including their login credentials
roles	Stores all the roles information, including the specific rights that each role has
regions	Stores all the regions
category	Stores all the categories
rights	Stores all the rights including all the routes of parents' page.
children	Stores all the children's rights,

	including all the routes of the children page

5.3.4 Data Dictionary

The function of a data dictionary is to serve as a centralized repository of detailed information about the data elements in a database. It includes metadata such as attribute names, descriptions, data types, constraints and example values.

5.3.3.1 Data Dictionary for inventory collection

The inventory collection is used to stores all the inventory details.

Table 5.2: Data Dictionary for inventory collection

Attribute	Data Type	PK/FK	Description	Nullable	Example Values
uid	ID generated by Firebase	PK	Unique identifier of inventory	No	zCBgl1Ls6TEdRNpEnwqz
categoryId	string	FK	The item belongs to which category	No	KsaNLqSQJzdPAGQu2TGP
createdBy	string	FK	Determine who created this item	No	9DYggiQF5KqLsPb3G2Qd
itemName	string		Name of the item	No	Rotiform 15” Vossen HF5
reorderPoint	number		Reorder point of the item	No	3

storageBoxNo	number		In which container box is the item stored?	No	6
subCompartment	string		Items are stored in which sub-compartment inside the container box?	Yes	A
weight	number		Weight of the item	No	13.6
length	number		Length of the item	No	1
width	number		Width of the item	No	407.4
thickness	number		The thickness of the item	No	166
brand	string		Brand name of the item	Yes	Vossen
model	string		Model of the item	Yes	HF-5
vendor	string		Vendor of the item	Yes	Beng San Tyre & Battery Sdn. Bhd.
description	string		Description of the item	Yes	HYBRID FORGED SERIESDiameters 19"-24" | Widths 8.5" - 12"
note	string		Additional note to	Yes	Vossen's all-new Hybrid Forged™ wheel, the HF-5, is

			remark additional information or context about the item, such as special handling instructions		derived from its forged counterpart, the S21-01.
imageURL	string		Uniform Resource Locator (URL) to the image of the item	Yes	https://firebasestorage.googleapis.com/v0/b/inventory-management-sys-f1f56.appspot.com/o/images%2Finventory%2Frc-upload-1681067544982-34?alt=media&token=7bb9d182-8417-4929-a8b4-9aa107f54737
createdTime	number		Created Time of the item	No	1681070435301

5.3.3.2 Data Dictionary for users collection

The user collection is responsible for storing comprehensive user details, encompassing their login credentials and other relevant information.

Table 5.3: Data Dictionary for users collection

Attribute	Data Type	PK/FK	Description	Nullable	Example Values
uid	ID generated by Firebase	PK	Unique identifier of the user	No	OD1p3KTfyGRlg0Oek3WF
roleId	Number	FK	The user belongs to which role	No	3
username	string	FK	Name of the user and will be used as a login credentials	No	elon
password	string		The password that will be used as a login credential, together with the username	No	test123
default	boolean		Determine if the user is a Root Administrator	No	False
region	number		Determine the user is responsible for which region	No	Zone 1
roleState	boolean		Determine the role of the user.	No	True

createdTime	number		The created time for this user	No	1681545127741

5.3.3.3 Data Dictionary for roles collection

The roles collection serves as a storage for all role-related information, encompassing the specific rights and privileges associated with each role.

Table 5.4: Data Dictionary for roles collection

Attribute	Data Type	PK/FK	Description	Nullable	Example Values
uid	ID generated by Firebase	PK	Unique identifier of roles	No	KJQ3aZmaqCD5VpSwdNpB
id	number		ID for role	No	1
rights	array		Name of the user and will be used as a login credentials	No	["/user-manage/add" "/user-manage/delete "/user-manage/update" "/user-manage/list "/access-manage/role/list" "/access-manage/access/list "/right-manage/role/update" "/right-manage/role/delete" "/right-manage/right/update" "/right-manage/right/delete"

					"/inventory-manage/list" "/inventory-manage/draft" "/audit-manage/audit" "/audit-manage/list" "/publish-manage/unpublished" "/publish-manage/published" "/publish-manage/sunset" "/user-manage" "/access-manage" "/audit-manage" "/publish-manage" "/home" "/user-profile" "/inventory-manage" "/inventory-manage/category" "/inventory-manage/update/:id" "/inventory-manage/add" "/inventory-manage/preview/:id"]

roleName	string		The title name for the Role	No	Super Administrator
roleType	number		Determine if the user is a Root administrator	No	1

5.3.3.4 Data Dictionary for regions collection

The regions collection is used to stores all the regions.

Table 5.5: Data Dictionary for regions collection

Attribute	Data Type	PK/FK	Description	Nullable	Example Values
uid	ID generated by Firebase	PK	Unique identifier of regions	No	IrKs5CgRGYtAbRsADHfO
id	number		ID for region	No	1
title	string		Title name of the region	No	Zone 1
value	string		Value of the region, usually the same as the title	No	Zone 1

5.3.3.5 Data Dictionary for category collection

The regions collection is used to stores all the categories.

Table 5.6: Data Dictionary for category collection

Attribute	Data Type	PK/FK	Description	Nullable	Example Values
uid	ID generated by Firebase	PK	Unique identifier of the category	No	NhfBR7zwdX91SewjZS4U
title	string		Title name of the category	No	Engine components and parts
value	string		The value of the category is usually the same as the title	No	Engine components and parts

5.3.3.6 Data Dictionary for rights collection

The rights collection is used to stores all the rights including all the routes of parents' page.

Table 5.7: Data Dictionary for rights collection

Attribute	Data Type	PK/FK	Description	Nullable	Example Values
uid	ID generated by Firebase	PK	Unique identifier of right	No	1D0jwUIKIX0e1HtJeDEj
id	Number		ID for right	No	1
title	string		The page's title will be used for navigation in the menu (for web applications only).	No	Homepage
grade	string		Determine the level of this page, which can be levels 1 and 2. Level 1 will be the parent page, while level 2 will be the child page.	No	1
key	number		The key specifies the location of a specific page within the web application's directory structure	No	/home

pagePermission	number		Decide if this page should be displayed or not	No	1

5.3.3.7 Data Dictionary for children collection

The children collection is used to stores all the children's rights, including all the routes of the children page.

Table 5.8: Data Dictionary for children collection

Attribute	Data Type	PK/FK	Description	Nullable	Example Values
uid	ID generated by Firebase	PK	Unique identifier of the children	No	E93jCvb78BDeVx06J3hk
rightId	Number	FK	The children belong to which right	No	7
id	string		ID for children	No	9
title	string		The page's title will be used for navigation in the menu (for web applications only).	No	Page Access Management
key	number		Determine the level of this page, which can be levels 1 and 2. Level 1 will be the parent page, while	No	/access-manage/access/list

			level 2 will be the child page.		
grade	number		The key specifies the location of a specific children's page within the web application's directory structure.	No	2
pagePermission	string		Decide if this children's page should be displayed or not	No	1

5.4 Conclusion

In conclusion, this chapter provided an overview of the system architecture and database design used in the project. The system architecture design included explanations of the ReactJS and React Native frameworks and the cloud services utilized in the system. The database design was presented using an Entity-Relationship Diagram (ERD) to depict the relationships between entities and their attributes. Additionally, entities' descriptions and a data dictionary were provided to offer a comprehensive understanding of the database design. Overall, this chapter laid the foundation for the development and implementation of the system by presenting a clear and well-structured system architecture and database design.

CHAPTER 6

SYSTEM IMPLEMENTATION

6.1 Introduction

This chapter provides an overview of setting up the project and implementing the entire system. It outlines each module that makes up the system, including a web application and a mobile application. The system modules have been designed based on the use cases and requirement specifications discussed in Chapter 4. To provide a deeper understanding of each module, this chapter provides further elaboration on their respective features and functionalities.

6.2 Project Setup

This project starts with creating a Firebase account on https://console.firebase.google.com/. Firebase served as a cloud database for web and mobile-based applications. Then, a Firebase project is created and set up with the project name. The Firestore database and Firebase Storage are added to this project to store data and images.

Next is to create and set up a React project to develop a web application. Node.js is needed for React application development. Node.js is downloaded from the official website: https://nodejs.org/en/download. Node.js comes with a package manager called npm (Node Package Manager), which provides access to thousands of packages and libraries that can be installed and used in the React app. The npm was used to install "create-react-app" by typing the following command in the command prompt "npm install -g create-react-app" to use the "create-react-app" command-line tool to create new React projects.

```
npx create-react-app <project-name>
```

Similarly, to create and set up a React Native application for developing a mobile application, Node.js is needed. The React Native command-line interface (CLI) must be installed by running the command "npm install -g react-native-cli". After the installation of CLI is completed, a

new React Native project can be created by running the command in the command prompt: -

```
react-native init <project-name>
```

Running this command will create a new project with the specified name and all the necessary files and dependencies. Android Studio needed to be installed to run the React Native application using the Android Emulator that comes with Android Studio, and it can be downloaded from https://developer.android.com/studio. To run the app, navigate to the project directory in the command prompt and run the “react-native run-android” command. This command will start a development server and launch the app in the Android emulator.

6.2.1 Firebase Setup

1. A Google account is needed to set up a Firebase project.
2. Once a Google account is prepared, go to https://console.firebase.google.com/ to create a new Firebase project by clicking the "Add project" button in the Firebase console, as shown below.

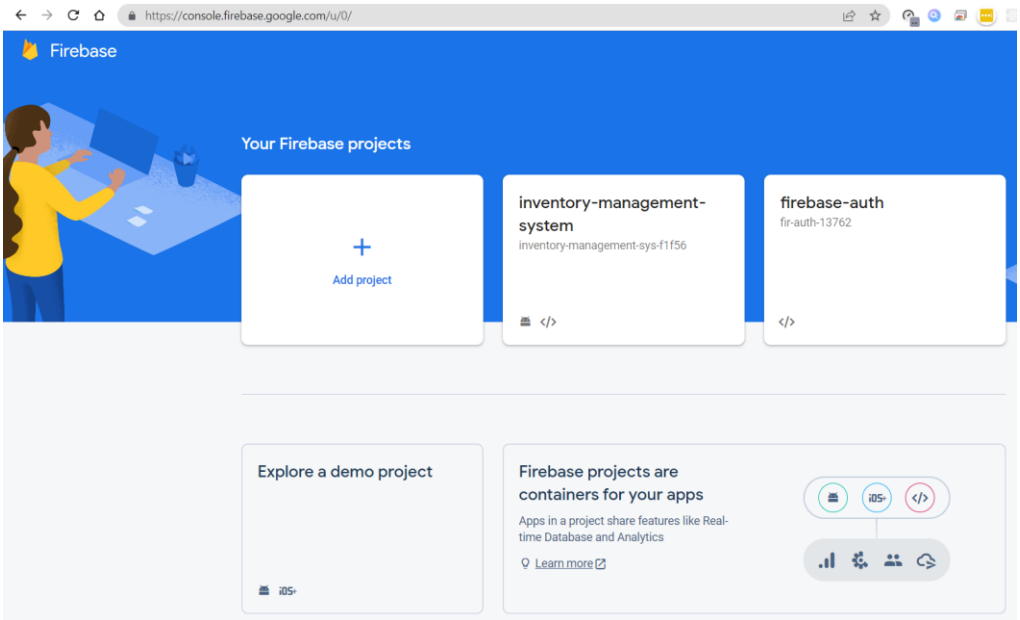


Figure 6.1: Firebase website to create a project (add project)

3. Follow the steps given by the Firebase console to complete the project creation process.

4. Once the project is created, a prompt to add Firebase to the application will be displayed, as shown in the figure below. This can be done by selecting the appropriate platform for your Android or Web app. Click on the respective red circle to begin adding Firebase to your app. Follow the on-screen instructions carefully to register the application and configure your Firebase project correctly.

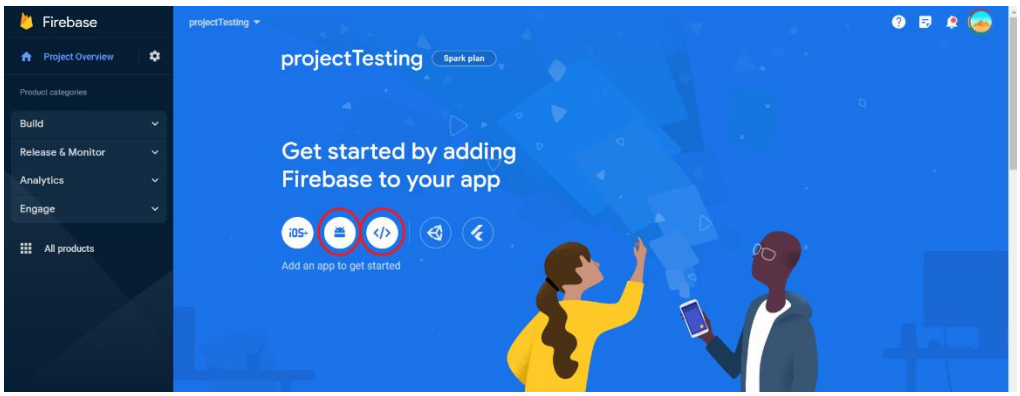


Figure 6.2: Process to add Firebase to mobile & web application

5. After successfully adding your application to Firebase, navigate to the Project Settings in the Firebase console. Scroll down to the bottom of the page, and follow the instructions to install the Firebase SDK to your project, as illustrated below.

   A. Installation of Firebase SDK to web application

      i. To begin using Firebase in your React web application project, you will need to install the Firebase SDK by running the following command in your terminal:

```
npm install firebase
```

      ii. Then, the next step is to create a Firebase folder within the src folder of your project. Create a JavaScript file to store your Firebase configuration within this Firebase folder.

      iii. Then copy the Firebase configuration from the Firebase console into this JavaScript file. Extra steps must be taken here to use the Firebase Storage, import “firebase/storage”, and create an instance to the Firebase storage as shown in the figure below.

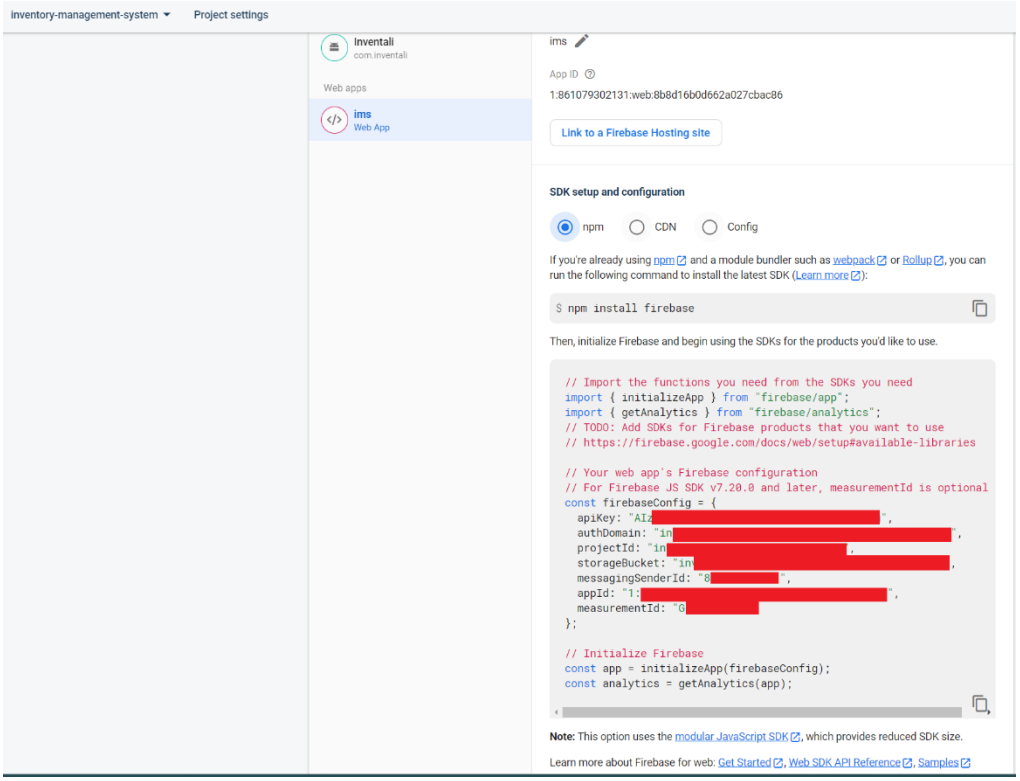


Figure 6.3: Firebase configuration shown in Firebase console

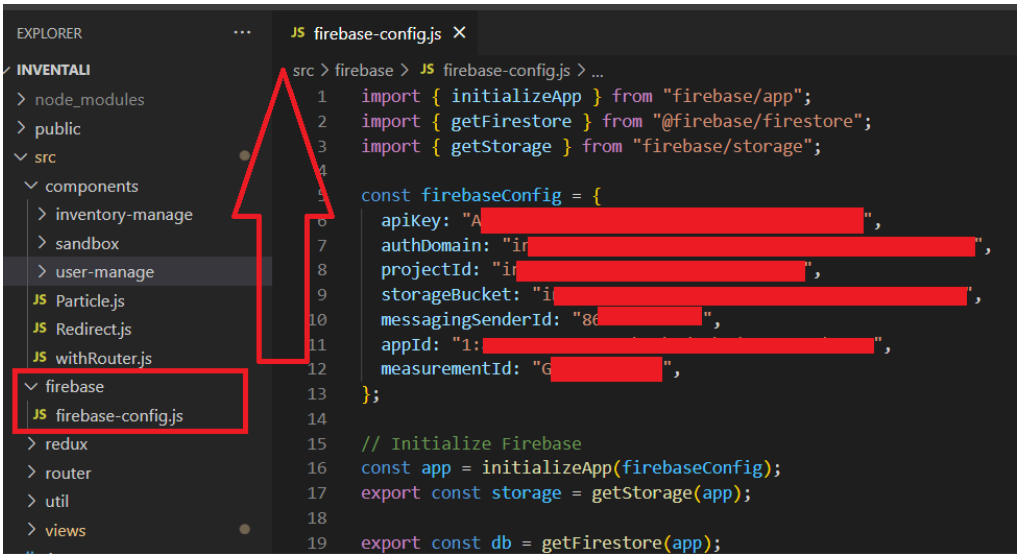


Figure 6.4: Firebase configuration pasted in the Firebase config file

iv. The Firebase service is now enabled in the React project.

2. Installation of Firebase SDK to mobile application

   i. To begin using Firebase in your React Native mobile application project, you will need to install React Native Firebase by running the following command in your terminal:

```
npm install @react-native-firebase/app
```

   ii. This time select the Android apps you created in the Firebase project instead of the web app. Then download the google-services.json file.

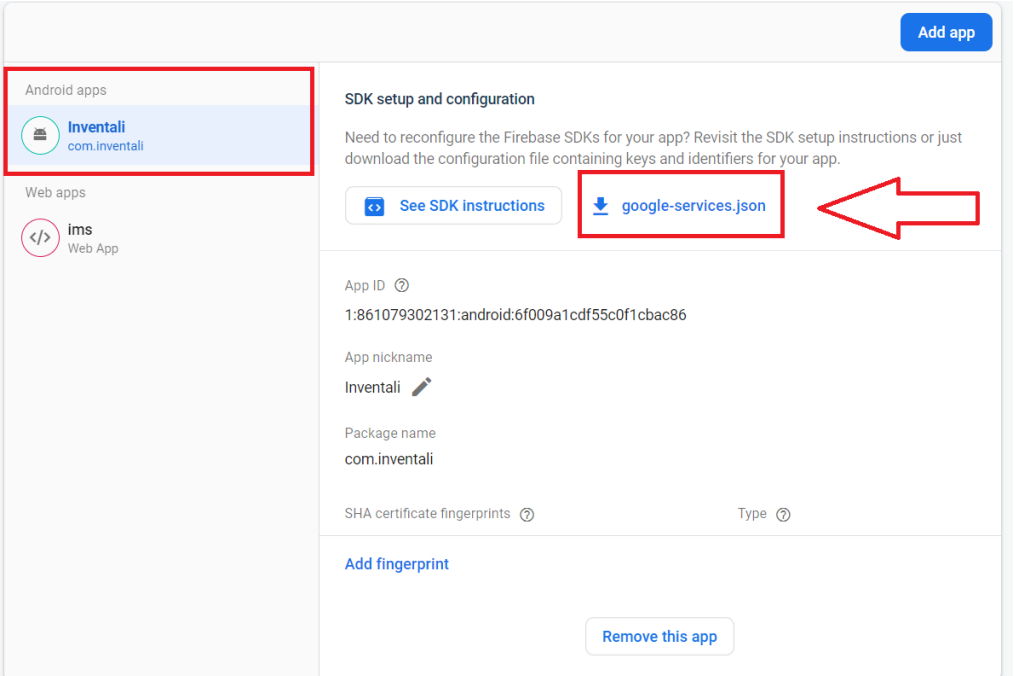


Figure 6.5: Firebase configuration file for Android mobile application

iii. Then, the next step is to move the downloaded google-services.json file and place it inside the React Native project at the following location: /android/app/google-services.json.

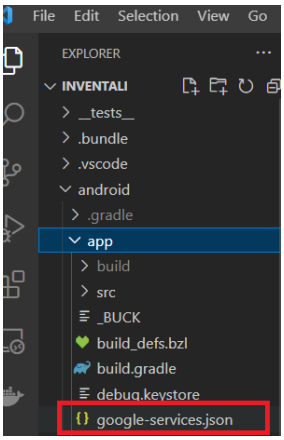


Figure 6.6: Location to place the google-services.json file

iv. The google-services plugin on the project has to be activated for Firebase on Android to access the credentials. Two files in the Android directory must be modified to activate the Firebase.

v. First, in your /android/build.gradle file, add the dependency for the google-services plugin:

```
    repositories {
        google()
        mavenCentral()
    }
    dependencies {
        classpath("com.android.tools.build:gradle:7.1.1")
        classpath("com.facebook.react:react-native-gradle-plugin")
        classpath("de.undercouch:gradle-download-task:5.0.1")
        classpath('com.google.gms:google-services:4.3.15')
        // Add me --- /\
        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}
```

Figure 6.7: Add dependency for the google-services plugin

vi. Add the following line into the /android/app/build.gradle file to finally apply the plugin:

```
apply plugin: "com.android.application"
apply plugin: 'com.google.gms.google-services' // <- Add this line
```

Figure 6.8: Apply the plugin

6.3 System Modules

This project will feature two applications, a web-based application and a mobile-based application, each with its own set of modules. The web and mobile applications have been designed to operate together seamlessly, and their respective modules have been developed to meet the system's requirements.

6.3.1 Modules for Web-based Application

The modules within the web-based application have been designed to facilitate Create, Read, Update, and Delete (CRUD) operations, enabling inventory staff to manage the inventory effectively. In addition to inventory management, the super administrator, also known as the Root administrator, can manage users, roles, and page access permissions. The super administrator has these management capabilities to efficiently supervise and administer the website and other users.

Table 6.1: Modules for mobile-based application

Module
6.3.1.1 Login
6.3.1.2 Dashboard (Homepage)
6.3.1.3 Profile
6.3.1.4 User Management
6.3.1.5 Role Management
6.3.1.6 Right Management (Page Access Management)
6.3.1.7 Inventory Management

6.3.1.1 Login Module

The Login module utilizes Firebase Firestore’s user collection to authenticate the user’s credentials. When users enter their username and password in the

form shown in the figure below, the Login module compares the entered credentials with the user collection in Firebase. This authentication process ensures that only authorized users can access the mobile application.

