RFID-INTEGRATED IOT FOR AUTOMATED VEHICLE AND VISITOR AUTHENTICATION IN UNIVERSITIES

BY

CHIN WHYE TING

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF INFORMATION TECHNOLOGY (HONOURS) (COMMUNICATIONS AND NETWORKING)

Faculty of Information and Communication Technology

(Kampar Campus)

FEBRUARY 2026

COPYRIGHT STATEMENT

This Final Year Project report is submitted in partial fulfillment of the requirements for the degree of Bachelor of Information Technology (Honours) (Communications and Networking) at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project report represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project report may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ACKNOWLEDGEMENTS

I would like to extend my sincerest gratitude and appreciation to my supervisor, Ms. Tan Lyk Yin, for her invaluable guidance, insightful feedback, and constant encouragement throughout the course of this project. Her expertise and support have been instrumental in navigating the challenges of developing this system.

My special thanks also go to the project moderator, Puan Athirah Binti Rosli, for her time and effort in evaluating this project and providing constructive suggestions for its improvement. I am also grateful to the lab officer, Mr. Fidaus, for his technical assistance and for ensuring the necessary hardware and equipment were available and functional.

Finally, my deepest thanks go to my family and friends for their unwavering support, patience, and encouragement throughout this journey.

ABSTRACT

Unauthorized vehicle entry and the illegal resale of parking permits are two major issues in university parking management that result in inefficiency, unfairness for registered users, and security vulnerabilities on campus. By designing and developing a full-stack IoT University Entry and Parking System (uPark) for automated visitor and vehicle authentication, this project tackles these problems through a cloud-integrated approach.

A Raspberry Pi 4 Model B serves as the system's edge foundation, utilizing an MFRC522 RFID reader for authorized staff and students and a Logitech USB webcam for QR-code-based visitor authentication. Node-RED serves as the central logic engine, orchestrating the system and maintaining real-time synchronization with a Google Firebase backend (Cloud Firestore and Realtime Database). The system architecture is designed to be highly secure and scalable, enabling real-time anti-passback enforcement and remote hardware control through a native Flutter mobile application.

The successful development of the functional prototype is documented in this final report. The system demonstrates the ability to read RFID tags, scan visitor QR codes, validate credentials against the cloud, and provide instant physical feedback through LED indicators. Additionally, the administrative dashboard allows for seamless user management and real-time remote camera control. By confirming the viability of the suggested solution to improve campus security and guarantee a more efficient and equitable parking environment, the successful implementation verifies the selected full-stack architecture.

Area of Study: Internet of Things, RFID Technology

Keywords: RFID, Internet of Things (IoT), Access Control, Flutter, Firebase, Campus Parking, Raspberry Pi 4

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

β	beta
Ω	Ohm (resistance)

LIST OF ABBREVIATIONS

5G	Fifth Generation
API	Application Programming Interface
CPU	Central Processing Unit
FR	Functional Requirement
GPIO	General Purpose Input Output
IDE	Integrated Development Environment
IoT	Internet of Things
IP	Internet Protocol
JSON	JavaScript Object Notation
NFR	Non-Functional Requirement
PWM	Pulse Width Modulation
QR	Quick Response
RAM	Random Access Memory
RBAC	Role-Based Access Control
RFID	Radio Frequency Identification
RTDB	Realtime Database
SDK	Software Development Kit
SPI	Serial Peripheral Interface
UI	User Interface

Chapter 1

Introduction

1.1 Project Background [1][2][3]

Universities are dynamic and high-traffic environments where effective vehicle access and parking management are essential for daily operations. There is a significant transportation problem within university locations as a result of the recent, unusually high growth in the number of students and passenger car owners [1]. Traditionally, campus entry relies heavily on manual verification methods, such as security guards visually checking physical vehicle stickers or issuing paper passes to visitors. Without an efficient and well-managed parking system, universities often grapple with issues such as traffic congestion, difficulties in finding available parking spots, and widespread frustration among campus users [2]. While this manual approach has been the standard for years, it introduces significant vulnerabilities and inefficiencies. During peak hours, manual checks frequently lead to severe traffic congestion at entry gates. Furthermore, physical stickers are easily transferred or illegally resold among students, making it difficult for security personnel to accurately verify the driver's true identity in real-time. Consequently, authorized users who have officially registered for permits often struggle to find available parking spaces because unauthorized vehicles occupy the lots. The difficulty in finding a parking spot is a significant issue, with studies in UTAR Kampar Campus revealing that up to 90% of students find it hard to find parking, and 80% reported being late to or missing classes as a direct result [3]. This not only causes frustration and disrupts academic schedules but also compromises campus security by allowing unverified individuals to enter the premises without proper tracking.

Advancements in the Internet of Things (IoT) and cloud computing present viable solutions to overcome these campus management challenges. Integrating Radio Frequency Identification (RFID) technology with cloud databases and mobile applications allows for automated, seamless, and highly secure access control. Unlike manual sticker checks, automated systems can instantly scan an RFID tag and cross-reference it with a live database to verify authorization, significantly reducing human error and processing time at the gates. Furthermore, the use of cross-platform mobile applications allows for innovative features, such as generating digital QR code invitations for visitors. For administrators and security staff, these technologies provide centralized management, real-time monitoring of gate activities, and automated, tamper-proof audit logs. For students and staff, an automated system enhances

convenience, speeds up entry, and ensures a fairer parking environment by enforcing strict digital rules, such as cloud-based anti-passback mechanisms that prevent multiple vehicles from exploiting a single credential.

This project proposes the development of a comprehensive, full-stack IoT University Entry and Visitor Authentication System, named uPark. The uPark system is designed to integrate edge hardware including RFID readers and QR code scanners, with a robust cloud backend using Firebase, and a dedicated mobile application for administrators, staff, and students. For the university management, the system offers features such as automated gate control, real-time cloud validation, and remote hardware management, thereby eliminating the heavy workload and inaccuracies of manual operations. By leveraging these modern digital technologies, the project aims to improve entry efficiency, eradicate the illegal sharing of parking permits, and enhance overall campus security. In the long term, the proposed system is expected to optimize parking resource utilization, establish a highly secure environment, and deliver a seamless daily commuting experience for the entire university community.

1.2 Problem Statement and Motivation

University campuses accommodate a massive daily influx of students, staff, and visitors, making effective parking and access management a critical operational priority. With rapid technological advancements, the expectation for seamless, secure, and automated facility access has grown significantly. However, many university parking systems still rely heavily on manual verification and offline physical credentials, creating a distinct gap between modern security standards and actual campus operations. This section addresses that gap by examining the specific limitations of current parking management methods and highlighting the motivation for developing a fully integrated, cloud-based IoT entry solution.

One major problem is the heavy reliance on physical vehicle stickers for access control, which has unfortunately fostered an environment of unauthorized credential sharing and illegal permit resale. Because traditional gates and offline readers cannot accurately track entry-and-exit states, students without valid permits can easily borrow or purchase physical stickers from others to bypass security. This unauthorized access not only diminishes the availability of limited parking spaces for legitimate, paying students but also compromises campus safety. An IoT-based RFID system equipped with a cloud-synchronized anti-passback mechanism

eliminates this issue by permanently linking a physical tag to a digital user profile and strictly preventing a single credential from being used for multiple simultaneous entries.

Another significant issue is the highly inefficient and insecure visitor management process. Many universities still depend on manual logbooks, paper passes, and verbal confirmations at the guardhouse to process guests, which routinely causes severe traffic bottlenecks during peak hours. Furthermore, these manual records are prone to human error, difficult to audit, and easily falsified by individuals attempting to enter the campus under false pretenses. By introducing a mobile application that allows staff to generate admin-approved QR code invitations, the university can drastically reduce gate waiting times, ensure that all visitors are securely pre-verified, and maintain a highly accurate, digital audit trail.

The lack of centralized, real-time monitoring and remote hardware control also remains a persistent challenge for campus security administrators. In traditional setups, security personnel cannot view live gate access logs, instantly revoke lost access cards, or remotely manage gate hardware without being physically present at the guardhouse. This lack of visibility severely limits their ability to respond to security incidents or system malfunctions promptly. Implementing a full-stack architecture that utilizes a real-time cloud database combined with an edge logic controller (Node-RED) empowers administrators with live dashboards, instant credential management, and the ability to remotely control hardware components—such as activating physical QR scanners—directly from a mobile device.

In summary, the reliance on easily shared physical stickers, outdated manual visitor logs, and decentralized management systems severely impacts campus security, operational efficiency, and user satisfaction. These interconnected problems highlight the urgent need for an intelligent, cloud-connected access control solution. By addressing these critical vulnerabilities, the proposed uPark system aims to eradicate unauthorized parking, streamline visitor processing, and provide administrators with robust, real-time management tools, ultimately supporting a safer and more orderly campus environment.

1.3 Project Objectives

Based on the challenges identified in the background and problem statement—namely unauthorized credential sharing, inefficient manual visitor processing, and the lack of real-time, centralized administrative control—this project sets out clear objectives to address these issues through a comprehensive full-stack IoT solution. The proposed uPark system is designed not only to eradicate unauthorized campus access and streamline visitor entry but also to empower

security personnel with robust, real-time monitoring capabilities. To achieve these outcomes, the following specific, measurable, achievable, relevant, and time-bound (SMART) objectives have been formulated in alignment with the project's overall aim:

Objective 1: To develop an automated RFID-based access control system equipped with a cloud-synchronized anti-passback mechanism.
Objective 2: To implement a secure QR-code-based mobile visitor management module to streamline the visitor registration and check-in process.
Objective 3: To design and deploy a real-time, cross-platform administrative dashboard for centralized user management and remote hardware control.

The primary objective of this project is to develop an automated RFID-based access control system equipped with a cloud-synchronized anti-passback mechanism. This development involves integrating physical RFID hardware with an edge logic controller (Node-RED) and linking it directly to a Cloud Firestore database. This system directly addresses the first problem of unauthorized permit sharing by binding physical credentials to digital user profiles and enforcing strict entry/exit status checks at the gate. The success of this objective is measured by the system's demonstrated ability to successfully authenticate valid users, log entry events to the cloud, and definitively deny access to a secondary vehicle attempting to use a credential that is already logged as "IN" on the campus.

The second objective is to implement a secure QR-code-based mobile visitor management module to streamline the visitor registration and check-in process. This objective directly addresses the inefficiencies associated with manual visitor logbooks. Using Flutter, a mobile application is developed to allow university staff to request visitor passes and administrators to approve them, generating a unique QR code tied to a specific Cloud Firestore document ID. The measurability of this objective is validated by the physical system's ability to scan the generated QR code, query the cloud database for the specific document ID, verify the approval status in real-time, and subsequently trigger the physical gate hardware to allow entry.

The third objective is to design and deploy a real-time, cross-platform administrative dashboard for centralized user management and remote hardware control. This directly tackles the limitations of traditional, decentralized security setups. This objective involves building a native mobile interface via Flutter and a web-based dashboard via Node-RED, both synchronized via Firebase Realtime Database and Firestore. These interfaces allow security

personnel to instantly register or revoke user credentials, view live gate access logs without refreshing, and remotely trigger physical hardware (such as waking up the physical QR scanner camera via the Realtime Database). The achievement of this objective is confirmed by successfully demonstrating these real-time data synchronizations and remote hardware triggers during system testing.

In summary, the objectives emphasize developing, implementing, and evaluating a robust, cloud-connected IoT access control system. All objectives are designed to be specific, measurable, achievable, and relevant to the project's aim of solving current campus parking vulnerabilities. These objectives are fully completed and validated within the final year project duration, ensuring that the outcomes remain realistic, manageable, and time-bound.

1.4 Project Contributions

The proposed full-stack uPark System contributes significantly to addressing the security vulnerabilities and operational inefficiencies faced by traditional campus management. By automating gate access and integrating cloud-based technologies, the system supports improved security, seamless visitor management, and centralized administrative control. These contributions ensure that universities can provide a secure, equitable, and efficient parking environment while optimizing their internal security operations.

The first contribution is eradicating unauthorized parking and restoring fairness for registered users. By implementing an automated RFID access system with a cloud-synchronized anti-passback mechanism, the project ensures that physical credentials cannot be illegally shared or reused simultaneously by multiple vehicles. This strict digital enforcement reduces congestion, maximizes the availability of limited parking spaces for legitimate permit holders, and significantly strengthens the campus's first line of defense against unverified entry.

The second contribution is streamlining the visitor registration and check-in process. By transitioning from manual paper logbooks to a secure, mobile-based QR code invitation system, guests can be pre-approved by administrators and processed instantly at the gate. This digital modernization significantly reduces traffic bottlenecks and waiting times at the guardhouse, eliminates human error in record-keeping, and provides security personnel with a tamper-proof, highly accurate digital audit trail of all non-student entries.

The third contribution is empowering security administrators with centralized, real-time management capabilities. The deployment of a cross-platform dashboard and the integration of the Firebase Realtime Database allow administrators to monitor live gate logs, instantly revoke

lost credentials, and remotely control edge hardware—such as waking up the physical QR scanner—directly from their mobile devices. This immediate oversight improves the security team's response time to potential incidents, reduces the burden of manual guardhouse operations, and greatly enhances overall system efficiency.

In summary, the contributions of this project lie in eliminating unauthorized permit sharing, modernizing visitor access, and providing real-time, remote administrative control. Together, these contributions not only solve existing vulnerabilities in traditional parking systems but also create long-term value for university management by establishing a smarter, more secure, and highly efficient campus infrastructure.

1.5 Project Scope and Direction

The project scope outlines the specific boundaries of the system by defining what will be included and what will be excluded. This ensures the project remains focused, feasible within the given timeframe, and strictly aligned with the core objectives of developing a secure, cloud-integrated university parking solution, uPark.

Inclusions:

- Provide a cross-platform mobile application developed using Flutter, featuring dedicated interfaces and functionalities for distinct user roles: Administrators, Staff, and Students.
- Integrate edge hardware components at the simulated checkpoint, specifically utilizing a Raspberry Pi 4, an MFRC522 RFID reader, and a USB webcam, orchestrated locally via Node-RED logic and Python scripts.
- Implement a robust cloud backend using Firebase (utilizing both Cloud Firestore and Realtime Database) to manage user credentials, log entry events, and enforce a real-time anti-passback mechanism.
- Enable a digital visitor management workflow that allows staff to request visitor passes and generates admin-approved QR codes for seamless guest authentication at the scanner.
- Support remote hardware control capabilities, allowing administrators to instantly wake up or shut down the physical USB webcam from the mobile app via Realtime Database triggers.

Exclusions:

- This project does not include the physical installation or mechanical integration of actual motorized boom gates; access "granted" or "denied" states are simulated using green and red LED indicators.
- This project does not include Automatic License Plate Recognition (ALPR/ANPR) technology or advanced AI computer vision, relying exclusively on secure RFID tags and QR codes for identification.
- This project does not include integration with existing university legacy databases or student information systems; the prototype is operate on a standalone, self-contained Firebase backend.
- This project does not cover a full-scale, campus-wide deployment, serving instead as a functional, single-checkpoint proof-of-concept.

In summary, the project scope strictly focuses on the development, hardware-software integration, and prototyping of an RFID and QR-code-based IoT access system. By clearly defining these boundaries, the project ensures a realistic development process while successfully demonstrating the core automated entry, cloud synchronization, and remote management functionalities. The following chapter, Chapter 2, will explore the literature review, analyzing existing technologies and past research relevant to this proposed system.

1.6 Report Organization

This report is organized into seven chapters, detailing the complete lifecycle of the project from initial concept to final evaluation.

Chapter 1: Introduction introduces the project background, identifies the problem statements regarding traditional campus parking, and outlines the project objectives, contributions, and scope.

Chapter 2: Literature Review examines the core technologies utilized in this project, such as mobile application frameworks, cloud databases, and IoT hardware, while comparing existing smart parking systems to highlight the proposed system's advantages.

Chapter 3: System Methodology justifies the selection of the Prototyping model for development. It also details the specific hardware and software requirements, project milestones, and estimated costs.

Chapter 4: System Design provides a blueprint of the project, featuring the overall system architecture, use case diagrams, activity diagrams, and a comprehensive database design using Google Firebase.

Chapter 5: System Implementation discusses the practical setup and coding phase. It explains the configuration of the edge hardware, the cloud backend, and the development of the Flutter mobile application, including specific logic flows like Role-Based Access Control (RBAC).

Chapter 6: System Evaluation and Discussion presents the results of the system testing. It analyzes hardware response times, database synchronization speeds, and documents the pass/fail functional test cases to prove that all project objectives were achieved.

Chapter 7: Conclusion and Recommendation summarizes the overall achievements of the project and suggests potential features and improvements for future development.

Chapter 2
Literature Review

2.1 Review of Technologies

This section reviews the core technologies utilized in the development of the IoT-integrated university entry and parking system. It explores the application domain, mobile development frameworks, and database solutions. Each technology is introduced, evaluated for its strengths and weaknesses, and summarized to justify its selection for this project.

2.1.1 Field of Study: Smart Campus Access Management [4]

The transformation of traditional universities into 'intelligent campuses' has become a global priority to improve operational efficiency and the quality of life for students and staff. Villegas-Ch et al. [4] argue that university campuses serve as ideal, controlled ecosystems for deploying and testing Internet of Things (IoT) infrastructures that can eventually be scaled up to smart city levels. Within this smart campus framework, parking and access management represent critical administrative processes that traditionally suffer from resource mismanagement, traffic congestion, and security gaps. Digital IoT-based solutions are therefore essential to modernize these administrative tasks, reduce human effort, and provide personalized, secure experiences for the campus community."

2.1.2 Mobile Application Frameworks

Mobile application frameworks provide the necessary tools and libraries to build software for mobile devices. To create a system accessible to administrators, staff, and students, an appropriate framework must be chosen. The main approaches include native development (specifically for Android or iOS) and cross-platform development (building for multiple OS from a single codebase).

2.1.2.1 Native Android (Java/Kotlin) [5]

These days, Android is the most widely used operating system worldwide, not just for mobile devices but also for desktop and laptop computers [5]. Native Android development involves building applications exclusively for the Android operating system using languages like Java or Kotlin.

Strengths: It provides the highest level of performance and allows developers to have direct, unrestricted access to the device's native hardware features (like cameras and GPS).

Weaknesses: The major drawback is that the code only works on Android devices. To release the app on Apple devices, a completely separate iOS application must be written from scratch, doubling the development time and effort.

2.1.2.2 React Native [6], [7]

Facebook unveiled React Native, an open-source cross-platform JavaScript framework, on March 15. It aims to assist developers and businesses who find it difficult to handle the complexity of creating cross-platform applications [6].

Strengths: It allows developers to write one codebase that works on both iOS and Android. It also has a massive community and a vast library of third-party plugins.

Weaknesses: Empirical performance evaluations demonstrate that React Native consumes significantly higher CPU and memory resources compared to Flutter and Native Android. This performance degradation is largely attributed to the overhead required by its JavaScript bridge during resource-intensive task [7].

2.1.2.3 Flutter [7]

Flutter is a modern open-source UI software development kit created by Google, utilizing the Dart programming language.

Strengths: Flutter offers a single codebase for iOS and Android and compiles directly to native machine code, completely avoiding the performance bridge used by React Native. Comparing to cross-platform frameworks demonstrate that Flutter offers superior performance over React Native. Specifically, Flutter provides comparable frame rates to native applications and demonstrates highly optimized response times even during hardware-intensive tasks, such as accessing device cameras [7].

Weaknesses: Applications built with Flutter tend to have a slightly larger file size upon download compared to native applications. Additionally, the Dart programming language has a smaller developer community than JavaScript.

2.1.2.4 Summary of Mobile Application Frameworks [8], [9]

Table 2.1 Mobile Application Frameworks

Framework	Strengths	Weaknesses
Native Android	Maximum performance; direct access to native hardware APIs.	Only works on Android; requires a separate codebase for iOS.
React Native	Single codebase for iOS/Android; large community.	Performance issues due to the JavaScript bridge.
Flutter	High performance (no bridge); excellent UI customization; strong Firebase integration.	Larger app file size; Dart is less commonly used than JavaScript.

Flutter is selected for this project. Native Android is not chosen because it requires building a completely separate codebase for iOS users, which doubles the development time. Because the system requires an application for students, staff, and admins across different mobile devices, a cross-platform solution is mandatory to save time. Recent benchmarking studies have shown that Flutter provides a significant performance advantage over React Native, particularly in UI rendering and responsiveness, due to its ability to compile directly to native machine code without the need for a JavaScript bridge [8], [9]. Flutter is chosen over React Native due to its superior performance and its native-like synergy with Firebase, which is essential for the system's real-time remote hardware control features.

2.1.3 Cloud Backend and Databases

The database serves as the digital memory of the system, storing user profiles, access logs, and hardware trigger commands. The choice of database determines how quickly and securely the mobile app and the physical hardware can communicate.

2.1.3.1 SQLite (Local Database) [10]

SQLite is a lightweight, serverless, and file-based relational database. Major web browsers, PCs, smart TVs, car media systems, and the PHP and Python programming languages all are using SQLite [10].

Strengths: It requires no internet connection to operate, makes local edge computing very fast, and is highly resource-efficient.

Weaknesses: It is strictly isolated to the local device (e.g., the Raspberry Pi). It cannot easily synchronize data with a mobile application or a cloud server, making remote management impossible.

2.1.3.2 MySQL [11]

MySQL is a traditional, widely used relational database management system (RDBMS) that stores data in structured tables. Under the terms of the General Public License (GPL), MySQL is a free implementation of a relational database management system (RDBMS). MySQL is available to all users without restriction, although it cannot be used as a commercial derivative product [11].

Strengths: It is highly reliable, excellent for complex data relationships, and follows a strict schema that ensures data consistency.

Weaknesses: It typically requires a dedicated backend server (like Node.js or PHP) to handle requests between the database and the mobile app. Furthermore, it is not inherently designed for instant, real-time data synchronization.

2.1.3.3 Firebase (Cloud Firestore & Realtime Database) [12], [13]

Firebase is a Cloud backend-as-a-service (BaaS) provided by Google. It offers NoSQL databases, specifically Cloud Firestore (for structured document storage) and Realtime Database (for low-latency JSON data).

Strengths: For IoT applications requiring sub-second data synchronization, Firebase's Realtime Database and Firestore have emerged as superior alternatives to traditional relational databases [12], [13]. The event-driven architecture of Firebase allows for instant state updates, which is critical when triggering physical hardware like entry gates and cameras [12].

Weaknesses: It is a NoSQL database, meaning complex querying and data filtering can be more difficult than in SQL. Additionally, it creates vendor lock-in to Google's ecosystem.

2.1.3.4 Summary of Cloud Backend and Databases

Table 2.2 Cloud Backend and Databases

Database	Strengths	Weaknesses
SQLite	Lightweight; operates offline; fast local queries.	No cloud synchronization; isolated to a single device.
MySQL	Excellent for structured data and complex queries; reliable.	Requires server setup; lacks built-in real-time synchronization.
Firebase	Real-time data sync; serverless architecture; fast IoT hardware triggering.	Complex queries are limited; vendor lock-in to Google

Firebase is selected as the cloud backend for this project. SQLite is rejected because it operates entirely offline, making remote synchronization with a mobile app impossible. Similarly, MySQL is not selected because it lacks built-in, real-time synchronization and requires dedicated server setups. Specifically, a hybrid approach is used: Cloud Firestore is utilized as the permanent storage for user credentials, RFID tags, and QR invitations, while the Realtime Database is utilized for its ultra-low latency to instantly trigger the Raspberry Pi hardware (e.g., waking up the webcam). This combination is superior to MySQL or SQLite for an IoT system requiring remote mobile control.

2.1.4 IoT Integration Platforms

An IoT integration platform serves as the central "brain" or middleware of the hardware setup. It is responsible for bridging the physical hardware sensors with the cloud database. Choosing the right platform dictates how easily custom logic can be implemented at the edge (on the device itself).

2.1.4.1 Node-RED [14]

Node-RED is an open-source, visual flow-based programming tool developed by IBM, designed specifically to wire together hardware devices, APIs, and online services. The complexity and time needed to develop IoT applications can be decreased with the help of Node-RED, an open-source visual programming tool that makes the process easier. It offers an easy-to-use interface for creating data flows between devices and services [14].

Strengths: It provides a highly intuitive drag-and-drop interface that simplifies complex data flows without requiring extensive coding. It is extremely lightweight, making it ideal for edge computing on devices like the Raspberry Pi. Furthermore, it easily executes external Python scripts and has dedicated packages for integrating with Firebase.

Weaknesses: While visual, it still requires a solid understanding of JSON and data routing. Its built-in dashboard UI is somewhat basic compared to dedicated mobile app builders.

