Restaurant Inventory & Management System using NFC

Chuah Wai Kit[1], Nur Syazliana binti Zul[2], Teoh Chee Jin[3], Dr. Ahmad Shahidan bin Abdullah[4]

[1, 2, 3, 4] Universiti Teknologi Malaysia, Malaysia

[1]chuahkit@graduate.utm.my; [2]nur.syazliana@graduate.utm.ny; [3]teohjin@graduate.utm.my; [4]ashahidan@utm.my

Abstract— ***Efficient inventory management is critical to addressing challenges such as manual errors, inefficiencies, and delays in data updates within the restaurant industry. This capstone project introduces a Smart Inventory Management System (SIMS) that leverages Near Field Communication (NFC) technology, the ESP32 microcontroller, and Firebase to streamline inventory tracking and ensure product quality. The system records comprehensive inventory data, including location, expiry date, manufacturing date, transaction type (e.g., check-in/check-out), item name, and serial number, which is transmitted via the ESP32 to Google Sheets through an Application Programming Interface (API) for real-time processing and validation. Additionally, an ESP32 camera captures images of items during check-in to assess product quality. These images are stored in a Google Drive database and analyzed using AI-powered Python scripts to detect potential defects, ensuring inventory integrity. The system features a multi-platform interface, available as a web-based application and an Android Package (APK), allowing users to manage inventory seamlessly across devices after securely creating an account. Firebase Authentication ensures secure user access, while Firestore provides scalable data storage for inventory records. Automated scripts in Google Apps Script process and update transaction details dynamically within Google Sheets, enhancing real-time inventory tracking. By aligning with Sustainable Development Goal 9 (Industry, Innovation, and Infrastructure), this project fosters technological innovation, enhances operational efficiency, and establishes resilient digital infrastructure tailored to the restaurant industry. Key project phases include frontend design, API integration, hardware-software interfacing, AI-driven defect analysis, and cross-platform accessibility development. By integrating Internet of Things (IoT), cloud technologies, and Artificial Intelligence (AI), this system delivers an innovative, scalable, and robust solution that addresses operational inefficiencies while ensuring product quality.***

Keywords— Near Field Communication (NFC), Espressif 32 (ESP32), Internet of Things (IoT), Android Package (APK), Smart Inventory Management System (SIMs), Artificial Intelligence (AI), Application Programming Interface (API), Sustainable Development Goals (SDG)

1. Conceive

The conceive phase forms the critical foundation of this project, where the focus lies on understanding the core challenges in restaurant inventory management, analyzing user needs, and formulating a technologically advanced solution. This phase is structured across three key areas: hardware configuration using Arduino IDE, software and user interface development, and prototype design using Fusion 360 for laser-cut acrylic components. Team members collaborated to address specific roles, with one member focusing on hardware integration, another on software development, and the third on prototype creation and documentation. By combining insights from interviews, surveys, and site visits, the team identified inefficiencies in traditional inventory management practices, such as reliance on manual processes and outdated tools. These challenges informed the design and development of the Smart Inventory Management System (SIMs), a solution that leverages NFC technology, ESP32 microcontrollers, and Firebase for real-time tracking and defect detection. This phase aligns with the objectives of Sustainable Development Goal 9 (Industry, Innovation, and Infrastructure) by fostering technological innovation and promoting operational efficiency.

1.1 Background Introduction

Inventory management within restaurants is essential but fraught with inefficiencies, including manual errors, stock discrepancies, and delays in updating records. A survey conducted during the initial research phase revealed that restaurant staff spend over 11 hours weekly on inventory-related tasks, often resulting in inaccuracies that disrupt operations. These inefficiencies are exacerbated by the lack of automation, making it difficult to track stock levels, manage expiration dates, and ensure product quality.

The Smart Inventory Management System (SIMs) addresses these challenges by introducing an automated solution that integrates NFC technology for inventory tracking, ESP32 microcontrollers for data transmission, and Firebase for secure storage. Hardware components, configured through Arduino IDE, capture real-time data, while the software provides an intuitive, cross-platform interface for seamless management. The system ensures operational resilience by addressing critical pain points, aligning with SDG 9's emphasis on innovation and infrastructure development.

1.2 End-user Input Data

To design a user-centric solution, data was collected from two key sources: (1) an interview with Izwan, a staff member from The Cube Enterprise, and (2) a survey distributed via Google Forms targeting restaurant staff and customers.

The interview with Izwan highlighted significant inefficiencies in manual inventory tracking and quality assurance processes. He detailed challenges such as stock discrepancies, delayed updates, and difficulties in maintaining product freshness. The survey, distributed to a broader audience, collected responses on challenges faced in inventory management and expectations for a solution. Respondents identified key pain points, including the inability to track items in real time, manage expiration dates efficiently, and monitor product quality.

The combined data from these sources provided a holistic understanding of user needs, enabling the team to design SIMs as an innovative, scalable solution to address these challenges effectively.

1.3 User Needs Analysis

The collected data were analyzed using clustering techniques to identify common themes and critical pain points. Key user demographics included restaurant managers, clerks, and customers. Managers required tools for overseeing stock levels and ensuring quality, clerks emphasized real-time tracking and error reduction, and customers prioritized timely delivery and fresh products.

Recurring challenges included reliance on spreadsheets, errors in manual tracking, and inefficiencies in updating stock levels. The survey results highlighted a strong demand for a system that automates inventory processes, provides real-time updates, and integrates quality assurance features. These findings informed the functional requirements of SIMs, ensuring the solution addressed user pain points comprehensively.

1.4 Personification and Design Statement

The insights gathered from Izwan and the survey participants were instrumental in shaping the project's design goals. Izwan, a staff member at The Cube Enterprise, represented the operational side of inventory management, highlighting issues such as manual errors, delayed restocking, and the lack of automated quality checks. Similarly, the survey responses from restaurant staff emphasized the need for real-time inventory tracking, while customers highlighted expectations for quality and timeliness.