Figure 6.9: Main login page

The code segment and screenshots below depict the function responsible for authenticating user credentials on a website. Upon entering their username and password, the system first checks for the existence of the username. If the username is not found, the user can contact the administrator to create an account. However, if the username exists, the system retrieves the user's information and compares the entered password with the stored password. The user is redirected to the dashboard (homepage) if the passwords match. If the passwords do not match, the system displays an error message indicating a password mismatch. Additionally, the login form verifies if any input fields are left empty and if any are detected, the user is prevented from logging in.

```
const verifyLogin = async (values) => {
  const usersCollection = collection(db, "users");
  const q = query(usersCollection, where("username", "==", values.username));
  const querySnapshot = await getDocs(q);
  if (querySnapshot.empty) {
    message.error("This username doesn't exist. Please contact administrator to create an account for you");
    return;
  }
  const userDoc = querySnapshot.docs[0];
  if (userDoc.data().password === values.password) {
    const rolesRef = collection(db, "roles");

    // Specify the field and value to search for
    const roleRef = query(rolesRef, where("id", "==", userDoc.data().roleId));
    // Use getDocs to retrieve the matching documents
    getDocs(roleRef)
      .then((roleSnapshot) => {
        roleSnapshot.forEach((role) => {
          const userExpandRole = { ...userDoc.data(), id: userDoc.id, role: role.data() };
          localStorage.setItem("token", JSON.stringify(userExpandRole));
          navigate("/app/home");
        });
      })
      .catch((error) => {
        console.error("Error fetching role document: ", error);
      });
    message.success("You have successfully login", 5);
  } else {
    message.error("Incorrect Password");
    return;
  }
};
```

Figure 6.10: Code segment for login function

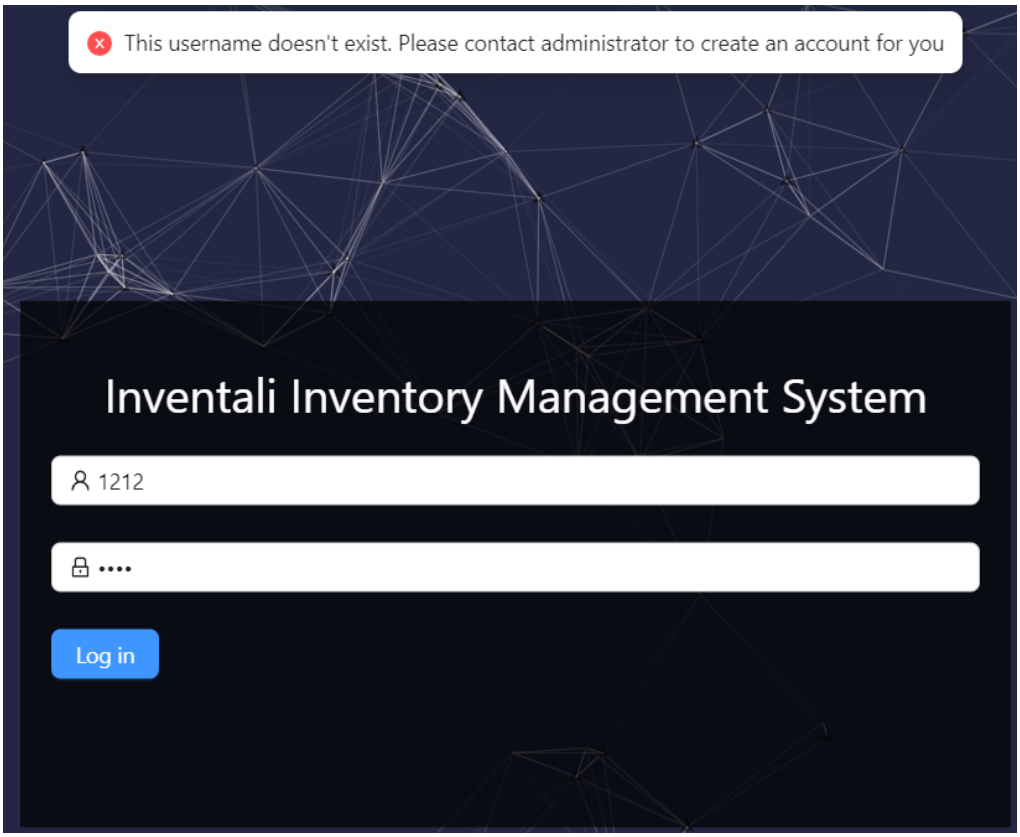


Figure 6.11: Login page showing the username does not exist

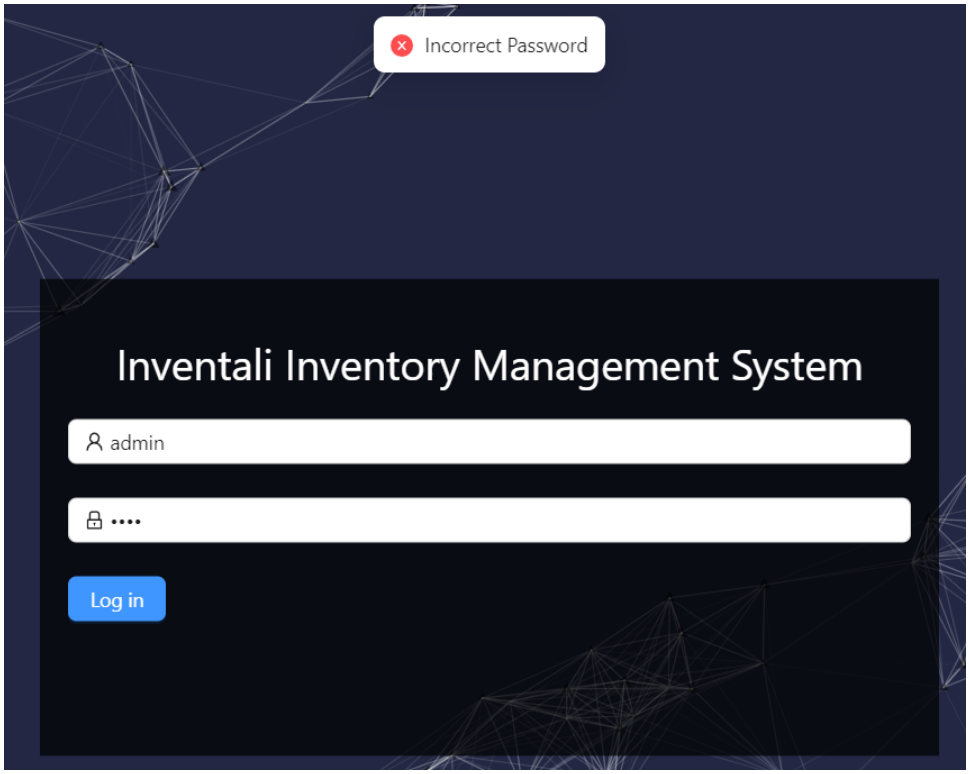


Figure 6.12: Login page showing the password mismatch

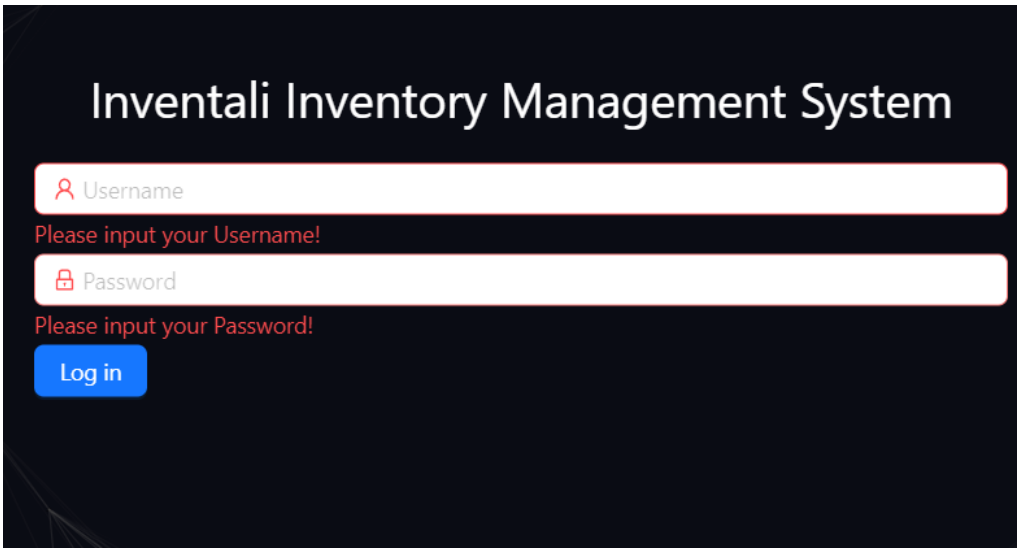


Figure 6.13: Login form verifies for empty input fields

6.3.1.2 Dashboard (Homepage) Module

Upon logging into the system, whether as a super administrator, inventory manager, or inventory staff, the user will be directed to the dashboard, also known as the homepage. Within the dashboard, the user can see an inventory summary, providing an overview of the inventory details. This summary includes the total number of items, total stocks, number of low-stock items,

number of out-of-stock items, list of items below reorder point and list of the out-of-stock item. The lists in the dashboard are interactive, allowing the user to click on any item and be directed to the corresponding item details page, where they can update the inventory item details.

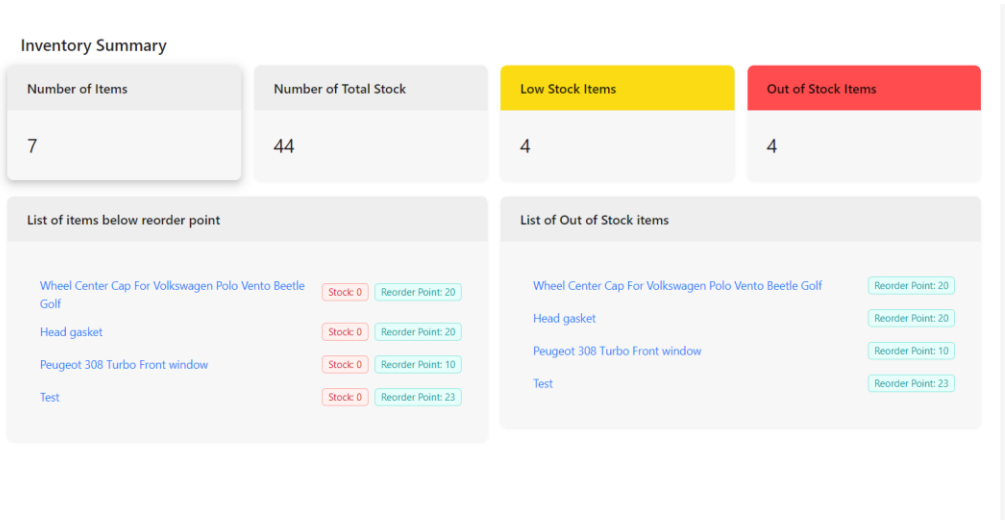


Figure 6.14: Dashboard (Homepage)

The dashboard module utilizes Firebase's Firestore to retrieve all necessary information to display on the dashboard. The information will be retrieved from the inventory collection and set the necessary information to the React state, such as the setTotalStock, setItemsBelowReorderLevelCount, setItemsBelowReorderLevel, setItemsOutOfStockCount and setItemOutOfStock . Once the state is saved, the dashboard module displays this information to the user on the dashboard using the state. The relevant code segment demonstrating this functionality is shown below.

```
const fetchData = async () => {
  try {
    const [inventorySnapshot, outOfStockSnapshot] = await Promise.all([
      getDocs(collection(db, "inventory")),
      getDocs(query(collection(db, "inventory"), where("stock", "<=", 0))),
    ]);

    const inventoryCount = inventorySnapshot.size;
    setInventoryCount(inventoryCount);

    let totalStock = 0;
    let count = 0;
    const belowReorderLevelItems = [];
    const outOfStockItems = [];

    inventorySnapshot.forEach((doc) => {
      const data = doc.data();
      totalStock += data.stock;
      if (data.stock < data.reorderPoint) {
        count++;
        belowReorderLevelItems.push({ id: doc.id, ...data });
      }
    });

    outOfStockSnapshot.forEach((doc) => {
      const data = doc.data();
      outOfStockItems.push({ id: doc.id, ...data });
    });

    setTotalStock(totalStock);
    setItemsBelowReorderLevelCount(count);
    setItemsBelowReorderLevel(belowReorderLevelItems);
    setItemsOutOfStockCount(outOfStockItems.length);
    setItemsOutOfStock(outOfStockItems);
  } catch (error) {
    console.error(error);
    message.error("Error fetching data from database: ", error, 5);
    setLoading(false);
  } finally {
    setLoading(false);
  }
};
```

Figure 6.15: Code segment for retrieving data for the dashboard (homepage)

6.3.1.3 Profile Module

Users can access the profile page and log out from the web application via the user icon located on the top right corner of the website. The profile module will display the user's profile details, as illustrated in the figure below.

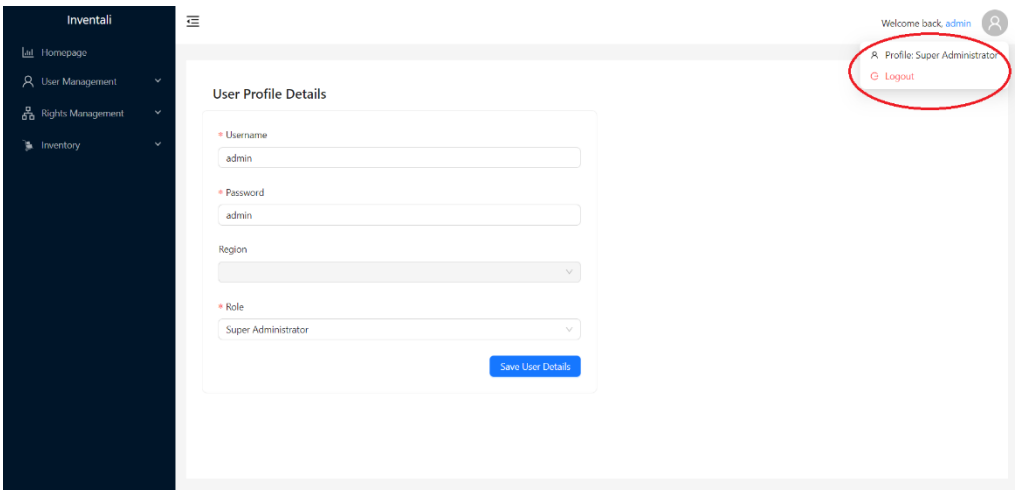


Figure 6.16: User Profile Page

In the figure below, the user module retrieves the user ID from the local storage saved when the user logs in. Using this ID, the user module fetches the other user details from Firebase's Firestore. The response obtained from Firestore is then saved to the React state using setCurrentItem. This response is also set to the form shown above, allowing the user to view and edit their profile details.

```
const getUserData = async () => {
  const { id } = JSON.parse(localStorage.getItem("token"));
  const userRef = doc(db, "users", id);
  getDoc(userRef)
    .then((userDoc) => {
      setCurrentItem(userDoc);
      if (userDoc.exists) {
        setTimeout(() => {
          const userData = userDoc.data();
          updateUserForm.current.setFieldsValue(userData);
          if (userData.roleId === 1) {
            setIsRegionDisabled(true);
          } else {
            setIsRegionDisabled(false);
          }
        }, 0);
      } else {
        console.log("No such document!");
      }
    })
    .catch((error) => {
      console.log("Error getting document:", error);
    });
};
```

Figure 6.17: Code segment for user profile to retrieve user’s details

6.3.1.4 User Management Module

The figure below displays the user management page, which shows a list of users. By default, this user management page is only accessible to the super administrator and inventory manager. The super administrator can perform Create, Read, Update, and Delete (CRUD) operations on any type of user, while the inventory manager can only perform CRUD on inventory staff users.

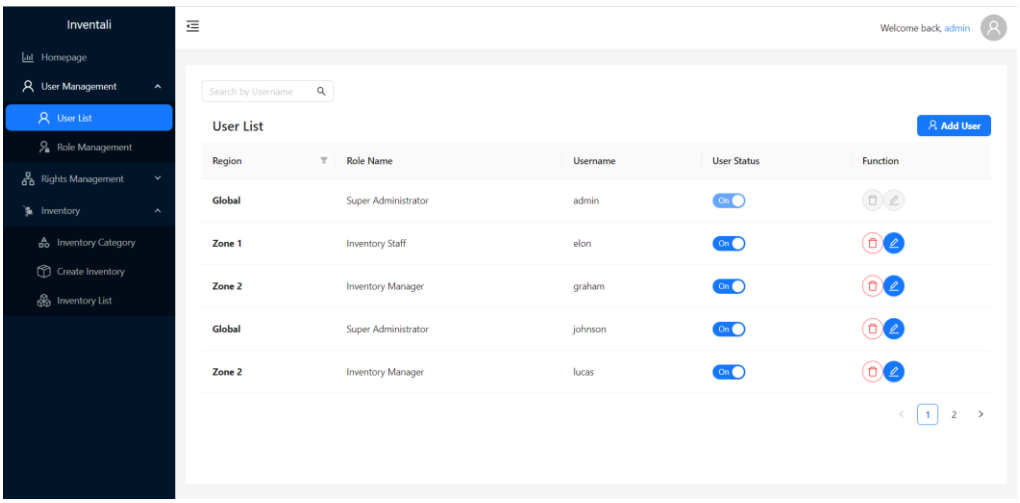


Figure 6.18: User Management Page showing the User List

The code segment below retrieves a list of user data and their roles from Firebase's Firestore, as shown in the figure. The user data and roles are separated into two collections, which must be retrieved separately and then processed to combine later. Once the data is combined, the getUsersExpandRole function returns the combined data.

```
const getUsersExpandRole = async () => {
  const usersCollection = query(collection(db, "users"), orderBy("username"));
  const rolesCollection = query(collection(db, "roles"));
  const usersSnapshot = await getDocs(usersCollection);
  const rolesSnapshot = await getDocs(rolesCollection);
  const usersData = usersSnapshot.docs.map((doc) => ({ ...doc.data(), id: doc.id }));
  const rolesData = rolesSnapshot.docs.map((doc) => ({ ...doc.data() }));

  const joinedData = usersData.map((user) => ({
    id: user.id,
    username: user.username,
    password: user.password,
    roleState: user.roleState,
    default: user.default,
    region: user.region,
    roleId: user.roleId,
    role: rolesData.filter((role) => user.roleId === role.id)[0],
  }));

  return joinedData;
};
```

Figure 6.19: Code segment to retrieve the list of data

The figure below shows a form that enables super administrators and inventory managers to create new users. In the example shown, a new inventory manager user is being created by a super administrator. The form requires the user to enter a unique username, fill in every input field, and make selections as necessary.

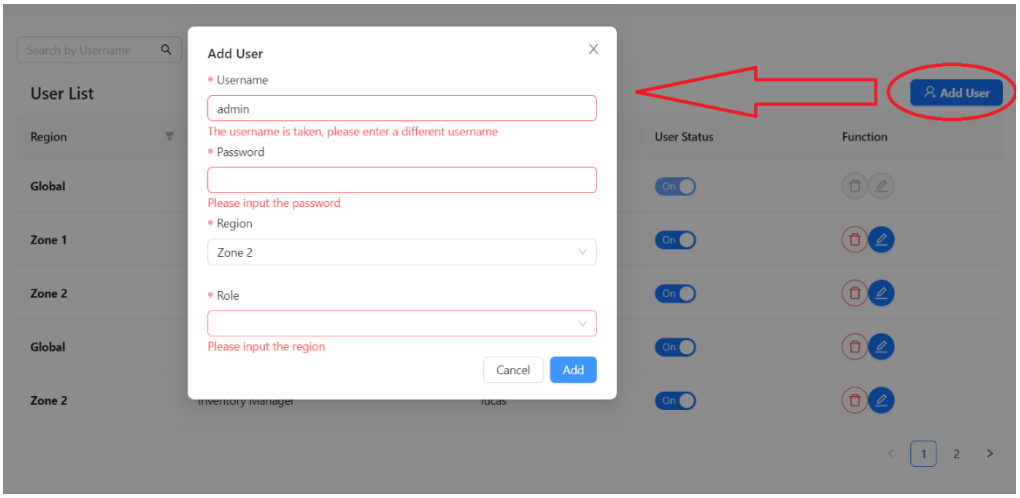


Figure 6.20: Add User Form

The following code segment demonstrates the function used to add a new user. Firstly, the function obtains a reference to the form and validates its fields. The function will display an error message on the screen if the validation fails. If the validation succeeds, the function adds the user's details to Firebase's Firestore using the addDoc function provided by the Firestore SDK. After the user is successfully added to Firestore, a success message is displayed, and the user list is updated accordingly. If the Firestore fails to add the user, an error message will be displayed accordingly.

```
const addUser = () => {
  addUserForm.current
    .validateFields()
    .then(async (value) => {
      console.log(value.username);
      const usersCollectionRef = collection(db, "users");
      addDoc(usersCollectionRef, { ...value, roleState: true, default: false, createdTime: Date.now() })
        .then((docRef) => {
          console.log("Document written with ID: ", docRef.id);
          return getDoc(docRef);
        })
        .then((doc) => {
          console.log("Added document data: ", doc.data(), doc.id);
          message.success(
            <>
              User: <Text strong>{doc.data().username}</Text> successfully added{" "}
            </>,
            5
          );
          setDataSource([...dataSource, { ...doc.data(), role: roleList.filter((item) => item.id === doc.data().roleId)[0], id: doc.id }]);
          setIsAddModalOpen(false);
          addUserForm.current.resetFields();
        })
        .catch((error) => {
          message.error("Error in adding the user, please try again");
          setIsAddModalOpen(false);
          console.error("Error adding document: ", error);
        });
    })
    .catch((err) => {
      message.error("Error in adding the user, please try again");
      setIsAddModalOpen(false);
      console.log(err);
    });
};
```

Figure 6.21: Code segment to add a user

The figure below displays the user details update form, accessible by the super administrator or inventory manager. This feature is similar to the "add user" function and allows for the update of user details. The form requires all input fields to be filled out, and the user must use a unique username to update an existing user.