2.1.4.2 ThingsBoard [15]

ThingsBoard is a comprehensive, open-source IoT platform designed for data collection, processing, and device management [15]. The platform's Device Management capabilities ensure secure provisioning and real-time monitoring of all hardware. All incoming data is

processed through its powerful Rule Engine, which allows for the creation of complex rules to trigger actions and automate responses based on specific conditions.

Strengths: It offers powerful, out-of-the-box data visualization dashboards and a robust rule engine for processing incoming telemetry data. It is highly scalable for enterprise-level deployments.

Weaknesses: It is heavily resource-intensive. Running a local instance of ThingsBoard on a standard single-board computer often leads to performance issues, making it less suitable for lightweight edge logic.

2.1.4.3 Blynk [16]

Blynk serves as the primary user-facing interface and control platform for the Smart Home system, providing a framework to easily create a powerful mobile application for both Android and iOS [16]. Chosen for its affordability and user-friendly design, Blynk allows homeowners to remotely monitor and control their household from a smartphone or PC

Strengths: It offers a highly user-friendly, drag-and-drop mobile app builder. It is excellent for quickly prototyping remote-control systems for hobbyist microcontrollers like Arduino or ESP8266.

Weaknesses: Developers are strictly locked into the Blynk ecosystem and UI. It lacks the deep, flexible backend data manipulation required to connect custom hardware scripts to third-party databases like Cloud Firestore.

2.1.4.4 Summary of IoT Integrated Platforms

Table 2.3 IoT Integrated Platforms

Platform	Strengths	Weaknesses
Node-RED	Visual flow-based; lightweight for edge computing; easily triggers local Python scripts and Firebase APIs.	Requires understanding of JSON payloads; built-in UI is relatively basic.
ThingsBoard	Excellent data visualization; strong enterprise device management.	Resource-intensive; overkill for simple edge logic.
Blynk	Rapid mobile UI building; easy plug-and-play setup.	Ecosystem lock-in; poor flexibility for custom database integrations.

Node-RED is selected as the IoT integration platform. ThingsBoard is not selected because it is heavily resource-intensive and overkill for simple edge logic on a Raspberry Pi. Since a custom Flutter app handles the user interface, a mobile builder like Blynk is unnecessary. Node-RED serves perfectly as the edge logic controller on the Raspberry Pi, orchestrating the physical hardware (executing Python scripts for RFID and the webcam) and securely formatting the payloads to synchronize with the Firebase Cloud database.

2.1.5 Single-Board Computers / Microcontrollers

To process hardware inputs and communicate with the cloud, a localized processing unit is required at the entry gate. Single-board computers offer the necessary processing power, operating system support, and GPIO (General Purpose Input/Output) pins for physical connections.

2.1.5.1 Raspberry Pi (Model 4) [17]

The Raspberry Pi serves as the central processing unit for the prototype setup. The Raspberry Pi was intentionally designed to be a very powerful and flexible computer at a fraction of the price of a standard computer so that anyone may use it to come up with innovative solutions to issues [17]. Chosen for its versatility and cost-effectiveness, it runs the software necessary to interface with the RFID reader, capture tag data, and manage communication with the backend system. The Raspberry Pi is a highly popular, low-cost single-board computer that runs a full Linux-based operating system.

Strengths: It offers an unbeatable balance of price and performance, featuring multiple GBs of RAM, built-in Wi-Fi, and a 40-pin GPIO header. It boasts a massive community, making troubleshooting and library integration (such as Python hardware libraries) extremely easy.

Weaknesses: Because it runs an entire operating system from a microSD card, it is susceptible to data corruption if the device suddenly loses power without safely shutting down.

2.1.5.2 NVIDIA Jetson Nano Dev Kit [18]

The NVIDIA Jetson Nano serves as the dedicated artificial intelligence (AI) engine for the system. The Jetson Nano was developed by NVIDIA to make powerful, embedded machine learning accessible for DIY projects, moving beyond its traditional high-end applications [18].

Strengths: It features a 128-core Maxwell GPU, providing massive parallel processing power capable of handling complex, real-time computer vision and heavy AI tasks efficiently.

Weaknesses: It is significantly more expensive than standard boards and draws more power. For a system relying on simple QR code scanning rather than heavy AI facial recognition, this board is heavily over-engineered.

2.1.5.3 BeagleBone Black [19]

The BeagleBone Black is utilized as the project's primary hardware interaction platform. A review in The Register describes it as a board designed for "hardware hackery," enabling users to get closer to fundamental components like UART and I²C buses, which are often hidden by modern abstraction layers [19].

Strengths: It excels in industrial applications, offering an impressive 92 expansion pins for diverse electronic interfacing, and it includes built-in onboard flash storage (eMMC), which is more stable than an SD card.

Weaknesses: It has a smaller community and a steeper learning curve compared to the Raspberry Pi, meaning less immediate support and fewer pre-built software libraries are available.

2.1.5.4 Summary of Single-Board Computers

Table 2.4 Single-Board Computers

Hardware	Strengths	Weaknesses
Raspberry Pi 4	Excellent price-to-performance; massive community support; fully supports Node-RED and Python.	SD card corruption risks upon sudden power loss.
NVIDIA Jetson Nano	Powerful GPU for real-time AI and machine learning tasks.	Expensive; high power draw; overkill for standard QR scanning.
BeagleBone Black	Industrial-grade stability; 92 GPIO pins; built-in flash storage.	Steeper learning curve; smaller community support.

The Raspberry Pi 4 is selected as the central host system. The NVIDIA Jetson Nano is not selected because it is too expensive and draws too much power; its powerful GPU is overkill since the system relies on simple QR code scanning rather than heavy AI facial recognition. The BeagleBone Black is also rejected because it has a steeper learning curve and a smaller community, making troubleshooting and finding pre-built Python hardware libraries more difficult. During the FYP1 prototype, the older Raspberry Pi 3 struggled with performance lag

when running Node-RED, Python scripts, and UI elements simultaneously. The upgraded RAM and processing capabilities of the Pi 4 provide the necessary power to smoothly handle the concurrent operations of the RFID reader, USB webcam, and real-time Firebase syncing without bottlenecking.

2.1.6 RFID Reader Technologies

Radio Frequency Identification (RFID) technology is used for the automated authentication of physical user credentials (vehicle stickers or cards). Different RFID frequencies dictate the read range, security, and application of the system.

2.1.6.1 RDM6300 (Low Frequency - 125 kHz)

The RDM6300 operates on a low-frequency band.

Strengths: Low-frequency waves penetrate non-metal materials like water and tissue very well. The modules are highly reliable and extremely inexpensive.

Weaknesses: They typically only support read-only tags, offer zero data encryption, and have a very slow data transfer rate with a strictly short range (under 10 cm).

2.1.6.2 MFRC522 (High Frequency - 13.56 MHz)

The MFRC522 module operates on a high-frequency band and interacts with MIFARE family cards.

Strengths: It provides an excellent balance of cost and functionality. It supports two-way communication (read and write capabilities) and incorporates cryptographic security protocols to protect the data on the tags.

Weaknesses: The read range is relatively short (up to a few centimeters), requiring users to physically "tap" or bring their card very close to the scanner.

2.1.6.3 SparkFun M6E Nano (Ultra-High Frequency - 860-960 MHz)

UHF RFID systems are designed for long-range identification.

Strengths: They offer exceptional read ranges (often several meters) and the ability to scan hundreds of tags simultaneously, making them ideal for highway toll collections or warehouse inventory.

Weaknesses: UHF signals are easily obstructed or reflected by liquids and metals. Furthermore, the hardware and necessary antennas are highly complex and expensive.

2.1.6.4 Summary of RFID Technologies

Table 2.5 RFID Technologies

Technology	Strengths	Weaknesses
RDM6300 (LF)	Reliable; penetrates non-metal materials well; very low cost.	Read-only; lacks security/encryption; slow data rate.
MFRC522 (HF)	Supports read/write functions; built-in security protocols; ideal for secure access control.	Requires close-proximity scanning ("tap-and-go").
SparkFun UHF	Long read range (meters); supports multi-tag scanning.	Expensive; signals easily disrupted by metal/water.

The MFRC522 (HF) reader is selected for this project. The RDM6300 (LF) is not selected because it is strictly read-only and lacks data encryption, which poses a security risk for access control. While UHF readers offer convenient long-range scanning, they are cost-prohibitive for a prototype and suffer from interference in dynamic environments. The MFRC522 perfectly suits the requirement for a deliberate, secure "tap" at a gate. Furthermore, its cryptographic features and ability to write data to tags align with the system's strict security requirements.

2.2 Review of Existing Systems / Applications

This section examines existing academic and commercial systems relevant to campus parking and access control. By reviewing prior research and existing applications, we can identify common approaches, their strengths, weaknesses, and the specific gaps that this project aims to address with its advanced, integrated solution.

2.2.1 Development of an RFID-based Campus Parking System [20]

Hidayat et al. (2021) [20] presented an experimental study focused on improving campus parking systems, which often rely on manual checks and fee collection. An RFID system can be used, for example, on campus or in a building's parking lot where the owner of the vehicle is someone who frequently visits or works there.

The system used RFID technology for campus parking access yields a number of significant advantages. Compared to conventional manual procedures, it delivers more efficiency by automating the process, making it more reliable, accurate, and quick. In addition to saving time

for those coming into and going out of the parking lot, this automation also helps to improve security at the entry points.

However, this system has several drawbacks. It does not define features that are outside of its primary emphasis, which is entry/exit control at the parking gate. The system lacks the granular role-based access control techniques required to handle varied user groups with particular access permissions, and the authentication offered seems inadequate, focusing on a simple subscriber ID check. With a restricted focus on automated gate entry, the developed system appears to exhibit only limited core functionality overall.

Strengths

- Increased Efficiency
- Reduced Time Wasted & Increased Security
- Robust, Faster, and More Accurate

Weakness

- Focuses only on Entry/Exit control
- Limited Authentication / No Granular Role-Based Access Control
- Basic Core Functionality

2.2.2 IoT-Based Smart Vehicle Parking System Using RFID [21]

Koya et al. (2024) [21] proposed an "intelligent and efficient solution" for modernizing vehicle parking management. Their project leverages the Internet of Things (IoT) and Radio-Frequency Identification (RFID) technologies with the aim of optimizing the parking process, enhancing user experience, and providing real-time monitoring and control. Using a parking management system would save time and human labor. The system uses RFID tags on vehicles and readers at entry/exit points for identification and automated access. Additionally, infrared sensors (IR sensors) are implemented to monitor the occupancy status of parking spaces, transmitting this real-time data on availability to a web admin panel. The hardware components used include a NodeMCU (ESP8266), RFID sensor, IR sensors, LCD, and Servo motors.

The IoT-Based Smart Vehicle Parking System using RFID presents several notable strengths. It effectively combines both RFID and IoT technologies, directly aligning with the core approach of the proposed project. The system provides real-time monitoring of parking space occupancy through IR sensors, which is a valuable feature that extends beyond basic gate control. It offers automated entry and exit processes linked to user accounts via RFID and

utilizes a capable IoT development board (NodeMCU ESP8266) for true internet connectivity and centralized management.

However, upon analysis, this system exhibits certain limitations relevant to our specific problems. Despite monitoring occupancy, the paper provides limited scope for in-lot enforcement, not clearly describing how it identifies or manages unauthorized vehicles parked within the lot based on their identity. It lacks the use of using role based access control (RBAC) needed for a complex environment like a campus with diverse user categories and access rules. Other than that, the paper lacks empirical results or validation from testing against the proposed metrics, limiting the demonstrated effectiveness of the solution.

Strengths

- Combining both RFID and IoT technologies
- Provides real-time monitoring of parking space occupancy
- Offers automated entry and exit processes using RFID and user accounts

Weakness

- Limited scope of In-Lot Enforcement
- Lacks the use of Role-Based Access Control (RBAC)
- Lack of Empirical Results/Validation

2.2.3 Smart Parking Management System Integrating ANPR and IoT Sensors [22]

Ditta et al. (2024) [22] proposed a Smart Parking Management System (SPMS) designed to address urban parking challenges. Their system integrates Automatic Number Plate Recognition (ANPR) through image processing and a sensor-based hardware system leveraging the Internet of Things (IoT), specifically IR sensors. The core aim is to automate parking processes, eliminate manual registration, and optimize space utilization by processing number plates at entry for identification and using IoT sensors to monitor real-time slot occupancy. This real-time data is transmitted to a web administration panel for efficient remote management, providing information on entry/exit times, parking duration, and billing costs.

The system supports flexible billing based on actual parking duration. The Smart Parking Management System offers several strengths. It effectively combines ANPR for vehicle identification at entry with IoT sensors for real-time parking space occupancy monitoring, providing a more comprehensive view than systems focused solely on gate access. Automate parking processes, eliminate manual registration is the core aim and also it strength.

However, analysis reveals certain limitations in this Smart Parking Management System. The design does not include an explicit mechanism for implementing granular role-based access control (RBAC) to manage different user categories with specific permissions. Besides that, the paper does not describe a method to prevent or detect the unauthorized use of a vehicle/plate linked to a registered account. As the system relies on ANPR, its performance is inherently susceptible to environmental conditions that can affect camera accuracy.

Strengths

- Provides real-time monitoring of parking space occupancy
- Combines ANPR for identification and IoT sensors for occupancy monitoring
- Automates entry via ANPR, eliminating manual registration

Weakness

- Absence of detailed Role-Based Access Control (RBAC) design
- No mechanism to address potential unauthorized plate/vehicle use
- Inherits the inherent limitations of ANPR technology

2.2.4 Proposed System and Summary

Table 2.6 Proposed System and Summary

Feature/System	**Hidayat et al. (2021)**	**Koya et al. (2024)**	**Ditta et al. (2024)**	**Proposed System**
Core Technology	RFID	RFID + IoT (IR Sensor)	ANPR + IoT (IR Sensor)	RFID + IoT + Cloud (Firebase) + Mobile App (Flutter)
Authentication	Entry/Exit via RFID	Entry/Exit via RFID & User Accounts	ANPR (License Plate)	Automated Vehicle/Visitor Authentication via RFID and QR Codes, with Cloud Validation
Role-Based Access Control (RBAC)	Absent	Absent	Absent	Implemented to manage diverse user categories and access rights (Admin, Staff, Student, Visitor)
Visitor Management	Not specified	Not specified	Limited/Manual (implied)	Secure QR code invitation system via mobile app,

				admin approval workflow.
Real-time Admin Control & Monitoring	Limited	Limited (web panel)	Limited (web panel)	Comprehensive live dashboard (Node-RED UI & Flutter App) for user management, live logs, and remote hardware control via Firebase RTDB.
Anti-Passback Mechanism	Absent	Absent	Absent	Implemented (Cloud-synchronized campus_status check)
Data Storage	Local (implied)	Local/Cloud (NodeMCU)	Local/Cloud (implied)	Cloud Firestore (permanent data) & Realtime Database (hot data/triggers)

Strengths of the Proposed System:

Comprehensive Integration: Unlike prior systems that focused narrowly on entry/exit or occupancy, this project integrates RFID, QR codes, a sophisticated cloud backend (Firebase), and native mobile applications for a holistic solution.

Granular Role-Based Access Control (RBAC): This is a key differentiator. The system implements RBAC across multiple user categories (Admin, Staff, Student, Visitor), allowing for sophisticated management of specific access privileges crucial for a university campus.

Enhanced Security & Fair Access: By combining cloud-based authentication with an anti-passback mechanism and QR code validation for visitors, the system ensures that only authorized individuals can access campus facilities, directly addressing unauthorized access and restoring fairness.

Real-time Cloud Synchronization & Remote Control: The use of Firebase (Firestore and Realtime Database) provides immediate data updates and allows for remote control of hardware components (like the webcam), enabling administrators to manage security dynamically from anywhere.

Modern User Experience: The Flutter mobile application provides a user-friendly interface for staff and administrators to manage invitations and credentials, while students and staff benefit from faster, automated entry.

Reliable Identification: Leveraging RFID for core authentication offers a more robust and less environmentally sensitive solution for access control compared to ANPR systems.

Chapter 3

System Methodology

3.1 System Development Models

The methodology for developing a software and IoT system requires a structured approach to guide the entire lifecycle, from initial planning to final deployment. Selecting an appropriate System Development Life Cycle (SDLC) model is crucial as it reduces project risks, ensures deliverables meet expectations, and provides a clear timeline for hardware and software integration. Several SDLC models exist, each with different workflows. This section compares three prominent models before selecting the most suitable one for this project.

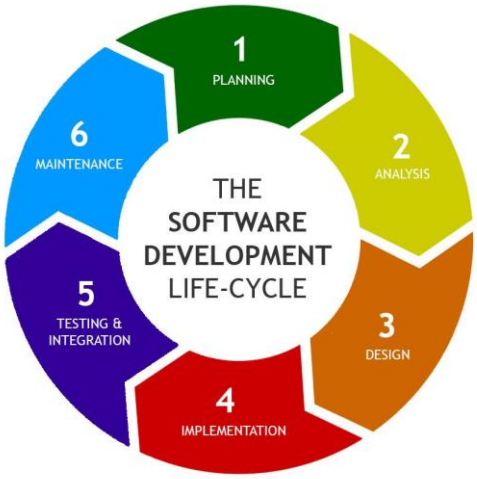


Figure 3.1.1 Software Development Life-Cycle

3.1.1 System Development Model 1: Waterfall Model

The Waterfall methodology is a traditional, linear approach to system development. It divides the project into distinct, sequential phases (Requirements, Design, Implementation, Testing, Deployment), where each phase must be fully completed before the next one begins.

Strengths: It is highly structured, simple to understand, and enforces rigorous documentation at every stage, making it easy to manage if requirements are fixed.

Weaknesses: It is extremely rigid. Once a phase is completed, it is difficult and costly to go back and make changes if unforeseen hardware or software integration issues arise.

Suitability: This model is not suitable for this IoT project, as hardware-software integration often requires trial, error, and frequent adjustments.

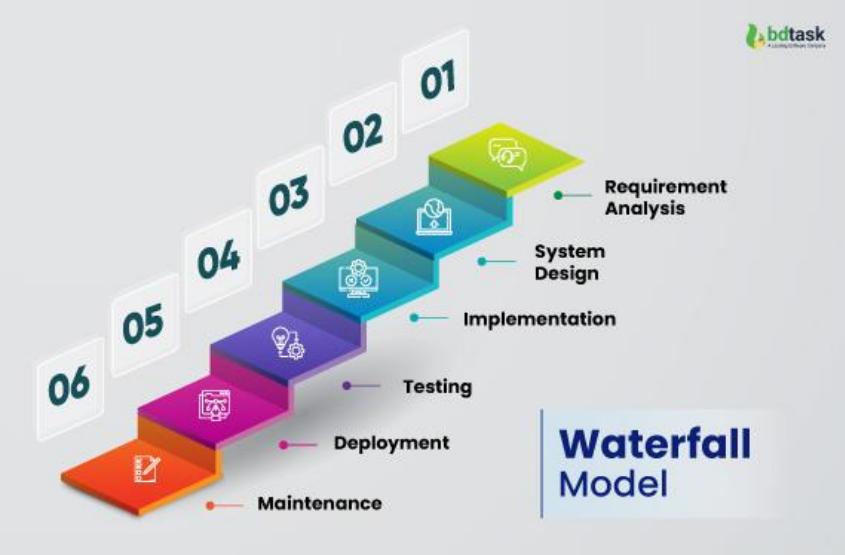


Figure 3.1.2 Waterfall Model

3.1.2 System Development Model 2: Agile Model

The Agile methodology is an iterative and collaborative approach that focuses on delivering software in small, incremental cycles called "sprints." It emphasizes continuous feedback from users, high team collaboration, and the flexibility to adapt to changing requirements rapidly.

Strengths: It is highly adaptable to changing requirements and ensures continuous testing, which generally leads to a higher-quality software product that meets user needs.

Weaknesses: It is difficult to predict final timelines and budgets. Furthermore, Agile is highly optimized for pure software development and struggles to accommodate the heavy, upfront physical hardware setups required in IoT projects.

Suitability: While flexible, this model is less suitable for the initial foundational hardware setup required in this project.

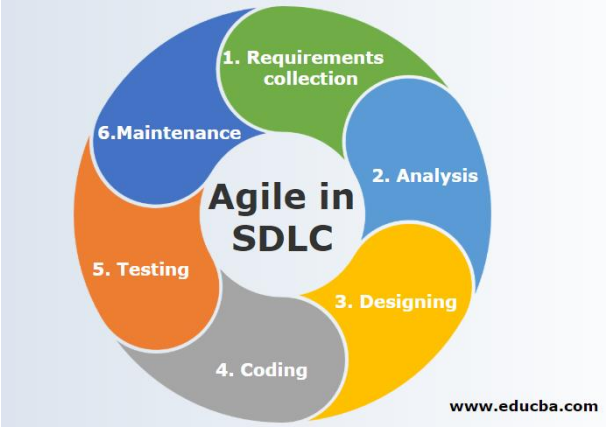


Figure 3.1.3 Agile Model

3.1.3 System Development Model 3: Prototyping Model

The Prototyping methodology is an iterative approach that centers on developing an early, working model (a prototype) of the system. This initial model is tested and evaluated, and the feedback gathered is used to refine the system in subsequent versions until the prototype evolves into the final, complete product.

Strengths: It is excellent for clarifying requirements and identifying technical integration challenges between hardware and software early in the development lifecycle.

Weaknesses: If not managed properly, it can lead to "scope creep" (endlessly adding new features), and users may mistake a fragile early prototype for a finished product.

Suitability: This model fits the project perfectly, as building a proof-of-concept is the best way to test the integration of RFID, webcams, Node-RED, and Firebase.

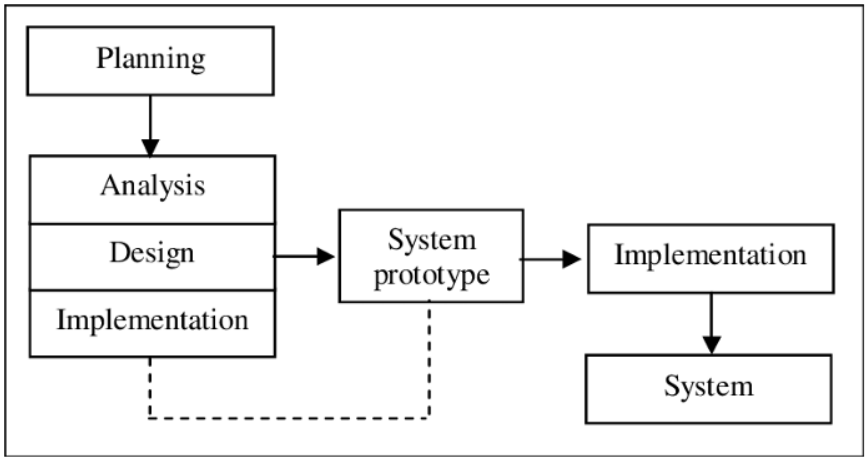


Figure 3.1.4 Prototype Model

3.1.4 Selected Model: Prototyping Model

After evaluating the options, the Prototyping Model is selected for this project. The Waterfall model is too rigid to handle the unpredictable nature of integrating physical edge hardware with cloud services. While Agile offers flexibility, it is less suited for projects requiring upfront hardware assembly. The Prototyping approach is chosen because it allows for the rapid development of a core working model (initially the basic RFID reader and local database in FYP1). By testing this early prototype, the project could safely scale up in FYP2 to include more complex features, such as the Flutter mobile app, QR scanning via a USB webcam, and real-time Firebase cloud synchronization, refining the system iteratively based on practical testing.

3.2 System Requirements

The successful development and deployment of the full-stack IoT University Entry and Parking System require a set of well-defined system requirements. These requirements are divided into two main categories: hardware and software. The hardware components ensure that the physical edge logic can run effectively at the simulated gate, while the software components provide the necessary tools, platforms, and cloud environments to build, test, and synchronize the system.

3.2.1 Hardware

3.2.1.1 Central Processing Unit: Raspberry Pi 4 Model B (4GB)

Figure 3.2.1 Raspberry Pi 4 Model B

The Raspberry Pi 4 Model B serves as the central edge processing unit for the IoT checkpoint. Upgraded from the Pi 3 used in FYP1 to overcome performance bottlenecks, this powerful single-board computer handles multiple concurrent tasks. It runs the Node-RED application, executes Python scripts for the RFID reader and the USB webcam, and manages constant internet communication with the Firebase cloud. Its 40-pin GPIO header interfaces directly with the MFRC522 reader and LED indicators.