Based on these findings, the design statement for the project was formulated as: "To develop an intelligent and scalable inventory management system that integrates real-time tracking using NFC technology, automated defect detection through ESP32 cameras, and a user-friendly, cross-platform interface to enhance operational efficiency and ensure product quality."

This design statement encapsulates the goals of the project, ensuring alignment with user needs and SDG 9's objectives of fostering innovation and sustainable infrastructure.

1.5 Idea Generation

The idea generation phase employed design thinking methodologies, beginning with brainstorming sessions to explore innovative solutions. The team conceptualized features such as NFC-based real-time tracking, ESP32-enabled defect detection, and a cross-platform interface for seamless inventory management. Ideas were evaluated based on feasibility, technological viability, and alignment with user needs.

Hardware design involved programming ESP32 microcontrollers using Arduino IDE for NFC and camera integration. The software team developed a responsive, user-friendly interface using HTML, CSS, and JavaScript. The prototype design team used Fusion 360 to create laser-cut acrylic models for demonstrating hardware functionality. Iterative testing and refinement ensured the selected ideas addressed operational inefficiencies effectively.

1.6 Conceptual Prototype Development

The conceptual prototype development spanned three areas: hardware configuration, software development, and prototype design. The hardware team, working with Arduino IDE, integrated NFC tags for inventory tracking and ESP32 cameras for capturing product conditions during check-ins. The software team created a multi-platform interface, accessible as a web-based application and an Android APK, allowing users to manage inventory across devices securely.

The prototype design team utilized Fusion 360 to design modular, laser-cut acrylic components for hardware assembly. These components demonstrated key functionalities, including real-time inventory updates to Firebase and AI-powered defect detection through Python scripts. The prototype was tested extensively to ensure it met the requirements identified in the conceive phase.

The prototype team initially planned to create a 3D-printed design for the hardware assembly, as it would provide a durable and customized structure. Fusion 360 was the primary software tool used for creating the 3D models. The reason for this decision was that there were tutorials and learning materials on sites like YouTube to learn about the process and improve upon their modelling skills.

The initial 3D-printed model incorporated the modular components for a specific fit with ESP32 microcontroller, NFC scanner and other hardware. Multiple design drafts were created in Fusion 360, and was adjusted based on feedback and tests. However, challenges arose during the manufacturing process, including:

1. Difficulty to achieve precise dimensions for hardware integration.
2. Limited access to high-quality 3D printers.
3. Limited time to complete the printing.

After failing to develop a functional 3D-printed prototype, the team decided to switch to an acrylic-based design. The transition involved redesigning the existing Fusion 360 models to suit laser cutting. Acrylic was chosen for its affordability, ease of customization, and availability.

Finally, the modular concept was maintained in the final prototype design, enabling simple assembly and integration of hardware components. Adapting the designs would not have been possible without Fusion 360, and it was essential in ensuring great laser cutting precision. The design process included:

1. Creating detailed 2D schematics for laser cutting.
2. Iterative testing and revisions to refine the assembly process.

The transition from 3D printing to acrylic highlighted several key lessons including flexibility in design which is the ability to adapt and iterate was crucial to overcoming manufacturing challenges, skill development where tutorials and hands-on practice with Fusion 360 significantly enhanced the team's design capabilities. Last but not least, material consideration is important because choosing the right material for prototyping is essential for balancing cost, functionality, and manufacturability.

To provide a comprehensive overview of the design process, the following visuals will be included:

1. Conceptual Prototype Picture: The works' early ideas and drafts are showcased.
2. 3D Printing Design: The initial 3D printed components and their challenges are highlighted.
3. Final Prototype Design: Showing acrylic based modular design.