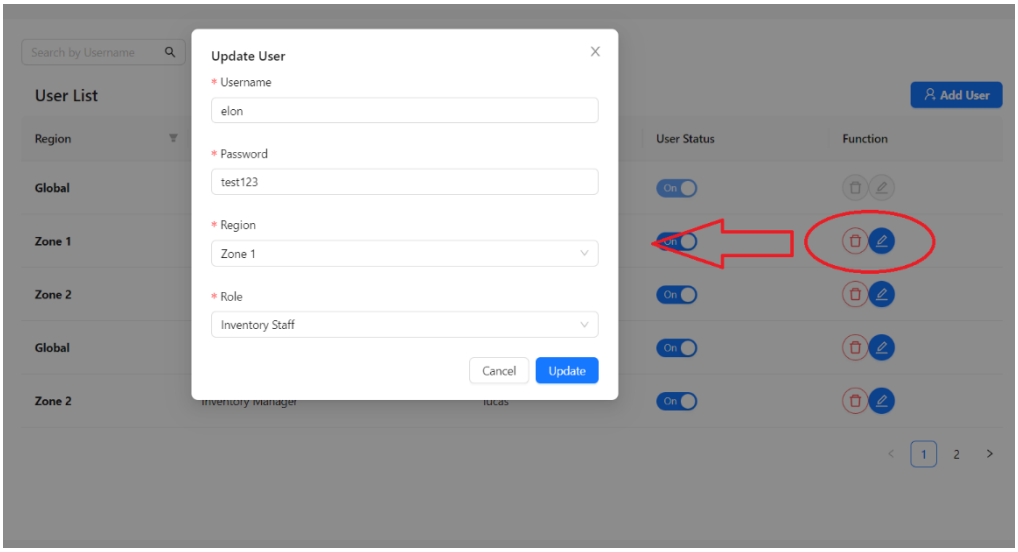


Figure 6.22: Update User Form

The figure below depicts the update function. It will be triggered when the user clicks the update button. The updateUserOk function begins by

using the doc() function to obtain a reference to the specified user that requires editing. This doc() function accepts three parameters: the database reference, the collection name, and the document ID that needs to be updated. Next, the obtained reference is used in the updateDoc() function, which accepts two parameters: the reference to the document (data) and the value in JSON object format. If the update action is successful, the modal will be closed, the user list will be updated accordingly, and a success message will be displayed. An error message will be displayed in the event of a failure. The doc() and updateDoc() function is provided by the Firebase SDK.

```
const updateUserOk = () => {
  updateUserForm.current.validateFields().then((value) => {
    setIsRegionDisabled(!isRegionDisabled);
    const userRef = doc(db, "users", currentItem.id);
    updateDoc(userRef, value)
      .then(() => {
        setIsUpdateModalOpen(false);
        setDataSource(
          dataSource.map((item) => {
            if (item.id === currentItem.id) {
              return { ...item, ...value, role: roleList.filter((data) => data.id === value.roleId)[0] };
            }
            return item;
          })
        );
        message.success(`User details for ${currentItem.username} updated successfully`, 5);
      })
      .catch((error) => {
        message.error(`Error updaing the user details for ${currentItem.username}`, 5);
        console.error("Error updating document: ", error);
      });
  });
};
```

Figure 6.23: Update User Form

The function in the figure below is used to delete a user. When the function is triggered, a confirmation modal will be displayed to confirm the delete action.

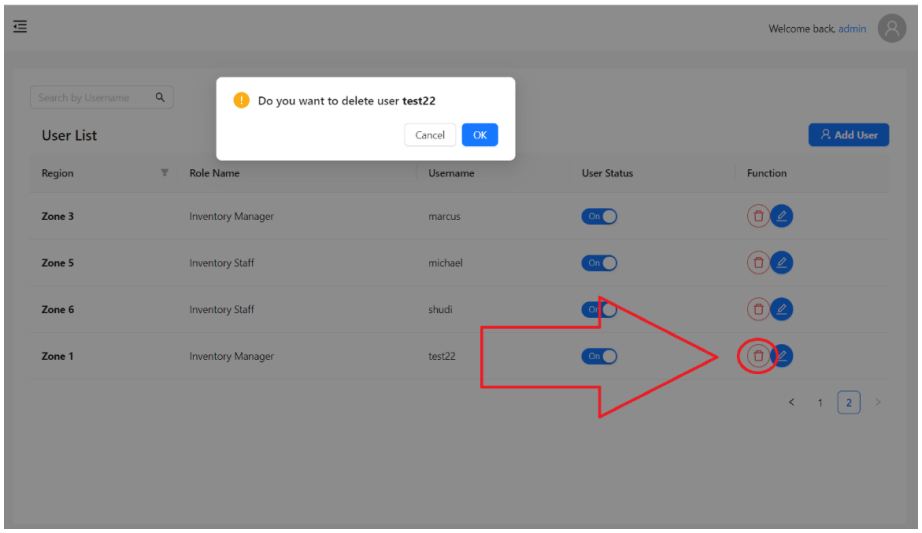


Figure 6.24: Delete User Function

If the user confirms the delete action, the function will obtain a reference to the user document using doc() function and then use the deleteDoc() function provided by Firebase's Firestore SDK to delete the specified user. If the delete action is successful, the user list will be updated accordingly using the JavaScript filter() function to filter out the deleted user, and a success message will be displayed. If the delete action fails, an error message will be displayed to the user.

```
const deleteUser = (item) => {
  const userDocRef = doc(db, "users", item.id);
  deleteDoc(userDocRef)
    .then(() => {
      message.success(`User ${item.username} deleted successfully `);
      setDataSource(dataSource.filter((data) => data.id !== item.id));
    })
    .catch((error) => {
      message.error(`Error deleting user ${item.username}`);
      console.error("Error deleting document: ", error);
    });
};
```

Figure 6.25: Code segment to delete user

The figure below illustrates two features, the searching and the sorting & filter feature. The Algolia extension powers the search feature, allowing users to search for specific user data in the table using the username. While the sorting & filtering feature is provided by the Ant Design component, allowing users to sort the table user data in ascending or descending order and use filters based on specific columns. These features allow users to easily find and organize the needed data, making their user management tasks more efficient.

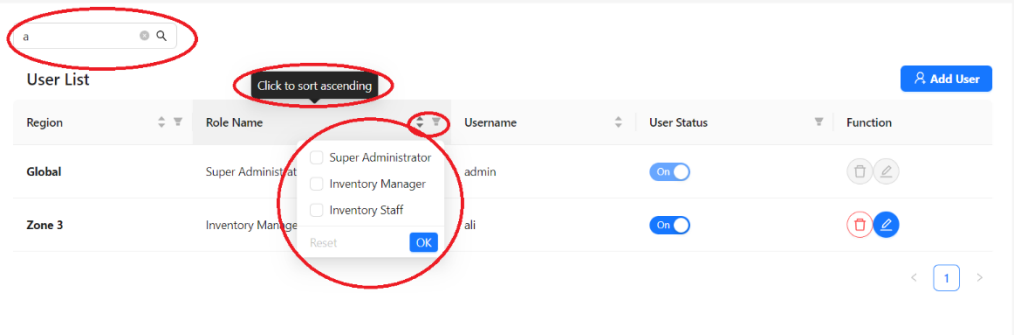


Figure 6.26: Searching and Sorting & filtering function

To use the Algolia Search Service in the user list screen, the Algolia JavaScript API Client must be installed through the npm package manager.

```
npm install algoliasearch
```

After installing the API client, it needs to be imported, and an instance of the API client must be created using the algoliasearch() function. The first parameter is the application ID, and the second is the API key. As mentioned in the project setup, these parameters can be obtained from the Algolia website. Once the API client instance is created, it can be used to obtain a reference to the "inventory" index that was created earlier. This index was also mentioned in the project setup.

```
//AlgoliaSearch
import algoliasearch from "algoliasearch";
const client = algoliasearch("8YK[redacted]", "[redacted]853a3a53");
const index = client.initIndex("inventory");
```

Figure 6.27: Initialization of an instance of the Algolia API client

The code segment shown in the figure below represents the search function. Firstly, the function checks the search box. If the search box is empty, the function retrieves the user data directly from the Firestore using the function getUsersExpandRole, which was mentioned at the beginning of this user management module to obtain the list of user data. However, if the search box contains any search query (any inputs), the function uses the Algolia SDK to perform a search by calling the index.search() function. The index.search() function returns a user data response that matches the value in the search box. The search function allows users to search for a specific user based on their search input, simplifying the process of finding the desired user.

```
if (searchQuery.trim() === "") {
  getUsersExpandRole()
    .then((usersExpandRole) => {
      const list = usersExpandRole;
      setDataSource(
        roleObj[roleId] === "superAdmin"
          ? list
          : [
              ...list.filter((item) => item.username === username),
              ...list.filter((item) => item.region === region && roleObj[item.roleId] === "branchAdmin"),
            ]
      );
    })
    .then(() => setSekeletonLoading(false))
    .catch((error) => {
      message.error("Error fetching users details from database", 5);
      console.error("Error fetching users: ", error);
    });
} else {
  index
    .search(searchQuery.trim())
    .then(async ({ hits }) => {
      const rolesCollection = query(collection(db, "roles"));
      const rolesSnapshot = await getDocs(rolesCollection);
      const rolesData = rolesSnapshot.docs.map((doc) => ({ ...doc.data() }));
      const list = hits.map((hit) => {
        let result = hit;
        result.role = rolesData.filter((role) => hit.roleId === role.id)[0];
        result.id = hit.objectID;
        return result;
      });
      setDataSource(
        roleObj[roleId] === "superAdmin"
          ? list
          : [
              ...list.filter((item) => item.username === username),
              ...list.filter((item) => item.region === region && roleObj[item.roleId] === "branchAdmin"),
            ]
      );
    })
    .then(() => setSekeletonLoading(false))
    .catch((err) => {
      message.error("Failed to search the user, please try again", 5);
      console.error(err);
    });
}
```

Figure 6.28: Code segment to search user using a username

The code segment below demonstrates how to implement filter and sorter functions for the user list. The "filters" attribute shown in the JSON object generates a filter menu in columns, allowing users to filter data based on a specific column. The "onFilter" function displays the filtered result. Additionally, the "sorter" function makes a column sortable, which allows users to sort data in ascending or descending order based on a specific column.

```
const columns = [
  {
    title: "Region",
    dataIndex: "region",
    filters: [...regionList.map((item) => ({ text: item.title, value: item.value })), { text: "Global", value: "Global" }],
    onFilter: (value, item) => {
      if (value === "Global") {
        return item.region === "";
      }
      return item.region === value;
    },
    sorter: (a, b) => a.region.localeCompare(b.region),
    key: "region",
    render: (region) => {
      return <b>{region === "" ? "Global" : region}</b>;
    },
  },
  {
    title: "Role Name",
    dataIndex: "role",
    filters: [...roleList.map((item) => ({ text: item.roleName, value: item.roleType }))],
    sorter: (a, b) => a.roleId - b.roleId,
    onFilter: (value, item) => {
      return item.roleId === value;
    },
    key: "role",
    render: (role) => {
      return role?.roleName;
    },
  },
```

Figure 6.29:Code segment to sort and filter user using user's details

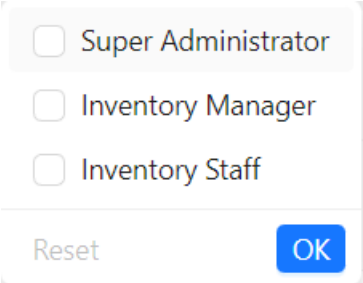


Figure 6.30: Filter menu

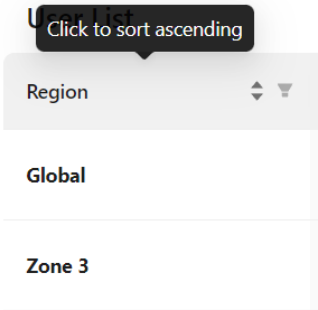


Figure 6.31: Sorter

6.3.1.5 Role Management Module

The figure below shows the Role Management Page with a Role Management List. This Role Management List aims to control and limit access to specific pages or features within this web application to each role. This allows the super administrators to grant or restrict access to certain pages or functionality based on the user's role or permission level. By implementing page access management, businesses can ensure that sensitive information or critical operations are only accessible to authorized individuals and reduce the risk of unauthorized access or misuse.

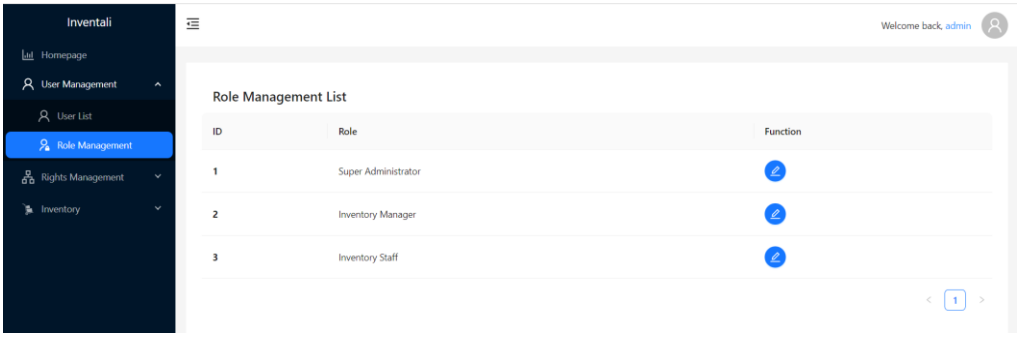


Figure 6.32: Role management page

The code segment shown below retrieves the list of roles from Firestore, which is similar to the process of getting the user list mentioned in the previous user management module. When the getRoles function is called, it uses the collection, query, and orderBy functions provided in the Firebase SDK to retrieve the list of roles data ordered by their id. The query function returns a reference to the list of roles data, and then the getDocs function is used to retrieve multiple documents from the roles collection. The getDocs function responds with the resolved data, and the data must be mapped out using the JavaScript map() function in order to set the roles data to the React state for displaying.

```
const getRoles = async () => {
  const rolesCollectionRef = query(collection(db, "roles"), orderBy("id"));
  const rolesSnapshot = await getDocs(rolesCollectionRef);
  const rolesData = rolesSnapshot.docs.map((doc) => ({ ...doc.data(), docId: doc.id }));
  setDataSource(rolesData);
};
```

Figure 6.33: Code segment to get the list of roles from Firestore

The figure below shows the Access Assignment for the Super Administrator, Inventory Manager, and Inventory Staff in default mode. As shown, Super administrators have complete control over all the features on the website, whereas other types of users may have limited access control over certain features. The Access Assignment for each role can only be modified by the Super administrator in default mode. The Super administrator may customize the access assignment for each role per their preferences.

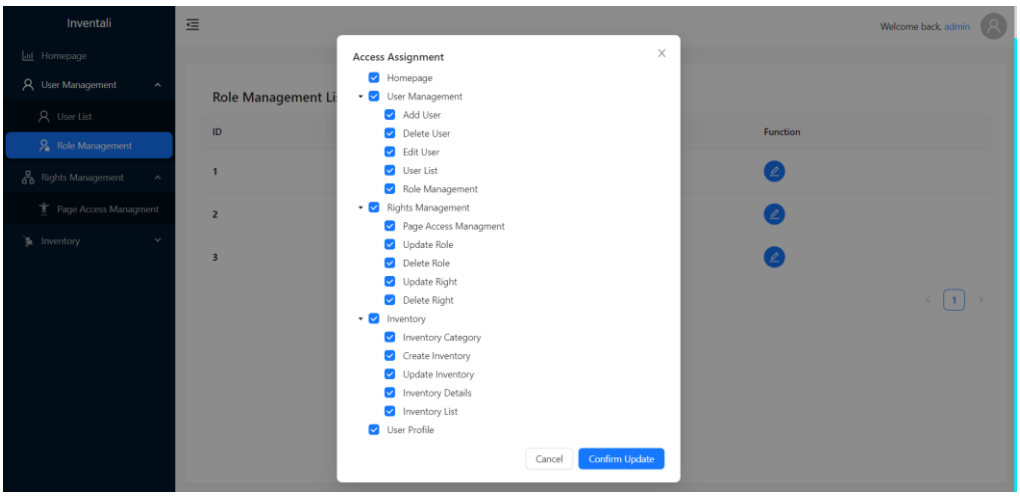


Figure 6.34: Access assignment for super administrator

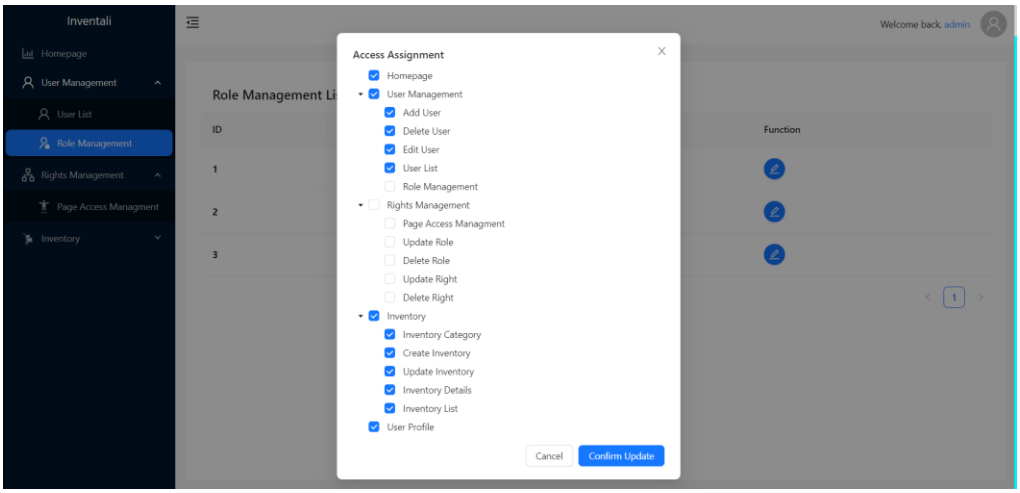


Figure 6.35: Access assignment for inventory manager

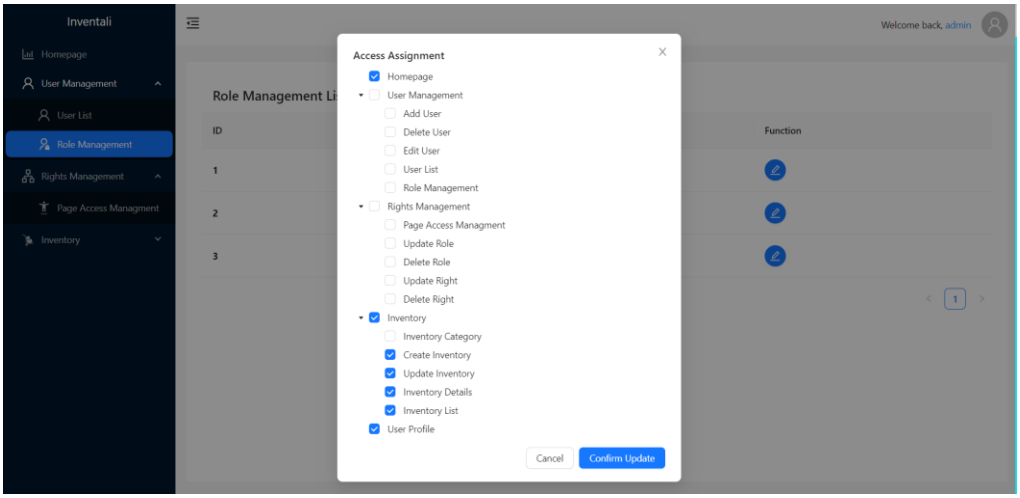


Figure 6.36: Access assignment for inventory staff

The following code segment contains the function getRightsEmbedChildren. This function retrieves data from the rights and children collections and combines them into a single list of objects. Each object in the list contains the rights data and its corresponding children data. The resulting list of objects is then set to the accessList state, which can be later used in the Ant Design's Tree Component.

```
const getRightsEmbedChildren = async () => {
  const rightsCollection = collection(db, "rights");
  const sortedRightsCollection = query(rightsCollection, orderBy("id"));
  const childrenCollection = collection(db, "children");
  const sortedChildrenCollection = query(childrenCollection, orderBy("id"));

  const rightsSnapshot = await getDocs(sortedRightsCollection);
  const rightsData = rightsSnapshot.docs.map((doc) => ({ ...doc.data() }));

  const childSnapshot = await getDocs(sortedChildrenCollection);
  const childData = childSnapshot.docs.map((doc) => ({ ...doc.data() }));
  //  combine the data from both collections based on the common "childId" field
  const joinedData = rightsData.map((right) => ({
    id: right.id,
    title: right.title,
    key: right.key,
    pagePermission: right.pagePermission,
    grade: right.grade,
    children: childData.filter((child) => right.id === child.rightId),
  }));
  setAccessList(joinedData);
```

Figure 6.37: Code segment to get the access assignment data from the Firestore

The Modal with the checkbox shown above is achieved using the help of Ant Design’s Tree Component and Modal Component. The Tree

component is wrapped in the modal component, as shown in the code segment below.

```
    </Skeleton>
    <Modal title="Access Assignment" open={isModalOpen} onOk={handleOk} onCancel={handleCancel}>
      <Tree checkable treeData={accessList} checkedKeys={currentSelectedAccess} onCheck={onCheck} checkStrictly={true} />
    </Modal>
  </Spin>
```

Figure 6.38: Code segment of implementing the Modal component and Tree component by the Ant Design

The following code segment shows a function that handles the update of Access Assignments when the user clicks the "Confirm Update" button. Firstly, the function obtains a reference to the specific role document that needs to be updated using the doc() function. Then, the updateDoc() function is used to update the role's rights access using the currentSelectedAccess array, which contains the selected access retrieved from the Ant Design's Tree Component. Once the update is complete, the modal is closed, and the checkbox is updated accordingly on the front end. An error message will be displayed if there is an update error.

```
const handleOk = () => {
  const roleRef = doc(db, "roles", currentItem.docId);
  updateDoc(roleRef, {
    rights: currentSelectedAccess,
  })
    .then(() => {
      setIsModalOpen(false);
      setDataSource(
        dataSource.map((item) => {
          if (item.id === currentItem.id) {
            return {
              ...item,
              rights: currentSelectedAccess,
            };
          }
          return item;
        })
      );
      message.success(`Access assignment for role: ${currentItem.roleName} has been successfully updated`, 5);
    })
    .catch((error) => {
      setIsModalOpen(false);
      message.error(`Error updating the access assignment for role: ${currentItem.roleName}`, 5);
      console.error("Error updating document: ", error);
    });
```

Figure 6.39: Code segment to update access assignment for any role

6.3.1.6 Right Management (Page Access Management) Module

The figure below displays the Page Access Management List, which differs from the Role Access Management List as it is used to turn on and off access for specific pages so that the access will apply to all types of users. This list allows the administrators to set page access according to the business operation needs.

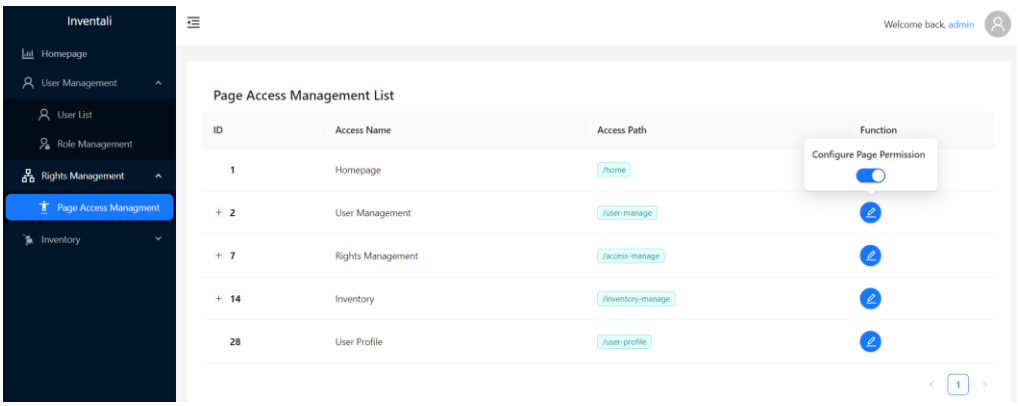


Figure 6.40: Page access management page

As shown in the figure below, the access control for some functions cannot be turned on and off since they are not physical pages but rather functions, such as the "Add User", "Delete User", and "Edit User" functions in the user management list.