Table 3.1 Raspberry Pi 4 Model B Specification & Requirement

Specification	**Requirement**
Model	Raspberry Pi 4 Model B
RAM	4GB LPDDR4-3200 SDRAM
Processor	Broadcom BCM2711, Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
Connectivity	2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, Gigabit Ethernet
I/O	40-pin GPIO header, 2x USB 3.0 ports, 2x USB 2.0 ports

3.1.1.2 RFID Reader: MFRC522 RFID Reader

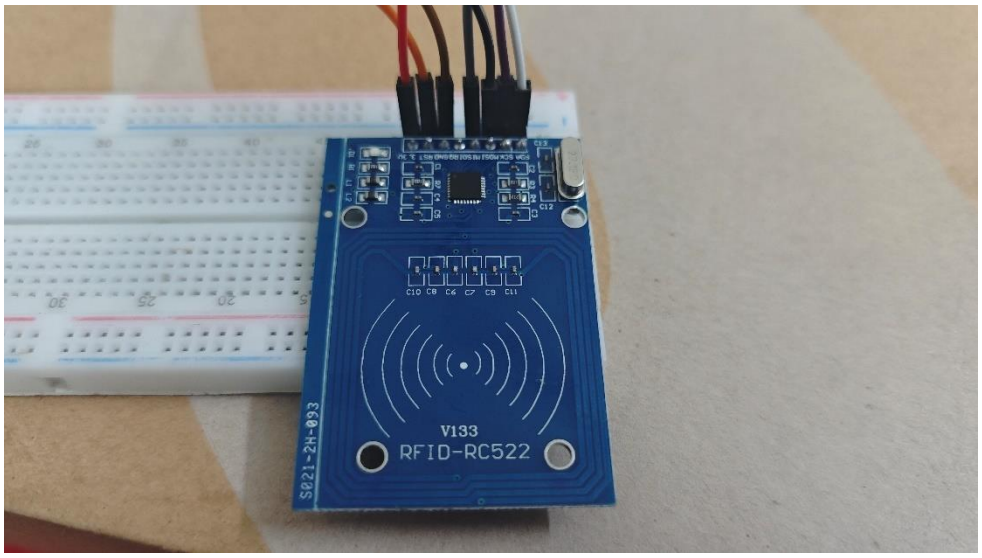


Figure 3.2.2 MFRC 522

The MFRC522 remains the primary hardware peripheral for reading the physical credentials of students and staff. It operates at 13.56 MHz and communicates with the Raspberry Pi via the SPI (Serial Peripheral Interface) protocol to quickly capture Unique Identifiers (UIDs) for the cloud anti-passback authentication check.

Table 3.2 MFRC552 Specification & Requirement

Specification	**Requirement**
Operating Frequency	13.56 MHz
Interface	SPI (Serial Peripheral Interface)
Supported Cards	MIFARE 1k, MIFARE 4k, MIFARE Ultralight
Operating Voltage	3.3V

3.1.1.3 User Credentials: MIFARE Classic 1K RFID Cards/Tags

Figure 3.2.3 MIFARE RFID Card/ Tag

These passive RFID cards serve as the physical credentials for registered university users. Each card contains a unique, non-changeable identifier (UID). In this upgraded system, the UID is no longer stored locally but is permanently linked to a user's digital profile within the Firebase Cloud Firestore.

Table 3.3 MIFARE 1K RFID Card/ Tag Specification & Requirement

Specification	**Requirement**
Operating Frequency	13.56 MHz
Type	Passive RFID Card/Tag
Memory	1KB
Compatibility	ISO/IEC 14443 Type A

3.1.1.4 Visual Scanner: Logitech C720 HD Webcam

Figure 3.2.4 Logitech C720 HD Webcam

A new addition to the FYP2 architecture, the Logitech C720 HD webcam acts as the physical QR code scanner at the simulated entry gate. When triggered remotely by the administrator via the Realtime Database, the webcam activates to capture the visitor's QR code displayed on their mobile device. The Raspberry Pi then processes the video frame to extract the embedded Firestore Document ID, which is subsequently used to verify the visitor's invitation status.

Table 3.4 Logitech C720 HD Webcam Specification & Requirement

Specification	**Requirement**
Model	Logitech C720 HD Webcam
Resolution	720p HD (1280 x 720)
Interface	USB 2.0
Focus Type	Fixed focus
Functionality	Real-time video frame capture for QR decoding

3.2.2 Software

3.2.2.1 Operating System: Raspberry Pi OS

Raspberry Pi OS (formerly Raspbian) provides the Debian-based Linux environment necessary to operate the edge hardware. It hosts all background services required to run Python, Node-RED, and the camera drivers seamlessly.

Table 3.5 Raspberry Pi OS Specification & Requirement

Specification	**Requirement**
Version	Raspberry Pi OS (x64 bit)
Requirement	A microSD card with at least 32GB of storage
Kernel	Linux

3.2.2.2 Integration Platform: Node-RED

Node-RED serves as the core edge logic engine. In this upgraded system, its primary role is to bridge the "dumb" Python hardware scripts with the Firebase cloud. It formats the captured RFID and QR data into complex JSON payloads (using the "Swiss Army Knife" structure) required by the node-red-contrib-firebase-admin package to execute Firestore writes.

Table 3.6 Node-RED Specification & Requirement

Specification	**Requirement**
Version	v3.0+
Runtime	Node.js v16+
Key Palettes	node-red-contrib-firebase-admin, node-red-node-pi-gpio
RAM Requirement	Recommended 512MB+

3.2.2.3 Cloud Backend: Firebase (Firestore & RTDB)

Replacing the local SQLite3 database from FYP1, Google Firebase serves as the full-stack cloud backend. Cloud Firestore (a NoSQL document database) permanently stores user records, visitor invitations, and gate logs. The Realtime Database (RTDB) acts as the "hot storage," creating low-latency listeners that allow the mobile app to trigger the physical hardware instantly.

Table 3.7 Firebase Specification & Requirement

Specification	Requirement
Type	Cloud-Hosted NoSQL & Realtime JSON Database
Services Used	Cloud Firestore, Realtime Database, Firebase Auth
Hosting	Google Cloud Platform
Data Format	JSON / Document-based Collections

3.2.2.4 Mobile Application Framework: Flutter

Flutter is utilized to build the front-end user interfaces for Students, Staff, and Administrators. It replaces the basic Node-RED web dashboard as the primary point of user interaction. It handles the creation of visitor invitations, renders the QR codes, displays real-time logs, and provides the UI for remote camera control.

Table 3.8 Flutter Specification & Requirement

Specification	Requirement
Programming Language	Dart (v3.0+)
Framework Version	Flutter SDK v3.0+
Target Platforms	Android (APK/AAB) and iOS
Key Packages	cloud_firestore, firebase_database, qr_flutter

3.2.2.5 Programming Language & Edge Libraries

Python 3 remains the primary language for interacting directly with the physical hardware components at the edge. Node-RED uses exec nodes to trigger these Python scripts. The scripts simply read the hardware inputs (RFID UID or QR text) and print them as string outputs, leaving all complex routing to Node-RED.

Table 3.9 Programming Languages' Name & Purpose

Component	Name/ Version	Purpose
Programming Language	Python 3	Runs the underlying hardware interface scripts triggered by Node-RED.
RFID Library	mfrc522 (SimpleMFRC522)	Provides functions to read the UID from MIFARE cards via SPI.
Camera Library	OpenCV (cv2) / PyZbar	Captures video frames from the USB webcam and decodes the embedded text from physical QR codes.

3.3 Functional and Non-Functional Requirements

3.3.1 Functional Requirements

Functional requirements define the specific behaviors and tasks the system must perform to meet its objectives. The functional requirements for this system are categorized by the user roles and the IoT edge logic.

Table 3.10 Functional Requirements

Category / Role	Functional Requirements	Description
IoT Edge Hardware (Node-RED)	FR1: RFID Authentication	The edge system must read an RFID tag, query the Firebase Firestore to verify the user's status, and trigger LED gate indicators.
	FR2: QR Code Scanning	The system must utilize the USB webcam to scan QR codes, extract the document ID, and verify the visitor invitation in Firestore.
	FR3: Anti-Passback Logic	The system must check the campus_status field in Firestore before granting entry. If a user is already "IN", subsequent entry scans must be denied.
Mobile App (Students & Staff)	FR4: Illegal Parking Report	Users must be able to capture location and photo evidence of parking violations. The data is uploaded, and a new document appears in the Firebase 'reports' collection.
Mobile App (Admin)	FR5: Remote Hardware Control	Admins must be able to tap a button in the app to write a "SCAN_START" command to the RTDB, which remotely turns on the physical Pi webcam.
Mobile App (Staff & Admin)	FR6: Visitor Invitation	Staff members must be able to generate visitor requests. Once approved by an Admin, the app must generate a scannable QR code containing the specific Firestore Document ID.

3.3.2 Non-Functional Requirements [23]

There are several non-functional requirements describing the performance standards and quality attributes that the uPark project must achieve to ensure that the system operates effectively and reliably. These requirements focus on how well the system performs rather than what the system does.

NFR001: Performance (Response Time)

Requirement: The system must respond to user interactions and hardware triggers rapidly to prevent vehicle queuing at the campus gates. Specifically, edge hardware authentication (RFID tapping or QR scanning) must process and trigger the physical LED indicators within 1.0 second. Furthermore, mobile application requests involving cloud synchronization—such as submitting a parking report or triggering the remote camera via the Realtime Database—should complete within 3.0 seconds under normal network conditions [23].

NFR002: Security

Requirement: The application must ensure secure handling of user data and campus access. This is achieved by implementing Firebase Authentication for secure login, enforcing Role-Based Access Control (RBAC) to restrict sensitive administrative features, and utilizing Cloud Firestore rules. Additionally, the system must enforce strict Cloud Anti-Passback logic to guarantee that a single physical credential cannot be exploited by multiple vehicles simultaneously.

NFR003: Reliability

Requirement: The IoT edge system and cloud backend must operate with high accuracy and consistency. The hardware sensors (MFRC522 and Logitech Webcam) must capture input data without misreads or crashes, and the system must correctly evaluate access permissions 100% of the time during functional operations. The system must also safely handle exceptions, such as blocking invalid email domains during the sign-up process.

NFR004: Usability

Requirement: The mobile application must feature a clean, intuitive, and user-friendly interface that requires minimal training for students, staff, and administrators. On the hardware

side, the system must provide immediate and clear physical feedback to users at the gate, specifically utilizing a Green LED for "Access Granted" and a Red LED for "Access Denied."

NFR005: Compatibility

Requirement: The uPark mobile application must be cross-platform, capable of running smoothly on both Android and iOS operating systems using a single Flutter codebase. The IoT edge logic (Node-RED and Python scripts) must be highly compatible with standard Debian-based Linux environments, specifically Raspberry Pi OS.

NFR006: Availability

Requirement: The system infrastructure must be highly available to support continuous campus operations. By utilizing Google Firebase as the cloud backend, the database guarantees high uptime and accessibility from any location. The edge controller (Raspberry Pi 4) must be capable of running continuously in a standby state, constantly listening to the Realtime Database for instant remote wake-up commands.

NFR007: Scalability

Requirement: The system architecture must be designed to accommodate future growth in the university population. The use of a NoSQL Cloud Firestore database ensures that the system can scale seamlessly to handle an increasing volume of registered users, visitor invitations, and permanent gate logs without degrading search speeds or requiring complex database restructuring.

3.4 Project Milestone

3.4.1 Project I (FYP1) Timeline

During FYP1, the focus was entirely on hardware acquisition, basic edge logic, and creating a standalone proof-of-concept. The milestones included setting up the Raspberry Pi OS, wiring the MFRC522 reader, and writing the Python scripts to capture UIDs. The backend logic was constrained to a local SQLite database and basic Node-RED flows to trigger LED indicators based on local authentication.

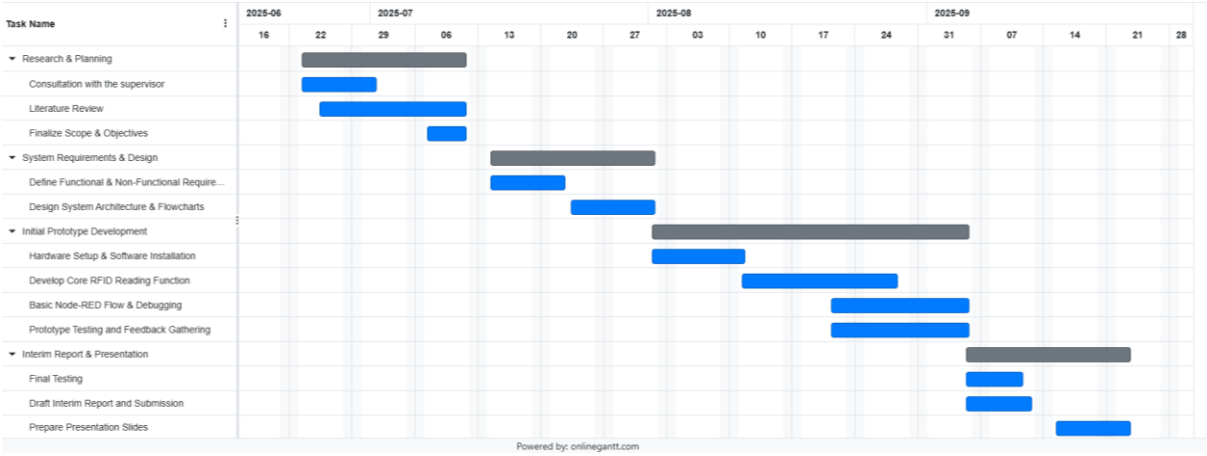


Figure 3.4.1 Gantt Chart for FYP1

3.4.2 Project II (FYP2) Timeline

FYP2 shifted focus to full-stack development, cloud integration, and mobile application deployment. Key milestones included migrating data from SQLite to Firebase Cloud Firestore. Following this, the Flutter mobile application was designed and developed with Role-Based Access Control (Admin, Staff, Student). Later milestones focused on implementing advanced IoT features, such as the USB webcam QR scanner, writing complex Node-RED payloads to update Firebase records for the anti-passback feature, and enabling remote camera control via the Realtime Database. Final weeks are dedicated to end-to-end system testing and report writing.

Figure 3.4.2 Gantt Chart for FYP2

3.5 Estimated Cost

Table 3.11 Estimated Project Cost

No.	Component / Item	Category	Estimated Cost (RM)
1	Raspberry Pi 4 Model B	Hardware	260.00 - 300.00
2	MFRC522 RFID Kit	Hardware	15.00
3	Basic USB Webcam	Hardware	50.00
4	Minor Electronic Components (LEDs, Jumper Wires)	Hardware	10.00

5	Flutter SDK, Node-RED, Python, Raspberry Pi OS	Software	0.00 (Open-Source)
6	Google Firebase ("Spark" Plan)	Cloud Services	0.00 (Free-Tier)
Total Estimated Cost			<RM390

The development of this project prioritizes cost-effectiveness by leveraging existing hardware and free, open-source software platforms.

As shown in Table 3.5.1, the primary hardware costs consist of the Raspberry Pi 4, the MFRC522 RFID kit, a basic USB webcam, and minor electronic components. The total estimated hardware cost is kept under RM 390.

There are no software costs associated with this project. The Flutter SDK, Node-RED, Python, and Raspberry Pi OS are completely free and open-source. Furthermore, the cloud infrastructure relies on Google Firebase's "Spark" plan, which provides generous free-tier quotas for both Cloud Firestore reads/writes and Realtime Database connections, easily covering the requirements of a prototype system. Therefore, the project incurs zero ongoing software or server subscription fees.

3.6 Concluding Remark

In summary, this chapter detailed the methodology and requirements necessary for developing the full-stack IoT parking system. The Prototyping model was justified as the most effective approach for integrating hardware and cloud software iteratively. The system requirements clearly defined the upgraded hardware, such as the Raspberry Pi 4 and USB webcam, alongside the modern software stack utilizing Flutter and Firebase. Finally, the functional requirements and milestones provided a clear roadmap of the system's advanced capabilities, including cloud anti-passback and remote hardware control, all achievable within a highly cost-effective budget.

Chapter 4

System Design

4.1 System Architecture

System architecture refers to the high-level structure and organization of a complex system, providing a blueprint for how various hardware and software components interact to achieve the project's objectives. It defines the system's design, including data flows, communication protocols, and processing responsibilities.

The system architecture of the "Full-Stack IoT University Entry and Parking System" is designed to provide a highly scalable, secure, and real-time platform for managing campus access. Moving beyond a simple localized setup, this architecture adopts a distributed model spanning three primary layers: the Client Layer, the Cloud Backend Layer, and the Edge IoT Layer. The system architecture is shown in Figure 4.1.

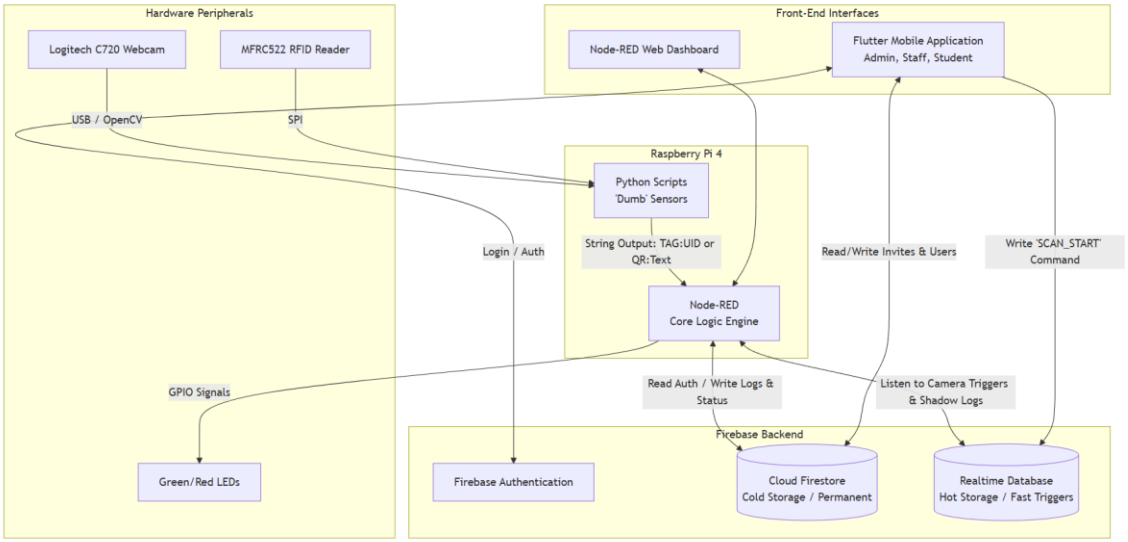


Figure 4.1.1 Project's System Architecture

Flutter Mobile Application (Application Layer):

This component serves as the primary user interface for the system, developed using the cross-platform Flutter framework. It provides tailored experiences based on user roles: Administrators, Staff, and Students. The application directly communicates with the cloud backend to perform tasks such as generating QR code visitor invitations, viewing access logs,

and managing user profiles. Crucially, it includes a remote hardware control feature, allowing administrators to push commands to the cloud to wake up or shut down the physical camera at the campus gate.

Firebase Cloud Backend (Data Layer):

Firebase serves as the centralized synchronization hub, eliminating the need for a traditional local database. The backend is strategically divided into two distinct database services to optimize performance:

Cloud Firestore: This NoSQL document database acts as the "Cold Storage" or permanent record. It securely stores rfid_users, visitor invitations, and permanent gate_logs. It is the primary source of truth for the anti-passback mechanism, tracking the campus_status (IN/OUT) of every user.

Realtime Database (RTDB): This JSON-based database acts as the "Hot Storage" for ultra-low latency operations. It is used to listen for remote camera triggers (e.g., gate_control/gate_IN/command) sent from the mobile app. It also holds a "Shadow Log" of recent entries to bypass Firestore read quotas and feed live data efficiently to the Node-RED web dashboard.

Firebase Authentication: Handles secure user sign-ups and logins for the Flutter mobile application, ensuring that only authorized personnel can access sensitive gate controls and logs.

Node-RED (Edge Logic Layer):

Hosted on the Raspberry Pi 4, Node-RED acts as the central orchestrator or "brain" of the physical checkpoint. It bridges the hardware with the cloud. Node-RED uses the rtdb-on node to listen for instant camera triggers. For permanent database updates, it utilizes the node-red-contrib-firebase-admin package. Because of the strict formatting requirements of this package, Node-RED is programmed to structure all Firestore write payloads as a "Swiss Army Knife" object (defining the exact collection, document, path, and data payload) to ensure seamless cloud synchronization.

Python Scripts & Hardware (Physical Layer):

The physical layer consists of the MFRC522 RFID reader, the Logitech C720 Webcam, a servo motor, and LED gate indicators. To maintain a clean architecture, the Python scripts interfacing with this hardware act strictly as "dumb sensors" and actuators. When executed via Node-RED

exec nodes, these scripts solely focus on reading the hardware input and outputting raw strings (e.g., TAG:12345 or decoded QR text). Furthermore, the servo motor actuation script uses Pulse Width Modulation (PWM) to open the physical gate and is programmed to automatically close it after a 5-second delay, as the prototype does not utilize a physical vehicle presence sensor. All subsequent logic, cloud validation, and decision-making are securely handled upstream by Node-RED.

In conclusion, the system architecture of the IoT University Entry and Parking System lays a robust foundation for its functionality, real-time synchronization, and security. By strictly separating the edge hardware sensing from the Node-RED logic orchestration and utilizing a hybrid Firebase cloud strategy, this architecture ensures a seamless and highly responsive user experience. Its modular design allows administrators to control physical gates remotely and enforce strict anti-passback rules securely from the cloud.

4.2 Use Case Diagram

A use case diagram description provides an overview of the interactions and functionality of a system or application from the perspective of its users. It uses visual elements like actors (users or external systems) and use cases to represent various scenarios in which users interact with the system to achieve a specific goal.

The use case diagrams below detail the system's users and their interactions with both the Flutter mobile application and the physical IoT gate hardware. In this project, there are three primary human actors (Admin, Staff, Student) and one system actor (IoT Edge Node).

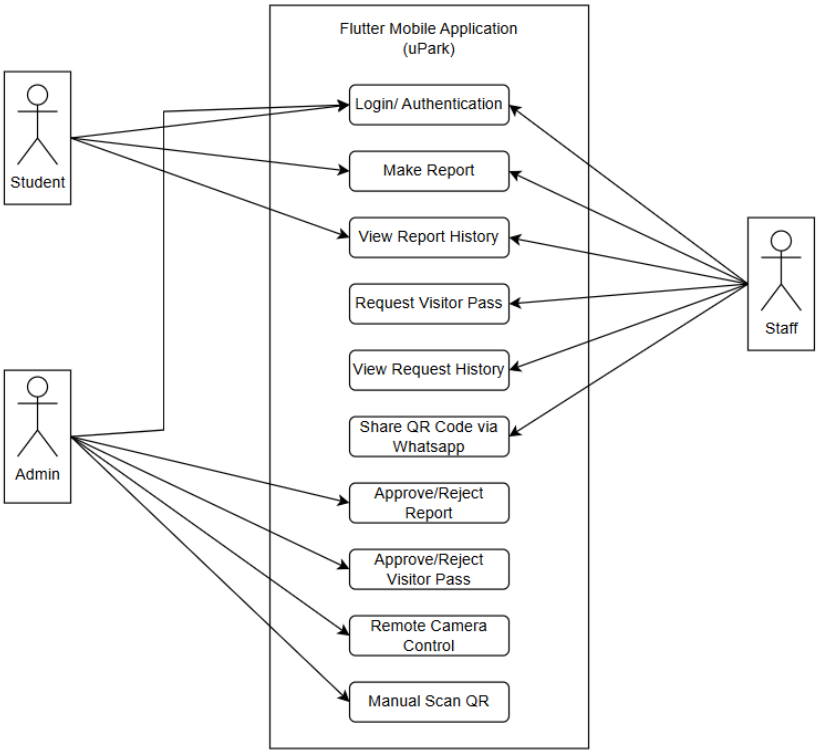


Figure 4.2.1 Flutter Mobile Application Use Case

Mobile Application Interactions Description:

Login / Authentication: Provides a unified entry point where users (Student, Staff, or Admin) can authenticate their credentials. Upon successful login, the system automatically determines the user's role and redirects them to their respective, personalized dashboard.

Make Report: Enables Students and Staff to file reports regarding illegal parking incidents on campus. This feature includes GPS-based location tagging and photographic evidence to ensure accuracy.

View Report History: Allows users to track the status of their submitted parking reports. The system provides real-time updates, indicating whether a report has been successfully submitted, approved, or rejected by the administration.

Request Visitor Pass: Empowers Staff members to submit formal entry requests for campus visitors, such as guest speakers or VIPs. This module collects visitor details to facilitate campus security protocols.

View Request History: Provides Staff with a comprehensive overview of their submitted visitor requests, allowing them to monitor the processing status of each invitation in real-time.

Share QR Code: Once an invitation is approved, this feature allows Staff to generate a unique, time-sensitive QR code and share it via instant messaging platforms (e.g., WhatsApp) to provide visitors with authorized campus entry.