1.6.1 Conceptual Prototype

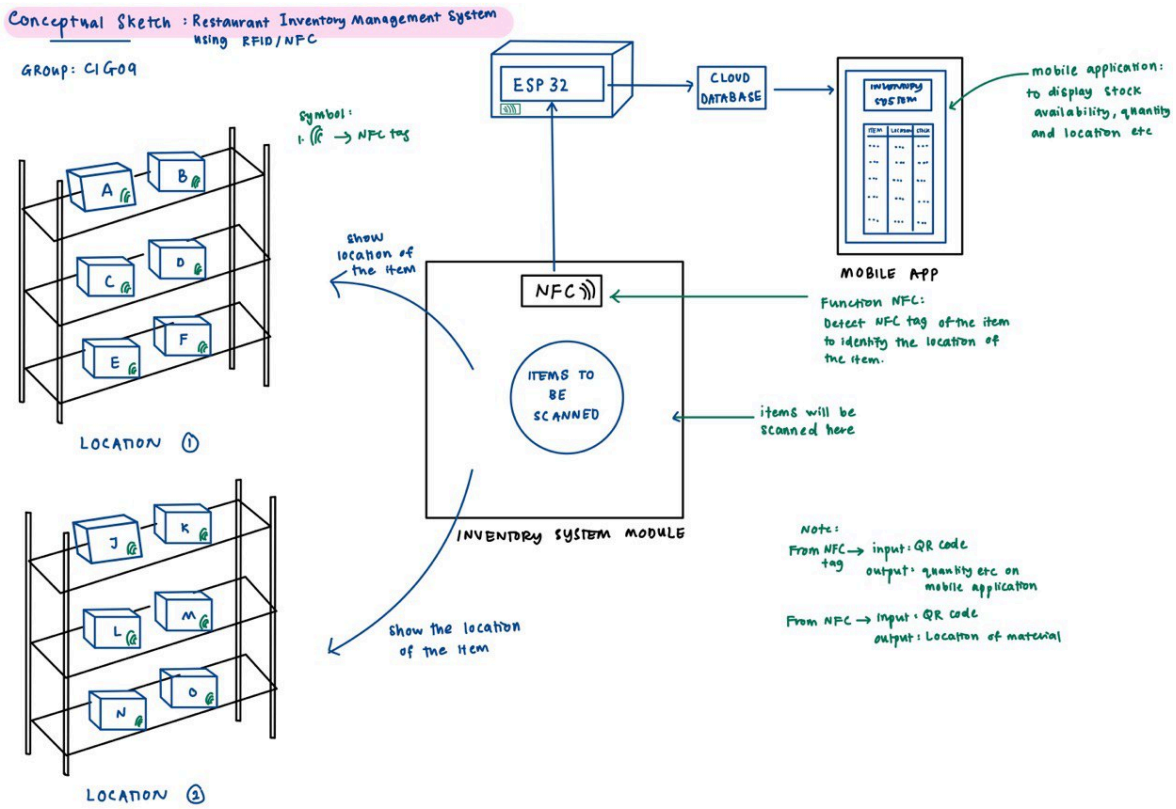

FIGURE 1.1 CONCEPTUAL PROTOTYPE OF SIMS AT THE INITIAL STAGE.

1.6.2 3D Printing Design

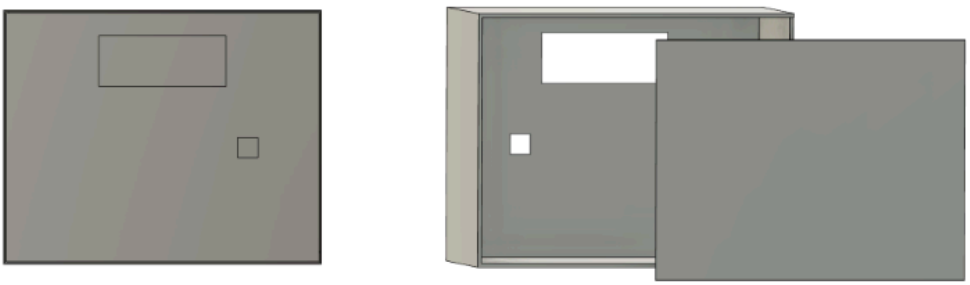

FIGURE 1.2: 3D PRINTING DESIGN ON FUSION 360.

1.6.3 Final Prototype Design

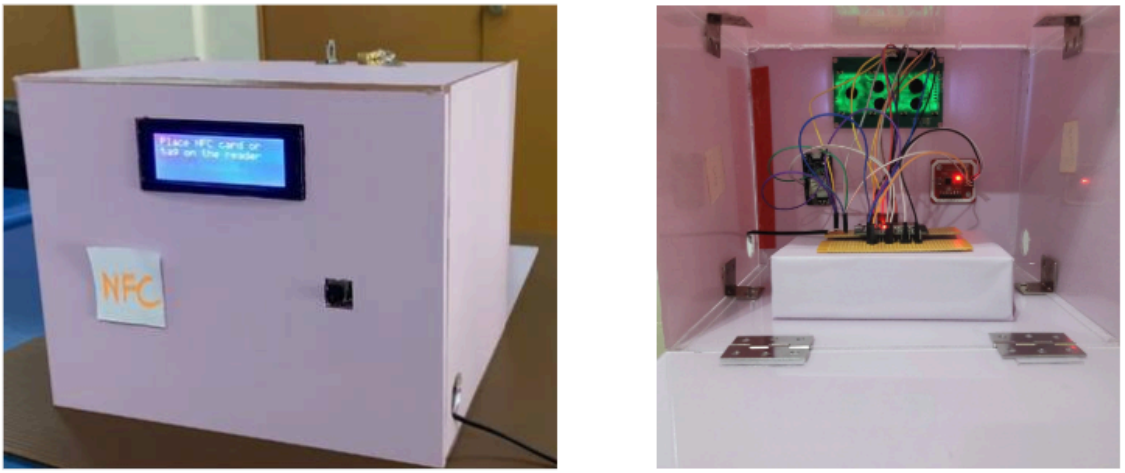

FIGURE 1.3: FINAL PROTOTYPE DESIGN FOR EECS PRESENTATION.

1.7 STEEP Analysis and SDG Requirements

1.7.1 Social Impact

The NFC-based inventory management system offers significant social benefits including improved Quality of Life for Staff: With the system, the time and energy used to count and track the inventory reduces because one only has to perform a scan to check the inventory level. The benefit of this time saving is that attention can be focused on another vital task, and therefore make work life balance. In addition, it will enhance efficiency. The solution minimizes human error so that the accuracy of the stock record improves, lessening stress and frustration as a result of inventory error. They can also build trust between team members as accurate records considerably help reinforce those relationships. Restaurants with a streamlined inventory system could turn these saved resources towards community engagement programs, like food donations, and thus improve their reputation.

1.7.2 Technological Advancements

The implementation of an NFC-based inventory system demonstrates technological innovation in which replacing manual processes with an automatic method improves processes' effectiveness and is compatible with other digital instruments.With this system, the integration of the NFC technology, the staff gets to explore the culture of adopting technological changes in the future. Besides, NFC readers and tags are compatible with most devices, they are flexible and scalable for future technological upgrades.

1.7.3 Economic Implications

The proposed solution provides various economic advantages such as cost savings. The system cuts labor costs by reducing the time needed for a manual stock taking as well as decreasing errors. The solution also helps in loss prevention. It removes the problem of overstocking and understocking and also prevents food or stock from getting spoilt hence improving financial management.

1.7.4 Environmental Impact

The inventory system supports sustainability as it helps to prevent waste and overstocking, and expiration of goods by providing the correct stock management. Also, replacing paper logs with a digital system supports the implementation of sustainable measures on the environment.