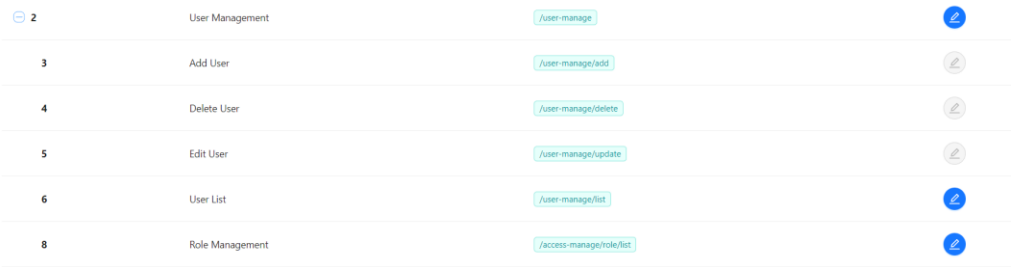


Figure 6.41: Page access management page expanded view

The code segment below shows the function used to switch the page permission of a specified page. The function switches the page permission based on the type of page, as the pages are separated into two collections. The parent page is stored in the rights collection, while the child pages are stored in the children collection. Similarly, this function will use the Firebase SDK to update the document.

```
const switchAccess = async (item) => {
  item.pagePermission = item.pagePermission === 1 ? 0 : 1;
  if (item.grade === 1) {
    const rightsCollectionRef = collection(db, "rights");
    const q = query(rightsCollectionRef, where("id", "==", item.id));
    getDocs(q)
      .then((querySnapshot) => {
        querySnapshot.forEach((docSnapshot) => {
          const docRef = doc(db, "rights", docSnapshot.id);
          updateDoc(docRef, { pagePermission: item.pagePermission })
            .then(() => {
              setDataSource([...dataSource]);
              message.success("Access permission updated successfully", 5);
            })
            .catch((error) => {
              message.error("Error upading access permission", 5);
              console.error("Error updating document: ", error);
            });
        });
      })
      .catch((error) => {
        message.error("Error upading access permission", 5);
        console.error("Error fetching documents: ", error);
      });
  } else {
    const childrenCollectionRef = collection(db, "children");
    const q = query(childrenCollectionRef, where("id", "==", item.id));
    getDocs(q)
      .then((querySnapshot) => {
        querySnapshot.forEach((docSnapshot) => {
          const docRef = doc(db, "children", docSnapshot.id);
          updateDoc(docRef, { pagePermission: item.pagePermission })
            .then(() => {
              setDataSource([...dataSource]);
              message.success("Access permission updated successfully", 5);
            })
            .catch((error) => {
              message.error("Error upading access permission", 5);
              console.error("Error updating document: ", error);
            });
        });
      })
      .catch((error) => {
        message.error("Error upading access permission", 5);
        console.error("Error fetching documents: ", error);
      });
  }
};
```

Figure 6.42: Update Access Function

6.3.1.7 Inventory Management Module

The figure below displays the Inventory Category List which includes features such as adding a new category, editing an existing category, and searching for a category.

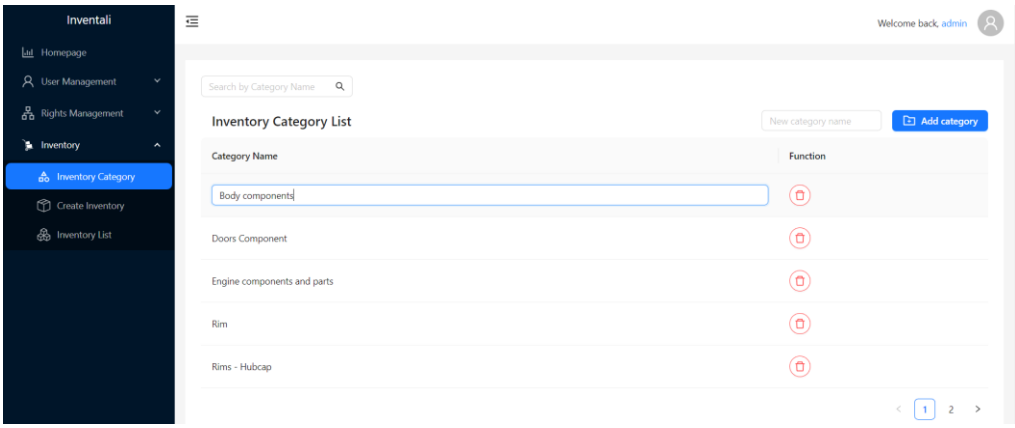


Figure 6.43: Updating an inventory in the inventory category list

The following code segment shows the function responsible for adding a category to the Firestore database. The addDoc() function from the Firebase SDK is used to achieve this. Once the category data list is obtained, it is set to the React state for later use in the Ant Design Table Component.

```
const addCategory = async (values) => {
  setSkeletonLoading(true);
  const categoryRef = collection(db, "category");
  await addDoc(categoryRef, { title: values.title, value: values.title })
    .then(() => getCategory().then((data) => setCategory(data)))
    .then(() => setSkeletonLoading(false))
    .then(() => message.success(`Category '${values.title}' has been added.`, 5))
    .catch((error) => {
      console.error("Error adding document: ", error);
      message.error(`Error adding category '${values.title}'.`, 5);
    });
};
```

Figure 6.44: Code segment for adding a category

The following code segment shows the function responsible for updating categories in the Firestore database. Similarly, this function uses the updateDoc() function from the Firebase SDK to update the category data.

```
const handleSave = async (record) => {
  console.log(record);
  setSkeletonLoading(true);

  const categoryRef = doc(db, "category", record.id);
  await updateDoc(categoryRef, { title: record.title, value: record.title })
    .then(
      () =>
        setCategory(
          category.map((item) => {
            if (item.id === record.id) {
              return { id: item.id, title: record.title, value: record.title };
            }
            return item;
          })
        ),
      setSkeletonLoading(false)
    )
    .then(() =>
      message.success(
        <>
          Category <Text strong>{record.title}</Text> has been successfully updated
        </>,
        5
      )
    );
};
```

Figure 6.45: Code segment for updating the existing category

The following code segment contains the search function responsible for searching for categories. If the search query input is empty, the function fetches the data directly from the Firestore. However, if the search query contains any input, the function fetches data from the Algolia server instead.

```
if (searchQuery.trim() === "") {
  fetchData();
} else {
  index
    .search(searchQuery.trim())
    .then(({ hits }) => {
      const list = hits.map((hit) => {
        let result = hit;
        result.id = hit.objectID;
        return result;
      });
      setCategory(list);
    })
    .then(() => setSkeletonLoading(false))
    .catch((err) => {
      console.error(err);
    });
}
```

Figure 6.46: Code segment for searching category

The image below depicts the inventory item creation form. This is the initial page where users must enter and select basic inventory information, which is the item name and category.

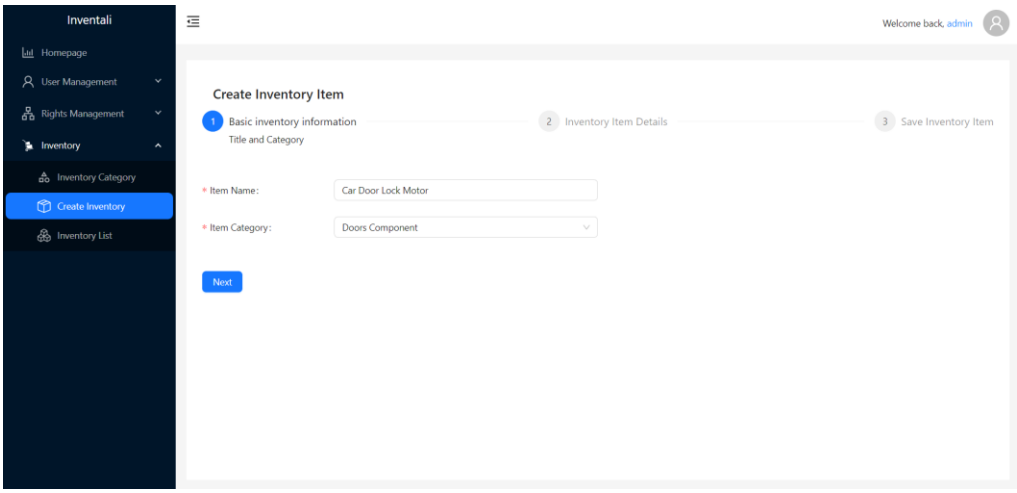


Figure 6.47: Create inventory form (basic inventory information)

The following image displays the second page of the inventory item creation form. On this page, users are required to enter and select inventory item details, including opening stock, reorder point, storage box number, sub-compartment, weight, dimensions, brand, model, vendor, description, note, and inventory image. This form also validates that these fields are required for input.

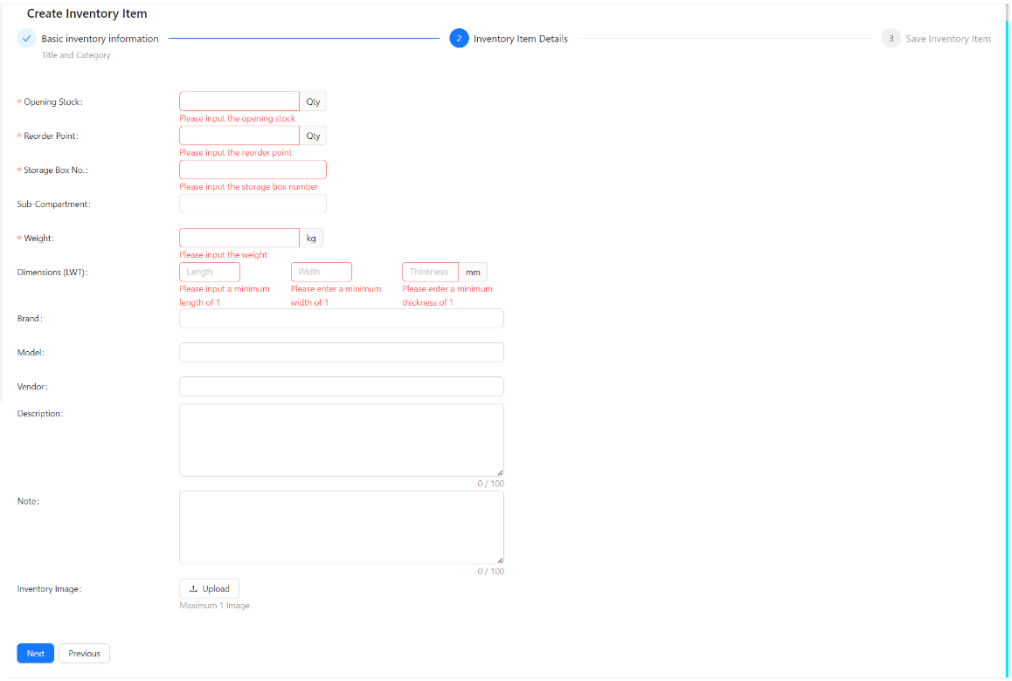


Figure 6.48: Create inventory form (inventory item details)

The final page of the inventory item creation form is the confirmation page.

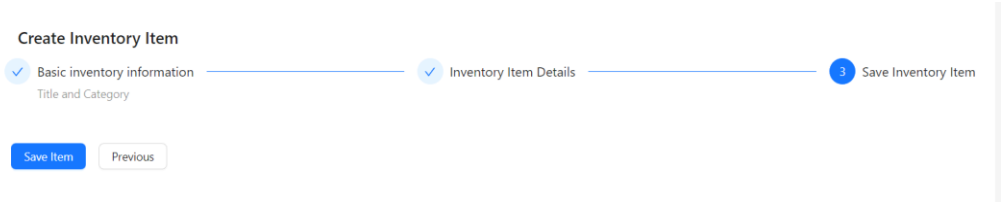


Figure 6.49: Save the inventory details

The code segment below shows the function for saving an inventory item. Firstly, the function obtains the reference to the inventory collection for later use in the addDoc() function. As previously mentioned, two pages of information need to be filled to create an inventory item, represented by two forms. The function retrieves all field values from both forms and sets them to corresponding variables. There are two cases to consider: if the user decides to create an inventory item without an inventory image, the function will call the addDoc() function to upload these inventory details. However, if the user creates an inventory item with an inventory image, the function will call an additional function, uploadBytes(), to upload the photo to a Firebase server known as Cloud Storage. After the inventory item is successfully created, the function will navigate the user to the inventory list and display a success message. An error message will be displayed if the function fails to create the inventory item.

```
const saveInventory = () => {
  const inventoryCollectionRef = collection(db, "inventory"); // reference to the collection
  try {
    const inventoryFormData = inventoryForm.getFieldValue();
    const addInventoryDetailsFormData = addInventoryDetailsForm.current.getFieldValue();
    const upload = addInventoryDetailsFormData.upload;
    console.log(upload);
    if (upload === undefined) {
      delete addInventoryDetailsFormData.upload;
      addDoc(inventoryCollectionRef, {
        ...inventoryFormData,
        ...addInventoryDetailsFormData,
        createdTime: Date.now(),
        region: User.region ? User.region : "Global",
        createdBy: User.id,
        roleId: User.roleId,
        imageURL: null,
      })
        .then(() => navigate("/app/inventory-manage/list"))
        .then(() => message.success("Inventory successfully added", 5));
    } else {
      const imageRef = ref(storage, `images/inventory/${upload[0].uid}`);
      uploadBytes(imageRef, upload[0].originFileObj)
        .then((snapshot) => {
          return getDownloadURL(snapshot.ref);
        })
        .then((downloadURL) => {
          delete addInventoryDetailsFormData.upload;
          addDoc(inventoryCollectionRef, {
            ...inventoryFormData,
            ...addInventoryDetailsFormData,
            createdTime: Date.now(),
            region: User.region ? User.region : "Global",
            createdBy: User.id,
            roleId: User.roleId,
            imageURL: downloadURL,
          })
            .then(() => navigate("/app/inventory-manage/list"))
            .then(() => message.success("Inventory successfully added", 5));
        });
    }
  } catch (error) {
    message.error("Error adding the inventory item");
    console.log(error);
  }
};
```

Figure 6.50: Code segment to create an inventory item

After a user creates an inventory item, it will be displayed in the inventory list. Users can utilize the search box to search for inventory items by the item name. Each column, except for the category column, has a sorting feature. The category column instead implements a filtering feature to allow for easier categorization and organization of the inventory list.

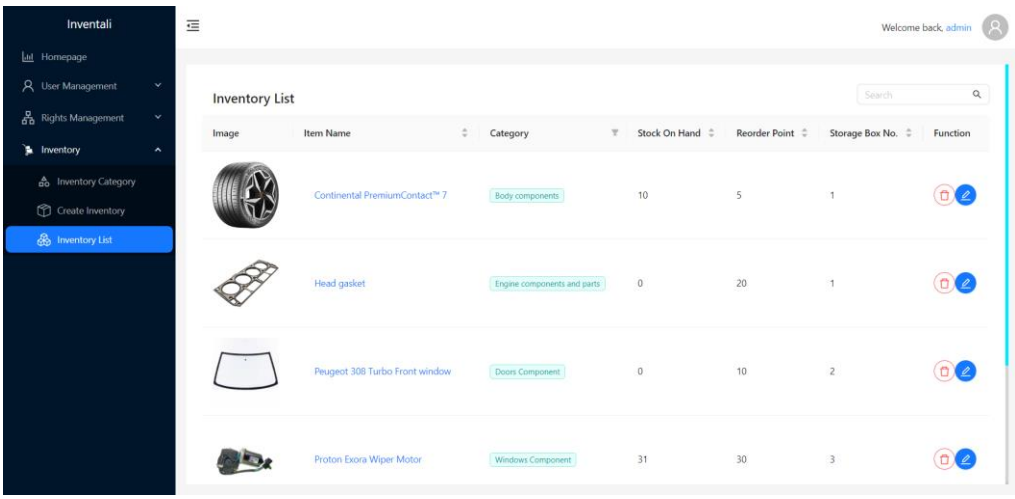


Figure 6.51: Inventory list

The following code segment retrieves a list of inventory data from Firebase. First, the function obtains a reference to the inventory collection. Then, it uses the query function to retrieve inventory data from the collection, sorting the inventory list in ascending order based on the item name using the orderBy function. Next, the getDocs function uses the query reference to retrieve the data in an array of JSON objects format. Finally, the function processes the data and returns it, which will be used by the Ant Design Table Component to display the inventory data.

```
const getData = async () => {
  const inventoryCollectionRef = collection(db, "inventory"); // reference to the collection
  const q = query(inventoryCollectionRef, orderBy("itemName"));
  try {
    const data = await getDocs(q);
    const filteredData = data.docs.map((doc) => {
      let inventories = doc.data();
      inventories.objectID = doc.id;
      return inventories;
    });
    return filteredData;
  } catch (error) {
    console.log(error);
  }
};
```

Figure 6.52: Code segment obtains data for the inventory list

Below are three figures illustrating the process of updating an inventory item. To initiate the update process, the user clicks on the blue pen edit icon, as circled in Figure 6.44.

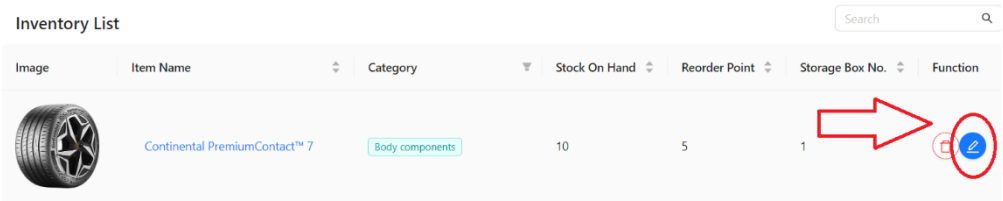


Figure 6.53: Edit inventory details

Once the user clicks on the edit icon, they will be directed to an update inventory item form similar to the form used to create an inventory item. In this update form, users can see the existing details that were previously saved. Users may choose to edit the existing details if needed or keep them unchanged.

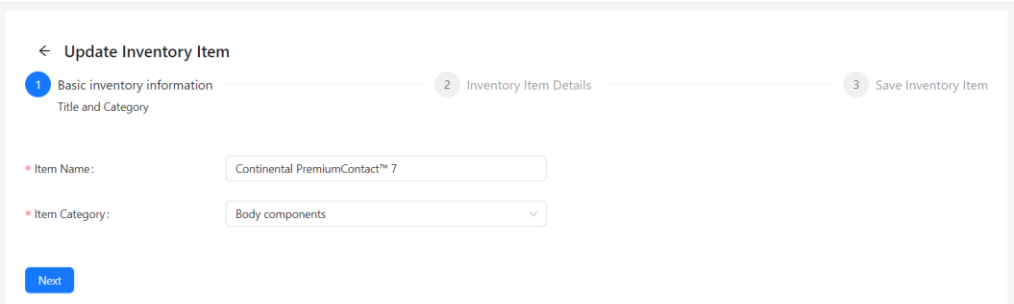


Figure 6.54: First page for updating inventory basic information

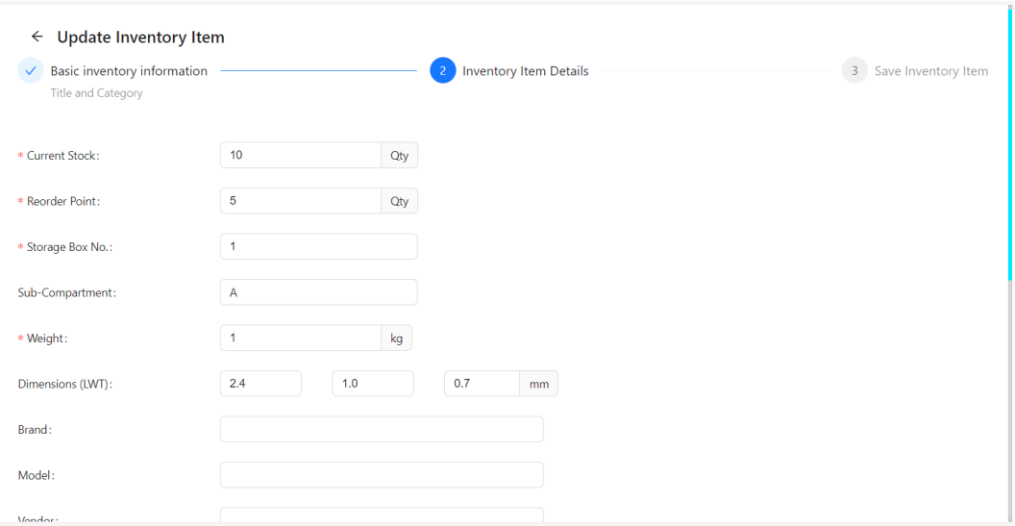


Figure 6.55: Second page for updating inventory item details

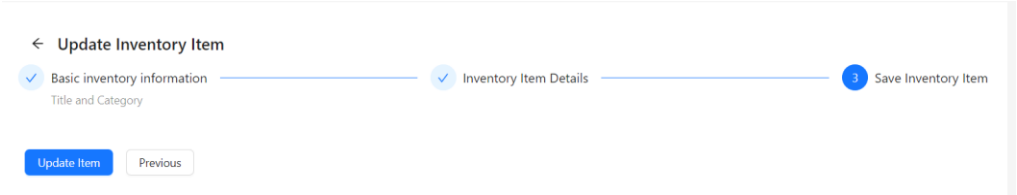


Figure 6.56: Confirmation page for updating an inventory item

The code segment below is similar to the createInventory function discussed earlier, which is used to create a new inventory item. However, there are some differences between these two functions, so the createInventory function cannot be reused here. First, this function obtains a reference to the inventory item using the inventory document ID. Then, it retrieves all field values from the form and sets them to corresponding variables. There are two conditions that can occur: the user updates the inventory without an inventory image, or the user updates the inventory with an image. As shown in the code below, there is an if-else conditional statement to handle the inventory image upload separately. The function uses the updateDoc() function to update the inventory with the new edited data.

```
const updateInventory = async () => {
  const inventoryDoc = doc(db, "inventory", inventoryId.id);
  try {
    setLoading(true);
    const inventoryFormData = inventoryForm.getFieldValue();
    const inventoryDetailsFormData = inventoryDetailsForm.getFieldValue();
    const upload = inventoryDetailsFormData.upload;
    console.log(upload);
    if (upload === undefined || upload.length === 0) {
      delete inventoryDetailsFormData.upload;
      updateDoc(inventoryDoc, {
        ...inventoryFormData,
        ...inventoryDetailsFormData,
        subCompartment: inventoryDetailsFormData.subCompartment ? inventoryDetailsFormData.subCompartment : null,
        brand: inventoryDetailsFormData.brand ? inventoryDetailsFormData.brand : null,
        model: inventoryDetailsFormData.model ? inventoryDetailsFormData.model : null,
        vendor: inventoryDetailsFormData.vendor ? inventoryDetailsFormData.vendor : null,
        description: inventoryDetailsFormData.description ? inventoryDetailsFormData.description : null,
        note: inventoryDetailsFormData.note ? inventoryDetailsFormData.note : null,
        updatedTime: Date.now(),
        region: User.region ? User.region : "Global",
        updatedBy: User.id,
        roleId: User.roleId,
        ...(isFileRemoved ? { imageURL: null } : {}),
      })
        .then(() => navigate("/app/inventory-manage/list"))
        .then(() => message.success("Inventory successfully updated", 5))
        .then(() => setLoading(false));
    } else {
      const imageRef = ref(storage, `images/inventory/${upload[0].uid}`);
      uploadBytes(imageRef, upload[0].originFileObj)
        .then((snapshot) => {
          return getDownloadURL(snapshot.ref);
        })
        .then((downloadURL) => {
          delete inventoryDetailsFormData.upload;
          updateDoc(inventoryDoc, {
            ...inventoryFormData,
            ...inventoryDetailsFormData,
            subCompartment: inventoryDetailsFormData.subCompartment ? inventoryDetailsFormData.subCompartment : null,
            brand: inventoryDetailsFormData.brand ? inventoryDetailsFormData.brand : null,
            model: inventoryDetailsFormData.model ? inventoryDetailsFormData.model : null,
            vendor: inventoryDetailsFormData.vendor ? inventoryDetailsFormData.vendor : null,
            description: inventoryDetailsFormData.description ? inventoryDetailsFormData.description : null,
            note: inventoryDetailsFormData.note ? inventoryDetailsFormData.note : null,
            updatedTime: Date.now(),
            region: User.region ? User.region : "Global",
            createdBy: User.id,
            roleId: User.roleId,
            imageURL: downloadURL,
          })
            .then(() => navigate("/app/inventory-manage/list"))
            .then(() => message.success("Inventory successfully updated", 5))
            .then(() => setLoading(false));
        });
    }
```

Figure 6.57: Code segment to update inventory details

The figure shown below is the function to delete an inventory item.