Approve / Reject Report: An administrative function that allows the Admin to review parking reports. The Admin can evaluate the evidence provided and either validate or dismiss the report, including an optional justification for rejection.

Approve / Reject Request: Grants the Admin the authority to oversee visitor requests. Only approved invitations result in the generation of a valid QR entry pass, ensuring restricted campus access.

Remote Camera Control: Offers the Admin the capability to remotely trigger the IoT camera system at the gate. This allows for manual control over entry point surveillance, enabling on-demand scanning for specialized security needs.

Manual Scan QR: Acts as a critical fallback mechanism. If the automated gate hardware is temporarily offline, the Admin can use their mobile device to scan a visitor's QR pass directly, manually verifying their entry authorization.

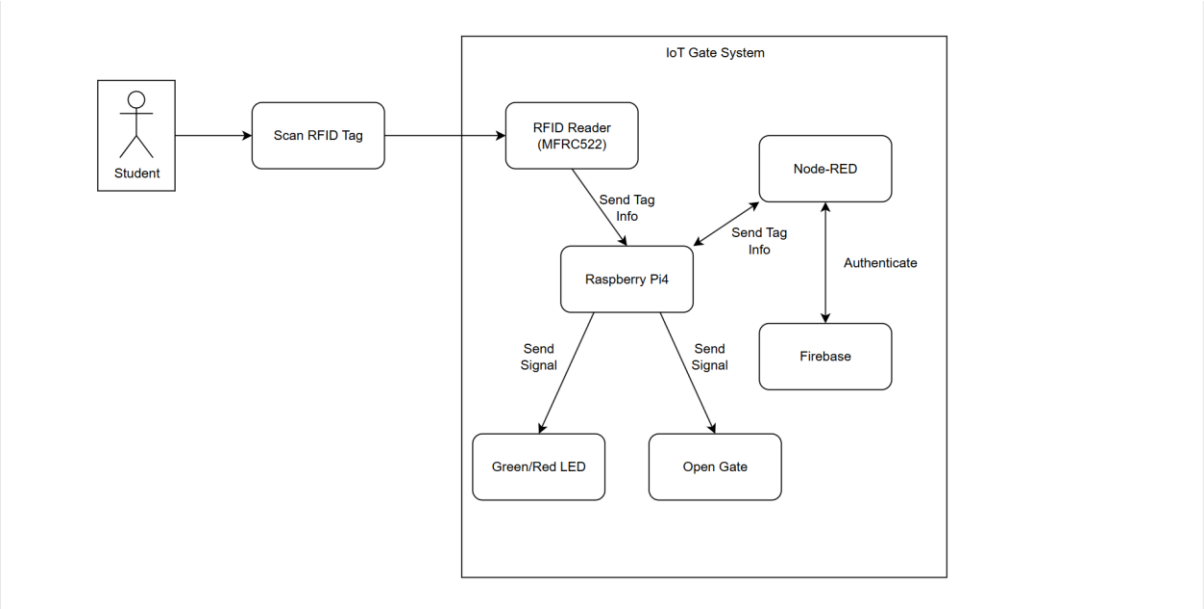


Figure 4.2.2 Iot Gate Use Case (Student)

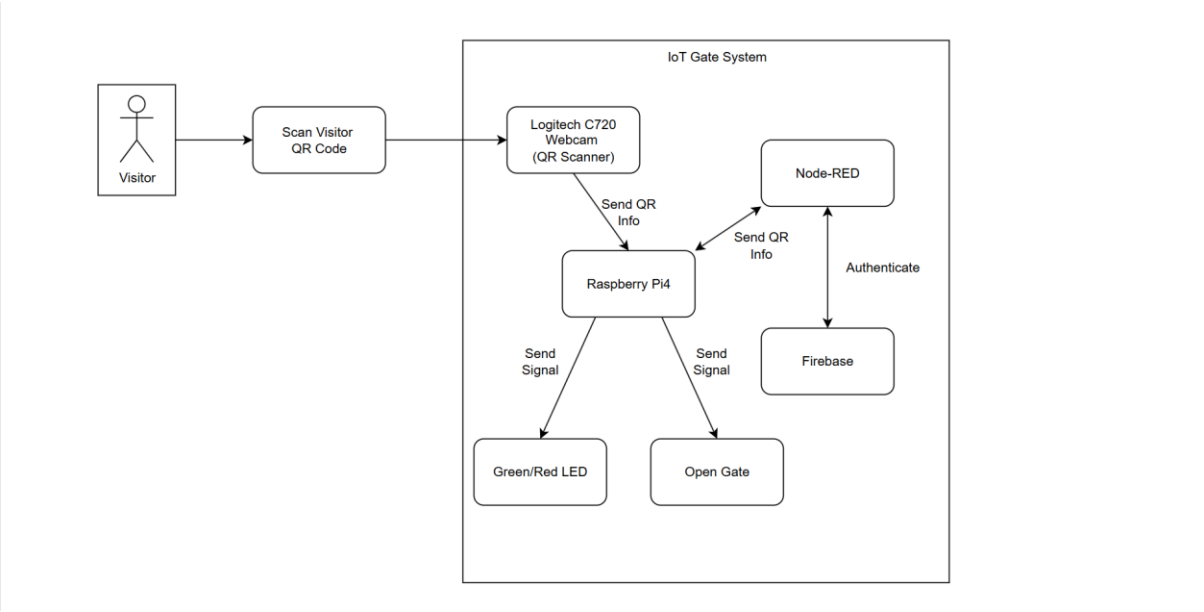


Figure 4.2.3 IoT Entry Gate Use Case (Visitor)

IoT Gate Interactions Description:

Student Access Flow (RFID)

Scan RFID Tag: The student presents their university-issued RFID tag at the physical campus gate.

RFID Reader (MFRC522): The reader scans the tag and transmits the raw identification number to the Raspberry Pi 4.

Raspberry Pi 4: Acting as an edge device, the Pi 4 receives the raw data and forwards it directly to the Node-RED software.

Node-RED & Firebase Authentication: Node-RED sends the tag information to the Firebase Cloud database. Firebase verifies the user's identity and checks the Cloud Anti-Passback system to ensure the student is not already inside the campus.

Gate Actuation: Once Firebase returns a successful verification, Node-RED sends a command back to the Raspberry Pi 4. The Pi 4 then triggers the GPIO pins to turn on the Green LED and operate the servo motor to open the gate. If unsuccessful, the gate won't open and the Red LED will flash.

Visitor Access Flow (QR Code)

Scan QR Code: The visitor presents the QR code—previously shared by a staff member—to the scanner at the gate.

Webcam (Logitech C720): The webcam captures the QR code, and a background script extracts the embedded invitation ID, sending it to the Raspberry Pi 4.

Raspberry Pi 4: The Pi 4 forwards this extracted data to Node-RED.

Node-RED & Firebase Authentication: Node-RED queries the Firebase database to check if the specific invitation ID is marked as "Approved".

Gate Actuation: Upon successful validation from the cloud, Node-RED instructs the Raspberry Pi 4 to flash the Green LED and open the gate for the visitor. If unsuccessful, the gate won't open and the Red LED will flash.

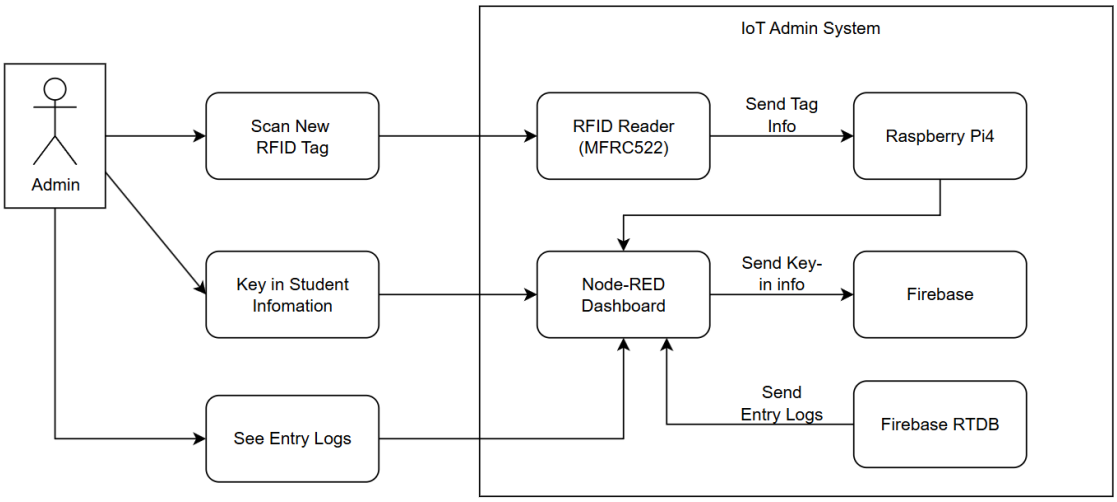


Figure 4.2.4 IoT Admin System Use Case

System Administrator Flow

Dashboard Management: The Administrator primarily interacts with the local Node-RED dashboard rather than the physical gate.

User Registration: To register a new user, the Admin scans a new RFID tag at the gate. The system temporarily saves this tag ID to its memory, allowing the Admin to easily type in the student's details on the dashboard without manually typing the long RFID number. Node-RED then saves this complete profile to Firebase.

System Monitoring: The Admin uses the dashboard to monitor real-time entry and exit logs (fetched from the Firebase Realtime Database) to maintain campus security and oversee traffic flow.

4.3 Activity Diagram

An activity diagram is a visual representation of the flow of activities or actions within a system or process. It is a type of UML (Unified Modeling Language) diagram commonly used in software engineering to describe the dynamic aspects of a system. These diagrams model the workflows, conditional logic, and sequence of events required to complete a specific functional requirement.

4.3.1 Activity Diagram: Cloud Anti-Passback Authentication (FR1 & FR3)

This activity diagram outlines the core automated process by which the IoT edge system reads an RFID tag, validates it against the cloud database, and enforces the anti-passback rule before granting physical access.

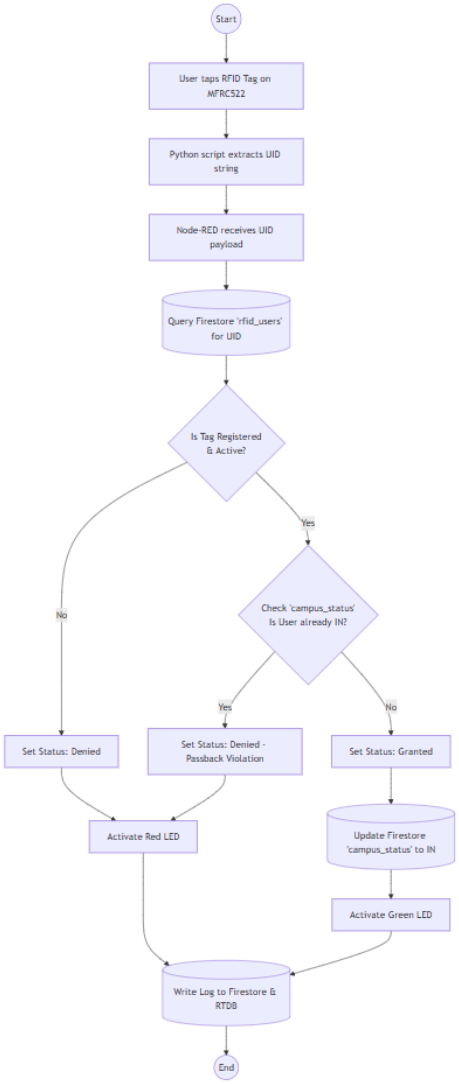


Figure 4.3.1 RFID Anti-Passback Flow

4.3.2 Activity Diagram: QR Visitor Validation (FR2)

This diagram illustrates the process of validating a visitor. It details how the Python script decodes the QR image to extract the Document ID, which is then verified in Firestore by Node-RED.

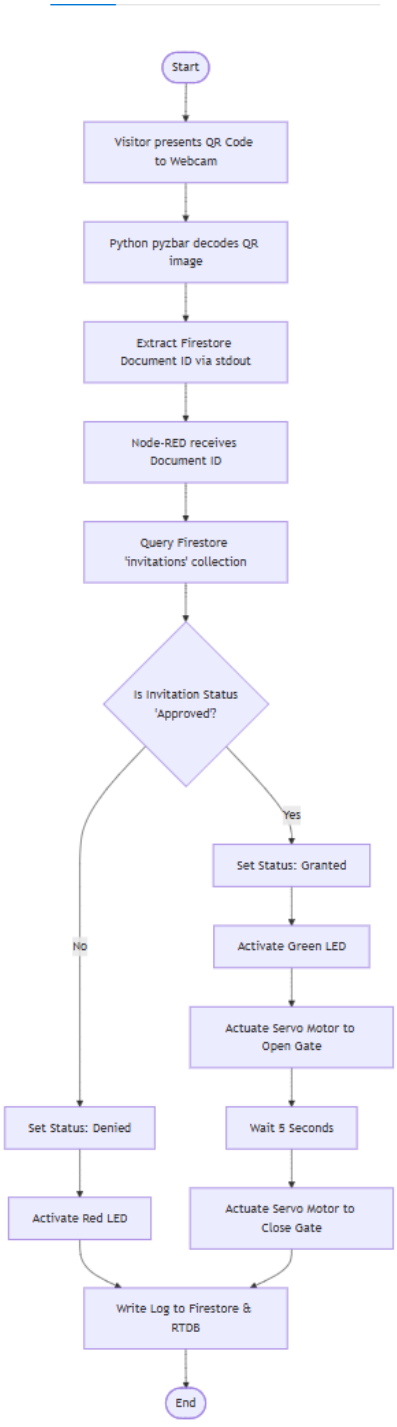


Figure 4.3.2 QR Visitor Validation

4.3.3 Activity Diagram: Illegal Parking Report (FR4)

This activity diagram details the workflow for students and staff to submit an illegal parking report, covering the GPS reverse geocoding process, the image capture and compression, and the final submission to the Firestore database.

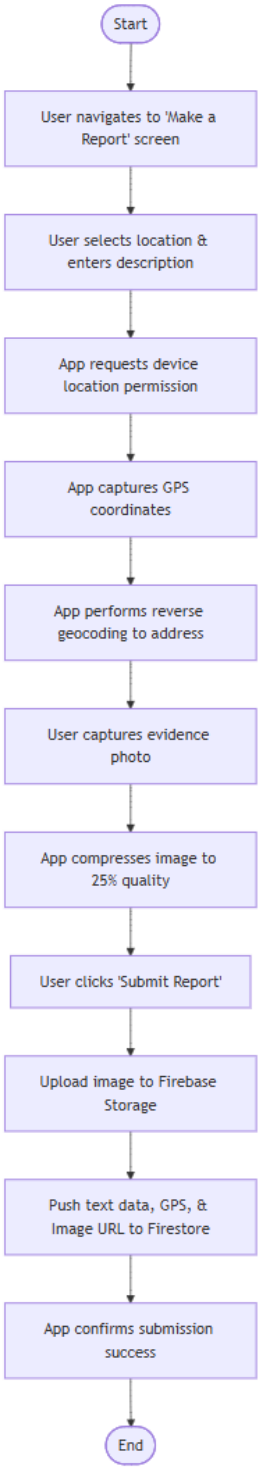


Figure 4.3.3 Illegal Parking Report

4.3.4 Activity Diagram: Remote Camera Control (FR5)

This diagram maps the cloud-to-edge communication workflow, showing how an Administrator uses the Flutter application to remotely wake up and shut down the physical USB webcam via the Realtime Database (RTDB).

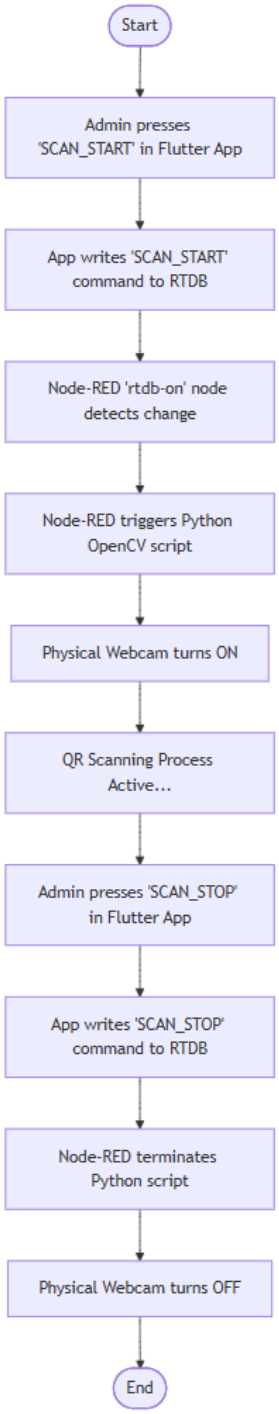


Figure 4.3.4 Remote Camera Control

4.3.5 Activity Diagram: Visitor Invitation Workflow (FR6)

This diagram outlines the Role-Based Access Control (RBAC) interaction between Staff and Administrators to successfully generate a secure QR code for a campus visitor.

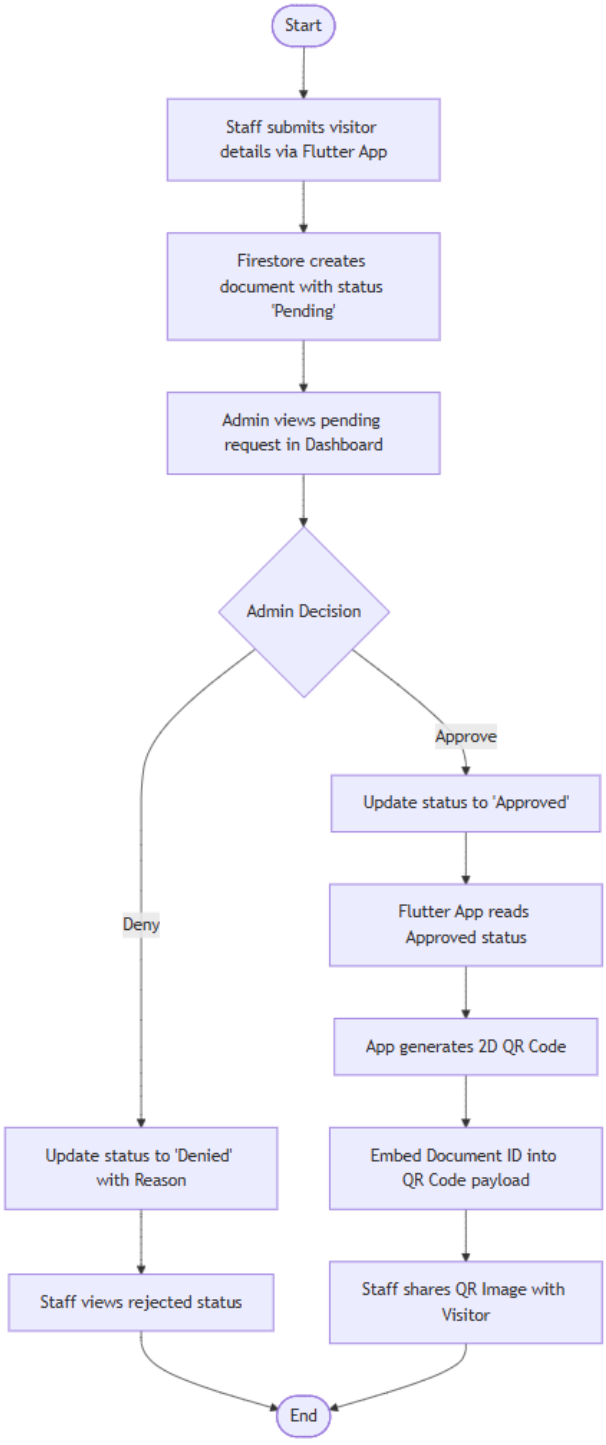


Figure 4.3.5 Visitor Invitation Workflow

4.4 Database Design

Database design is a crucial aspect of the "Full-Stack IoT University Entry and Parking System" project, as it defines the structure and organization of the cloud backend that stores critical user credentials, visitor invitations, hardware commands, and system logs. Because this system requires both permanent record-keeping and ultra-low latency hardware triggers, a hybrid NoSQL approach is utilized using Google Firebase.

The architecture is divided into Cloud Firestore (for permanent, structured document storage) and the Realtime Database (RTDB) (for fast, JSON-based hardware commands). Proper database design here ensures data integrity, rapid edge-to-cloud retrieval, and seamless integration between the Flutter mobile application and the Node-RED IoT edge node. Below is the detailed structure of the database design.

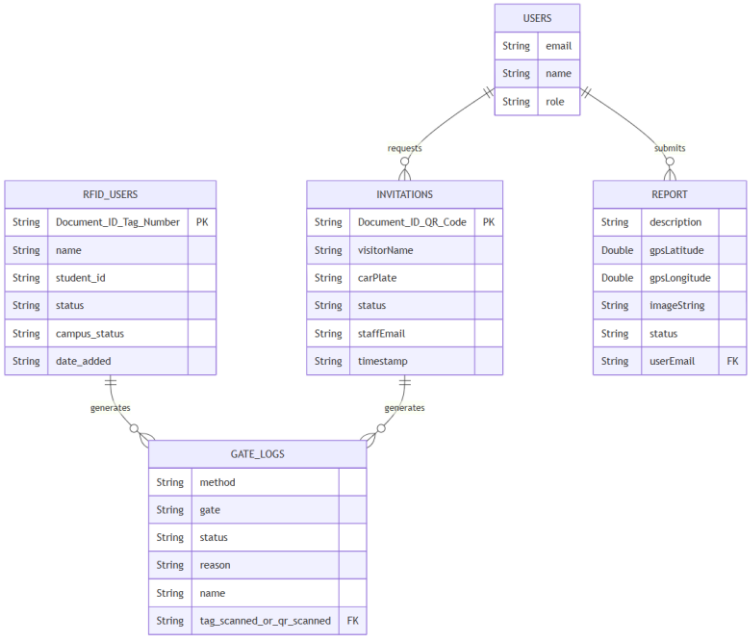


Figure 4.4.1 Databases

4.4.1 Cloud Firestore Collections (Permanent Storage)

4.4.1.1 rfid_users Collection

This collection stores the profiles of registered students and staff who possess physical MIFARE 1K RFID cards. The Document ID is explicitly set as the physical RFID tag number to enable instant O(1) read operations from the hardware.

Table 4.1 rfid_users Collection

Field Name	Data Type	Description
Document ID	String	The physical RFID Tag Number (e.g., 1075872668062). Acts as Primary Key
name	String	Full name of the registered user
student_id	String	The official university ID number
status	String	Account status ("Active" or "Banned").
campus_status	String	Current physical location ("IN" or "OUT"). Used for Cloud Anti-Passback.
date_added	Timestamp	The date and time the user was registered in the system.

4.4.1.2 invitations Collection

This collection manages visitor passes requested via the Flutter app. The auto-generated Document ID is directly embedded into the QR code.

Table 4.2 invitations Collection

Field Name	Data Type	Description
Document ID	String	Auto-generated ID. Encoded into the physical QR Code.
visitorName	String	Full name of the expected guest.
carPlate	String	Vehicle license plate number of the visitor.
status	String	Current state of the invite ("pending", "Approved", or "Denied").
purpose	String	Stated reason for visiting the campus.
location	String	Specific building or area the visitor is heading to.
date / time	String	The scheduled date and time of the visit.
staffEmail	String	The email of the staff member who requested the pass.
timestamp	Timestamp	The exact time the invitation was created.

4.4.1.3 gate_logs Collection

This collection acts as the permanent, immutable audit trail for every gate interaction.

Table 4.3 gate_logs Collection

Field Name	Data Type	Description
Document ID	String	Auto-generated by Node-RED upon entry/exit.
method	String	The authentication method used ("RFID" or "QR").
gate	String	Direction of travel ("IN" or "OUT").
status	String	Gate physical response ("OPEN" or "LOCKED").
reason	String	Explanation if locked (e.g., "Anti-Passback: Already Inside", "Unknown Tag").
name	String	The name retrieved from the user or invitation document.
tag_scanned / qr_scanned	String	The raw ID string presented to the scanner.
timestamp	String	ISO formatted timestamp of the event.

4.4.1.4 users Collection

This collection is utilized natively by the Flutter application for role-based access control (RBAC) upon Firebase Authentication login.

Table 4.4 users Collection

Field Name	Data Type	Description
Document ID	String	Auto-generated Firebase Authentication UID.
email	String	The registered UTAR email address.
name	String	The display name of the app user.
role	String	RBAC identifier ('student', 'staff', or 'admin').

4.4.1.5 report Collection

This collection handles the community-driven illegal parking reporting feature submitted via the Flutter application.