1.7.5 Political Implications

The solution supports regulatory and policy alignment such as compliance with food safety standards by managing stock with expiration dates, the system ensures adherence to food safety regulations. Alignment with industry trends matches the governmental initiatives for the enhancement of the technological innovation in the food processing industry.

The NFC-based inventory management system contributes to several SDGs and specifically SDG 9 (Industry, Innovation, and Infrastructure) where the integration of NFC technology promotes industrial innovation.

TABLE 1.1 STEEP ANALYSIS OF THE DEVELOPED PROTOTYPE

Aspect	Contribution
Sociological	Allow staff to quickly scan and check inventory levels, reducing the time spent on manual counting and tracking. The system minimises human error, improving accuracy in stock records.
Technological	Introduce an modern inventory system, replacing outdated manual processes and NFC readers and tags are compatible with the devices staff uses.
Economic	Reduces labour costs by decreasing the time needed for manual stock-taking and the NFC system minimises loss and overstocking by keeping a precise record of inventory.
Environmental	Reduces waste because with accurate tracking, the restaurant can manage inventory more efficiently, reducing food spoilage and waste. and promoting eco-friendly practices within the restaurant as the system will replace paper logs for inventory.
Political	Compliance with food safety standards, as the system helps manage stock with expiration dates. By adopting a modern NFC inventory system, the project aligns with industry trends toward automation and digital solutions.

1.8 Innovative Features

The Smart Inventory Management System (SIMs) introduces a suite of innovative features that distinguish it from traditional inventory management solutions. These features address the specific challenges faced by restaurant managers, clerks, and suppliers, offering a modern, scalable, and user-centric approach to inventory management.

One of the core innovations of SIMs is its real-time inventory tracking system powered by Near Field Communication (NFC) technology. By integrating NFC tags on each inventory item, the system enables instant and accurate data capture, including item name, serial number, location, manufacturing date, expiry date, and transaction type (e.g., check-in or check-out). This data is transmitted seamlessly via ESP32 microcontrollers, configured using Arduino IDE, to Firebase for secure storage. The real-time nature of this feature eliminates manual errors, reduces the time spent on inventory updates, and ensures that staff are always informed about stock levels.

Another groundbreaking feature is the automated defect detection system. Leveraging an ESP32 camera module, SIMs capture high-resolution images of items during check-in to assess their quality. These images are stored in a Google Drive database and processed using AI-powered Python scripts to detect potential defects.

This feature ensures that only high-quality products are entered into the inventory, minimizing waste and enhancing customer satisfaction. The use of AI-driven analysis demonstrates the project's commitment to integrating advanced technologies for improved operational efficiency.

SIMs also excels in multi-platform accessibility. Designed with a user-friendly interface, the system is available as both a web-based application and an Android APK, enabling users to manage inventory on various devices. This flexibility ensures that restaurant staff can access inventory records, update stock levels, and review quality checks anytime and anywhere. Firebase Authentication adds an additional layer of security, ensuring that only authorized users can access the system.

The integration of modular design and hardware prototyping adds another dimension to the system's innovation. Using Fusion 360, the team developed modular laser-cut acrylic components to house the hardware securely and aesthetically. This approach not only enhances the durability of the physical system but also ensures scalability, allowing additional components to be incorporated as needed.

SIMs incorporates automated transaction updates through Google Apps Script, which dynamically processes transaction types in Google Sheets. This feature reduces the burden on staff by automating routine administrative tasks, further streamlining operations. The system's ability to adapt to real-time updates and process inventory data efficiently makes it a comprehensive solution for modern inventory management.

Furthermore, SIMs is built with a strong focus on scalability and adaptability. The system is designed to handle expanding inventory sizes and support additional features, such as predictive analytics for stock forecasting. By combining IoT, AI, and cloud computing technologies, SIMs ensures that it can grow with the needs of its users, providing long-term value.

These innovations collectively position SIMs as a transformative tool in restaurant inventory management. By addressing critical pain points such as real-time tracking, quality assurance, and cross-platform accessibility, SIMs redefines how inventory is managed, contributing to the advancement of Sustainable Development Goal 9 (Industry, Innovation, and Infrastructure) through technological innovation and resilient system design. This holistic approach ensures that SIMs is not only a functional system but also a forward-looking solution that meets the demands of the modern restaurant industry.

1.9 Any Other Related Data / Activities

In addition to the primary activities outlined in the previous sections, several supplementary tasks and data collection efforts contributed to the development of the Smart Inventory Management System (SIMs). These activities provided critical insights and ensured the comprehensiveness and robustness of the solution.

One of the additional efforts involved the creation and refinement of a survey questionnaire distributed via Google Forms to restaurant staff and customers. This questionnaire was designed to capture nuanced perspectives on the challenges faced in inventory management, focusing on issues such as manual errors, time inefficiencies, and quality assurance. The survey also gathered user feedback on the proposed features of SIMs, such as NFC-enabled tracking and AI-powered defect detection. The responses informed design decisions, ensuring that the system addressed real-world pain points effectively.

Another significant activity was the collaborative refinement of hardware configurations. Team members conducted extensive testing on the ESP32 microcontroller and ESP32 camera to optimize their performance and ensure seamless integration with NFC tags and Firebase. Debugging sessions using Arduino IDE played a crucial role in identifying and resolving issues related to hardware-software interfacing, particularly in real-time data transmission and image capture.

Prototype evaluation and iteration were also key activities during the project. Using Fusion 360, the team designed and fabricated laser-cut acrylic enclosures for the hardware. These enclosures underwent multiple iterations to ensure durability, modularity, and ease of assembly. Feedback from prototype testing was used to refine the design, aligning it with the operational needs of restaurant environments.