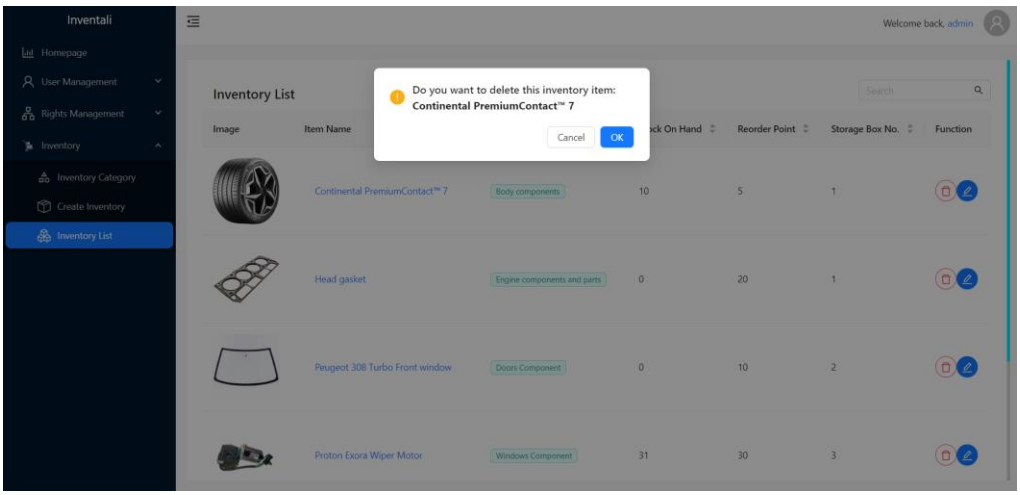


Figure 6.58: Function to delete an inventory item

The figure below shows the code segment used to delete an inventory item. Like the previous functions, this function first obtains the reference to the inventory document using the inventory ID and the doc() function. Then, the deleteDoc function uses the reference to delete the inventory item from Firestore. Once the item is deleted, a success message will be displayed, and the inventory list in the front end will be updated accordingly. In case of any errors, while deleting the inventory item, an error message will be shown to the user.

```
const deleteInventory = async (item) => {
  setLoading(true);
  const inventoryDoc = doc(db, "inventory", item.objectID);

  try {
    await deleteDoc(inventoryDoc).then(
      () => setLoading(false),
      message.success("Inventory successfully deleted", 5),
      setDataSource(dataSource.filter((data) => data.objectID !== item.objectID))
    );
  } catch (error) {
    message.error(error);
  }
};
```

Figure 6.59: Code segment to delete an inventory item

6.3.2 Modules for Mobile-based Application

Table 6.2: Modules for mobile-based application

Module
6.3.2.1 Login
6.3.2.2 Dashboard module (Homepage)
6.3.2.3 Profile
6.3.2.4 Inventory Management (Only view inventory item details and update stock)
6.3.2.5 QR scanning

6.3.2.1 Login Module

The login module for the mobile application is similar to the web application. It utilizes Firebase Firestore's user collection to authenticate the user's credentials. When users enter their username and password in the form shown in the figure below, the login module compares the entered credentials with the user collection in Firebase. This authentication process ensures that only authorized users can access the mobile application.

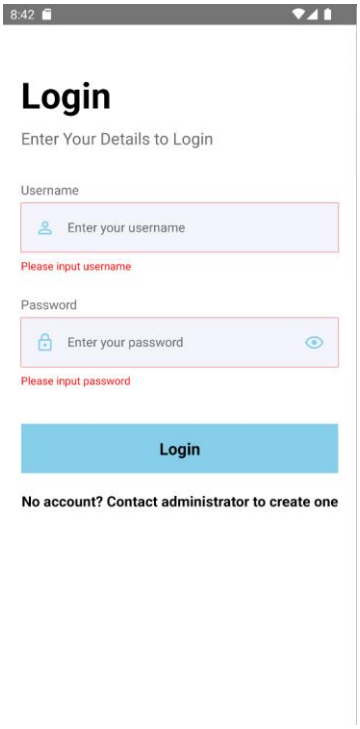


Figure 6.60: Mobile main login screen

According to the code segment and mobile screenshots below contains the function that authenticates user credentials. When a user enters their username and password, the system first checks for the existence of the entered username. If the username does not exist in the system, the user is prompted to contact the administrator to create an account. If the username exists, the system retrieves the user information and compares the entered password with the user's stored password. The user is redirected to the dashboard (homepage) if the passwords match. However, if the passwords do not match, the system displays an error message indicating a password mismatch. The login form verifies if input fields are left empty, and the user cannot log in if any are detected.

```
const handleLogin = async () => {
  setLoading(true);
  try {
    const userRef = firestore()
      .collection('users')
      .where('username', '==', inputs.username.trim());
    const userQuerySnapshot = await userRef.get();
    if (userQuerySnapshot.empty) {
      setShowErrorDialog(true);
      setErrorTitle('Invalid username');
      setErrorMessage(
        "The username that you've entered doesn't exist. Contact administrator",
      );
      setLoading(false);
      console.log("This username doesn't exist");
      console.log('No matching documents.');
      return;
    }
    const userDoc = userQuerySnapshot.docs[0];
    const userData = userDoc.data();
    if (userData.password == inputs.password.trim()) {
      const rolesCollection = firestore().collection('roles');
      const roleQuery = rolesCollection.where(
        'id',
        '==',
        userDoc.data().roleId,
      );
      const roleSnapshot = await roleQuery.get();
      if (roleSnapshot.empty) {
        Alert.alert("Authentication Error: Failed to retrieve user's roles");
        console.log('Error: No matching documents for role.');
        return;
      } else {
        const roleDoc = roleSnapshot.docs[0];
        const userExpandRole = {
          ...userDoc.data(),
          id: userDoc.id,
          role: roleDoc.data(),
        };
        try {
          await AsyncStorage.setItem('user', JSON.stringify(userExpandRole));
        } catch (e) {
          Alert.alert(`Error setting item in temporarily storage, ${e}`);
        }
        setLoading(false);
        navigation.navigate('Root');
      }
    } else {
      setLoading(false);
      setShowErrorDialog(true);
      setErrorTitle('Inavalid Password');
      setErrorMessage(
        "The password that you've entered is incorrect. Please try again",
      );
      setInputs({...inputs, password: ''});
    }
  } catch (error) {
    console.error('Error logging in:', error);
  }
};
```

Figure 6.61 Code segment to handle login for mobile application

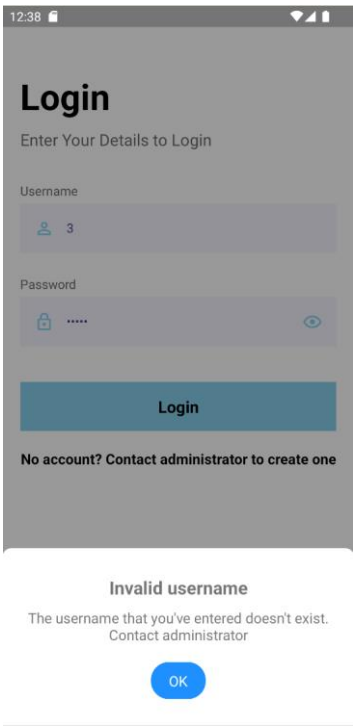


Figure 6.62: Login screen showing the username does not exist

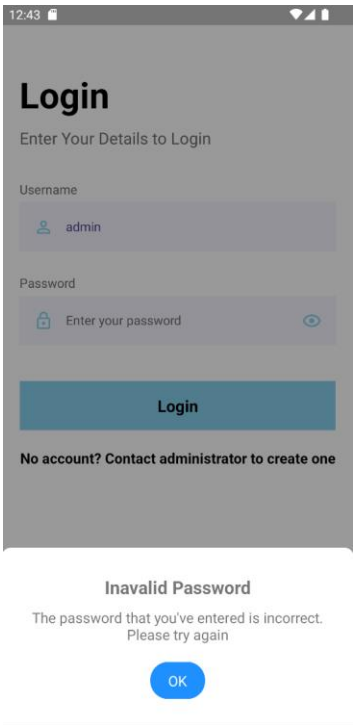


Figure 6.63: Login screen showing the password mismatch

6.3.2.2 Dashboard Module

After the user is logged into the mobile application, the user will be redirected to the dashboard module, as shown in the figure below. Regardless of whether they are a super administrator, inventory manager, or inventory staff, they will be able to view the inventory summary, which provides a comprehensive overview of the inventory details. This summary includes information such as the total number of items, total stocks, number of low-stock items, number of out-of-stock items, a list of items below the reorder point, and a list of out-of-stock items. The interactive lists in the dashboard allow the user to simply tap on any item and be directed to the corresponding item details page, where they can update the inventory item details as required.

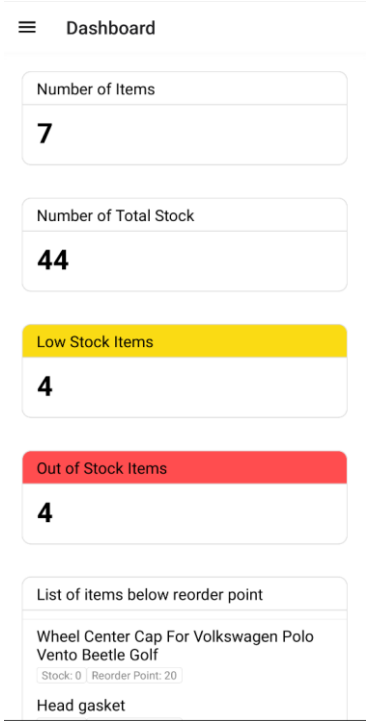


Figure 6.64: Dashboard (homepage)

The dashboard module utilizes Firebase's Firestore to retrieve all necessary information to display on the dashboard. The information will be retrieved from the inventory collection and set the necessary information to the React state, such as the setTotalStock, setItemsBelowReorderLevelCount, setItemsBelowReorderLevel, setItemsOutOfStockCount and setItemOutOfStock . Once the state is saved, the dashboard module displays this information to the user on the dashboard using the state. The relevant code segment demonstrating this functionality is shown below.

```
useEffect(() => {
  const unsubscribe = firestore()
    .collection('inventory')
    .onSnapshot(querySnapshot => {
      let totalStock = 0;
      let count = 0;
      const belowReorderLevelItems = [];
      const outOfStockItems = [];
      querySnapshot.forEach(doc => {
        const data = doc.data();
        totalStock += data.stock;
        if (data.stock < data.reorderPoint) {
          count++;
          belowReorderLevelItems.push({id: doc.id, ...data});
        }
        if (data.stock <= 0) {
          outOfStockItems.push({id: doc.id, ...data});
        }
      });
      setInventoryCount(querySnapshot.size);
      setTotalStock(totalStock);
      setItemsBelowReorderLevelCount(count);
      setItemsBelowReorderLevel(belowReorderLevelItems);
      setItemsOutOfStockCount(outOfStockItems.length);
      setItemsOutOfStock(outOfStockItems);
      setLoading(false);
    });
  return () => unsubscribe();
}, []);
```

Figure 6.65: Code Segment for retrieving data for dashboard (homepage)

The mobile application features a navigation bar, depicted in the figure below, also referred to as a navigation drawer in React Navigation. Users can utilize this navigation bar to navigate to other screens throughout the mobile application. Additionally, a logout button is at the bottom of this navigation bar, enabling users to log out from the mobile application conveniently.

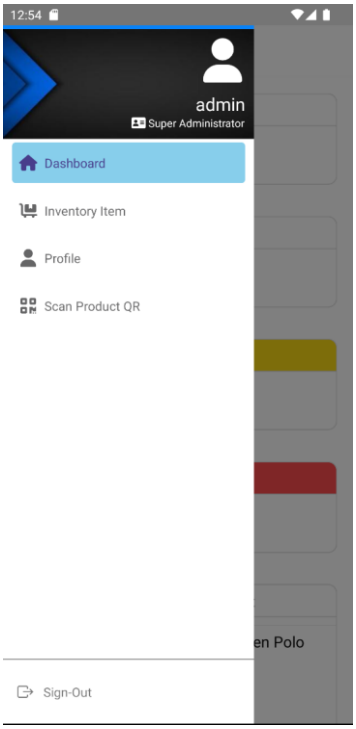


Figure 6.66: Navigation bar (navigation drawer)

The code segments below are used to implement the Navigation Bar (Navigation Drawer) using the React Navigation Component.

```
<Drawer.Navigator
  drawerContent={props => <DrawerComponent {...props} />}
  screenOptions={{
    drawerActiveTintColor: COLORS.darkBlue,
    drawerActiveBackgroundColor: COLORS.blue,
    drawerLabelStyle: {
      marginLeft: -24,
      fontFamily: 'EduQLDBeginner-Regular',
    },
  }}>
  <Drawer.Screen
    name="Dashboard"
    component={HomeScreen}
    options={{
      drawerIcon: ({color}) => (
        <FontAwesomeIcon icon="fa-solid fa-house" color={color} size={20} />
      ),
    }}></Drawer.Screen>
  <Drawer.Screen
    name="Inventory Item"
    component={InventoryListScreen}
    options={{
      drawerIcon: ({color}) => (
        <FontAwesomeIcon
          icon="fa-solid fa-cart-flatbed"
          color={color}
          size={20}
        />
      ),
    }}></Drawer.Screen>
  <Drawer.Screen
    name="Profile"
    component={ProfileScreen}
    options={{
      drawerIcon: ({color}) => (
        <FontAwesomeIcon icon="fa-solid fa-user" color={color} size={20} />
      ),
    }}></Drawer.Screen>
  <Drawer.Screen
    name="Scan Product QR"
    component={ScanQRScreen}
    options={{
      drawerIcon: ({color}) => (
        <FontAwesomeIcon
          icon="fa-solid fa-qrcode"
          color={color}
          size={20}
        />
      ),
    }}></Drawer.Screen>
  {/* Different with stack is that Drawer does not need to use onPress to navigate to different page */}
</Drawer.Navigator>
```

Figure 6.67: Code segment to implement the Navgiation Bar (Navigation Drawer)

6.3.2.3 Profile Module

In the mobile application, users can easily access their profile information by clicking the profile option in the navigation drawer. Once selected, the profile module will display user details, including username, user id, role, and region, as illustrated in the figure below. However, it's important to note that, unlike the web application, the mobile app does not allow users to modify their profile details. This is because the mobile app is a companion to the web application, allowing users to conveniently access inventory details and update inventory stock while on the go.

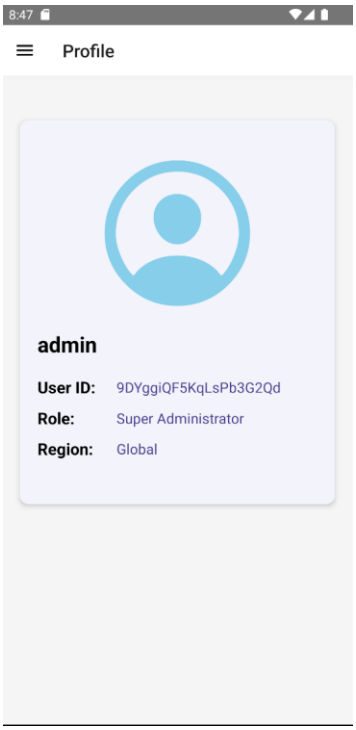


Figure 6.68: Code segment to delete an inventory item

The following code segment demonstrates the functioning of the user module. It retrieves the unique user ID from the local storage (AsyncStorage), previously saved upon user login. This ID is then used to retrieve the corresponding user details from Firebase's Firestore. The retrieved response is then stored in the React state by calling the setUser to set the user state. The saved state is then used to display the user details on the profile screen.

```
const getUserData = async () => {
  try {
    // Get the user ID from AsyncStorage
    const userData = await AsyncStorage.getItem('user');
    const userId = JSON.parse(userData).id;

    // Get the user document from the users collection
    const userRef = firestore().collection('users').doc(userId);
    userRef
      .get()
      .then(userDoc => {
        if (userDoc.exists) {
          // Set the user state to the document data
          setUser({...userDoc.data(), id: userDoc.id});
          // Get the role document for the user's roleId
          const roleRef = firestore()
            .collection('roles')
            .where('id', '==', userDoc.data().roleId);
          roleRef
            .get()
            .then(querySnapshot => {
              if (querySnapshot.size > 0) {
                // Set the role state to the role name of the first matching document
                setRole(querySnapshot.docs[0].data().roleName);
              } else {
                setShowErrorDialog(true);
                setErrorTitle('Error');
                setErrorMessage('Unable to retrieve user role');
                console.log(
                  'No role document found for roleId:',
                  userDoc.data().roleId,
                );
              }
            })
```

Figure 6.69: Code segment retrieve user data for profile module

6.3.2.4 Inventory Management Module

The figure below illustrates the inventory item list screen, which displays all items created in the web application and now synchronized with the mobile app. Additionally, users can use the search box to search for inventory items by their respective item names conveniently.

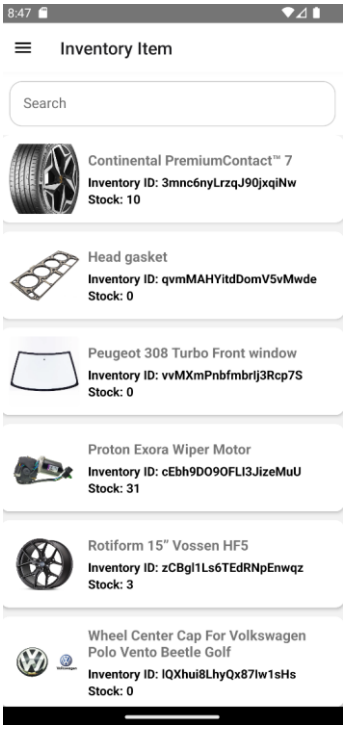


Figure 6.70: Inventory List Screen

The following code segment presents the getInventoryData function, which is utilized by the inventory list module to retrieve the list of inventory data. This function uses the firestore() method to fetch data from the inventory collection. The data is ordered by the inventory item name and limited to a maximum of 5 items per pagination. The retrieved data is subsequently saved to a React state, which can be utilized later to display the inventory items in the inventory list screen. Additionally, the setLastDoc state is used to capture the last document that was retrieved, which is crucial for pagination purposes - this last document serves as a cursor to the last retrieval. In situations where there is more data to retrieve, the getMoreData function is triggered instead of the getInventoryData function.

```
const getInventoryData = async () => {
  try {
    setRefreshing(true); // set refreshing state to true
    setLoading(true); // set loading to true while fetching data
    const querySnapshot = await firestore()
      .collection('inventory')
      .orderBy('itemName')
      .limit(LIMIT)
      .get();
    const documents = querySnapshot.docs.map(doc => ({
      id: doc.id,
      ...doc.data(),
    }));
    setInventoryData(documents); // update state with retrieved data
    setLastDoc(querySnapshot.docs[querySnapshot.docs.length - 1]); // set lastDoc to last document retrieved
    setLoading(false); // set loading to false when done fetching data
  } catch (error) {
    console.log(error);
    setLoading(false);
  } finally {
    setRefreshing(false); // set refreshing state to false when done fetching data
  }
};
```

Figure 6.71: Function to get inventory data

```
// function to retrieve next page of inventory data
const getMoreData = async () => {
  try {
    setLoading(true);
    const querySnapshot = await firestore()
      .collection('inventory')
      .orderBy('itemName')
      .startAfter(lastDoc)
      .limit(LIMIT)
      .get();
    const documents = querySnapshot.docs.map(doc => ({
      id: doc.id,
      ...doc.data(),
    }));
    setInventoryData([...inventoryData, ...documents]); // update state with retrieved data from next page
    setLastDoc(querySnapshot.docs[querySnapshot.docs.length - 1]); // set lastDoc to last document retrieved from next page
    setLoading(false);
  } catch (error) {
    console.log(error);
    setLoading(false);
  }
};
```

Figure 6.72: Function to get more inventory data

The diagram presented below illustrates the process described earlier in the code segment. It visually represents how the getInventoryData function is executed to retrieve inventory data from Firestore and how the pagination works when there is more data to retrieve. This diagram helps better understand the flow of the code and its logic.

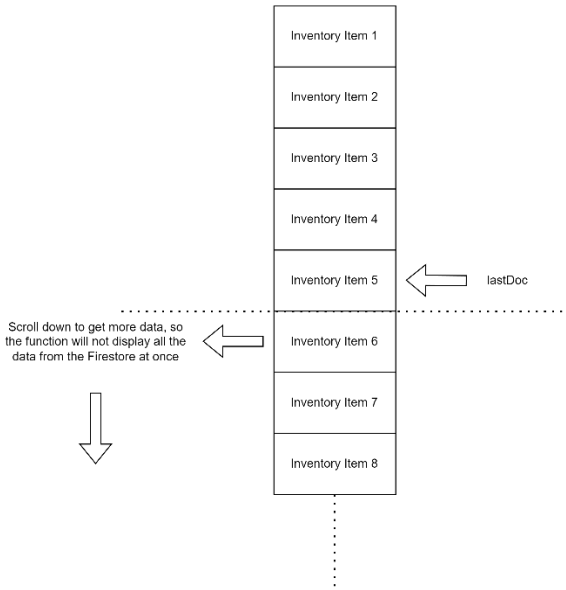


Figure 6.73: Inventory List Screen

Users will be directed to the inventory details screen when they click on an item in the inventory list.