Table 4.5 report Collection

Field Name	Data Type	Description
Document ID	String	Auto-generated by Firestore upon submission.
description	String	User-provided text detailing the parking violation.

gpsLatitude	Double	The exact GPS latitude coordinate captured by the mobile device.
gpsLongitude	Double	The exact GPS longitude coordinate captured by the mobile device.
imageString	String	Base64 encoded string of the photographic evidence.
location	String	A textual description of the campus location.
status	String	Admin review status ("approved" or "denied").
userEmail	String	Email of the user who submitted the report for accountability.
timestamp	String	ISO formatted timestamp of the report submission.

4.4.2 Realtime Database (RTDB) Structure (Hot Storage)

The RTDB is utilized strictly for operations requiring sub-second latency, bypassing the heavier overhead of Firestore document reads.

Table 4.6 Realtime Database

Node Path	**Data Type**	**Description**
gate_control/gate_in/command	String	Written by the Admin via the Flutter app. Value is either "SCAN_START" (wakes up Raspberry Pi webcam) or "SCAN_STOP".
gate_logs	JSON Object	A lightweight "shadow log" pushed by Node-RED. Used exclusively to populate the live Node-RED web dashboard without exhausting Firestore read quotas.

4.4.3 Database Relationships and Data Binding

Because Firebase operates on a NoSQL document model, it does not use traditional SQL "Foreign Keys." Instead, this system utilizes Implicit Foreign Keys to seamlessly connect the physical edge hardware to the cloud architecture:

- **Hardware-to-Cloud (RFID Binding):** The physical integer read by the MFRC522 Python script is used directly as the Document ID to query the rfid_users collection. This allows Node-RED to perform a direct lookup without needing to search through fields.

- **Hardware-to-Cloud (QR Binding):** The text payload decoded by the OpenCV Python script from the visitor's phone is the exact auto-generated Document ID of their record in the invitations collection.
- **Audit Traceability:** Every document in the gate_logs collection securely records the tag_scanned or qr_scanned. This allows an Administrator investigating a suspicious log to copy the ID, search the respective collections, and precisely identify the user or visitor involved.

4.5 Circuit Design

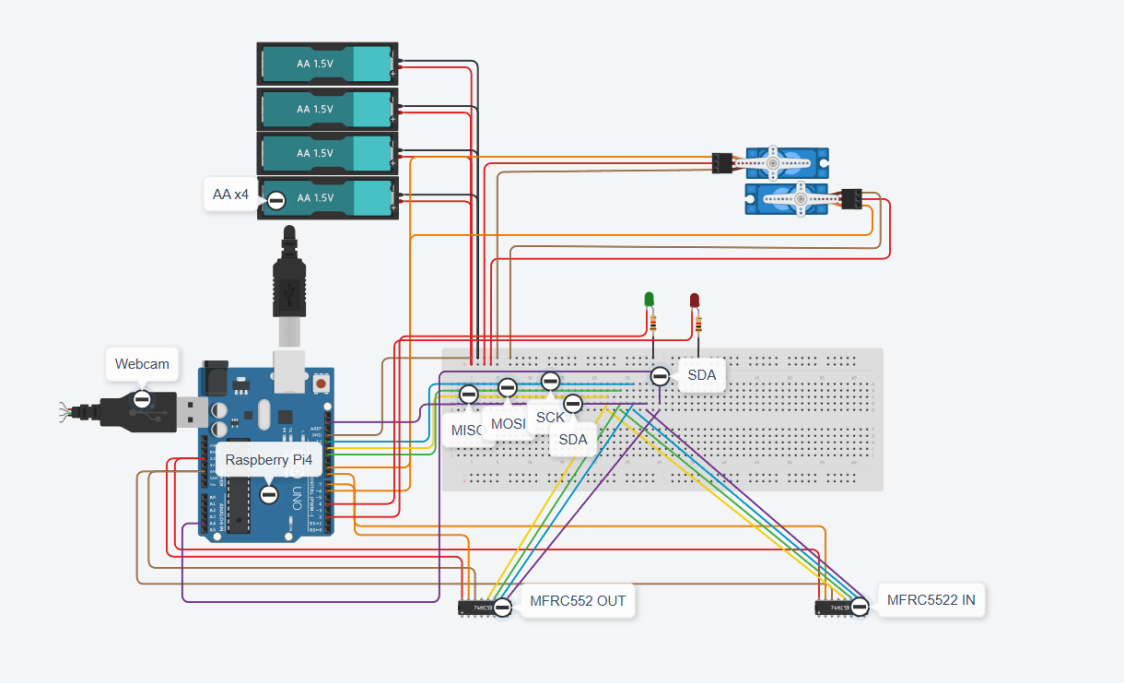


Figure 4.5.1 System Circuit Diagram

The circuit design of the edge hardware details the physical wiring and communication interfaces between the Raspberry Pi 4 and its peripheral components (Note: Tinkercad components are used for representation due to software library limitations; the actual implementation utilizes a Raspberry Pi 4). A breadboard is utilized to safely route power and distribute signals among the dual RFID readers, LED indicators, and servo motors. The complete hardware assembly is designed to ensure stable power delivery and accurate data transmission.

To manage entry and exit authentication simultaneously, two MFRC522 RFID readers are integrated into the system using the Serial Peripheral Interface (SPI) protocol. The core SPI communication pins—MISO (GPIO 9), MOSI (GPIO 10), and SCK (GPIO 11)—are shared

between both readers via the breadboard. To differentiate between the "Entry" and "Exit" scanners, unique Chip Select (SDA) pins are assigned: GPIO 8 is connected to the Entry reader, while GPIO 7 is connected to the Exit reader. Unique Reset (RST) pins are also assigned using GPIO 25 and GPIO 24, respectively. Both readers are powered directly by the Raspberry Pi's 3.3V output pins.

Visual feedback is provided through Green and Red LEDs. The Green LED is driven by GPIO 26, and the Red LED is driven by GPIO 17. Both LEDs are connected in series with 220-ohm resistors routing to the common ground to prevent excessive current draw and protect the GPIO pins.

For the physical gate actuation, two SG90 servo motors are deployed. The Pulse Width Modulation (PWM) signal for the first motor is controlled via GPIO 13, while the second motor utilizes GPIO 18. Because servo motors draw significant current during actuation—which can cause the Raspberry Pi to crash or reboot due to voltage drops—an external power supply (a 4x AA battery pack providing approximately 6V) is implemented. The positive terminal of the battery pack is routed directly to the VCC of both servo motors, while the negative terminal is tied to the common ground of the Raspberry Pi to complete the circuit and synchronize the PWM signals. Finally, the Logitech webcam operates independently of the GPIO pins and is connected directly via one of the Raspberry Pi's native USB ports.

4.6 Concluding Remark

In summary, this chapter has provided a comprehensive overview of the design and architectural framework for the "Full-Stack IoT University Entry and Parking System." By adopting a distributed architecture that bridges edge hardware with Google Firebase's cloud-native services, the project ensures a scalable and highly responsive system capable of handling real-time authentication tasks. The use case and activity diagrams illustrated the dynamic interactions between the mobile application, the physical entry gate hardware, and the centralized cloud backend. Furthermore, the detailed database design successfully established a robust data structure, utilizing a hybrid storage strategy—Cloud Firestore for permanent audit trails and Realtime Database for instant hardware orchestration—to meet the project's stringent security and performance goals. This design foundation serves as the blueprint for the practical implementation and development of the working prototype, ensuring that all functional requirements are supported by a secure and efficient infrastructure.

Chapter 5

System Implementation

5.1 Software and Hardware Setup and Configuration

In this chapter, the procedures and steps required to prepare the development environment, configure the hardware, and integrate essential software elements are detailed. This phase ensures that the edge hardware (Raspberry Pi 4), the cloud backend (Firebase), and the mobile application (Flutter) are properly initialized and connected to guarantee the smooth operation of the system. The coding explanations and specific logic flows will be discussed in the subsequent section.

5.1.1 Environment Preparation: Edge Hardware and Development Tools

This section outlines the setup of the primary development tools, Integrated Development Environments (IDEs), and the hardware operating system.

Step 1: Raspberry Pi 4 OS and Core Libraries Setup

The foundation of the edge IoT system is the Raspberry Pi 4. The 64-bit Raspberry Pi OS was flashed onto a 32GB microSD card using the Raspberry Pi Imager. Once booted, the terminal was used to update the system and install the required Python libraries for the hardware peripherals. Specifically, mfrc522 and RPi.GPIO were installed for the RFID reader, while opencv-python and pyzbar were installed to enable the Logitech C720 webcam to capture video frames and decode QR codes.

- sudo apt update -y && sudo apt upgrade -y
- pip3 install mfrc522 RPi.GPIO opencv-python pyzbar

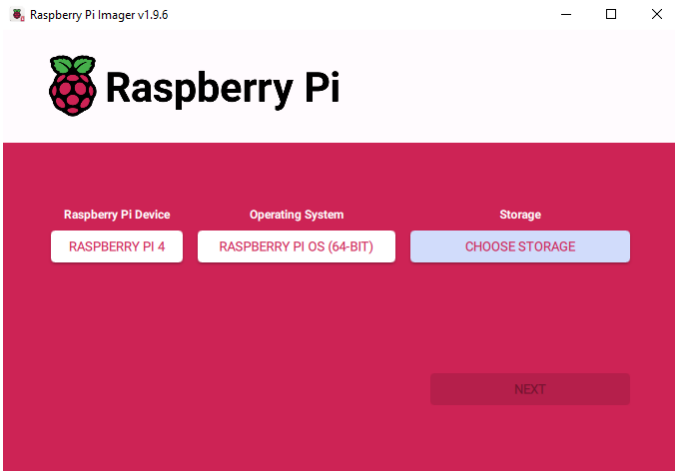


Figure 5.1.1 Raspberry Pi Imager

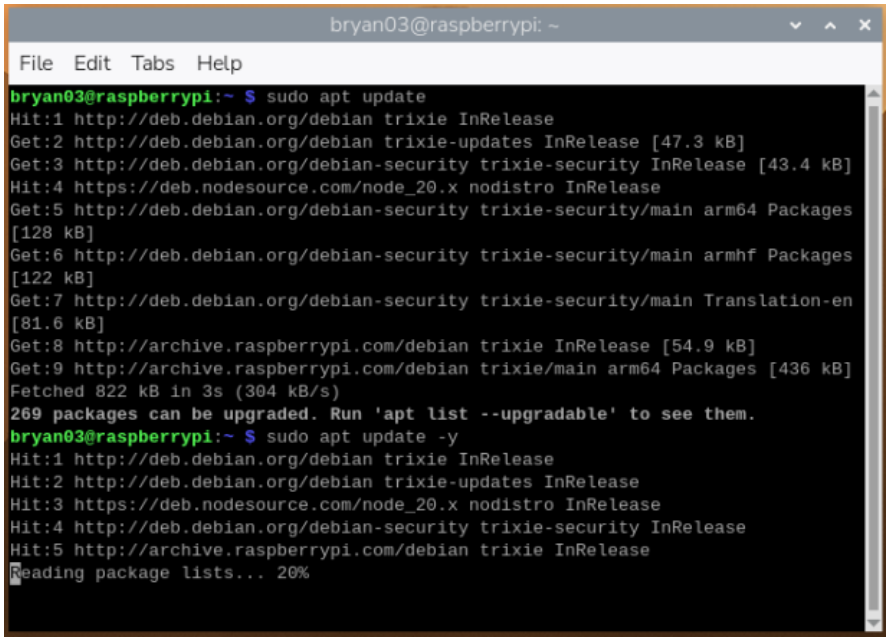


Figure 5.1.2 Use *sudo apt update -y* to update the system

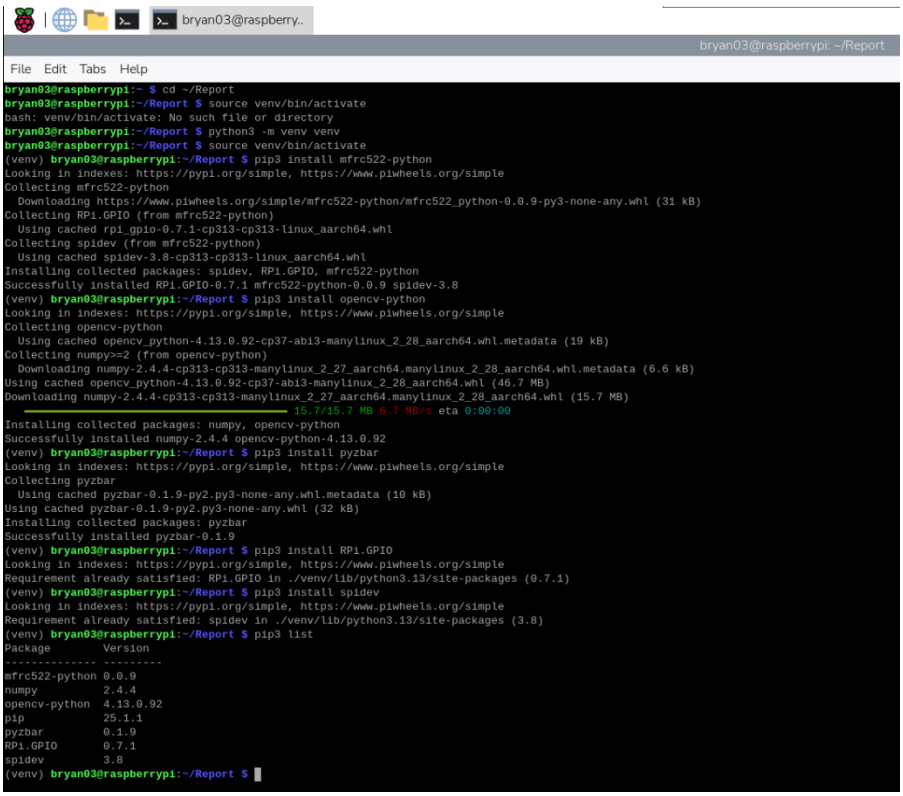


Figure 5.1.3 Install and View Python Libraries

Step 2: Node-RED Installation on Raspberry Pi

Node-RED acts as the core logic engine. It is installed using the official installation script, as shown in Figure 5.4, which also automatically configures the compatible Node.js runtime. Figure 5.4 shows the terminal when Node-RED is successfully executed.

- bash <(curl -sL https://raw.githubusercontent.com/node-red/linux-installers/master/deb/update-nodejs-and-nodered)

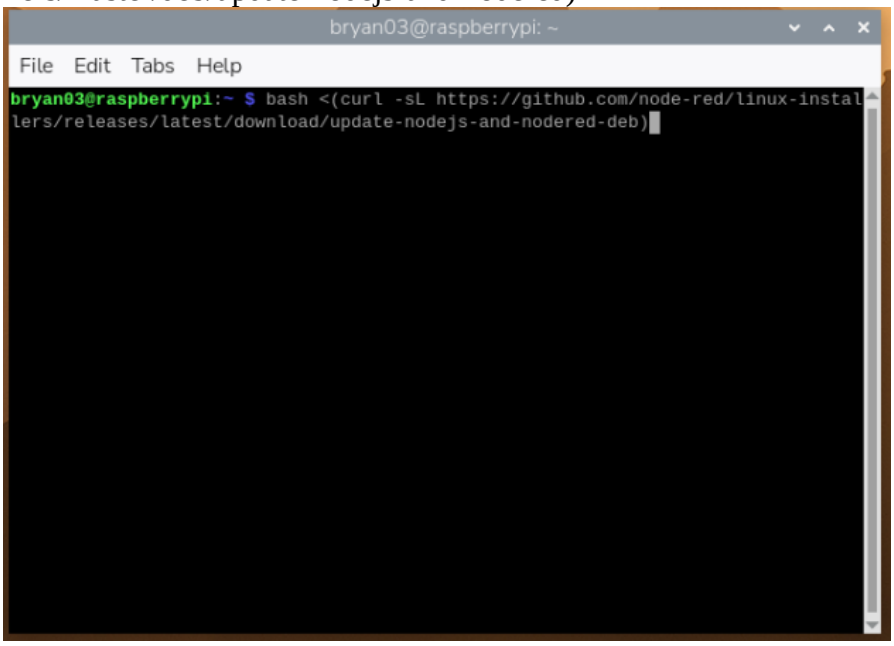


Figure 5.1.4 Install Node-RED

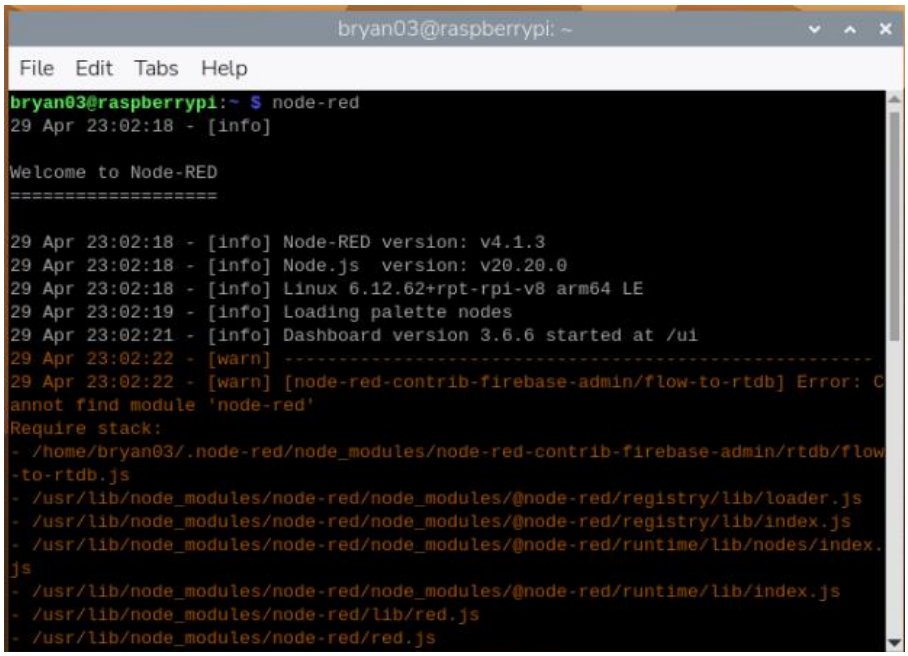


Figure 5.1.5 Use *node-red* to start Node-RED to determine is installed

Step 3: Flutter SDK and Mobile IDE Setup

To develop the cross-platform mobile application, the Flutter SDK (version 3.0+) was downloaded and added to the system's environment variables. Visual Studio Code (VS Code) and Android Studio were selected as the primary IDEs. The Android SDK and virtual emulators were configured within Android Studio to allow for real-time mobile application testing.

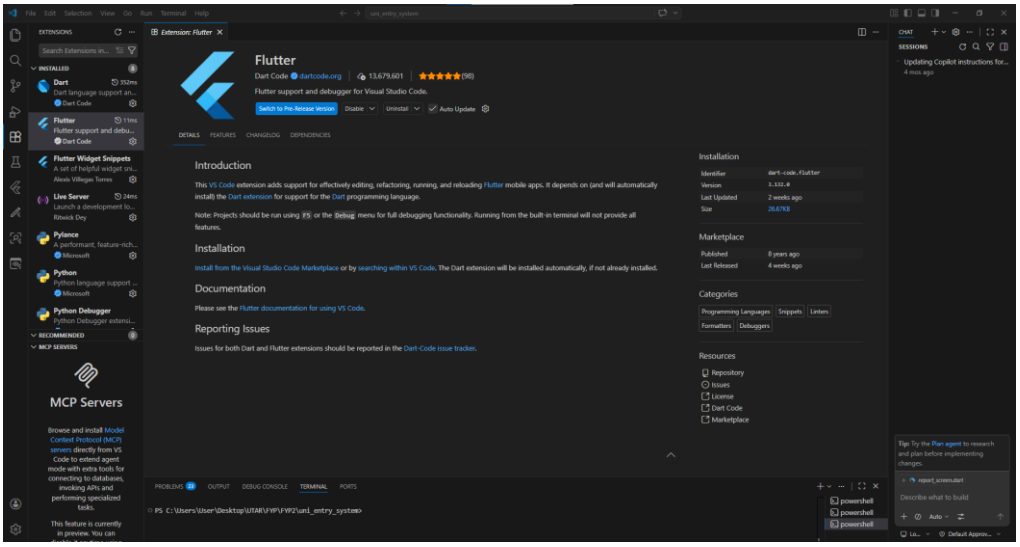


Figure 5.1.6 In Visual Studio Code Search "Flutter" in extension and install

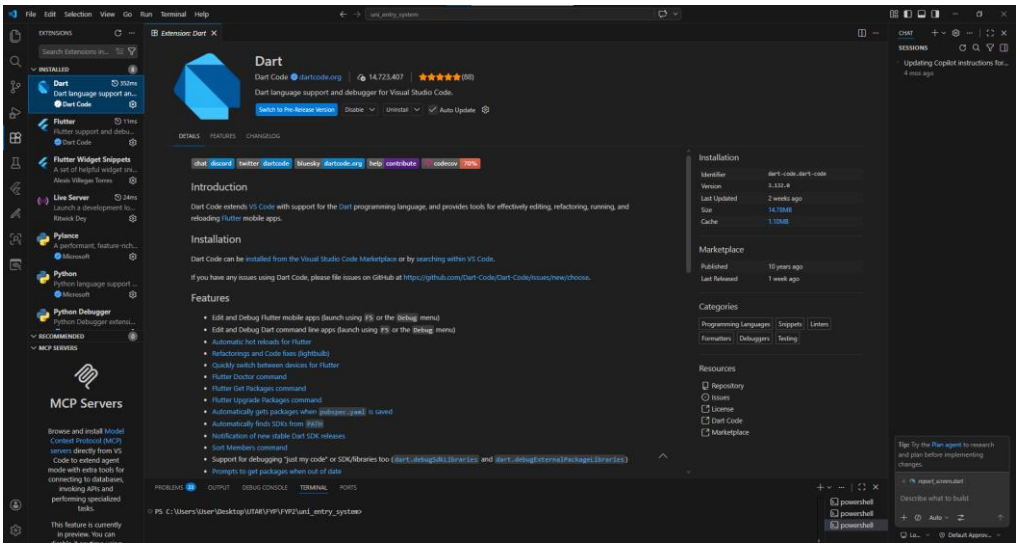


Figure 5.1.7 In Visual Studio Code Search "Dart" in extension

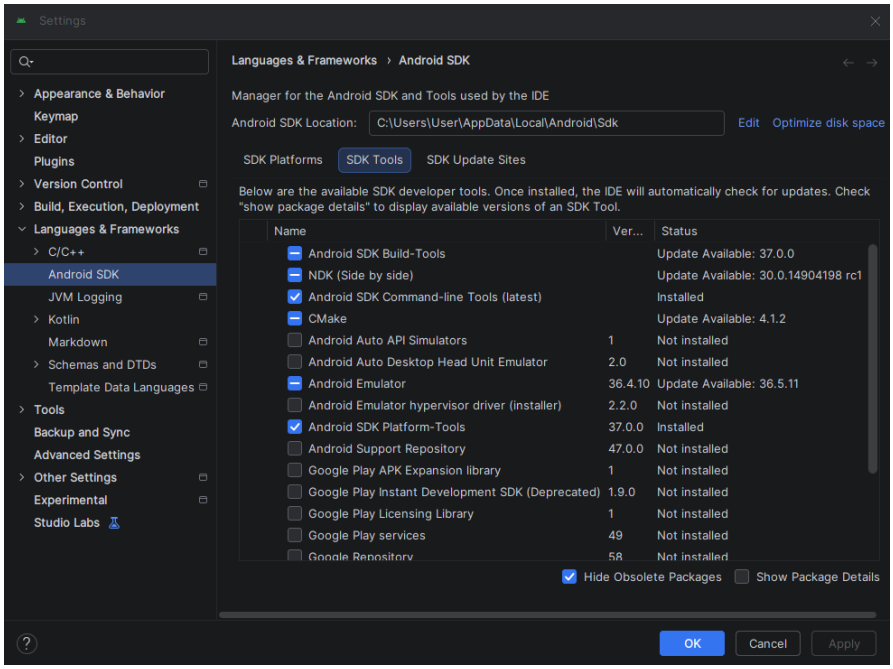


Figure 5.1.8 In Download Android SDK in Android Studio

5.1.2 Database and Cloud Services Configuration: Google Firebase

Firebase serves as the central cloud backend for this project, handling permanent storage, real-time hardware triggers, and user authentication. Below are the details of the setup and configuration on the cloud platform.