Additionally, the team developed and tested automated scripts using Google Apps Script to dynamically process inventory data in Google Sheets. These scripts were fine-tuned to handle various scenarios, such as discrepancies in transaction records and synchronization issues. This automation minimized manual effort and enhanced the accuracy of inventory tracking.

A time-motion study was conducted to compare the efficiency of SIMs against traditional inventory management methods. The study revealed significant improvements in processing time, error reduction, and overall operational efficiency. These findings were critical in demonstrating the value proposition of the system to stakeholders.

Lastly, the team engaged in stakeholder presentations and discussions, including interactions with restaurant owners and staff, to validate the system's features and gather additional feedback. These sessions helped identify potential areas for future enhancements, such as integrating predictive analytics for stock forecasting and expanding compatibility with third-party restaurant management software.

These supplementary activities ensured a holistic approach to the design and development of SIMs, reinforcing its ability to address the diverse challenges of inventory management in the restaurant industry while aligning with the innovation goals of Sustainable Development Goal 9 (Industry, Innovation, and Infrastructure).

2. Design

The design phase of this project focuses on translating the conceptual framework developed during the Conceive phase into tangible, functional elements. This stage involves the detailed planning and creation of components directly relevant to the Smart Inventory Management System (SIMs). The design process incorporates both hardware and software elements, ensuring seamless integration between physical devices, backend infrastructure, and user-facing interfaces.

Key design elements include hardware schematics for the NFC-enabled ESP32 microcontroller and ESP32 camera module, software flowcharts detailing user interaction and data processing, and the physical prototype modeled using Fusion 360 for laser-cut acrylic components. Additionally, the user interface was designed to be intuitive and accessible, leveraging modern web technologies such as HTML, CSS, and JavaScript, while maintaining compatibility with Android through APK deployment.

Throughout this phase, diagrams, charts, and drawings were created using specialized tools to ensure precision and clarity. These visual aids not only facilitated team collaboration but also ensured that stakeholders could easily understand and evaluate the design's feasibility. Each design element was iteratively refined based on user feedback, hardware testing, and system integration requirements, ensuring alignment with project objectives and the innovation goals of Sustainable Development Goal 9 (SDG 9).

The following subsections detail the specific designs created during this phase, accompanied by explanations of their relevance, functionality, and development tools used. These designs represent the culmination of collaborative efforts to create a robust, scalable, and user-centric inventory management solution.

2.1 Flowchart for Coding

The flowchart for the coding process of the Smart Inventory Management System (SIMs) outlines the logical flow of user interactions, backend operations, and output generation, focusing on app development. This section provides a detailed explanation of the steps illustrated in the flowchart, emphasizing the modularity and efficiency of the code structure. The coding process was implemented primarily using HTML, CSS, and JavaScript for the front-end and Firebase for the backend.

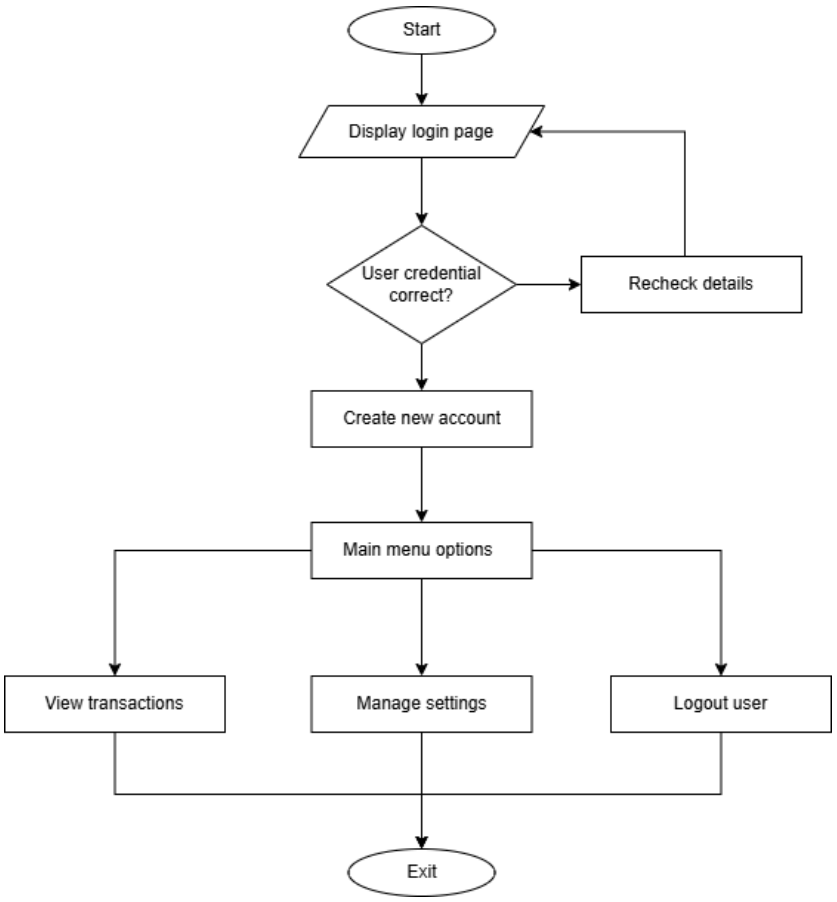

FIGURE 2.1: FLOWCHART FOR MOBILE APP DESIGN.

On the other hand, this flowchart explains the logical flow of the Google Apps Script used to update transaction details dynamically in a Google Sheet. The script automates the process of identifying and categorizing inventory transactions based on serial numbers, ensuring real-time updates for better inventory tracking.

The Python script is used for defect detection of inventory boxes, leveraging a MobileNetV2 model pre-trained on the ImageNet dataset, with additional custom layers added for binary classification. The model predicts whether an item is Good or Defective based on images captured during inventory check-ins. The script is divided into several logical stages, which are explained in detail below.