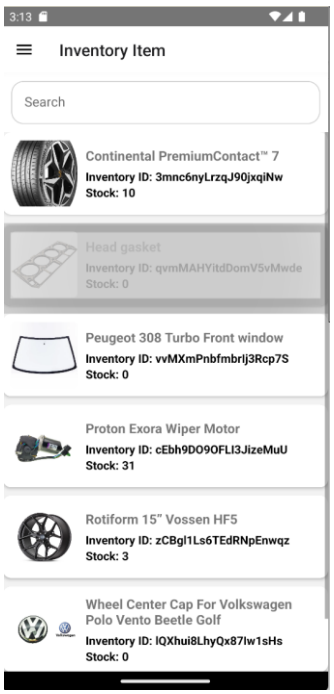


Figure 6.74: User clicks an item on the inventory list

The following code segment demonstrates how each inventory item is rendered on the inventory list screen. Each inventory item is wrapped inside a React Native component called TouchableOpacity, which functions like a clickable button. When users click on an inventory item, they are redirected to the inventory details screen, and the inventory ID is passed as a prop. The inventory ID is then obtained and used to retrieve the corresponding inventory details for display on the inventory details screen.

```
const renderItem = ({item}) => (
  <TouchableOpacity
    style={styles.touchable}
    onPress={() =>
      navigation.navigate('InventoryDetailsScreen', {inventoryId: item.id})
    }>
    <View style={styles.itemContainer}>
      <Image style={styles.itemImage} source={{uri: item.imageURL}} />
      <View style={styles.itemDetails}>
        <Text style={styles.itemName}>{item.itemName}</Text>
        <Text style={styles.itemDescription}>Inventory ID: {item.id}</Text>
        <Text style={styles.itemDescription}>Stock: {item.stock}</Text>
      </View>
    </View>
  </TouchableOpacity>
);
```

Figure 6.75: Code segment for every inventory item in the list

The inventory details screen displays all the relevant details, including an item image and a QR code at the bottom. Additionally, users can update the stock of the inventory item directly from this screen.

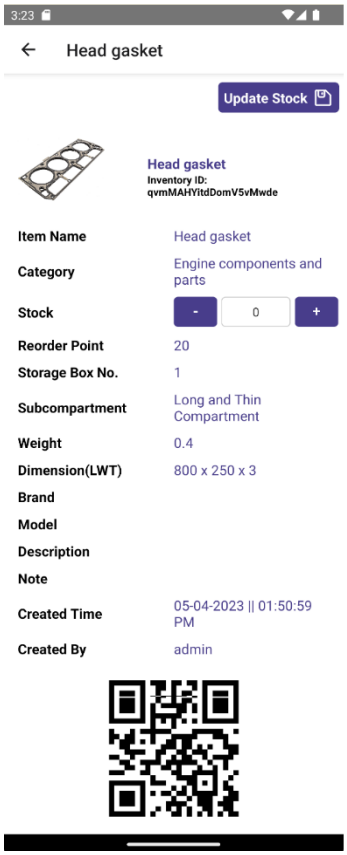


Figure 6.76: Inventory Details Screen

The following code segment demonstrates the getData() function, which is responsible for obtaining inventory details from the inventory collection. First, the function obtains a reference to the inventory item using the inventory ID. Once the data is retrieved, the response is saved to a React state for display on the inventory details screen.

```
const getData = async () => {
  console.log(inventoryId);
  const inventoryRef = firestore()
    .collection('inventory')
    .doc(inventoryId.replace(/[.,\/#!$%\^&\*;:{}=\-_`~()]/g, ''));
  await inventoryRef
    .get()
    .then(doc => {
      if (doc.exists) {
        setCurrentStock(doc.data().stock);
        setInventoryDetails({...doc.data(), id: doc.id});
      } else {
        setInventoryDetails(undefined);
        setShowErrorDialog(true);
        console.log('No inventory document found with ID: ', inventoryId);
      }
    })
    .catch(error => {
      console.log('Error getting inventory document: ', error);
    });
};
```

Figure 6.77: Code segment to get inventory details of an inventory item

6.3.2.5 QR Scanning Module

The figure below displays the QR scanning screen, which users can access through the navigation bar. Users can scan QR codes on this screen using the camera on their device. The screen also includes a button that allows users to turn the flash on or off for scanning QR codes in low light conditions, ensuring that they can quickly and easily scan QR codes regardless of the lighting situation.

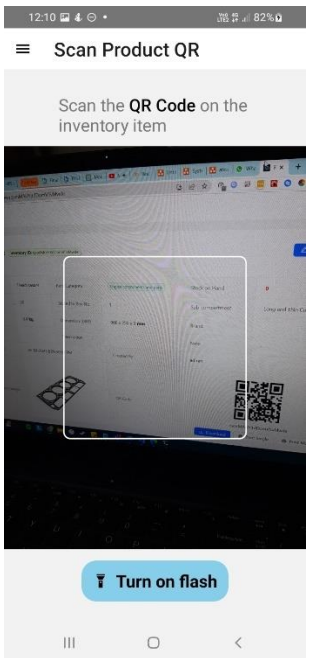


Figure 6.78: QR Scanning Screen

The figure below shows the process of scanning a QR code pasted on an inventory item box. Once the user scans the QR code using the app's built-in QR scanner, they will be automatically directed to the corresponding inventory details page, as shown below.

Figure 6.79: Demonstration of scanning a QR code on an inventory item box.

If the QR code scanning successfully matches the ID of any inventory item, the user will be redirected to the inventory details screen. On this page, the user can view all the inventory details, including the inventory image, QR code, and the option to update the stock level.

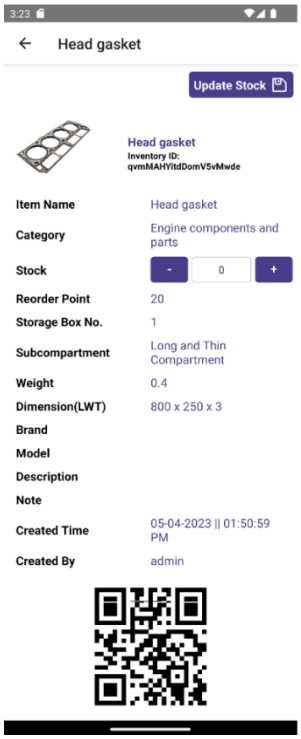


Figure 6.80: Successfully scanned QR code matching inventory item ID

However, if the QR code scanned does not match any inventory item ID, the app cannot redirect the user to the corresponding inventory details page. Instead, an error message will inform the user that the scanned QR code is not associated with any inventory item.

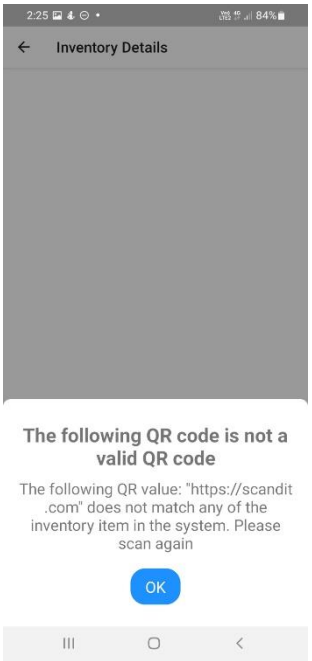


Figure 6.81: Scanned QR code with no matching inventory item ID.

The code segment below demonstrates how the "react-native-qrcode-scanner" library is used in the mobile application to enable QR code scanning functionality. This library is popular among developers due to its simplicity and several useful features. It supports iOS and Android platforms and includes built-in support for handling camera permissions, customizable UI components, on/off flash, and different QR code types. Each QR code represents the ID for a particular inventory item, and the application will redirect to the corresponding inventory details screen using the inventory ID. The redirect is handled through the onSuccess function.

```
export default function ScanQR(props) {
  const [isFlashOn, setIsFlashOn] = useState(false);
  const onSuccess = e => {
    props.navigation.navigate('InventoryDetailsScreen', {
      inventoryId: e.data,
    });
  };
  const toggleFlash = () => {
    setIsFlashOn(!isFlashOn);
  };
  return (
    <QRCodeScanner
      reactivate={true}
      reactivateTimeout={2000}
      flashMode={
        isFlashOn
          ? RNCamera.Constants.FlashMode.torch
          : RNCamera.Constants.FlashMode.off
      }
      fadeIn={true}
      onRead={onSuccess}
      showMarker={true}
      topContent={
        <View>
          <Text style={styles.centerText}>
            Scan the <Text style={styles.textBold}>QR Code</Text> on the
            inventory item
          </Text>
        </View>
      }
      bottomContent={
        <Button
          style={styles.button}
          title={isFlashOn ? 'Turn off flash' : 'Turn on flash'}
          onPress={toggleFlash}
          iconName={isFlashOn ? 'flashlight-off' : 'flashlight'}
        />
      }
      markerStyle={styles.marker}
    />
  );
}
```

Figure 6.82: Code segment for QR scanning functionality in the mobile app.

6.4 System Deplyoment

Both the web application and mobile application have efficient version control through Git and are hosted on GitHub repositories. Git, a widely-used version control system, enables developers to track changes, collaborate on code, and easily manage different versions of the project. With the help of Git, code updates and enhancements can be implemented smoothly while maintaining a structured development workflow.

The project repositories are hosted on GitHub, a web-based platform for version control and collaboration. GitHub provides a centralized location where developers can store, manage, and share their code repositories. It offers features such as pull requests, branching, and issue tracking, facilitating effective collaboration among team members and making it easier to review and merge code changes.

In addition, the project's web application is hosted on an Amazon Web Services (AWS) server using AWS Amplify. The web application can be access at this domain, www.inventali.com. AWS Amplify is a development platform that simplifies the process of deploying and scaling modern web applications. By leveraging AWS's robust infrastructure, the website can ensure reliable and scalable hosting, accommodating high traffic loads without compromising performance. AWS Amplify provides automatic scaling, caching, and content delivery network (CDN) capabilities, optimizing the website's performance and ensuring fast loading times for visitors.

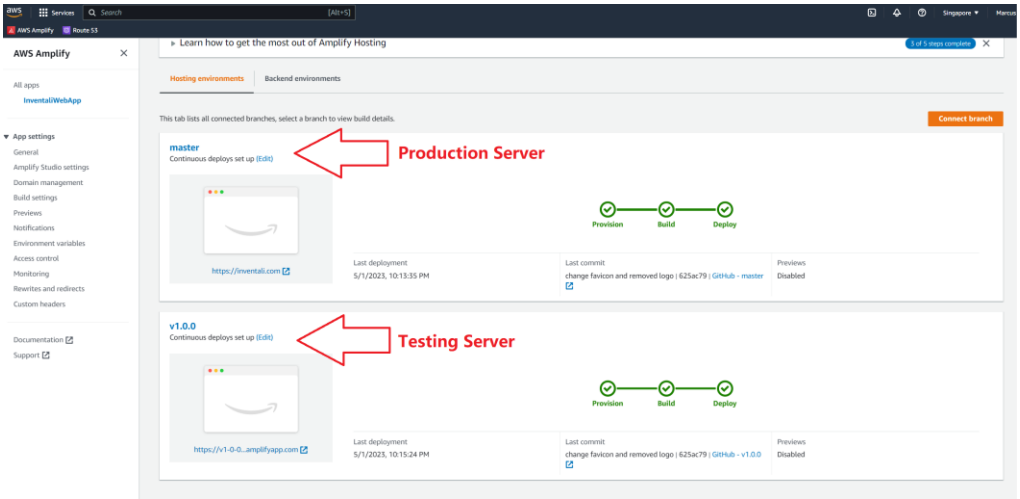


Figure 6.83: Demonstration of continuous deployment and integration (CI/CD)

One of the key advantages of AWS Amplify is its seamless integration with Git-based workflows, such as GitHub. Through this integration, developers can establish continuous deployment and integration (CI/CD) pipelines, enabling automated building, testing, and deployment of the web application

to AWS. This streamlines the development process, allowing for quick iterations and efficient collaboration among team members.

6.5 Conclusion

In conclusion, Chapter 6 provides an overview of the system implementation process. It begins with the project setup, including the creation of a Firebase account and project setup with Firestore database and Firebase Storage for data and image storage. The chapter then explains the setup of a React project for the web-based application and a React Native project for the mobile-based application, along with the installation of necessary dependencies and tools.

The system modules are divided into web-based and mobile-based applications. The web-based application includes modules such as Login, Dashboard, Profile, User Management, Role Management, Right Management (Page Access Management), and Inventory Management. The mobile-based application consists of modules like Login, Dashboard, Profile, Inventory Management (viewing item details and updating stock), and QR scanning.

Both the web and mobile applications utilize Git for efficient version control and are hosted on GitHub repositories, providing a centralized location for code management and collaboration. The web application is hosted on an AWS server using AWS Amplify, which simplifies the deployment and scaling process. AWS Amplify's integration with Git enables continuous deployment and integration (CI/CD) pipelines, automating the building, testing, and deployment of the web application.

Overall, this chapter highlights the key steps involved in setting up the project and implementing the system, while also outlining the modules for each application. The use of Git, GitHub, Firebase, React, React Native, AWS Amplify, and CI/CD pipelines ensures a structured development workflow.

CHAPTER 7

SYSTEM TESTING

7.1 Introduction

In this project, both unit testing and usability testing were conducted to ensure the fulfilment of functional and non-functional requirements of the system, which include both the web and mobile applications.

7.2 Unit Testing

This project utilises unit testing to test every function manually to ensure that the web and mobile applications meet the requirement specification. The unit testing process is divided into web application unit testing and mobile application unit testing. This approach helps verify both applications' functionality and ensures that all functional and non-functional requirements are fulfilled. Below is an overview of the test cases and their corresponding results.

7.2.1 Unit testing for the Web Application

Table 7.1: Unit Testing of login module (web application)

Test Module	Log in module		**Test Title**	Log in to the user account	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-001	Log in with valid credentials	1. Enter the registered username and password 2. Click Log In button	i. Registered username ii. Valid password	Display login successfully message and redirected to the homepage	**Pass**
UNT-002	Log in with an invalid username	1. Enter an unregistered username and password 2. Click Log In button	i. Unregistered username ii. password	Display username does not exist error message	**Pass**
UNT-003	Log in with a valid username and an invalid password	1. Enter registered username but invalid password 2. Click Log In button	i. Registered username ii. Invalid password	Display password mismatch error message.	**Pass**
UNT-004	Log in with an empty	1. Click the Login button	No test data	The system will	**Pass**

	username and password			display an error message indicating that inputs are missing or empty.	
UNT-005	Log in to a suspended account	1. Enter the registered username and password 2. Click Log In button	i. Registered username ii. Valid password	Display account had been suspended error message	**Pass**
UNT-006	Log out from the web application	3. Click logout button	No test data	Display logout had been successfully message and redirected to the login page	**Pass**

Table 7.2: Unit Testing of profile module (web application)

Test Module	Profile Module		**Test Title**	Edit Profile	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-007	Edit personal profile details with a new	1. Navigate to the user profile.	i. New unique username	Display profile details successfully updated	**Pass**

	username, password, region, and role.	2. Edit profile details using a unique username, password, and region and select a role. 3. Click the save user details button	ii. New password iii. New Region iv. New role	message.	
UNT-008	Edit personal profile details with duplicated username	1. Navigate to the user profile 2. Edit profile details using a duplicated username 3. Click the save user details button	i. Duplicated username	Display an error message that the username is already taken	**Pass**
UNT-009	Edit personal profile details with empty inputs	1. Navigate to the user profile 2. Remove all or some profile details and leave	i. Empty username ii. Empty password	The system will display an error message indicating that inputs are missing	**Pass**

		it empty 3. Click the save user details button		or empty.	

Table 7.3: Unit testing of the homepage (dashboard) module to display inventory summary (web application)

Test Module	Homepage (dashboard)		**Test Title**	Displaying Inventory Summary on the Dashboard	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-010	User access to the dashboard	1. Navigate to the dashboard	No test data	All the inventory data in the dashboard is displayed accordingly	**Pass**

Table 7.4: Unit testing of the homepage (dashboard) module to show items that below reorder point and out of stock (web application)

Test Module	Homepage (dashboard)		**Test Title**	Displaying Inventory Summary on the Dashboard	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-011	User access the	1. Navigate to the	No test data	The system will	**Pass**

	dashboard to check items below reorder point	dashboard 2. The user clicks one of the items inside reorder point list		redirect the user to the corresponding inventory item details page and display all the item details accordingly, including the QR code and the image of the inventory item.	
UNT-012	A user accesses the dashboard to check items that are out of stock	1. Navigate to the dashboard 2. The user clicks one of the items that are inside out of stock item list	No test data	The system will redirect the user to the corresponding inventory item details page and display all the item details accordingly, including the QR code and the image of the inventory	**Pass**

				item.	

Table 7.5: Unit testing of the user management module (web application)

Test Module	User Management		**Test Title**	Displaying a list of user	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-013	User access to the user management page	1. Navigate to the user management page	No test data	The system will display the first five user data	**Pass**
UNT-014	A user clicks on the next page of the user list on the management page	1. Navigate to the user management page 2. The user clicks the next page button	No test data	The system will display the next five user data	**Pass**
Test Module	User Management		**Test Title**	Search users from the user list	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-015	Search user from the user list with the full username	1. Navigate to the user management page 2. The user inputs the search query in the	Full username: lucas	The system will display the user with username lucas only	**Pass**

		search field with the full username.			
UNT-016	Search user from the user list with partial input of username	1. Navigate to the user management page 2. The user inputs the search query in the search field with a partial username.	Partial username: lu	The system will display the user with the username lukas and lucas	**Pass**
Test Module	User Management		**Test Title**	Add a user to the user list	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-017	Add a user to the user list	1. Navigate to the user management page 2. Click on the add user button 3. Filled in all the necessary details 4. Click the add button in the modal	i. Username ii. Password iii. Region iv. Role	The system has successfully added the user to the user list and the Firestore database.	**Pass**

UNT-018	Add a user to the user list with empty fields	1. Navigate to the user management page 2. Click on the add user button 3. Click the add button in the modal	No test data	The system will display an error message for any empty required fields.	**Pass**
Test Module	User Management		**Test Title**	Delete the user from the user list	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-019	Delete the user from the user list	1. Navigate to the user management page 2. Click on the delete button for the user to wish to remove. 3. Confirm the deletion of the user when prompted.	No test data	The system has successfully removed the user from the user list and the Firestore database.	**Pass**
Test Module	User Management		**Test Title**	Edit user from the user list	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**

UNT-020	Edit user details of a user by using the same username	1. Navigate to the user management page 2. Click on the edit button for the user that needs to be edited. 3. Filled in all the necessary details that required changes and used the same username	i. Password ii. Region iii. Role	The system has successfully updated the user with new user details in both the user list and the Firestore database.	**Pass**
UNT-021	Edit user details of a user by using the duplicated username	1. Navigate to the user management page 2. Click on the edit button for the user wishes to edit 3. Filled in all the necessary details that required changes and	i. Duplicated username ii. Password iii. Region iv. Role	Display an error message that the username is already taken	**Pass**

		used a duplicate username			
UNT-022	Edit user details of a user with empty inputs	1. Navigate to the user management page 2. Click on the edit button for the desired user that needs to be edited 3. Remove all or some user details and leave it empty	No test data	The system will display an error message indicating that inputs are missing or empty.	**Pass**
Test Module	User Management		**Test Title**	Toggle the user status of a user	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-023	Toggle the user status	1. Navigate to the user management page 2. Toggle the user status button for any user	No test data	When a user clicks the toggle button, the user status will be updated both in the front end and the Firestore database.	**Pass**

Table 7.6: Unit testing of the role management module (web application)

Test Module	Role Management		**Test Title**	Displaying a list of role	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-024	User access the role management page	1. Navigate to the role management page	No test data	The system will display the list of role	**Pass**
Test Module	User Management		**Test Title**	Edit the access assignment for each role	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-025	Edit access assignment of a role	1. Navigate to the role management page 2. Click on the edit button for the desired role that needs to be edited. 3. Check or uncheck the desired access assignments to remove, add, or keep them.	i. Access assignments	The system has successfully updated the role with new access assignments in the front end and Firestore databases.	**Pass**

Table 7.7: Unit testing of page access management module (web application)

Test Module	Page Access Magnement		**Test Title**	Displaying a list of page access	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-026	User access the page access management page	1. Navigate to the page access management page	No test data	The system will display the list of page access	**Pass**
Test Module	Page Access Magnement		**Test Title**	Configure the page permission of each page	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-027	Edit access assignment of a role	1. Navigate to the page access management page 2. Click the edit button for the desired page access to configure its permission settings.	No test data	Upon clicking the toggle button of a page, the page permission setting of a page will be updated in both the front end and the Firestore database.	**Pass**

Table 7.8: Unit testing of inventory management module – Inventory Category Page (web application)

Test Module	Inventory Management – Inventory Category Page		**Test Title**	Displaying a list of the inventory category	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-028	User access the inventory category page	1. Navigate to the inventory category page	No test data	The system will display the first five inventory categories	**Pass**
UNT-029	The user navigates to the next page of the inventory category list	1. Navigate to the inventory category page 2. The user clicks the next page button	No test data	The system will display the next five inventory categories	**Pass**
Test Module	Inventory Management – Inventory Category Page		**Test Title**	Search category from the inventory category list	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-030	Search categories from the inventory category list with the full	1. Navigate to the inventory category page.	Full search: rims - hubcap	The system will display the category name with rims -	**Pass**

	category name	2. The user inputs the search query with the full category name in the search field.		hubcap	
UNT-031	Search user from the inventory category list with partial input of category name.	1. Navigate to the user management page 2. The user inputs the search query in the search field with a partial search	Partial search: rims	The system will display the categories start with rims wording	**Pass**
Test Module	Inventory Management – Inventory Category Page		**Test Title**	Add category to the category list	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-032	Add category to the category list	1. Navigate to the inventory category list page 2. Fill in the category name in the input field 3. Click the add category	i. Category name	The system has successfully added the category to the category list and Firestore.	**Pass**

		button			
UNT-033	Add a category to the category list with an empty input field	1. Navigate to the inventory category list page 2. Click the add category button	No test data	The system will display an error message for an empty category name	**Pass**
Test Module	Inventory Management – Inventory Category Page		**Test Title**	Delete a category from the category list	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-034	Delete a category from the category list	1. Navigate to the inventory category management page 2. Click on the delete button for the category wishes to remove. 3. Confirm the deletion of the user when prompted.	No test data	The system has successfully removed the category from the category list and the Firestore database.	**Pass**
Test Module	Inventory Management – Category Page		**Test Title**	Edit a category in the category list	

Test Case ID	Test Case Description	Execution Steps	Test Data	Expect Result	Status
UNT-035	Edit category name	1. Navigate to the inventory category management page. 2. Click on the category row that needs to be edited. 3. Fill in the new category name.	i. New category name	The system has successfully updated the category with a new name in the category list and the Firestore database.	**Pass**
UNT-036	Edit a category name of a category with empty inputs	1. Navigate to the inventory category management page. 2. Click on the category row that needs to be edited. 3. Left the category name input fields empty	No test data	The system will display an error message indicating that inputs are missing or empty.	**Pass**

Table 7.9: Unit testing of inventory management module – Inventory list page (web application)

Test Module	Inventory Management – Inventory List Page		**Test Title**	Displaying a list of the inventory item	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-037	User access the inventory list page	1. Navigate to the inventory list page	No test data	The system will display the first five inventory item	**Pass**
UNT-038	The user navigates to the next page of the inventory item list	1. Navigate to the inventory list page 2. The user clicks the next page button	No test data	The system will display the next five inventory item	**Pass**
Test Module	Inventory Management – Inventory List Page		**Test Title**	Search inventory items from the inventory list	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-039	Search inventory items from the inventory list with full inventory item names.	1. Navigate to the inventory list page. 2. The user inputs the search query with the full inventory item name in the search	Full search: head gasket	The system will display the inventory item named “head gasket”.	**Pass**

		field.			
UNT-040	Search inventory items from the inventory list with partial input of inventory name.	1. Navigate to the inventory list page 2. The user inputs the search query in the search field with a partial search	Partial search: proton	The system will display the inventory item named “Proton”.	**Pass**
Test Module	Inventory Management – Inventory List Page		**Test Title**	Click an inventory item in the inventory list	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-041	A user clicks an inventory item in the inventory list to redirect to the inventory item details page.	1. Navigate to the inventory list page 2. The user clicks a desired inventory item in the inventory list	No test data	The system will redirect the user to the corresponding inventory item details page and display all the item details accordingly, including the QR code and the image of the inventory	**Pass**

				item.	
Test Module	Inventory Management – Inventory List Page		**Test Title**	Delete an inventory item from the inventory list	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-042	Delete an inventory item from the inventory list	1. Navigate to the inventory list page 2. Clicking the delete button for a specific inventory item will remove it from the list. 3. Confirm the deletion of the inventory item when prompted.	No test data	The system has successfully removed the inventory item from the inventory list and the Firestore database.	**Pass**
Test Module	Inventory Management – Inventory List Page		**Test Title**	Edit an inventory item from the inventory list	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-043	Edit inventory item details	1. Navigate to the inventory list page 2. Click on the edit button	New inventory data depending on the user choose what	The system has successfully updated the inventory item	**Pass**

		for the inventory item that needs to be edited. 3. Change the necessary details or keep them for basic inventory information on the first page. 4. Click next button 5. Change the necessary details or keep them for inventory item details on the second page. 6. Click next button 7. Click update item button	attribute to update	with new details in the inventory list and the Firestore database.	
UNT-044	Edit inventory item details with empty required inputs on the	1. Navigate to the inventory list page 2. Click on the edit button	No test data	The system will display an error message for any	**Pass**

	first or second pages.	for the inventory item that needs to be edited. 3. Left required inputs in either the first page or second page empty		required field left empty.	