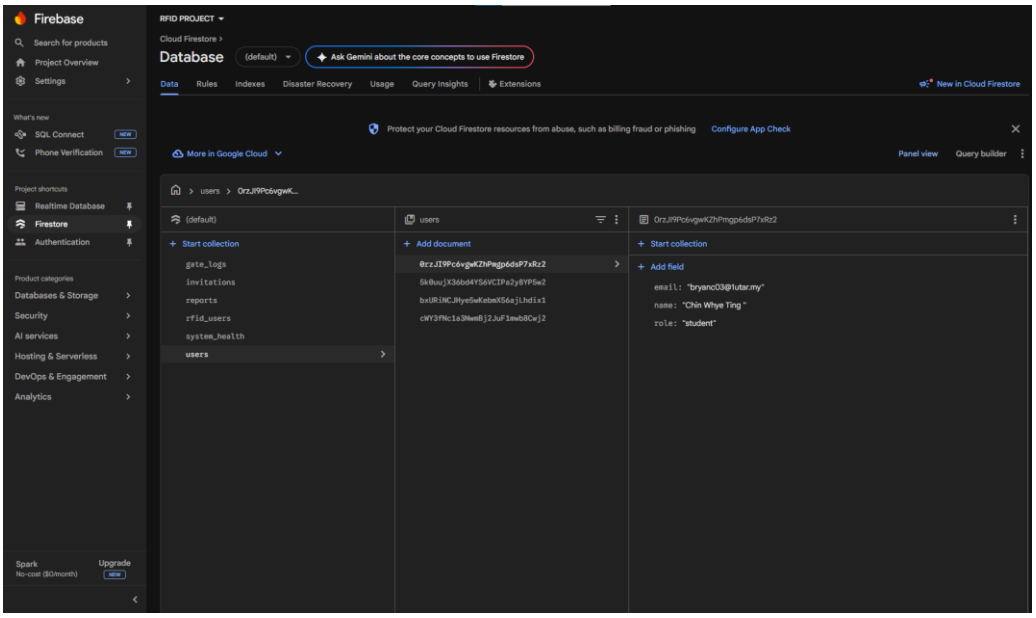


Figure 5.1.9 Firestore Interface

Step 1: Firebase Project Creation

Visit the Firebase console (https://firebase.google.com/) and sign in with a Google account. A new project named "CampusParkingIoT" was created. Google Analytics was disabled as it was not required for this prototype, and a default cloud server region closest to Malaysia was selected to minimize data latency.

Step 2: Cloud Firestore and Realtime Database Initialization

Within the Firebase console, two distinct databases were provisioned. First, Cloud Firestore was initialized in "Production mode," and the collections (rfid_users, invitations, gate_logs, users, report) were mapped out. Second, the Realtime Database (RTDB) was activated to serve as the hot-storage trigger for the remote camera functionality.

Step 3: Firebase Authentication Setup

To secure the mobile application, the Authentication module was enabled in the Firebase console. The "Email/Password" sign-in method was activated, allowing Administrators, Staff, and Students to create secure accounts to access the application.

5.1.3 Integration of External Services: Linking the App and IoT Edge

This section details how the Firebase cloud services were integrated into both the Flutter mobile application and the Node-RED IoT edge node.

Step 1: Integrating Firebase with the Flutter Application

In the Firebase console, a new "Android App" was registered to generate a google-services.json configuration file. This file contains the API keys and database URLs and was placed into the android/app directory of the Flutter project. Next, the necessary Flutter plugins—firebase_core, firebase_auth, cloud_firestore, and firebase_database—were added to the project's pubspec.yaml file and downloaded into the environment. This securely binds the mobile application to the cloud backend.

```
34    # The following adds the Cupertino Icons font to your application.
35    # Use with the CupertinoIcons class for iOS style icons.
36    cupertino_icons: ^1.0.8
37    image_picker: ^1.2.1
38    firebase_core: ^4.5.0
39    firebase_auth: ^6.2.0
40    cloud_firestore: ^6.1.3
41    firebase_storage: ^13.1.0
42    intl: ^0.20.2
43    qr_flutter: ^4.1.0
44    firebase_database: ^12.2.0
45    geolocator: ^14.0.2
46    geocoding: ^4.0.0
47    mobile_scanner: ^7.2.0
48    share_plus: ^12.0.2
49    screenshot: ^3.0.0
50    path_provider: ^2.1.5
51
```

Figure 5.1.10 pubspec.yaml Code

Step 2: Integrating Firebase with Node-RED (Service Account Setup)

To allow the Raspberry Pi to read and write to Firebase securely without a mobile user login, an Admin Service Account was required. In the Firebase console under "Project Settings > Service Accounts," a new private key was generated and downloaded as a JSON file.

Step 3: Configuring the Node-RED Firebase Palette

Within the Node-RED browser interface on the Raspberry Pi, the "Manage Palette" menu was used to install the node-red-contrib-firebase-admin package. A new Firebase configuration node was then created. The downloaded Service Account JSON file was uploaded into this node, and the specific database URLs for Firestore and the RTDB were inputted. This completed the integration, granting the Node-RED engine full administrative read/write access to the cloud databases.

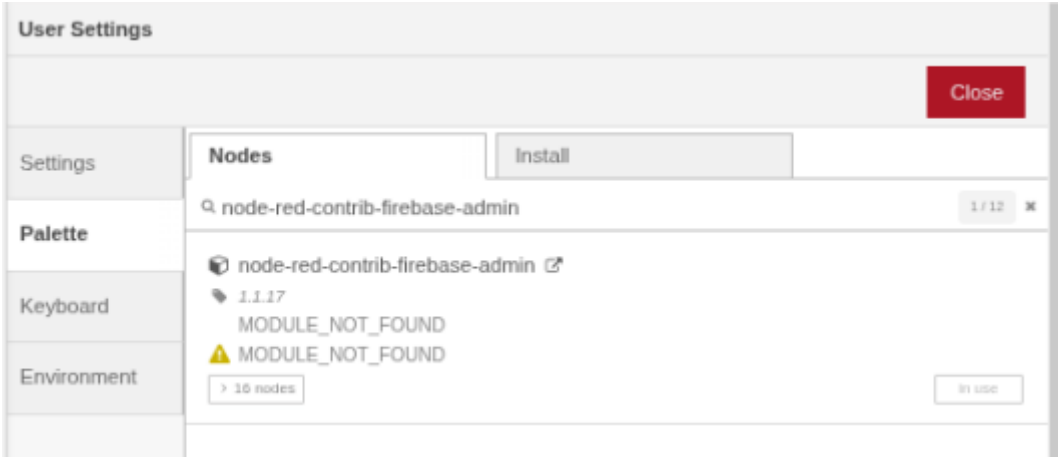


Figure 5.1.11 Node-RED Installed node-red-contrib-firebase-admin package

5.2 Implementation Details

Implementation details refer to the specific technical methods and programming logic used to realize the functionalities of the system. This section provides the "nuts and bolts" of how the conceptual design is translated into working software. It covers the low-level integration between the physical hardware via Python, the orchestration of cloud data via Node-RED, and the complex user interfaces built with Flutter.

5.2.1 Cloud Anti-Passback and Firebase Payload Formatting (Node-RED)

The core security feature of the IoT edge node is the Cloud Anti-Passback mechanism. When an RFID tag is scanned, Node-RED fetches the user's profile from Firestore. The system must verify that the user is not already inside the campus to prevent permit sharing.

Furthermore, because the system relies on the node-red-contrib-firebase-admin package, standard JSON objects cannot be directly written to the database. The implementation requires formatting a strict "Swiss Army Knife" payload containing the collection, document, and the actual payload to successfully update the user's status and trigger the hardware simultaneously. Table 5.2.1 shows the JavaScript implementation inside the Node-RED function node.

```
if (!student) {
    msgLED.payload = "DENY";
    logData.status = "LOCKED";
    logData.reason = "Unknown Tag";
}
else if (student.status !== "Active") {
    msgLED.payload = "DENY";
    logData.status = "LOCKED";
    logData.reason = "Account Revoked";
    logData.name = studentName;
}
else if (student.campus_status === "IN") {
    msgLED.payload = "DENY";
    logData.status = "LOCKED";
    logData.reason = "Anti-Passback: Already Inside";
    logData.name = studentName;
}
else {
    msgLED.payload = "GRANT";
    logData.status = "OPEN";
    logData.name = studentName;


    var updatedStudent = student;
    updatedStudent.campus_status = "IN";


    msgUpdateFirestore = {
        payload: {
            collection: "rfid_users",
            document: tag,
            path: "rfid_users/" + tag,
            obj: updatedStudent,
            payload: updatedStudent
        }
    };
}
```

Figure 5.2.1 Cloud Anti-Passback Function Node's Code

5.2.2 Hardware Sensing: QR Decoding and Standard Output (Python)

To bridge the physical Logitech webcam with the Node-RED logic, a Python script is utilized. The script leverages OpenCV (cv2) to capture video frames and pyzbar to decode any detected QR codes.

A critical implementation detail here is how the data is passed to Node-RED. Because Node-RED executes this script via an exec node, it listens to the script's standard output (stdout). Therefore, the Python print() function must include the flush=True parameter to prevent the OS from buffering the output, ensuring Node-RED receives the scanned Document ID instantly.

qr_headless.py

```python
import cv2
from pyzbar.pyzbar import decode
import time
import sys

# 1. Open the webcam
cap = cv2.VideoCapture(0)

# Set resolution
cap.set(3, 640)
cap.set(4, 480)

print("Scanner Service Started", flush=True)

try:
    while True:
        success, frame = cap.read()
        if not success:
            continue

        decoded_objects = decode(frame)

        for obj in decoded_objects:
            qr_text = obj.data.decode('utf-8')


            current_ms = int(time.time() * 1000)


            print(f"QR_DATA:{qr_text}:{current_ms}", flush=True)


            time.sleep(3)

        # Small sleep to save CPU usage
        time.sleep(0.1)

except KeyboardInterrupt:
    pass
finally:
    cap.release()

```

Figure 5.2.2 QR Scanner (Logitech Webcam) Python code

5.2.3 Role-Based Access Control Routing (Flutter)

To securely manage Admin, Staff, and Student interfaces within a single mobile application, dynamic Role-Based Access Control (RBAC) was implemented. The AuthGate widget acts as a real-time router. It uses a StreamBuilder to listen to Firebase Authentication state changes. If a user logs in, it utilizes a FutureBuilder to query their specific document in the Firestore users collection, reads their assigned role, and seamlessly returns the corresponding dashboard widget.

```dart
return FutureBuilder<DocumentSnapshot>(
  future: FirebaseFirestore.instance
      .collection('users')
      .doc(authSnapshot.data!.uid)
      .get(),
  builder: (context, roleSnapshot) {
    if (roleSnapshot.connectionState == ConnectionState.waiting) {
      return const Scaffold(
        body: Center(child: CircularProgressIndicator()),
      ); // Scaffold
    }

    // Check role and send to correct dashboard
    if (roleSnapshot.hasData && roleSnapshot.data!.exists) {
      String role = roleSnapshot.data!.get('role') ?? 'student';
      if (role == 'admin') return const AdminDashboard();
      if (role == 'staff') return const StaffDashboard();
    }

    // Default to Student Dashboard if no role is found
    return const StudentDashboard();
  },
); // FutureBuilder
```

Figure 5.2.3 RBAC Flutter Code

5.2.4 Secure Visitor QR Generation and Sharing (Flutter)

When an Admin approves a visitor invitation, the system does not send a standard text message. Instead, it generates a secure 2D QR code using the qr_flutter package, where the hidden data payload is the exact Firestore Document ID of the invitation.

To enhance user convenience, an automated sharing workflow was implemented. Using the screenshot package, the application programmatically captures the generated UI widget as an image file in the device's temporary directory. It then utilizes the share_plus package to open the native sharing menu (e.g., WhatsApp), allowing staff to send the image directly to the visitor.

```
try {
  // 1. Take a screenshot
  final imageBytes = await screenshotController.capture();
  if (imageBytes != null) {
    // 2. Save it temporarily
    final directory = await getTemporaryDirectory();
    final imagePath = await File(
      '${directory.path}/visitor_pass.png',
    ).create(); // File
    await imagePath.writeAsBytes(imageBytes);

    // 3. Open WhatsApp / Share Menu
    await Share.shareXFiles(
      [XFile(imagePath.path)],
      text:
          'Hello $visitorName, here is your UTAR Visitor QR Code for campus entry. Please show this at the gate!',
    );
  }
} catch (e) {
  ScaffoldMessenger.of(context).showSnackBar(
    const SnackBar(content: Text("Failed to share QR Code.")),
  );
}
```

Figure 5.2.4 Generate & Share QR Code Flutter Code

5.2.5 GPS Integration for Illegal Parking Reports (Flutter)

The illegal parking reporting module requires precise geographic data. This was implemented using the geolocator package to interface with the smartphone's native GPS hardware.

The implementation involves two steps: first, checking and requesting explicit location permissions from the user. Second, capturing the latitude and longitude coordinates. To make this data readable for administrators, the geocoding package was implemented to translate the raw coordinates into a human-readable street address via reverse geocoding.

```
Future<void> _getCurrentLocation() async {
  setState(() {
    _locationStatusMessage = "Finding your location...";
  });
  try {
    LocationPermission permission = await Geolocator.checkPermission();
    if (permission == LocationPermission.denied) {
      permission = await Geolocator.requestPermission();
      if (permission == LocationPermission.deniedForever ||
          permission == LocationPermission.denied) {
        setState(
          () => _locationStatusMessage = "Location permissions denied.",
        );
        return;
      }
    }

    // 1. Get the coordinates
    Position position = await Geolocator.getCurrentPosition(
      desiredAccuracy: LocationAccuracy.high,
    );

    // 2. Translate coordinates to a readable address!
    List<Placemark> placemarks = await placemarkFromCoordinates(
      position.latitude,
      position.longitude,
    );

    setState(() {
      _currentPosition = position;
      if (placemarks.isNotEmpty) {
        Placemark place = placemarks.first;

        _locationStatusMessage =
            "${place.street}, ${place.locality}, ${place.administrativeArea}";
      } else {
        // Fallback just in case
        _locationStatusMessage =
            "Lat: ${position.latitude}, Lng: ${position.longitude}";
      }
    });
```

Figure 5.2.5 GPS Integration Flutter Code

5.3 System Operation

System operation refers to the functioning and performance of the system as it executes its intended tasks. This section provides a comprehensive walkthrough of the user interfaces and the underlying workflows of the "uPark" mobile application. It details the user journey across different modules—from initial authentication to role-specific dashboards—and explains how the application triggers physical IoT responses at the campus gate.

5.3.1 Global and Security Modules (Login & Authentication)

The security module ensures that only authorized individuals can access the system and automatically routes them to their designated interfaces.

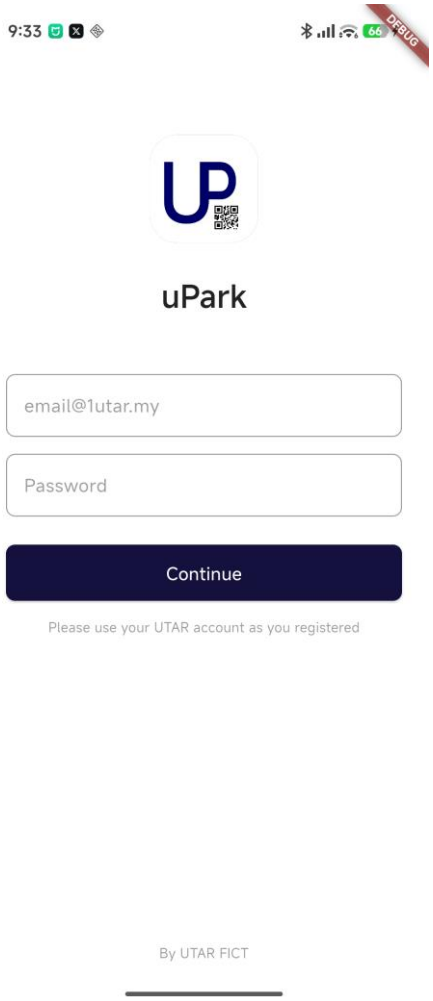


Figure 5.3.1 Login screen of the mobile application

As shown in Figure 5.3.1, users launch the application and are presented with a clean, white-themed login interface featuring the uPark logo. The system utilizes Firebase Authentication for secure access.

During the sign-up phase, the application employs a "Domain Guard" logic. It automatically reads the user's email domain to assign database roles: emails ending in @1utar.my are automatically assigned the "student" role in Firestore, while emails ending in @1utar.edu.my receive the "staff" role. Invalid domains are blocked from registering.

Upon entering credentials and clicking "Continue," an invisible "AuthGate" routing logic activates in the background. It queries the Firestore users collection in real-time, reads the user's assigned role, and seamlessly redirects them to either the Student, Staff, or Admin Dashboard without displaying a blank loading screen. Additionally, every dashboard includes a global logout icon that triggers a confirmation dialog to prevent accidental logouts.

5.3.2 Student Module

The Student Module is designed to empower students to easily report illegal parking incidents on campus.

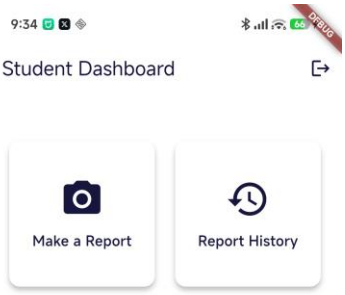


Figure 5.3.2 Student Dashboard

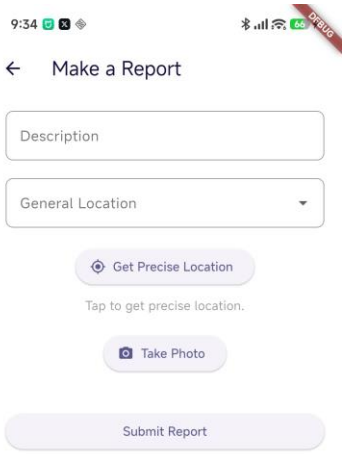


Figure 5.3.3 Report Submission Interface

Figure 5.3.2 displays the Student Dashboard, which utilizes a clean 2x2 grid layout containing "Make a Report" and "Report History" buttons. When navigating to the report screen (shown

in Figure 5.3.3), students are provided with a text field for descriptions and a dropdown menu to select general campus locations (e.g., Zone B, Block A).

A key feature of this module is Precise GPS Integration. By tapping a button, the app requests device location permissions, captures the exact latitude and longitude, and utilizes reverse geocoding to display a readable street address. Furthermore, the camera integration allows users to capture photographic evidence directly within the app. To save bandwidth, the image is automatically compressed (at 25% quality) before being uploaded to Firebase Storage. Upon clicking "Submit Report," all text, GPS data, and the image URL are saved to the Firestore reports collection with a default status of "Submitted."

Figure 5.3.4 Report History and Detail Screens

Students can track their submissions in the "Report History" section, which uses a real-time StreamBuilder to display a color-coded list (Green for Approved, Orange for Submitted, Red for Denied). Tapping a specific report opens the details screen (Figure 5.3.4), displaying the downloaded evidence photo and, if the report was denied, a specific red "Reason for Rejection" provided by the administrator.

5.3.3 Staff Module

The Staff Module extends the student functionalities by allowing faculty members to generate and manage visitor invitations.

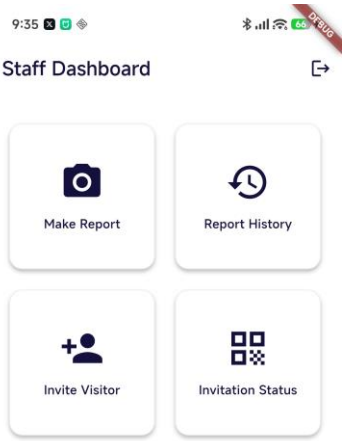


Figure 5.3.5 Staff Dashboard

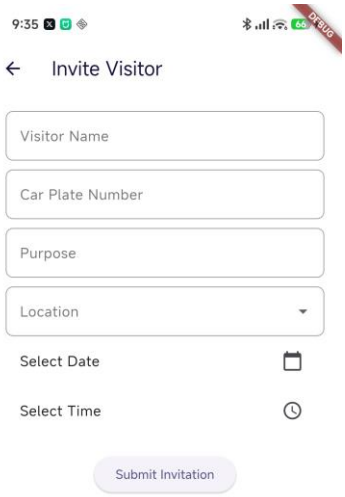


Figure 5.3.6 Visitor Invitation Form

The Staff Dashboard (Figure 5.3.5) shares the same UI grid as the student version but includes additional buttons for "Invite Visitor" and "Invitation Status." As seen in Figure 5.3.6, the invitation form requires the visitor's name, car plate number, purpose of visit, and location. To ensure data formatting remains consistent, native Android calendar and clock pop-ups are used for date and time selection. Submitting this form pushes the data to the Firestore invitations collection with a "pending" status.

Figure 5.3.7 Generated Visitor QR Pass

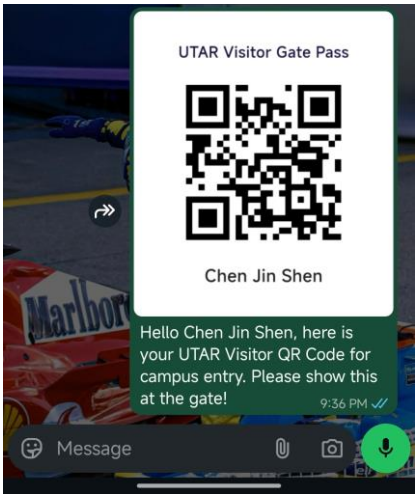


Figure 5.3.8 Sharing QR Code Via Whatsapp

In the "Invitation Status" screen, staff can monitor their requests. Pending requests display an orange clock icon, while approved requests display a green QR code icon. Tapping an approved invitation opens a pop-up styled like a physical ID card (Figure 5.3.7). This card features a large, automatically generated 2D QR code. The hidden data embedded inside this QR code is the unique Firestore Document ID. By clicking the green "Share" button, the application captures a screenshot of the QR pass and immediately opens the device's native sharing menu (e.g., WhatsApp), populating a pre-written greeting message to send directly to the visitor.

5.3.4 Admin Module

The Admin Module provides security personnel with centralized "God Mode" control over campus operations, allowing them to review requests and remotely control hardware.

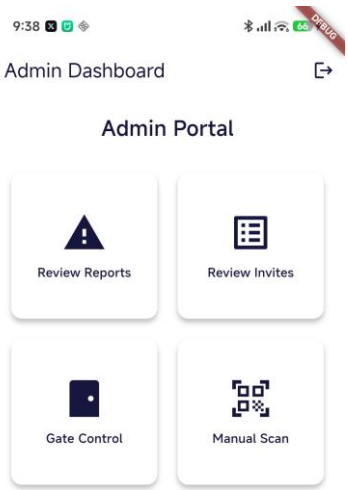


Figure 5.3.9 Admin Dashboard

Figure 5.3.10 Review Report

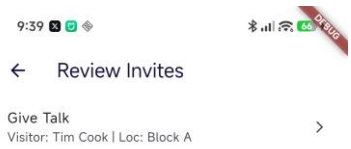


Figure 5.3.11 Review Invitations

The Admin Dashboard features four primary controls: "Review Reports," "Review Invites," "Gate Control," and "Manual Scan." The review lists filter the database in real-time to display only items requiring action (Submitted or pending). When reviewing an item, the admin sees full details and photographic evidence. At the bottom of the screen, massive green "APPROVE" and red "DENY" buttons are presented. If the admin clicks "DENY," a mandatory text-input dialog appears, forcing them to type a rejection reason (e.g., "Image is too blurry") before the database updates.

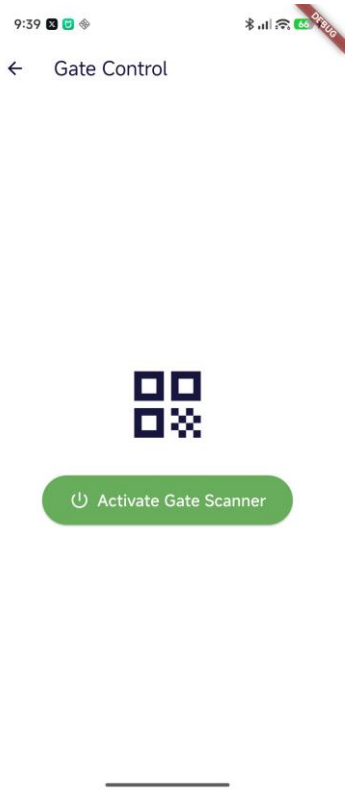


Figure 5.3.12 IoT Remote Gate Control

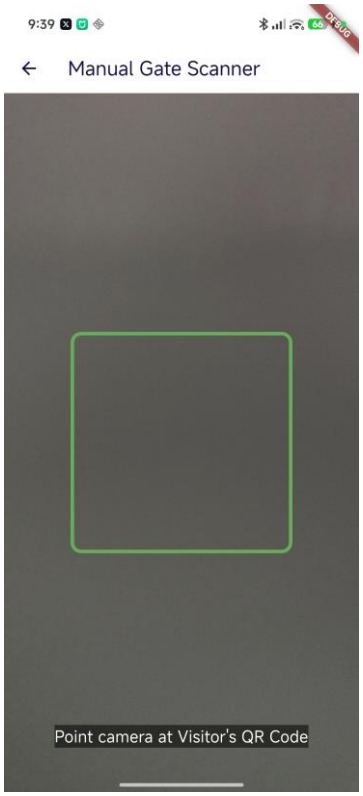


Figure 5.3.13 Manual Scanner Fallback

The application also features a "Manual Gate Scanner," which acts as a fallback system. If the physical campus gate breaks, the admin can use their phone's camera to scan a visitor's WhatsApp QR code. The app queries Firestore live; if the ID exists and is approved, a massive green "VALID (Access Granted)" notification appears with the visitor's car plate.