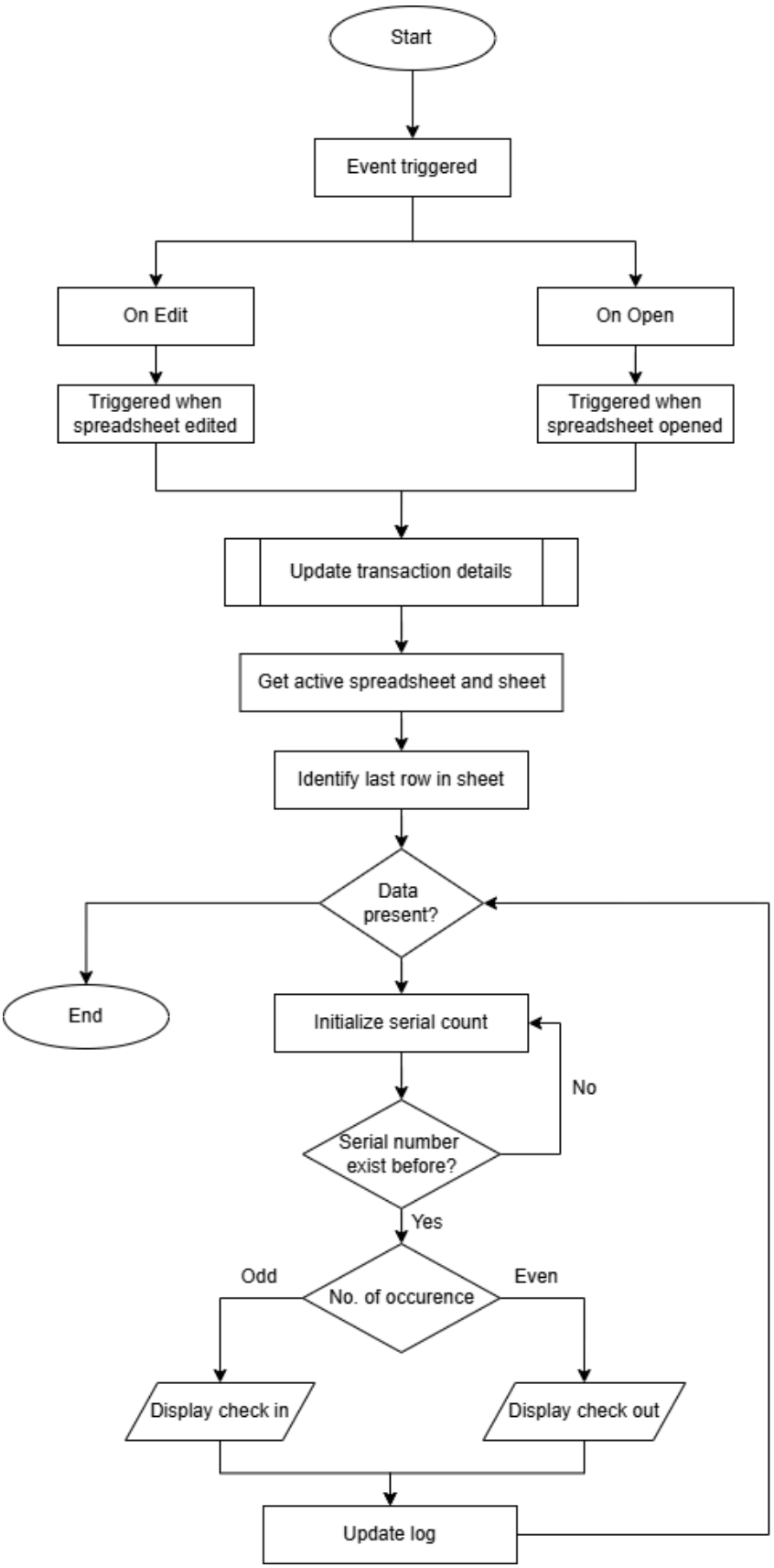

FIGURE 2.2: FLOWCHART FOR DATABASE TRANSACTION IDENTIFICATION.