Table 7.10: Unit testing of inventory management module – Create inventory page (web application)

Test Module	Inventory Management – Create inventory page		**Test Title**	Add an inventory item to the inventory list	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-045	Add an inventory item to the inventory list	1. Navigate to the create inventory item page 2. Fill in all the necessary details for basic inventory information on the first page. 3. Click next button 4. Fill in all the necessary details for inventory	i. Item name ii. Item category iii. Opening stock iv. Reorder point v. Storage box no. vi. Sub-compartment vii. Weight	The system has successfully added the inventory item details to the inventory list and Firestore and stored the inventory image in Firebase Cloud Storage.	**Pass**

		item details on the second page. 5. Uploaded an inventory image 6. Click next button 7. Click save item	viii. Dimensions ix. Brand x. Model xi. Description xii. Note xiii. Inventory image		
UNT-046	Add an inventory item to the list with mandatory input fields left empty on the first page.	1. Navigate to the create inventory item page. 2. Left the required input fields empty for basic inventory information on the first page 3. Click next button	No test data	The system will display an error message for any required field left empty.	**Pass**
UNT-047	Add an inventory item to the list with mandatory input fields	1. Navigate to the create inventory item page. 2. Fill in all the required	i. Item name ii. Item category	The system will display an error message for any	

	left empty on the second page.	details for basic inventory information on the first page. 3. Click next button 4. Left the required input fields empty for inventory item details on the second page. 5. Click next button		required field left empty.	

7.2.2 Unit testing for Mobile application

Table 7.11: Unit Testing of login module (mobile application)

Test Module	Log in module		**Test Title**	Log in to the user account	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-048	Log in with valid credentials	1. Enter the registered username and password 2. Click Log In button	i. Registered username ii. Valid password	Display login successfully message and redirected to the	**Pass**

				homepage	
UNT-049	Log in with an invalid username	1. Enter an unregistered username and password 2. Click Log In button	i. Unregistered username ii. password	Display username does not exist error message	**Pass**
UNT-050	Log in with a valid username and an invalid password	1. Enter registered username but invalid password 2. Click Log In button	i. Registered username ii. Invalid password	Display password mismatch error message.	**Pass**
UNT-051	Log in with an empty username and password	1. Click the Login button	No test data	The system will display an error message indicating that inputs are missing or empty.	**Pass**
UNT-052	Log in to a suspended account	1. Enter the registered username and password 2. Click Log In button	i. Registered username ii. Valid password	Display account had been suspended error message	**Pass**

UNT-053	Log out from the mobile application	1. Click the logout button in the navigation bar	No test data	Redirect to the login page	**Pass**

Table 7.12: Unit Testing of profile module (mobile application)

Test Module	Profile Module		**Test Title**	Display profile details	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-054	User access the profile page	1. Navigate to the user profile screen	No test data	All the profile details of the logged-in user are displayed accordingly	**Pass**

Table 7.13: Unit testing of the homepage (dashboard) module to display inventory summary (mobile application)

Test Module	Homepage (dashboard)		**Test Title**	Displaying Inventory Summary on the Dashboard	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-055	User access to the dashboard	1. Navigate to the dashboard	No test data	All the inventory data in the dashboard is	**Pass**

				displayed accordingly	

Table 7.14: Unit testing of the homepage (dashboard) module to show items that below reorder point and out of stock (web application)

Test Module	Homepage (dashboard)		**Test Title**	Displaying Inventory Summary on the Dashboard	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-056	User access the dashboard to check items below reorder point	1. Navigate to the dashboard 2. The user clicks one of the items inside reorder point list	No test data	The system will redirect the user to the corresponding inventory item details page and display all the item details accordingly, including the QR code and the image of the inventory item.	**Pass**
UNT-057	A user accesses the dashboard to check	1. Navigate to the dashboard	No test data	The system will redirect the user to the	**Pass**

	items that are out of stock	2. The user clicks one of the items that are inside out of stock item list		corresponding inventory item details page and display all the item details accordingly, including the QR code and the image of the inventory item.	

Table 7.15: Unit testing of inventory management module – Inventory list page (mobile application)

Test Module	Inventory Management – Inventory List Page		**Test Title**	Displaying a list of the inventory item	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-058	User access the inventory list page	1. Navigate to the inventory list page	No test data	The system will display the first five inventory item	**Pass**
UNT-059	The user scrolls down to display more inventory items	1. Navigate to the inventory list page 2. The user scrolls down	No test data	The system will continue displaying the next inventory	**Pass**

		to display more inventory items		items until the end of the inventory list is reached.	
Test Module	Inventory Management – Inventory List Page		**Test Title**	Search inventory an item from the inventory list	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-060	Search inventory items from the inventory list with full inventory item names.	1. Navigate to the inventory list page. 2. The user inputs the search query with the full item name in the search field.	Full search: head gasket	The system will display the inventory item named “head gasket”.	**Pass**
UNT-061	Search inventory items from the inventory list with partial input of inventory name.	1. Navigate to the inventory list page 2. The user inputs the search query in the search field with a partial search	Partial search: proton	The system will display the inventory item name with “proton”.	**Pass**

Test Module	Inventory Management – Inventory List Page		**Test Title**	Click an inventory item in the inventory list	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-062	A user clicks an inventory item in the inventory list to redirect to the inventory item details page.	3. Navigate to the inventory list page 4. The user clicks a desired inventory item in the inventory list	No test data	The system will redirect the user to the corresponding inventory item details page and display all the item details accordingly, including the QR code and the image of the inventory item	**Pass**
Test Module	Inventory Management – Inventory List Page		**Test Title**	Edit the stock of an inventory item from the inventory list	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-063	Edit the stock of an inventory item	1. Navigate to the inventory list page 2. Click an inventory item	i. Stock	The system has successfully updated the inventory item	**Pass**

		from the inventory list. 3. Update the stock quantity 4. Click update stock button		with new details in the inventory list and the Firestore database.	
UNT-064	Edit the stock of an inventory item with empty input	1. Navigate to the inventory list page 2. Click an inventory item from the inventory list. 3. Update the stock quantity by removing it	No test data	The mobile application prevents the user from entering negative values or other data types, and the system will only allow the input of numerical values greater than or equal to zero.	**Pass**

Table 7.16: Unit testing of QR scanning module (mobile application)

Test Module	QR Scanning module		**Test Title**	Scan the QR code on an inventory item	
Test Case ID	**Test Case Description**	**Execution Steps**	**Test Data**	**Expect Result**	**Status**
UNT-065	The user scanned a QR code attached to an inventory item registered in the system through the web application.	1. Navigate to the QR-scanning screen 2. Direct the device's camera towards a QR code attached to an inventory item registered in the system through the web application.	No test data	Upon scanning a QR code on an inventory item, the system should automatically direct the user to the screen displaying the relevant inventory details.	**Pass**
UNT-066	The user scanned a QR code attached to an inventory item not registered in the system through the web application.	1. Navigate to the QR-scanning screen. 2. Direct the device's camera towards a QR code attached to an inventory item not	No test data	After scanning a QR code on an inventory item, the system should notify the user that the QR code is not registered in the	**Pass**

		registered in the system through the web application.		system and not linked to any inventory item.	

7.3 System Usability Testing

In this project, the System Usability Scale (SUS) is being used to evaluate the usability of both web and mobile applications. The SUS was selected for its "quick and dirty" methodology, which provides reliable results for usability testing. This approach is particularly useful in this project, as it involves only one person completing the system within a short timeframe, including both the mobile and web applications. The SUS questionnaire consists of 10 questions with five response options ranging from strongly agree to neutral to strongly disagree, and each option is assigned a score of 1 to 5, with five representing strongly agree and one representing strongly disagree. Since its creation in 1986 by John Brooke, the System Usability Scale (SUS) has evolved into an industry standard and has been referenced in over 1,300 articles and publications. The SUS is advantageous because it can produce reliable results with small sample sizes, making it an efficient testing tool. Moreover, it is a valid measure that effectively distinguishes between usable and unusable systems (usability.gov, n.d.).

The table below displays the User Satisfaction Survey template used to conduct the usability testing, comprising two sections. Section A includes the 10 rating questions, as mentioned earlier, while section B consists of three open-ended questions that allow respondents to provide brief comments on the current system.

Table 7.17: Template of User Satisfaction Survey (Brooke. J, 1986)

Participant No.: Name:					
Question	Strongly Disagree (1)	(2)	Neutral (3)	(4)	Strongly Agree (5)
1. I think I would like to use this system frequently.					
2. I found the system unnecessarily complex.					
3. I thought the system was easy to use.					
4. I think that I would need the support of a technical person to be able to use this system.					
5. I found the various functions in this system were well integrated.					
6. I thought there was too much inconsistency in this system.					
7. I would imagine that most people would learn to use this system very quickly.					
8. I found the tool system cumbersome to use.					
9. I felt very confident using the system.					
10. I needed to learn a lot of things before I could get going with this system.					

1. What do you like best about the system?
2. What do you like least about the system?
3. Do you have any suggestions for improving the current system?

7.3.1 Test Scenarios of Usability Testing

Table 7.18: Usability Testing Scenario to Act as an inventory staff

Test Scenarios to act as an inventory staff
Scenario 1 – Login to an account created by the admin to access the system
Imagine you are an inventory staff and have been provided with login credentials to access the inventory management system's web and mobile applications. Your task is to log in to the application using the provided username and password. What would you do to access the system? Username: lucas Password: test123
Scenario 2 – Interact with the dashboard
Imagine that you are an inventory staff. You need to regularly monitor the inventory levels of the warehouse to ensure that the stock is maintained at an optimal level. Your task is to review the inventory summary in the dashboard and identify any items that have fallen below their reorder point or are out of stock. How would you access this information in both the web and mobile applications?
Scenario 3 – Create an inventory item
Imagine that you are an inventory staff, you have been tasked with adding a new inventory item to the system, and a moderator has provided you with the necessary information about an item. What would you do to add this inventory item through the web application? Hints: you may need to add a category if it does not exist.
Scenario 4 – Search for an inventory item
Imagine that you are an inventory staff tasked to search for an inventory item known as a “Head Gasket”. How would you search for this item in the web and mobile applications?
Scenario 5 – Delete an inventory item
Imagine that you are an inventory staff. You have been tasked with removing an inventory item called "Spark Plug" from the system because it is no longer available in the warehouse. What would you do to delete this inventory item in the web application?
Scenario 6 – Edit an inventory item

Imagine that you are an inventory staff and have been tasked to edit the information of an inventory item you created just now. How would you edit this inventory item's information in the web application?
Scenario 7 – Edit Personal details
Imagine that you are an inventory staff, and you wish to change the personal details in your profile, such as your password. What would you do to edit the personal details in the web application?
Scenario 8 – Scan QR Code in mobile application
Imagine that you are an inventory staff, and you wish to retrieve the details of an inventory item by scanning a QR code. A QR code is pasted on the box of an inventory item prepared by the moderator. How would you get the details of the item by scanning the QR code in the mobile application?
Scenario 9 – Update the stock amount of an inventory item in the mobile application
Imagine that you are an inventory staff and wish to update an inventory item's stock amount. You may access the inventory item details page by searching it or scanning the QR code. How would you update the mobile application's inventory stock amount?
Scenario 10 – Logout from the system
Imagine you are an inventory staff who wishes to log out of the system. What would you do logout from the web application and mobile application

Table 7.19: Usability Testing Scenario to act as super administrator or inventory manager

Test Scenarios to act as a super administrator or inventory manager
Scenario 11 – Login to an account as a super administrator or inventory manager
Imagine you are a super administrator or inventory manager. You have been provided with login credentials to access the inventory management system's web and mobile applications. Your task is to log in to the application using the provided username and password. What would you do to access the system?

Scenario 12 – Create a new inventory staff user
Imagine that you are a super administrator or inventory manager. You need to create a new account for a new employee to manage inventory in the warehouse. What would you do to create this account in the web application?
Scenario 13 – Delete a user from the system
Imagine that you are a super administrator or inventory manager. You need to delete an existing account from the system. What would you do to delete this account in the web application?
Scenario 14 – Search for a user
Imagine that you are a super administrator or inventory manager. You have been tasked to search for a user known as “lukas”. What would you do to search for this user in the web application?
Scenario 15 – Edit a user's details
Imagine that you are a super administrator or inventory manager. You have been tasked to edit the user details of the user you created just now. What would you do to edit the user details in the web application?
Scenario 16 – Change the access assignment for inventory staff
Imagine that you are a super administrator. You have been tasked to change the access assignment for the inventory staff to limit them from accessing the inventory management module. What would you do to change the access assignment in the web application?
Scenario 17 – Change the page permission of a specified page
Imagine that you are a super administrator. You have been tasked to change the page permission for the inventory module, so everyone cannot access it. What would you do to change the page permission in the web application?

7.3.2 Results of Usability Testing

Five respondents were selected to provide feedback on 17 test scenarios during the usability testing process, as outlined in section 7.3.1. The recorded responses of each tester can be found in Appendix B.

The respondent's answers are analyzed to calculate the SUS score by assigning a corresponding number score to each response. The overall SUS score can then be tabulated using the following framework:

i. One is subtracted from the score for all odd-numbered questions to get the actual score.
ii. Five is subtracted from the score for all even-numbered questions to get the actual score.
iii. The total score of all questions answered by a single participant is then added up and multiplied by 2.5 to obtain the percentage score.
iv. The percentage scores for each participant are then summed up and divided by the total number of participants. In this case, the total percentage is divided by 5.

The SUS score for each participant can be determined by following the method shown above. It is essential to understand that the SUS score is a total number out of 100, not a percentage. The average SUS score for a project is 68, which means that a score of 68 will just put you at the 50th percentile, and SUS scores above or below the average might give a quick indication of how usable the design solution is overall (Smyk, 2020).

Table 7.20: General guideline on the interpretation of SUS score (UIUX Trend, n.d.)

SUS Score	Grade	Adjective Rating
> 80.3	A	Excellent
68 – 80.3	B	Good
68	C	Okay
51 – 68	D	Poor
< 51	F	Awful

Based on the testing results, as shown in the table below, the system received an average usability score of 90%, corresponding to a Grade A rating. This indicates that both the mobile and web applications are highly usable and user-friendly

Table 7.21: Summary of User Satisfaction Survey Results

Participants Name	**Usability score for each question**										**Total**	**Percentage (%)**
	1	2	3	4	5	6	7	8	9	10		
Hiew Khai Hang	4	3	4	3	4	4	3	4	4	4	37	92.5
Lim Leong Dong	4	4	3	3	4	4	4	4	4	4	38	95
Ong Lip Wei	4	4	3	3	4	4	4	4	4	4	38	95
Shu Juin Cheng	3	3	3	3	4	4	3	4	3	3	33	82.5
Wong Tack Hwa	3	4	3	4	4	4	2	4	3	3	34	85
Average SUS Score												**90**
Grade												**A**

In addition to the System Usability Scale (SUS) used in the System Usability Testing, some open-ended questions were also prepared to allow respondents to provide brief comments on the current system. This approach helped gather valuable feedback on users' feelings and thoughts towards the implemented system, complementing the quantitative data obtained through the SUS. The open-ended questions used are listed below:

1. What do you like best about the system?
2. What do you like least about the system?
3. Do you have any suggestions for improving the current system?

The table below presents the system's most liked features and functionalities based on participant feedback. However, no least-liked features and functionalities were identified during the study.

Table 7.22: Summary of Participants' Top Liked Features of the System

Summary of Participants' Top-Liked Features of the System
The dashboard provides the user with direct access to the list of items that have reached the reorder point and a list of out-of-stock items without having to check them one by one in the inventory list.
The seamless integration between the mobile and web applications is one of the system's standout features that I most liked.
Able to print out multiple QR at once
I like the create inventory function since it allows the users to enter information in more detail.
The QR scanning feature of the mobile application that I can update the inventory stock on the go

Despite no complaints or least-liked features and functionalities being identified during the study, participants provided suggestions for improving the current system, as shown in the table below. These recommendations are valuable for enhancing the system's usability and overall effectiveness.

Table 7.23: Summary of suggestions for improving the system as recommended by participants

Summary of Suggestions for Improving the System as Recommended by Participants
The name for Role Management in the navigation bar should change to Role/Access Management to allow users to understand the page's function easily.
The interface for the mobile application can be better in terms of UI design.
The weight unit is currently fixed and cannot be changed, limited to kilograms only. It would be beneficial to allow users to choose their preferred units of measurement, such as pounds or grams, for greater

flexibility and convenience.
The details in the inventory details screen are very close together and hard to read, might consider putting some space between them.
Updating the profile will not immediately take effect in the top header bar. It might update this feature so the user does not need to re-login to see the effect.

CHAPTER 8

CONCLUSION AND RECOMMENDATION

8.1 Conclusion

The purpose of this chapter is to provide a conclusion to this project. All objectives outlined in Chapter 1 were successfully achieved which, includes:

1. To develop a web application for inventory management of automotive parts that allows CRUD operations.
2. To develop a mobile application for checking inventory details.
3. To include a Quick Response (QR) code scanning feature for reading inventory details and updating stocks in the mobile application.

The project has successfully fulfilled its goal of creating an inventory management system with mobile and web applications to assist businesses in streamlining their inventory management processes, resulting in increased ease of inventory management.

8.2 Limitations and recommendations for future works

Throughout this system's development and testing phases, various limitations were identified by both myself and the participants involved in the usability testing. These limitations will be summarized in the following section, and recommendations for future work will be provided. This will help address any

current limitations and ensure the system's performance can be optimized in future iterations.

Limitation	Recommendation
The name of the Role Management Module in the navigation bar of the web application was found to be confusing by some users as they did not understand its function. However, upon explanation, they were able to comprehend its purpose.	Change the name of the Role Management Module to Role/Access Assignment Management module.
The UI interface for the mobile application lacks visual appeal.	To enhance the mobile application's visual appeal, implement a UI component like React Native Paper component into the React Native project.
The current version of the system only supports the metric unit of kilograms (kg) in the create inventory form and update inventory form in the web application, without the option to change the weight unit.	To increase flexibility and convenience, it would be beneficial to support users in choosing their preferred units of measurement, such as pounds or grams.
The information displayed on the inventory details screen appears congested and may be difficult to read for some users. To improve the readability and overall user experience	To enhance the readability of the inventory details screen, introduce an additional spacing between the details.
Updating the profile will not immediately take effect in the top header bar. It might update this	To improve user experience, implement a feature that updates the user profile in the top header bar

feature so the user does not need to re-login to see the effect.	without requiring them to log out and log back in to see the changes take effect using the Redux component.
The dashboard lacks interactivity, making it difficult for inventory staff to make informed decisions based on the summary details displayed.	To improve the interactivity of the dashboard, it would be beneficial to add visual aids such as charts or graphs that represent the data more comprehensively. This can provide a clearer picture of the inventory status, allowing staff to make informed decisions quickly.
There is no dedicated management page to handle the system's vendor, brand, and model attributes. Users must manually input these fields as strings, which can lead to inconsistencies and difficulties in data management.	To address this, creating a management page specifically for these attributes would be beneficial, allowing users to manage and reuse them in the system easily. This would increase data consistency and streamline the inventory management process.
The current implementation of the login module does not incorporate any token-based authentication mechanism, which poses a potential security vulnerability to the system.	One possible solution to address the security vulnerability in React or React Native applications is incorporating security libraries such as JSON Web Token (JWT) or OAuth.
The stock alert feature currently does not provide real-time notifications to the user's mobile application.	A possible solution is integrating Firebase Cloud Messaging (FCM) to enable push notifications for the stock alert feature. This would allow users to receive instant alerts on their mobile devices when stock levels reach a certain threshold.

REFERENCES

Algolia, 2023. *How Algolia works.* [online] Available at: <https://www.algolia.com/doc/guides/getting-started/how-algolia-works/> [Accessed 13 April 2023]

Algolia, n.d. *Firestore search with Algolia.* [online] Available at: <https://www.algolia.com/developers/firebase-search-extension/> [Accessed 15 April 2023]

Bachara, M., 2019. *3 Common Auto Parts Inventory Issues and How to Fix Them.* [online] Available at: <https://www.procountwest.com/mikes-blog/3-common-auto-parts-inventory-issues> [Accessed 27 June 2022].

Dikov, D., 2020. *Obsolete Inventory and How to Deal with It.* [online] Available at: <https://medium.com/magnimetrics/obsolete-inventory-and-how-to-deal-with-it-1aa0f5772e27> [Accessed 27 June 2022].

Ethiraj, A., 2022. *Top 7 Inventory Management Challenges & Solutions.* [online] Available at: <https://www.hakunamatatatech.com/our-resources/blog/inventory-management-most-common-challenges-and-solution-to-overcome-in-2022/> [Accessed 27 June 2022].

Farias, R., 2019. *How to set up Firestore and Algolia using Cloud Functions.* [online] Available at: <https://rsfarias.medium.com/how-to-set-up-firestore-and-algolia-319fcf2c0d37> [Accessed 10 April 2023]

Hayes, A., 2022. *Inventory management*. [online] Available at: <https://www.investopedia.com/terms/i/inventory-management.asp> [Accessed 26 June 2022].