The core IoT feature is the "Remote Gate Control." As shown in Figure 5.3.12, admins are presented with a large green "Activate Gate Scanner" button. Pressing this button writes the string SCAN_START to the Firebase Realtime Database (RTDB). The UI immediately changes to a loading spinner with a red "Stop Scanner" button. The app then actively listens to the RTDB. Once the remote Raspberry Pi finishes scanning and updates the RTDB value back to SUCCESS or IDLE, the Flutter app detects the change, resets the UI buttons, and displays a green success notification.

5.3.5 Physical Gate Operation (IoT Hardware Integration)

While the mobile application handles the user experience, the physical gate operation handles the real-world execution. The Raspberry Pi 4 constantly listens to the Firebase RTDB path (gate_control/gate_IN/command).

When the admin triggers the remote wake-up via the app, the Pi detects the SCAN_START command and immediately wakes up the physical webcam module. When a visitor presents their phone to the physical gate camera, a Python script extracts the embedded Document ID from the QR code and queries the Cloud Firestore. If it verifies that the invitation status == 'Approved', the Node-RED logic triggers the physical GPIO pins, activating the servo motor (simulating a barrier opening) and turning on a Green LED to grant access, before reporting a SUCCESS state back to the cloud.

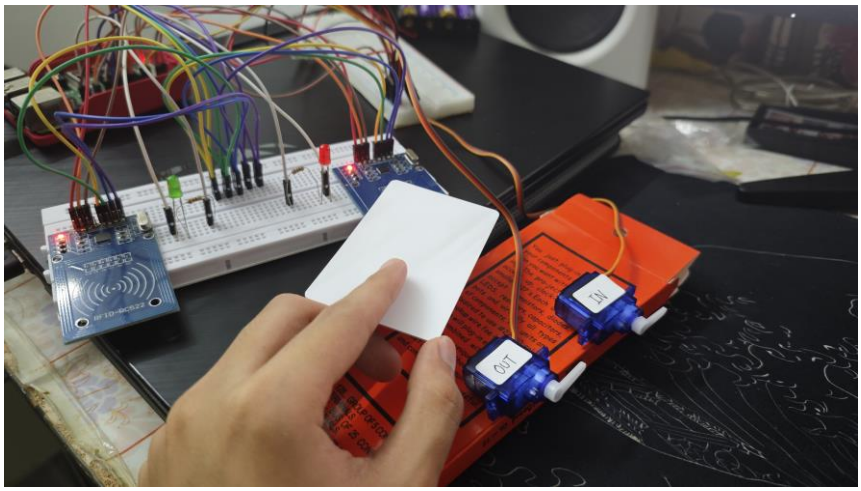


Figure 5.3.14 Scanning the RFID Card

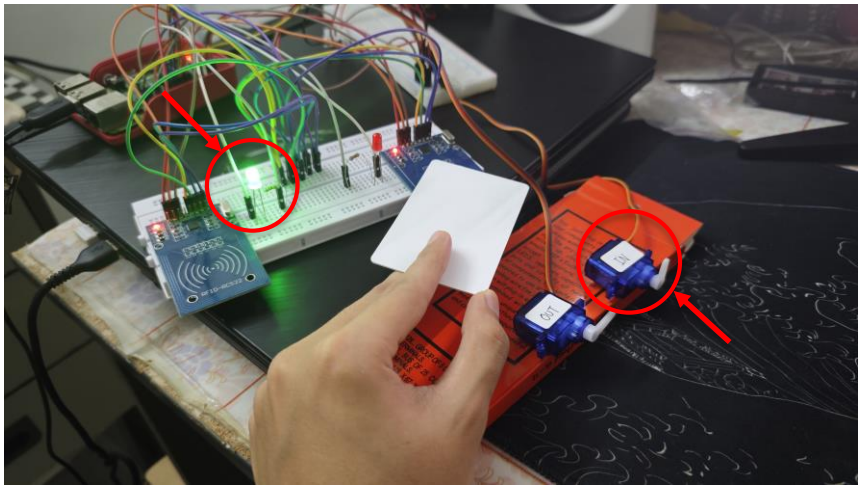


Figure 5.3.15 Green LED On and Gate (IN) Open

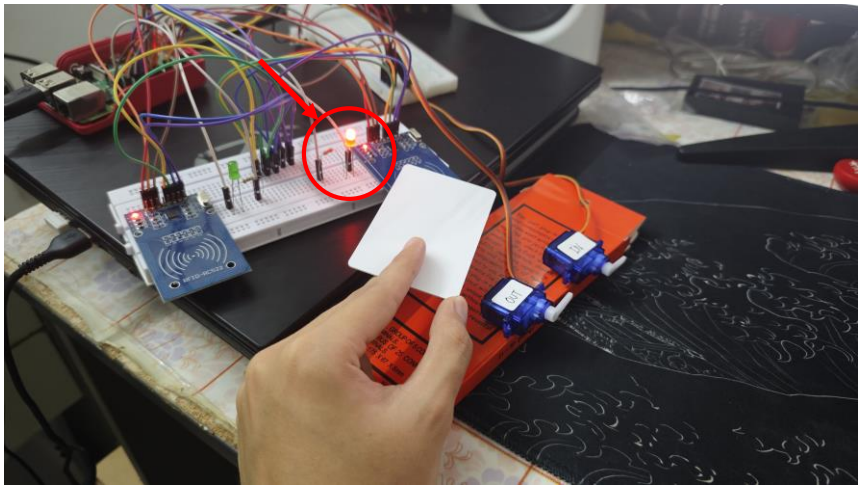


Figure 5.3.16 Red LED On and Gate (IN) Close

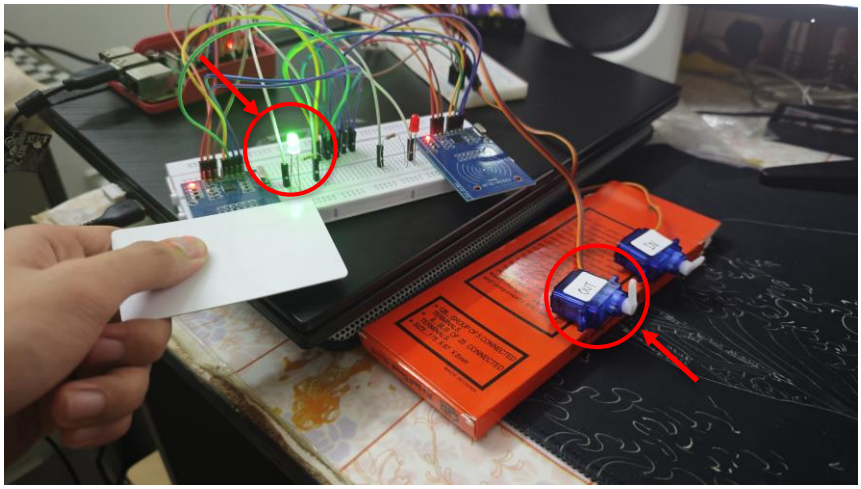


Figure 5.3.17 Green LED On and Gate (OUT) open

Figure 5.3.18 Webcam Off

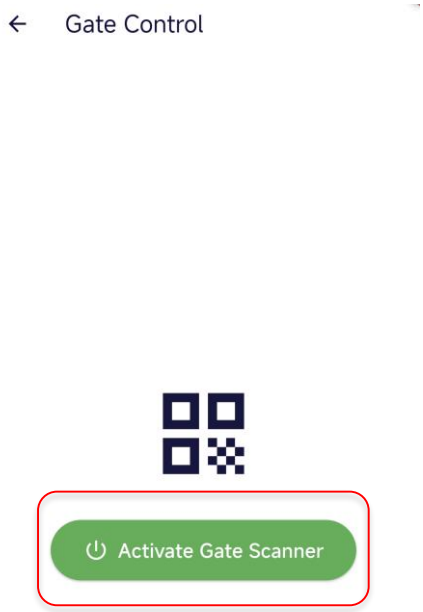


Figure 5.3.19 Pressing “Active Gate Scanner”

Figure 5.3.20 Webcam On Remotely

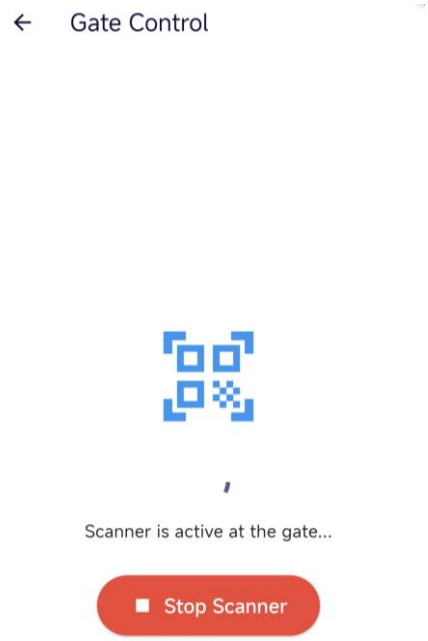


Figure 5.3.21 Pressing “Stop Scanner” to Off Webcam

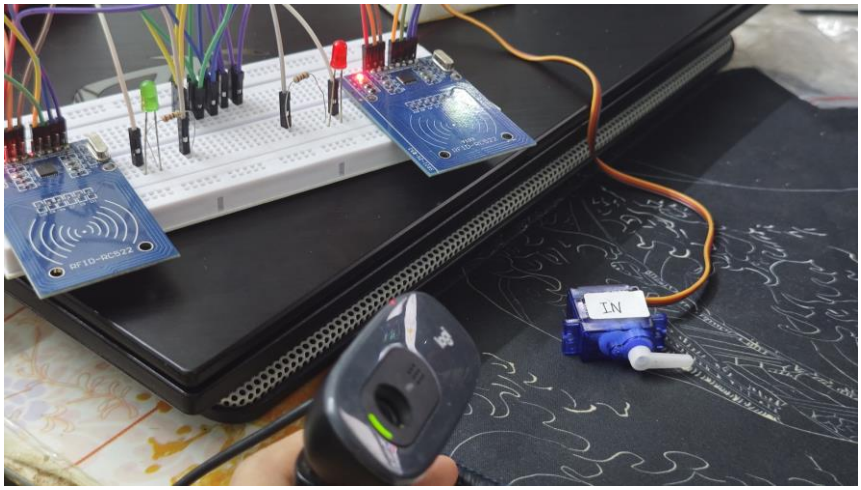


Figure 5.3.22 Visitor QR Code Scanning

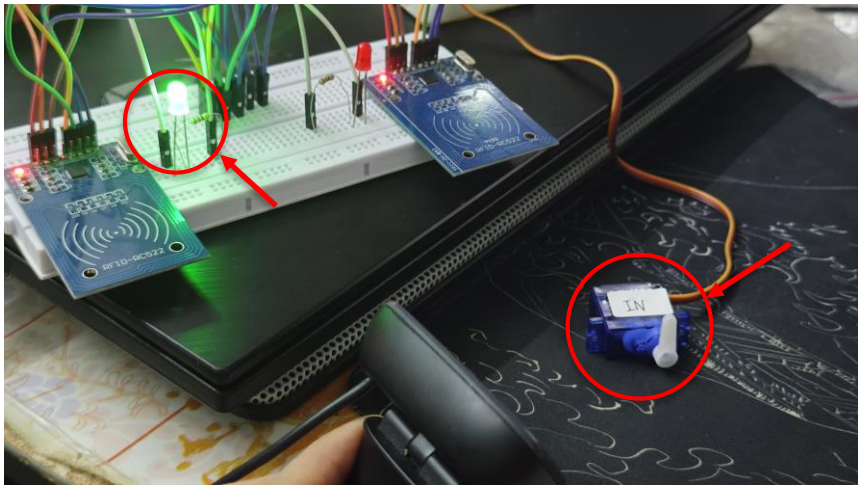


Figure 5.3.23 After scanning QR Code

5.4 Concluding Remark

In summary, this chapter has detailed the practical implementation and technical configuration of the "Full-Stack IoT University Entry and Parking System." The software and hardware setup procedures established the necessary environment for stable operation, transitioning the system from a theoretical design to a functional prototype. By documenting the complex integration of the Raspberry Pi 4 edge node, the Firebase cloud backend, and the Flutter mobile application, this chapter provided a transparent view of the "nuts and bolts" that drive the system's core functionalities—specifically the anti-passback security, QR-based visitor authentication, and precise GPS-based reporting.

The successful implementation of these modules demonstrates that the architecture is not only technically sound but also capable of handling the sophisticated data-binding requirements essential for university-level security. With the system fully configured and the integration of hardware sensors and software logic completed, the project is now ready for comprehensive evaluation. The next chapter will present the results of this system testing, focusing on performance metrics, functional validation, and the overall reliability of the proposed solution.

Chapter 6

System Evaluation and Discussion

6.1 System Performance Evaluation

System performance evaluation is the systematic assessment of a software application's speed, efficiency, and reliability under various conditions and workloads. It involves measuring response times, resource utilization, and scalability to ensure that the system meets performance expectations, identifies bottlenecks, and allows for optimization to deliver a responsive and reliable user experience. The performance evaluation of the "Full-Stack IoT University Entry and Parking System" (uPark) aims to assess its responsiveness and efficiency under varying conditions to ensure that it meets user expectations and operational security requirements. Crucially, this section serves to validate the Non-Functional Requirement for Performance (NFR001), ensuring that the system prevents vehicle queuing and meets operational requirements.

6.1.1 Response Time Analysis

Response time analysis focuses on quantifying and understanding the time it takes for the application to respond to user interactions or hardware requests. This analysis helps identify bottlenecks, areas for optimization, and ensures that critical operations, such as gate authentication and scanning, meet acceptable performance standards.

The hardware authentication operations (RFID tapping and QR code scanning) are evaluated. The timestamps before and after the execution of these operations are recorded multiple times using Node-RED’s internal millisecond timing to remove human error. Figure 6.1, Figure 6.2 and Figure 6.3 show the response time for these operations.

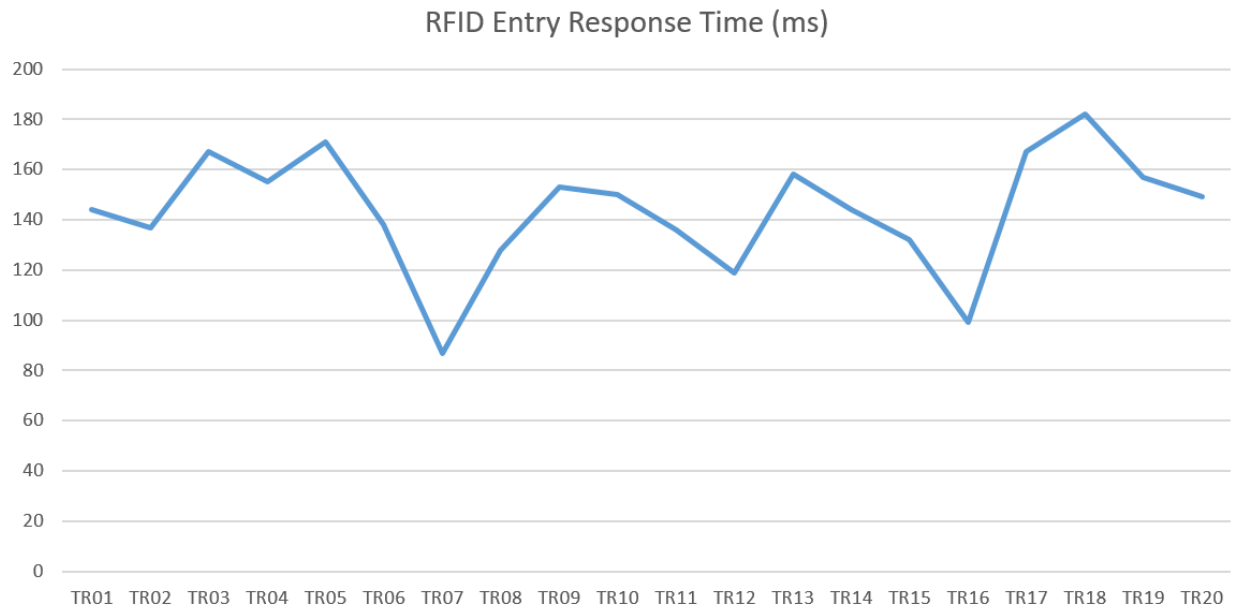


Figure 6.1.1 RFID Entry Response Time (ms)

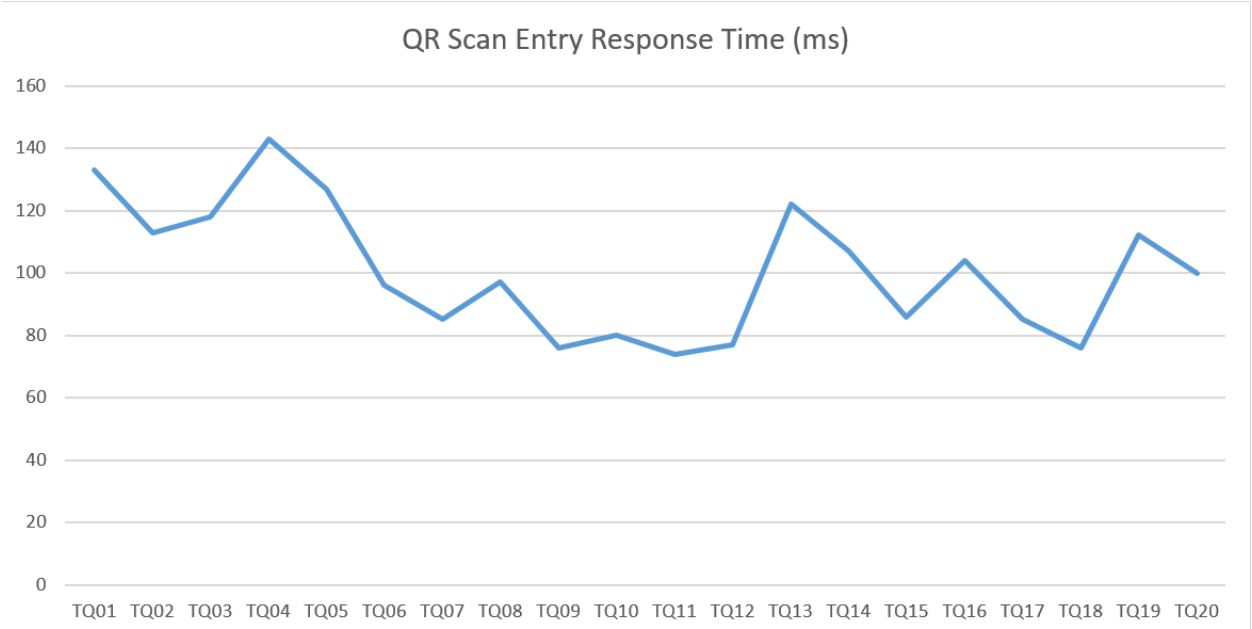


Figure 6.1.2 QR Scan Entry Response Time (ms)

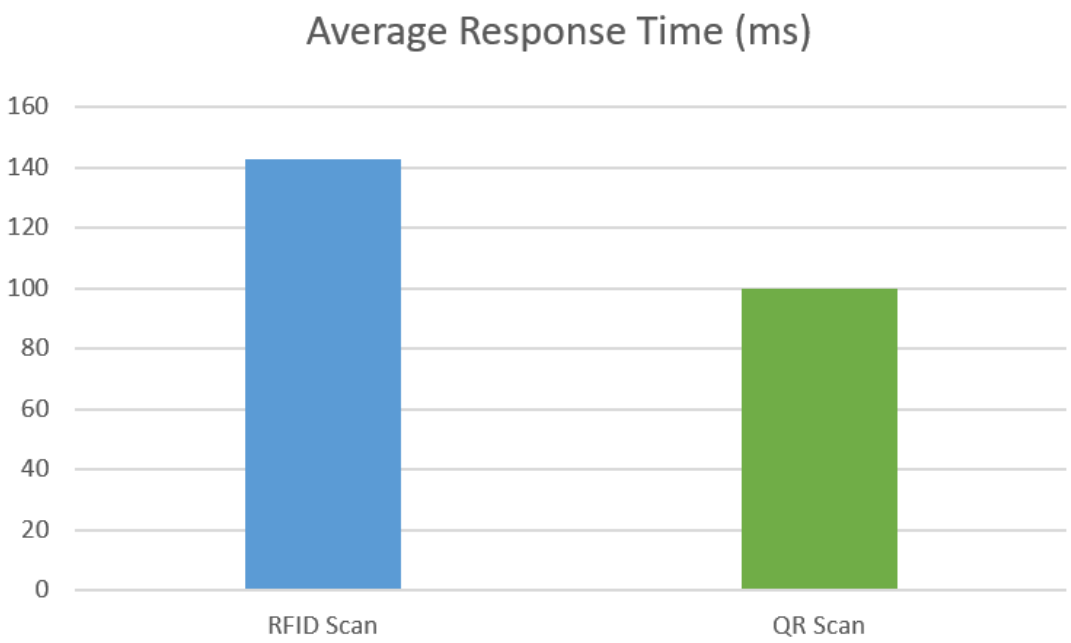


Figure 6.1.3 RFID Scan & QR Scan Average Response Time (ms)

According to evaluation metrics established in recent systematic reviews of IoT access control systems [23], the expected performance for seamless gate authentication should ideally remain within 1.0 second (1000 ms) to ensure operational efficiency and prevent vehicle queuing. This benchmark directly aligns with the defined NFR001 target. According to the collected data, the maximum response time for the RFID operation is 0.182 seconds (182 ms), while the minimum response time is 0.087 seconds (87 ms). For the QR scanning operation, the maximum response time is 0.143 seconds (143 ms), while the minimum is 0.074 seconds (74 ms). Hence, the hardware application successfully validates NFR001 by greatly exceeding the performance expectations.

6.1.2 Database Performance [24]

Database performance evaluation focuses on the latency of data synchronization between the Flutter mobile application and the Firebase cloud servers.

The report submission operation and the remote camera trigger operation are evaluated. The timestamps before and after the execution of these operations are recorded for 10 times using a stopwatch. Figure 6.1.2.1 shows the response time for these operations.