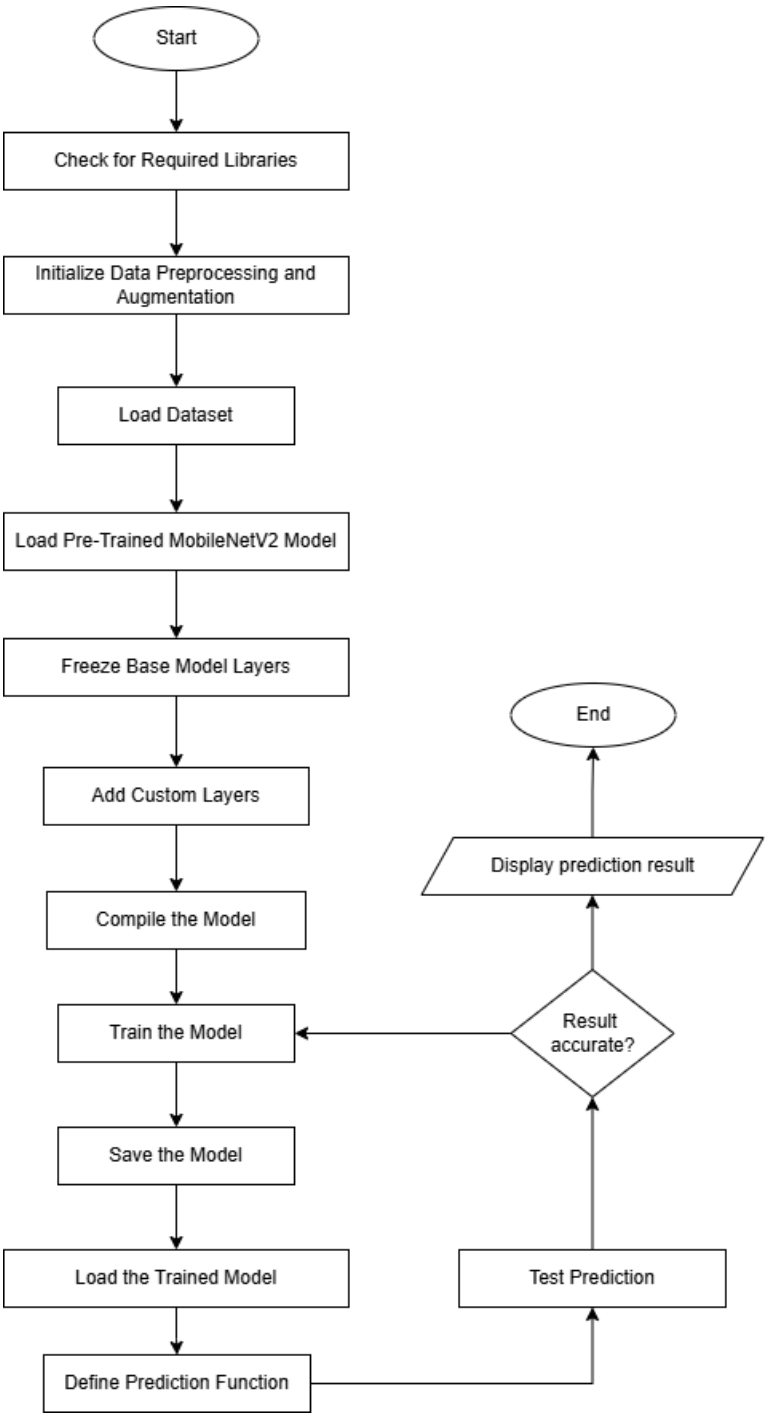

FIGURE 2.3: FLOWCHART FOR INVENTORY DEFECTS DETECTION USING MACHINE LEARNING.

2.2 Block Diagram / System Architecture

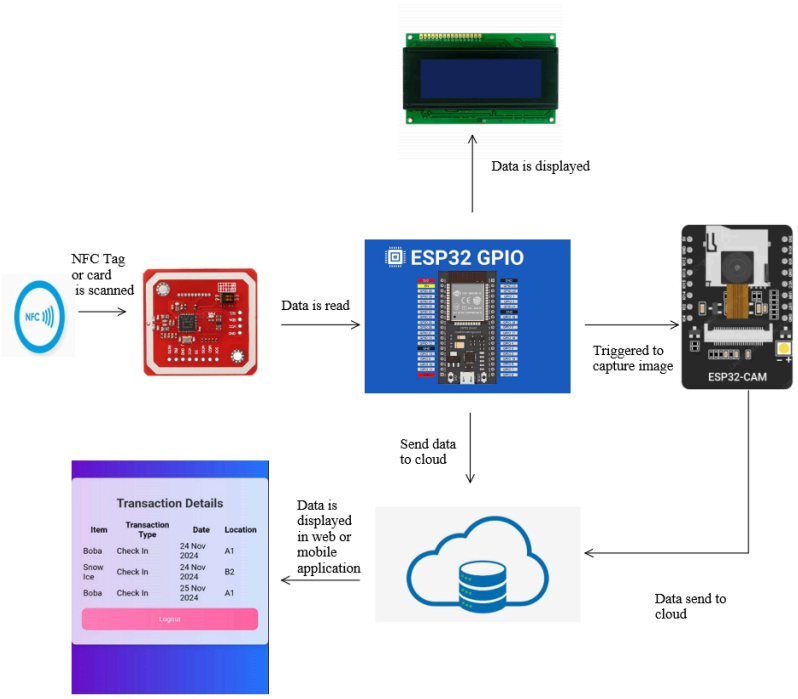

FIGURE 2.4: BLOCK DIAGRAM FOR THE RESTAURANT INVENTORY AND MANAGEMENT SYSTEM USING NFC.

Initially, the NFC tag or Mifare card stored the required data such as items, location, manufactured date and expiry date. The tag or card is scanned using PN532 module, a NFC reader to read all the data which is then extracted by the ESP32. The I2C communication protocol, a synchronous, multi-master, multi-slave, packet-switched protocol ideal for short-distance communication is used between the ESP32 and PN532 modules.

The ESP32 will send the data read to Google Sheets via API for storage. Other than the data in the tag or card, the time the tag or card is scanned and transaction status will also be stored in the Google Sheets. Besides, the information about the item and its location will be displayed in the LCD.

After scanning, the ESP32 will trigger the ESP32 CAM to capture the image of the item. UART communication is a common protocol used for asynchronous communication, providing a simple way to transfer data over serial connection is used for ESP32 and ESP32 CAM. If the image is failed to capture, the ESP32 CAM will notify the ESP32 and ESP32 will tell the ESP32 CAM to capture the image again. The image captured is sent to Google Drive via Google App Script where the time captured for the image is displayed to match with the data in the Google Sheets. All the data in the cloud storage can be displayed in a web interface or mobile application.