Javatpoint, n.d. *ReactJS Architecture*. [online] Available at: <https://www.javatpoint.com/reactjs-architecture> [Accessed 16 April 2023]

Kamble, S., 2021. A QR code technology for centralised inventory management system. International Research Journal of Engineering and Technology (IRJET), [e-journal] 8(4). https://www.irjet.net/archives/V8/i4/IRJET-V8I4289.pdf

Macas, C.V.M., Aguirre, J.A.E. and Arcentales-Carrión, R., 2021. Inventory management for retail companies: A literature review and current trends. 2021 Second International Conference on Information Systems and Software Technologies (ICI2ST), [e-journal] pp. 71-78. https://doi.org/10.1109/ICI2ST51859.2021.00018

Martinez, P., 2020. *What is Evolutionary Prototype?* [online] Available at: <https://mockitt.wondershare.com/prototyping/evolutionary-prototyping.html> [Accessed 29 June 2022]

Nikolaieva, A., n.d. *8 Best Software Development Methodologies*. [online] Available at: <https://www.uptech.team/blog/software-development-methodologies> [Accessed 1 July 2022]

Oladele, A., n.d. *Top 15 Software Development Methodologies: Benefits and Drawbacks.* [online] Available at: <https://www.velvetech.com/blog/software-development-methodologies/> [Accessed 30 June 2022]

Patil, A., 2022. *Deep dive into React Native's New Architecture*. [online] Available at: <https://medium.com/coox-tech/deep-dive-into-react-natives-new-architecture-fb67ae615ccd> [Accessed 16 April 2023]

Pisuwala, U., 2022. *The benefits of ReactJS and reasons to choose it for your project*. [online] Available at: < https://www.peerbits.com/blog/reasons-to-choose-reactjs-for-your-web-development-project.html> [Accessed 28 June 2022]

Smyk, A., 2020. *The System Usability Scale & How It's Used in UX*. [online] Available at: <https://xd.adobe.com/ideas/process/user-testing/sus-system-usability-scale-ux/> [Accessed 11 April 2023]

Stevenson, D., 2018. *What is Firebase? The complete story, abridged*. [online] Available at: < https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0> [Accessed 28 June 2022]

UIUX Trend, n.d. *Measuring and Interpreting System Usability Scale (SUS).* [online] Aavailable at: <https://uiuxtrend.com/measuring-system-usability-scale-sus/> [Accessed 17 April 2023]

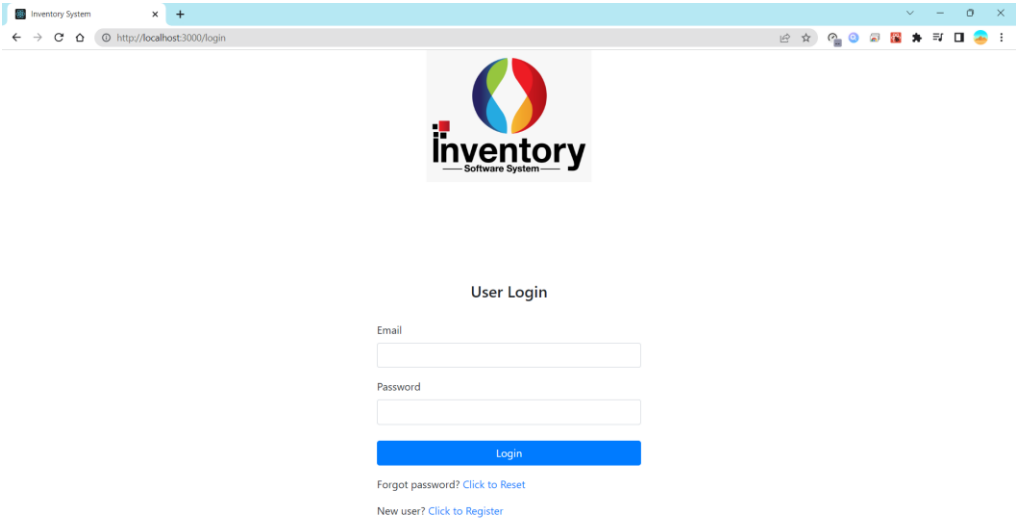
usability.gov, n.d. *System Usability Scale (SUS).* [online] Available at: <https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html> [Accessed 16 April 2023]

Vazquez, A., 2021. *How to Improve Stock Control with a Barcode Inventory System.* [online] Available at: <https://www.g2.com/articles/barcode-inventory-system> [Accessed 27 June 2022].

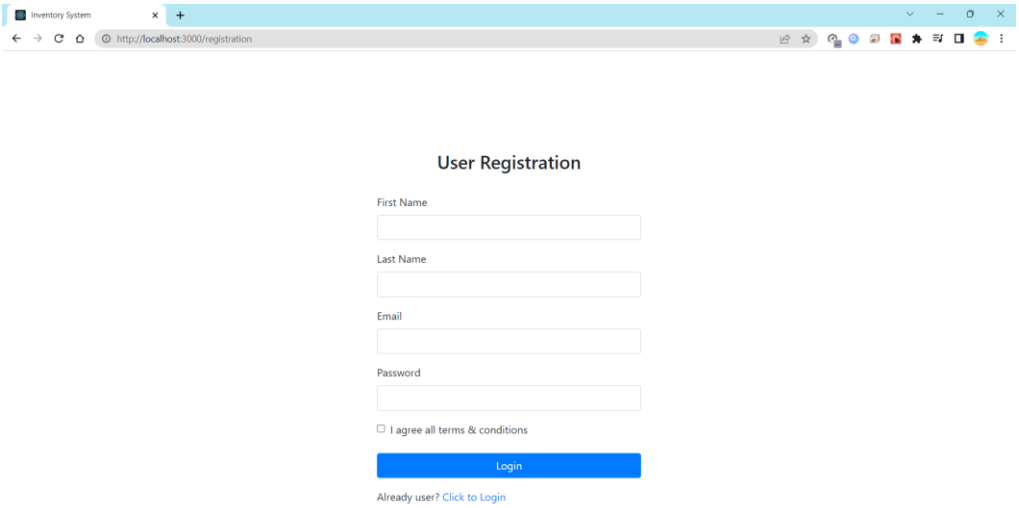
Weiss, D., n.d. *How to Avoid Inventory Overstocks and Understocks*. [online] Available at: <https://www.skunexus.com/blog/how-to-avoid-inventory-overstocks-and-understocks> [Accessed 28 June 2022]

APPENDICES

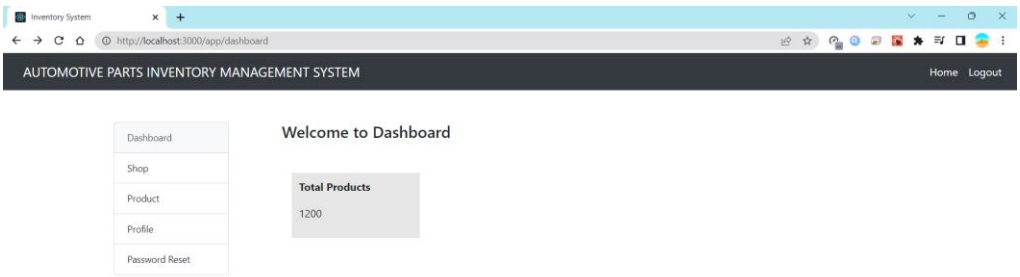
Appendix A: Low-fidelity Prototypes for Web Application



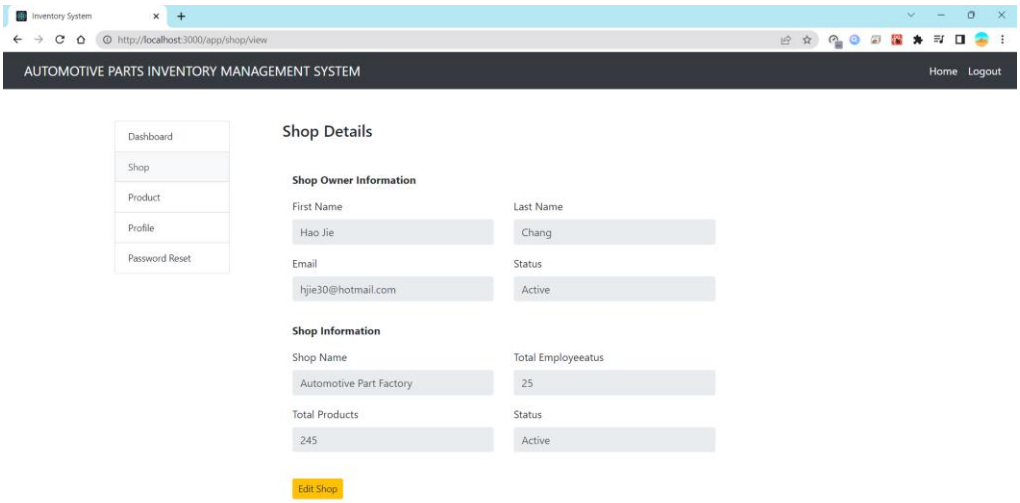
Login Page



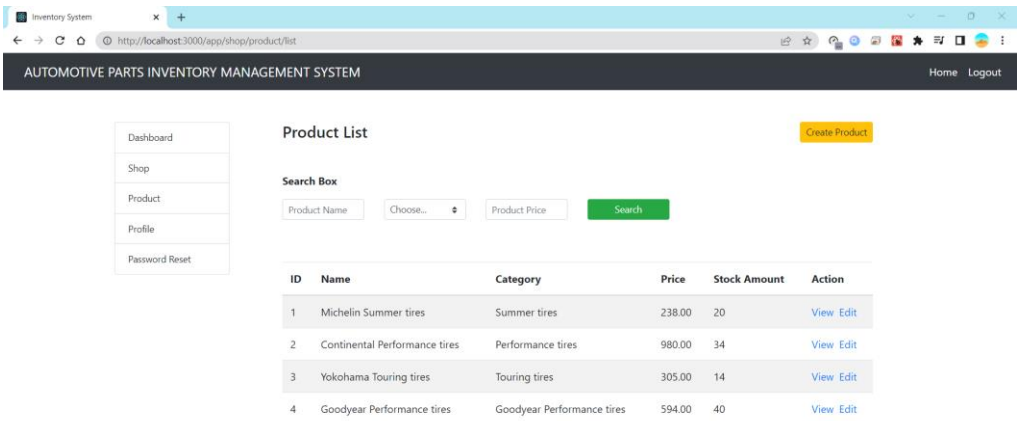
Registration Page



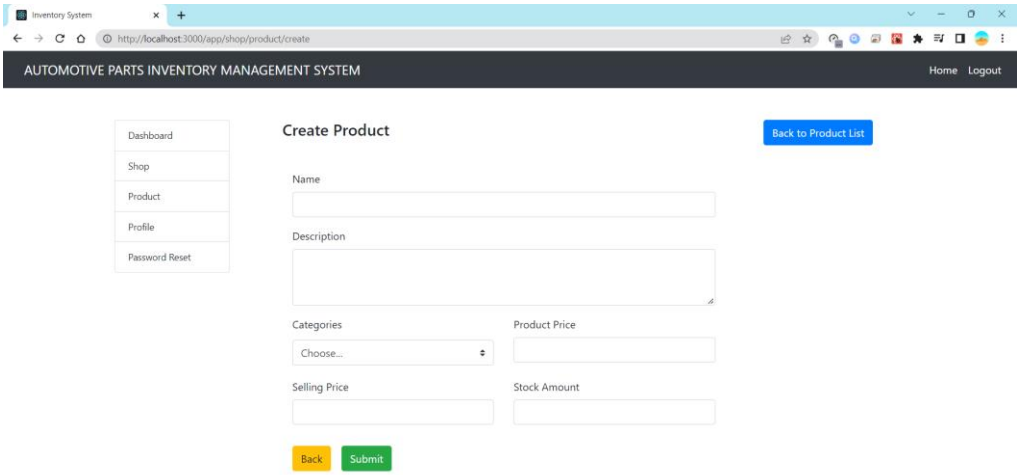
Main page (Dashboard)



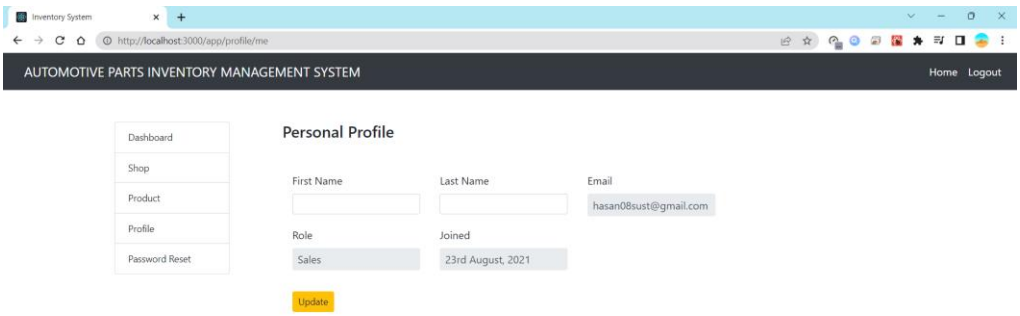
Shop Details Page



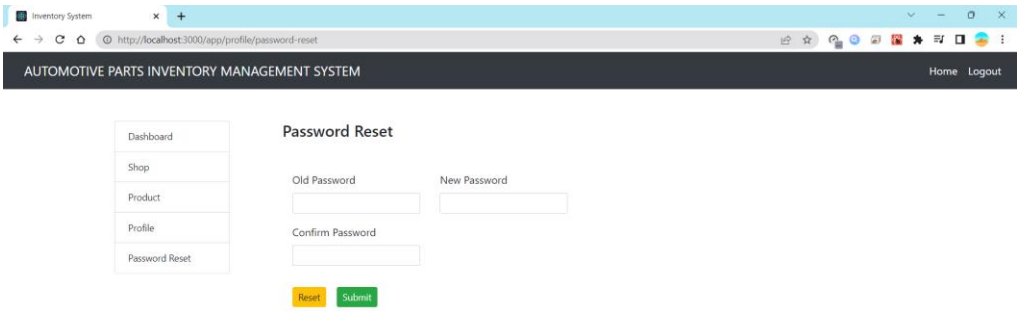
Product List Page



Create Product Form Page

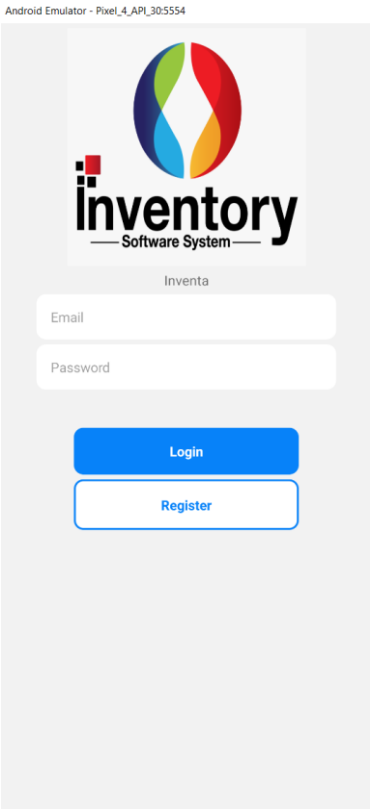


Profile Page

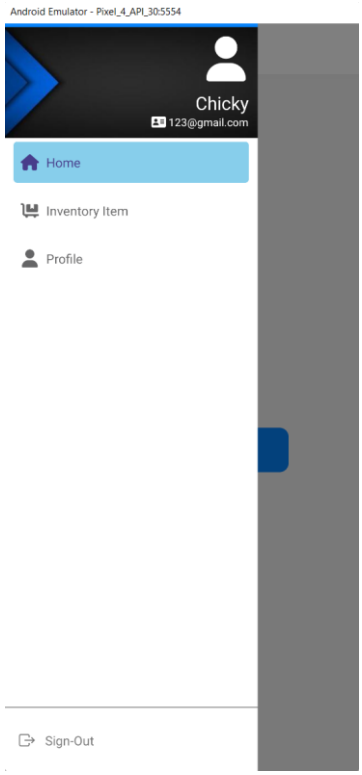


Password Reset Page

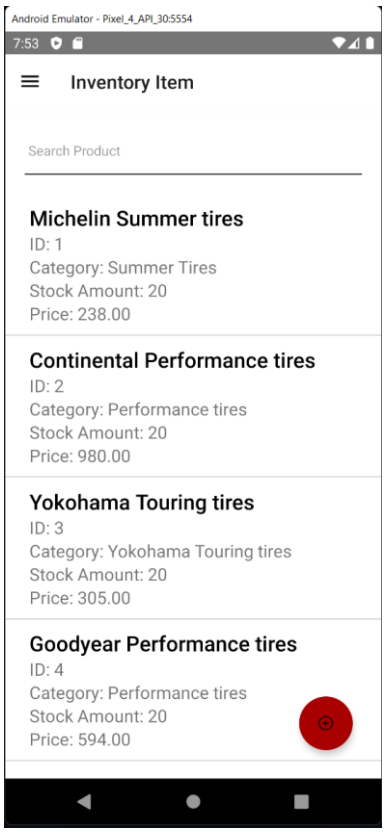
Appendix B: Low-fidelity Prototypes for Web Application



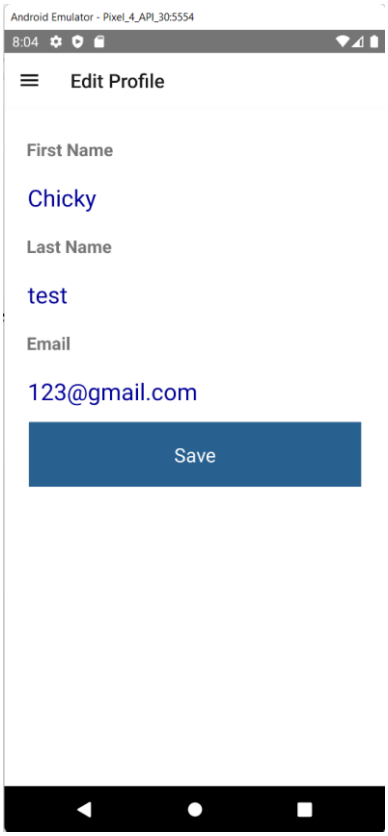
Login and Registration Page



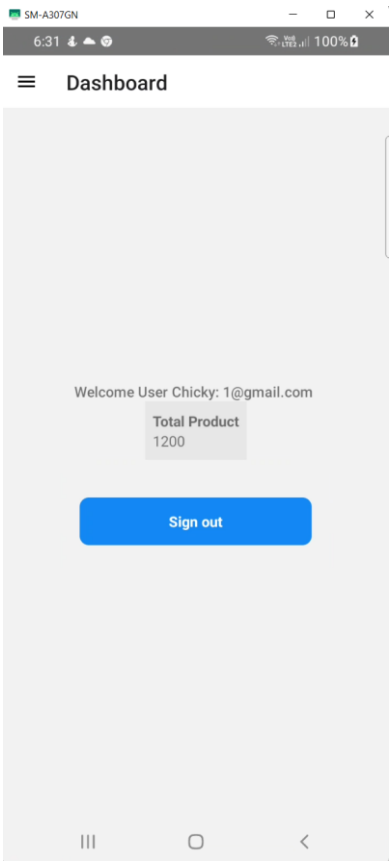
Navigation Drawer



Item List (Product List) Page & Search Item Page



Edit Profile Page



Dashboard Page

Scan Product QR page

Appendix C: Usability test responses

Participant No.: 1 Name: Hiew Khai Hang					
Question	Strongly Disagree (1)	(2)	Neutral (3)	(4)	Strongly Agree (5)
1. I think I would like to use this system frequently.					✓
2. I found the system unnecessarily complex.		✓			
3. I thought the system was easy to use.					✓
4. I think that I would need the support of a technical person to be able to use this system.		✓			
5. I found the various functions in this system were well integrated.					✓
6. I thought there was too much inconsistency in this system.	✓				
7. I would imagine that most people would learn to use this system very quickly.				✓	
8. I found the tool system cumbersome to use.	✓				
9. I felt very confident using the system.					✓
10. I needed to learn a lot of things before I could get going with this system.	✓				

1. What do you like best about the system?

The dashboard provides the user with direct access to the list of items that have reached the reorder point, as well as a list of out-of-stock items without having to check it one by one in the inventory list

2. What do you like least about the system?

   None

3. Do you have any suggestions for improving the current system?

   The name for Role Management in the navigation bar should change to Role/Access Management to allow user to understand the function of the page easily.

Participant No.: 2 Name: Lim Leong Dong					
Question	Strongly Disagree (1)	(2)	Neutral (3)	(4)	Strongly Agree (5)
1. I think I would like to use this system frequently.					✓
2. I found the system unnecessarily complex.	✓				
3. I thought the system was easy to use.				✓	
4. I think that I would need the support of a technical person to be able to use this system.		✓			
5. I found the various functions in this system were well integrated.					✓
6. I thought there was too much inconsistency in this system.	✓				
7. I would imagine that most people would learn to use this system very quickly.					✓
8. I found the tool system cumbersome	✓				

to use.					
9. I felt very confident using the system.					✓
10. I needed to learn a lot of things before I could get going with this system.	✓				

1. What do you like best about the system?

   The seamless integration between the mobile and web applications is one of the system's standout features that I most liked

2. What do you like least about the system?

   None

3. Do you have any suggestions for improving the current system?

   The interface for the phone can be better in terms of UI design.

Participant No.: 3 Name: Ong Lip Wei					
Question	Strongly Disagree (1)	(2)	Neutral (3)	(4)	Strongly Agree (5)
1. I think I would like to use this system frequently.			✓		
2. I found the system unnecessarily complex.	✓				
3. I thought the system was easy to use.					✓
4. I think that I would need the support of a technical person to be able to use this system.	✓				
5. I found the various functions in this system were well integrated.				✓	
6. I thought there was too much inconsistency in this system.		✓			

7. I would imagine that most people would learn to use this system very quickly.					✓
8. I found the tool system cumbersome to use.	✓				
9. I felt very confident using the system.					✓
10. I needed to learn a lot of things before I could get going with this system.	✓				

1. What do you like best about the system?

   Able to print out multiple QR at once

2. What do you like least about the system?

   None

3. Do you have any suggestions for improving the current system?

   The weight unit is currently fixed and cannot be changed, limited to kilograms only. It would be beneficial to allow users to choose their preferred unit of measurement, such as pounds or grams, for greater flexibility and convenience.

Participant No.: 4 Name: Shu Juin Cheng					
Question	Strongly Disagree (1)	(2)	Neutral (3)	(4)	Strongly Agree (5)
1. I think I would like to use this system frequently.				✓	
2. I found the system unnecessarily complex.		✓			
3. I thought the system was easy to use.				✓	
4. I think that I would need the support of a technical person to		✓			

be able to use this system.					
5. I found the various functions in this system were well integrated.					✓
6. I thought there was too much inconsistency in this system.	✓				
7. I would imagine that most people would learn to use this system very quickly.				✓	
8. I found the tool system cumbersome to use.	✓				
9. I felt very confident using the system.				✓	
10. I needed to learn a lot of things before I could get going with this system.		✓			

1. What do you like best about the system?

   I like the create inventory function since it is allow the users to enter their information in more detail.

2. What do you like least about the system?

   None

3. Do you have any suggestions for improving the current system?

   The details in the inventory details screen is very close together and hard to read.

Participant No.: 5 Name: Wong Tack Hwa					
Question	Strongly Disagree (1)	(2)	Neutral (3)	(4)	Strongly Agree (5)
1. I think I would like to use this system frequently.				✓	
2. I found the system unnecessarily complex.	✓				
3. I thought the system was easy to use.				✓	
4. I think that I would need the support of a technical person to be able to use this system.	✓				
5. I found the various functions in this system were well integrated.					✓
6. I thought there was too much inconsistency in this system.	✓				
7. I would imagine that most people would learn to use this system very quickly.			✓		
8. I found the tool system cumbersome to use.	✓				
9. I felt very confident using the system.				✓	
10. I needed to learn a lot of things before I could get going with this system.		✓			

1. What do you like best about the system?

   <u>The QR scanning feature of the mobile application that I can update the inventory stock on the go.</u>

2. What do you like least about the system?

None

3. Do you have any suggestions for improving the current system?

   Updating the profile will not immidiately take effect in the top header bar, might update this feature, so user not need to relogin to see the effect.