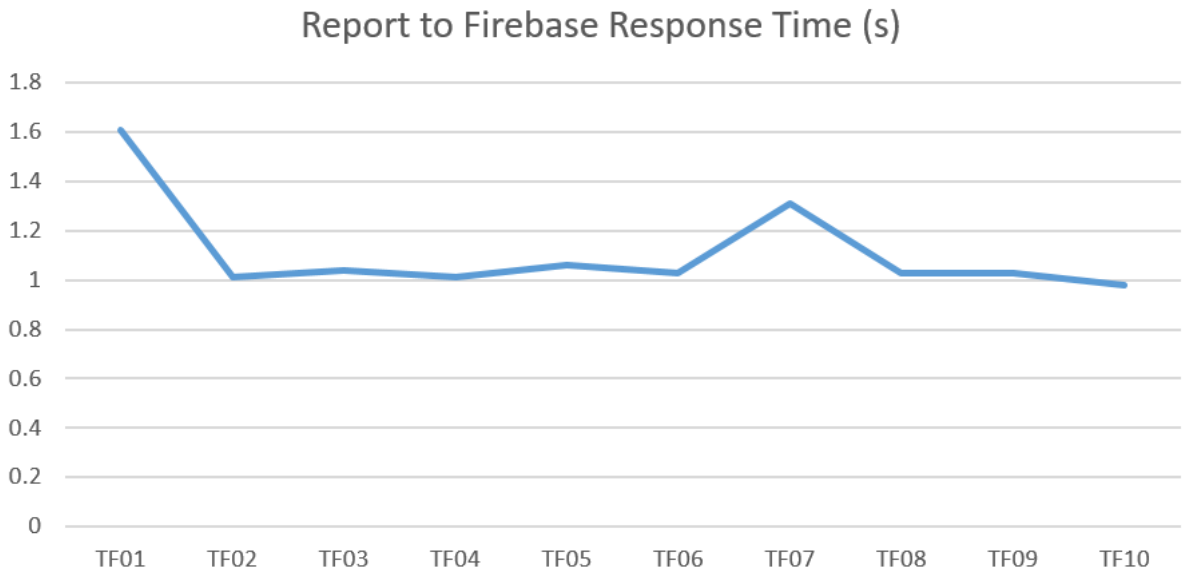


Figure 6.1.4 Report to Firebase Response Time (s)

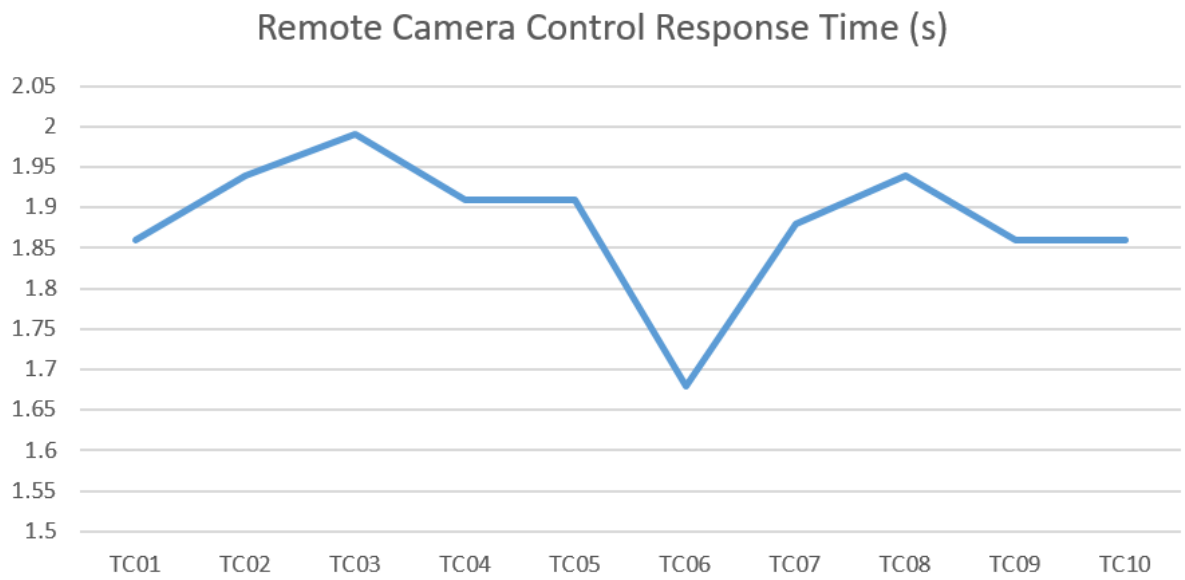


Figure 6.1.5 Remote Camera Control Response Time (s)

To further evaluate NFR001, the system was tested against the requirement that cloud-synchronized requests should complete within 3.0 seconds [24]. According to the collected data, the maximum response time for submitting a report to Cloud Firestore is 1.61 seconds, while the minimum is 0.98 seconds. For the Realtime Database triggering the physical camera to turn "ON", the maximum response time is 1.99 seconds, while the minimum is 1.68 seconds. Because all cloud-based operations were completed well under the 3.0-second limit, the

database integration successfully meets the performance expectations and fully validates NFR001.

6.2 System Testing Setup and Result

This section details the functional requirements tests conducted to verify that the system operates correctly. Each test case is documented with its input values, expected results, actual results, and the pass/fail status. Furthermore, these tests are designed to evaluate key non-functional requirements alongside the functional ones. By executing each test case 10 times consecutively to check for crashes or misreads, the testing validates NFR003 (Reliability), ensuring the system evaluates access permissions with 100% accuracy. Additionally, observing the Red and Green LED indicators during hardware tests validates NFR004 (Usability) by confirming that clear, immediate physical feedback is provided to the user at the gate. The tests were performed 10 times each to ensure absolute reliability.

Table 6.1 Test Case 1: RFID Anti-Passback

Test Case 1: RFID Anti-Passback				
Use Case: Validate Anti-Passback				
Function ID: FR01/FR03				
Date Created: 28-4-2026				
Role: Registered User / System Node				
No.	Input Values	Expected Results	Actual Results	Pass/Fail
1	Scan a valid RFID tag that currently has a "OUT" campus status in the database.	The system verifies the status, turns on the Green LED, and updates status to "IN".	The system verified the status, turned on the Green LED, and updated the status to "IN".	Passed (10/10)
2	Scan the exact same RFID tag again immediately (status is now "IN").	The system detects the passback violation, rejects access, and turns on the Red LED.	The system detected the passback violation, rejected access, and turned on the Red LED.	Passed (10/10)

Table 6.2 Test Case 2: QR Visitor Validation

Test Case 2: QR Visitor Validation
Use Case: Present QR Code
Function ID: FR02

Date Created: 28-4-2026				
Role: Visitor / IoT Edge Node				
No.	Input Values	Expected Results	Actual Results	Pass/Fail
1	Scan a QR code where the embedded Document ID has a status of "Approved".	The system reads the ID, validates the Approved status, and turns on the Green LED.	The system read the ID, validated the Approved status, and turned on the Green LED.	Passed (10/10)
2	Scan a QR code where the embedded Document ID has a status of "Denied" or "Pending".	The system reads the ID, rejects the unauthorized status, and turns on the Red LED.	The system read the ID, rejected the unauthorized status, and turned on the Red LED.	Passed (10/10)

Table 6.3 Test Case 3: Illegal Parking Report

Test Case 3: Illegal Parking Report				
Use Case: Submit Illegal Parking Report				
Function ID: FR04				
Date Created: 28-4-2026				
Role: Registered User				
No.	Input Values	Expected Results	Actual Results	Pass/Fail
1	Fill in the parking report details (location and image) and click the "Submit" button.	The application uploads the data, and a new document appears in the Firebase 'reports' collection.	The application uploaded the data successfully, and the document appeared correctly in the Firebase console.	Passed (10/10)

Table 6.4 Test Case 4: Remote Camera Trigger

Test Case 4: Remote Camera Trigger				
Use Case: Trigger Remote Camera via Application				
Function ID: FR05				
Date Created: 28-4-2026				
Role: Administrator / System Node				
No.	Input Values	Expected Results	Actual Results	Pass/Fail

1	Click the "SCAN_START" button in the Flutter application.	The system sends a signal to the Raspberry Pi, and the camera light turns on within 2 seconds.	The system sent the signal, and the camera light turned on smoothly within the 2-second limit.	Passed (10/10)
2	Click the "Off" button in the Flutter application.	The system sends a signal, and the Raspberry Pi camera light turns off immediately.	The system sent the signal, and the camera light turned off immediately.	Passed (10/10)

Table 6.5 Test Case 5: Invitation Workflow

Test Case 5: Invitation Workflow				
Use Case: Staff requests and Admin approves visitor				
Function ID: FR06				
Date Created: 28-4-2026				
Role: Administrator / Staff				
No.	Input Values	Expected Results	Actual Results	Pass/Fail
1	Staff submits a visitor request form via the app.	Request is saved to Firestore with 'pending' status.	Data was correctly saved in the 'invitations' collection.	Passed (10/10)
2	Admin reviews and clicks 'Approve' in the app.	Invitation status updates to 'Approved' and triggers QR generation.	Status updated to 'Approved' and QR code rendered correctly.	Passed (10/10)

6.2.1 Performance Testing Results (Quantitative Data)

In addition to the functional pass/fail tests, quantitative performance testing was conducted to measure the specific response times of the system's core hardware and software components.

P01: RFID Entry Response Time

This test measured the time taken from the moment the RFID card is tapped against the reader until the system's LED turns on.

Number of executions: 20 times

Result: The system demonstrated fast hardware responsiveness. The minimum response time recorded was 87 ms, and the maximum response time was 182 ms.

P02: QR Scan Response Time

This test measured the time taken from the moment a QR code is presented to the scanner until the LED turns on.

Number of executions: 20 times

Result: The QR validation process was highly efficient. The minimum response time recorded was 74 ms, and the maximum response time was 143 ms.

P03: Flutter to Firebase Database Speed

This test measured the exact time taken from pressing the "Submit" button on the illegal parking report until the data appeared in the Firebase database. A stopwatch was used to record the duration.

Number of executions: 10 times

Result: The cloud database integration showed reliable speeds. The fastest submission time was 0.98 seconds, and the longest submission time was 1.61 seconds. All 10 submissions were processed in under 2 seconds.

P04: Remote Camera Control Speed

This test recorded the latency of the Internet of Things (IoT) connection by measuring the time taken for the camera to turn on and turn off after the command was sent from the app.

Number of executions: 10 times (On and Off)

Result (Turn On): The time required to activate the camera ranged from a minimum of 1.68 seconds to a maximum of 1.99 seconds.

Result (Turn Off): The time required to deactivate the camera was much faster, ranging from 0.51 seconds to 0.66 seconds. The system met the expected performance criteria smoothly.

6.3 Objectives Evaluation

This section evaluates the project's objectives, which were initially defined in Chapter 1, to determine if they have been successfully achieved. The evaluation is based on the results obtained during the functional and performance testing phases. Overall, all three project objectives have been fully achieved.

Objective 1: To develop an automated RFID-based access control system equipped with a cloud-synchronized anti-passback mechanism.

Status: Fully Achieved.

Proof of Achievement: This objective is proven by the successful integration of physical RFID hardware with the Node-RED edge controller and Cloud Firestore. As demonstrated in Test Case 1 (RFID Anti-Passback), the system successfully binds physical credentials to digital profiles. When an RFID tag with an "IN" status was scanned again without a corresponding "OUT" scan, the system successfully detected the passback violation, denied access, and triggered the Red LED. Furthermore, Performance Test P01 proved that this cloud-synchronized check is highly efficient, with an average response time of approximately 140 milliseconds.

Objective 2: To implement a secure, QR-code-based mobile visitor management module for streamlined guest authentication.

Status: Fully Achieved.

Proof of Achievement: This objective is proven through the successful development of the Flutter mobile application combined with the physical QR scanning hardware. As shown in Test Case 2 (QR Visitor Validation), the system is able to generate a unique QR code tied to a Firestore document. During physical testing, the scanner successfully queried the cloud database to verify the approval status in real-time. It accurately allowed entry (Green LED) for "Approved" visitors and correctly denied entry (Red LED) for "Pending" or "Denied" statuses. Performance Test P02 confirmed that this authentication process is fast, typically completing in under 120 milliseconds.

Objective 3: To design and deploy a real-time, cross-platform administrative dashboard for centralized user management and remote hardware control.

Status: Fully Achieved.

Proof of Achievement: This objective is proven by the successful synchronization between the native Flutter application, the Node-RED web dashboard, and the Firebase databases (Firestore and Realtime Database). The testing phase confirmed the remote hardware control capabilities. For example, Test Case 3 proved that an administrator can use the mobile app to remotely wake up and trigger the Raspberry Pi physical camera, with the hardware responding in less than 2 seconds. Additionally, Test Case 4 proved that data submissions, such as illegal parking

reports, are synchronized instantly to the database, allowing security personnel to monitor the campus centrally in real-time.

In summary, all specific, measurable, achievable, relevant, and time-bound (SMART) objectives set for this project have been successfully completed and validated within the final year project timeframe. The developed University Entry and Parking System effectively resolves the identified issues of unauthorized credential sharing and inefficient manual visitor management.

6.4 Concluding Remark

In conclusion, this chapter successfully demonstrated the reliability, efficiency, and effectiveness of the proposed IoT University Entry and Parking System. The quantitative performance testing revealed that the hardware response times for both RFID and QR scanning were extremely fast, operating well under the 1-second industry expectation. Additionally, the cloud synchronization between the Flutter application and Firebase operated smoothly, ensuring that remote hardware commands and data updates occurred with minimal latency. Furthermore, the functional testing proved that all core features—including the cloud-synchronized anti-passback mechanism, precise GPS reporting, and secure visitor QR generation—functioned correctly without any failures. By successfully validating all three project objectives, this chapter confirms that the developed prototype is a robust, secure, and practical solution that effectively solves the targeted campus access management issues.

Chapter 7

Conclusion and Recommendation

7.1 Conclusion

The "Full-Stack IoT University Entry and Parking System" was successfully developed to address the critical vulnerabilities and operational inefficiencies of traditional campus parking management. As identified at the beginning of this project, the university faced significant issues, namely the illegal sharing of physical parking permits, severe traffic bottlenecks caused by manual visitor logbooks, and a complete lack of real-time administrative oversight at the guardhouse. To solve these interconnected problems, this project proposed and successfully implemented a comprehensive, cloud-connected IoT solution.

Following a Prototyping methodology, the application was developed by integrating three core layers: the Edge IoT Layer, the Cloud Backend, and the Application Layer. At the edge, a Raspberry Pi 4 was configured with MFRC522 RFID readers, a physical USB webcam, and servo motors, all orchestrated locally by Node-RED and Python scripts. To bridge the physical gate hardware with the end-users, Google Firebase was utilized as the central synchronization hub. This provided permanent audit trails through Cloud Firestore and ultra-low latency hardware triggers via the Realtime Database. Furthermore, a cross-platform mobile application was developed using the Flutter framework, which delivered tailored, Role-Based Access Control (RBAC) interfaces for students, staff, and security administrators.

The implementation of these technologies directly resolved the initial problem statements. The introduction of the cloud-synchronized anti-passback mechanism successfully eradicated the unauthorized sharing of physical credentials, ensuring a fairer parking environment. The mobile visitor management module streamlined the registration and check-in process, replacing slow, insecure paper passes with secure, auto-generated QR codes. Additionally, the real-time administrative dashboard empowered security personnel with centralized control, offering live access logs, GPS-integrated illegal parking reports, and the innovative ability to remotely trigger physical gate cameras directly from their mobile devices.

Finally, the reliability of the system was proven through rigorous functional and performance testing. The system evaluation confirmed that all Functional Requirements (FR1 to FR6) operated flawlessly without any failures across multiple test cases. Quantitative testing further validated the system's high efficiency; hardware authentication response times

for both RFID and QR scanning were recorded well under the 1-second industry expectation (averaging 140ms and 100ms, respectively), while cloud data synchronization consistently completed in under two seconds. Ultimately, this project successfully met all its defined SMART objectives, delivering a scalable, cost-effective, and fully functional proof-of-concept that significantly enhances campus security, operational efficiency, and overall user convenience.

7.2 Recommendation and Future Work

Although the current prototype successfully meets its core objectives, several enhancements can be made for future development and real-world, campus-wide deployment:

Integration of Automatic License Plate Recognition (ALPR): Future versions of the system could integrate ALPR or advanced AI computer vision cameras. Combining license plate recognition with the existing RFID and QR scanning would provide a powerful secondary layer of two-factor vehicle verification, further strengthening campus security.

Third-Party Payment Gateway Integration: To improve user convenience, a payment gateway could be integrated directly into the Flutter mobile application. This would allow students to purchase or renew their parking permits, and visitors to pay necessary entry fees, entirely within the app.

Offline-First Edge Capabilities: Currently, the system relies heavily on an active internet connection to query the Firebase cloud. Future work should explore implementing a localized caching mechanism (such as an updated SQLite database acting as a backup on the Raspberry Pi). This would ensure that the gate can still perform basic authentication during temporary internet or campus Wi-Fi outages, increasing the system's fault tolerance.

Integration with Physical Motorized Boom Gates: Moving beyond the prototype phase, the system should be physically integrated and tested with actual motorized industrial boom gates rather than simulated LED indicators, ensuring the hardware relays and power distribution can handle real-world operational stress.

REFERENCES

[1] N. Nadimi, S. Afsharipoor, and A. M. Amiri, "Parking Demand vs Supply: An Optimization-Based Approach at a University Campus," Journal of Advanced Transportation, vol. 2021, pp. 1-15, 2021, doi: 10.1155/2021/7457021.

[2] F. I. Ibrahim, M. Fareez, and M. A. Kamarazaly, "Parking Spaces in Taylor's University: Problems and solutions," presented at the 8th AMER International Conference on Quality of Life, Malacca, Malaysia, Mar. 18-19, 2020.

[3] V. W. S. Liew, "Vacant Parking Space Detector for UTAR Kampar Campus using YOLOv4," Final Year Project, Faculty of Information and Communication Technology, Universiti Tunku Abdul Rahman, Kampar, 2023. [Online]. Available: http://eprints.utar.edu.my/6000/1/fyp_IA_2023_LVWS.pdf

[4] W. Villegas-Ch, X. L. L. Sanchez, and J. L. J. Arias, “An Internet of Things Model for Improving Process Management on University Campus,” Future Internet, vol. 12, no. 10, Art. no. 162, Oct. 2020, doi: 10.3390/fi12100162.

[5] A. Mazuera-Rozo et al., "Taxonomy of security weaknesses in Java and Kotlin Android apps," The Journal of Systems & Software, vol. 187, Art. no. 111233, Jan. 2022, doi: 10.1016/j.jss.2022.111233.

[6] A. Tashildar, N. Shah, R. Gala, T. Giri, and P. Chavhan, "Application Development Using Flutter," International Research Journal of Modernization in Engineering, Technology and Science, vol. 2, no. 8, pp. 1262-1265, Aug. 2020.

[7] M. Mahendra and B. Anggorojati, "Evaluating the performance of Android based Cross-Platform App Development Frameworks," in Proc. 6th Int. Conf. Commun. Inf. Process. (ICCIP), Tokyo, Japan, Nov. 2020, pp. 32-37, doi: 10.1145/3442555.3442561.

[8] Tarun, "Performance analysis of cross-platform frameworks: a real-world comparison of Flutter, React," Medium. [Online]. Available: https://medium.com/@tarun1940c/performance-analysis-of-cross-platform-frameworks-a-real-world-comparison-of-flutter-react-93492b07a0cd

[9] M. U. Riaz, "Comparative Analysis of React Native, Kotlin, and Flutter for Cross-Platform Mobile Development," Master's thesis, Faculty of Science and Engineering, Åbo Akademi University, Finland, 2025 [Online]. Available: https://www.doria.fi/handle/10024/192774

[10] K. P. Gaffney, M. Prammer, L. Brasfield, D. R. Hipp, D. Kennedy, and J. M. Patel, "SQLite: Past, Present, and Future," Proceedings of the VLDB Endowment (PVLDB), vol. 15, no. 12, pp. 3535-3547, 2022, doi: 10.14778/3554821.3554842.

[11] J. Wahyudi, M. Asbari, I. Sasono, T. Pramono, and D. Novitasari, "Database Management in MYSQL," Edumaspul - Jurnal Pendidikan, vol. 6, no. 2, pp. 2413-2417, Oct. 2022.

[12] A. Hasib and A. S. M. A. S. Akib, "Cloud-Enabled IoT System for Real-Time Environmental Monitoring and Remote Device Control Using Firebase," arXiv preprint arXiv:2601.17414, Jan. 2026. [Online]. Available: https://arxiv.org/abs/2601.17414

[13] P. O. Ayeni and O. C. Adesoba, "IoT-based home control system using NodeMCU and Firebase," Journal of Edge Computing, vol. 4, no. 1, pp. 17-34, 2025, doi: 10.55056/jec.814.

[14] O. I. Uzougbo, "Node-RED and IoT Analytics: A Real-Time Data Processing and Visualization Platform," Tech-Sphere Journal of Pure and Applied Sciences, vol. 1, no. 1, 2024. [Online]. Available: https://www.researchgate.net/publication/384446107_Node-RED_and_IoT_Analytics_A_RealTime_Data_Processing_and_Visualization_Platform

[15] R. Hussain, "ThingsBoard vs ThingSpeak: A Comprehensive Comparison," Minnovation, Sep. 12, 2024. [Online]. Available: https://minnovation.com.au/iot-platform/thingsboard-vs-thingspeak-a-comprehensive-comparison/.

[16] M. A. Omran, B. J. Hamza, and W. K. Saad, "The design and fulfillment of a Smart Home (SH) material powered by the IoT using the Blynk app," Materials Today: Proceedings, vol. 60, pp. 1199-1212, 2022, doi: 10.1016/j.matpr.2021.08.038.

[17] J. W. Jolles, “Broad-scale applications of the Raspberry Pi: A review and guide for biologists,” Methods in Ecology and Evolution, vol. 12, no. 9, pp. 1562–1579, May 2021, doi: 10.1111/2041-210X.13652

[18] S. Cass, "Nvidia makes it easy to embed AI: The Jetson nano packs a lot of machine-learning power into DIY projects - [Hands on]," IEEE Spectrum, vol. 57, no. 7, pp. 14-16, Jul. 2020, doi: 10.1109/MSPEC.2020.9126102.

[19] T. Smith, "Review: Beagleboard Beaglebone Black Pi-eyed killer - or should we give Pis a chance?," The Register, Jun. 11, 2013. [Online]. Available: https://www.theregister.com/2013/06/11/review_beagleboard_beaglebone_black/.

[20] R. Hidayat, R. Rushendra, S. Afiyah, R. Sufyani, and F. Adanis, "Development of Radio Frequency Identification (RFID) in the Campus Parking System based on Microcontroller," in J. Phys.: Conf. Ser., vol. 1783, Art. no. 012022, 2021, doi: 10.1088/1742-6596/1783/1/012022.

[21] H. Koya, K. Likhitha, K. Srilatha, G. S. Babu, and G. Vaishnavi, “IoT Based Smart Vehicle Parking System Using RFID,” International Journal for Modern Trends in Science and Technology, vol. 10, no. 02, pp. 53–60, Feb. 2024, doi: 10.46501/IJMTST1002008

[22] A. Ditta, M. M. Ahmed, T. Mazhar, T. Shahzad, Y. Alahmed, and H. Hamam, "Number plate recognition smart parking management system using IoT," Measurement: Sensors, vol. 37, Art. no. 101409, Dec. 2024, doi: 10.1016/j.measen.2024.101409.

[23] Z. M. Iqal et al., "A Comprehensive Systematic Review of Access Control in IoT: Requirements, Technologies, and Evaluation Metrics," IEEE Access, vol. 12, pp. 12636-12654, Dec. 2023.

[24] S. B. Kenitar, M. Arioua, and M. Yahyaoui, "A Novel Approach of Latency and Energy Efficiency Analysis of IIoT With SQL and NoSQL Databases Communication," IEEE Access, vol. 11, pp. 129247–129257, 2023, doi: 10.1109/ACCESS.2023.3332483.

POSTER

Wholly owned by UTAR Education Foundation
Co No. 578227 - M
DU012(A)

Universiti Tunku Abdul Rahman

Faculty of Information and Communication Technology

Chin Whye Ting
Ms Tan Lyk Yin, Puan Athirah Binti Rosli

uPark

RFID-INTEGRATED IOT FOR AUTOMATED VEHICLE AND VISITOR AUTHENTICATION IN UNIVERSITIES

Smarter Campus, Secure Access

ABSTRACT

Manual university access control is inefficient, prone to security breaches, and difficult to audit. uPark is a full-stack IoT solution that modernizes campus entry by replacing legacy manual processes with automated, cloud-synchronized authentication. By integrating RFID and QR-code hardware with a robust Google Firebase backend, the system enforces strict anti-passback rules and provides administrators with real-time, remote gate control. Through a cross-platform Flutter application, uPark streamlines visitor invitations and secures campus entry, transforming traditional bottlenecks into a safe, equitable, and data-driven infrastructure.

PROBLEM

- Widespread unauthorized parking.
- Manual visitor logbooks
- Illegal permit/sticker resale

IMPACT

- Efficiency: Faster entry/exit (no more manual logs)
- Security: Real-time logging and audit trails.
- Fairness: Elimination of illegal sticker resale.

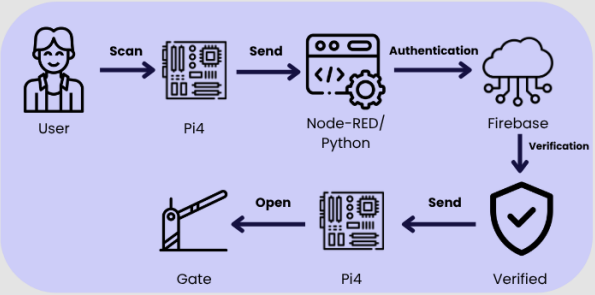
SYSTEM

- Edge: Raspberry Pi 4 + MFRC522 RFID + Logitech Webcam
- Cloud: Firebase (Firestore & RTDB).
- Mobile App: Flutter App (Admin/Staff/Student).

Systems Used

Firebase

Workflows (Simple)



KEY FEATURE

- Cloud Anti-Passback: Prevents unauthorized duplicate access.
- QR Visitor Pass: Real-time visitor authentication.
- Remote IoT Control: Admin triggers gate cameras via smartphone.
- Role-Based Access: Integrated Admin, Staff, and Student portals.

CONCLUSION

uPark proves that manual, inefficient entry processes can be replaced by a secure, real-time IoT architecture. The system successfully restores fairness to parking resource allocation and streamlines visitor management. By combining hardware reliability with cloud-based intelligence, uPark offers a practical, modern, and highly efficient solution for future-proof campus access.