2.3 Electrical Circuit Schematic

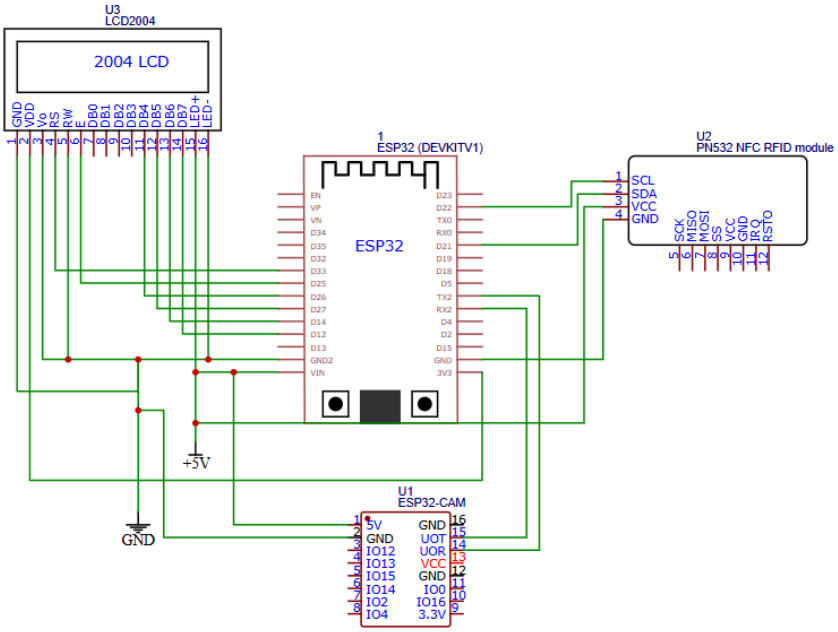

FIGURE 2.5: SCHEMATIC DIAGRAM OF ESP-32 MICROCONTROLLER CONNECTION.

- ESP32 CAM: UART communication
  - Transmitter, U0T connects to the receiver, RX2 of ESP32
  - Receiver, U0R connects to the transmitter, TX2 of ESP32
  - VCC connects to 5V
  - GND connects to ground
- PN532: I2C protocol
  - SDA connects to pin 21 of ESP32
  - SCL connects to pin 22 of PN532
  - VCC connects to 5V
  - GND connects to ground
- LCD: 4-bit data transmission
  - D4 connects to D12 of ESP32
  - D5 connects to D14 of ESP32
  - D6 connects to D26 of ESP32
  - D7 connects to D27 of ESP32
  - RS (Register Select) connects to D25 of ESP32
  - E (Enable) connects to D33
  - BLA connects to 5V
  - VSS, V0, RW and BLK connect to ground
  - VDD connects to 3.3V of ESP32

2.4 Printed Circuit Board (PCB)/Donut/Strip Board Layout

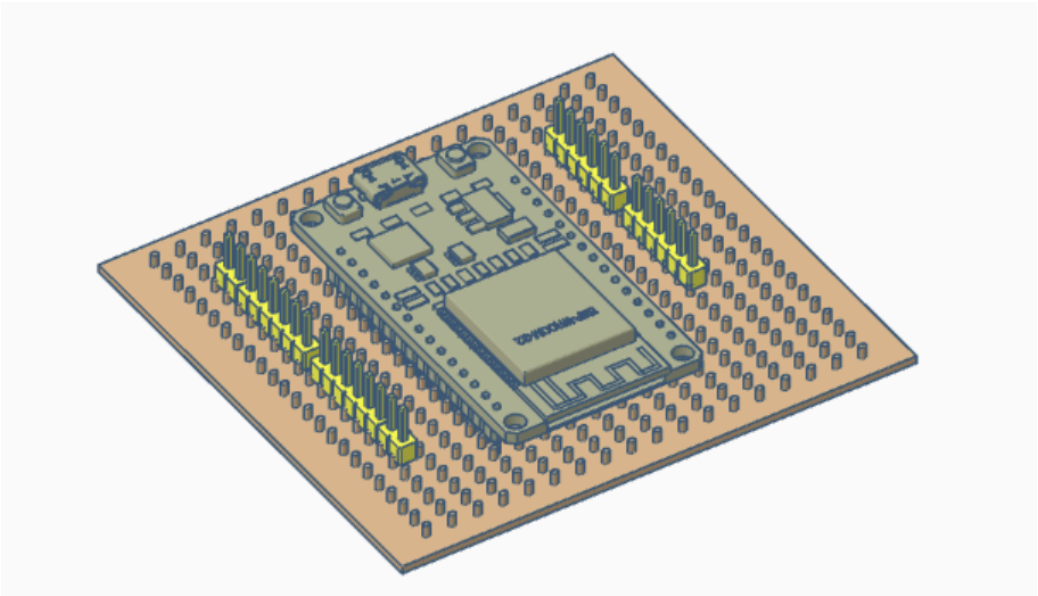

FIGURE 2.6: STRIPBOARD LAYOUT TO MOUNT ESP32.

The ESP32 is mounted on the stripboard with each of its pins soldered to the individual copper tracks. This is to ensure that each track only contains a connection to a single ESP32 pin to avoid any unintended short circuits or interference of the adjacent pins. A male header is placed in series with the corresponding pin of the ESP32, allowing for easy connections to external components. In addition to this, a USB power cable with two wires, 5V and ground is soldered onto the stripboard to supply power to the ESP32 and other external components if needed. This layout ensures a well organized power distribution and great connectivity of all pins.

2.5 Mechanical Design Drawing

The mechanical design of the Smart Inventory Management System (SIMs) provides the structural framework necessary to house its operational components while ensuring functionality, durability, and aesthetic appeal. The framework is constructed using robust materials designed to withstand environmental conditions and mechanical stress, with modular components allowing scalability and ease of maintenance. The system dimensions are carefully calculated to balance spatial efficiency and user ergonomics, with interaction points such as touchscreens and storage compartments positioned at optimal heights and angles for convenience. Strategic mounting points are provided for sensors, including RFID, weight, and proximity sensors, ensuring accurate data collection without interference. Detailed assembly instructions accompany the design, with clearly labeled components, step-by-step guides, and a comprehensive list of required tools and fasteners. Aesthetic considerations include smooth finishes, modern design elements, and internal pathways for cable management to maintain a clean and professional appearance. The design also incorporates protective enclosures for embedded sensors, dedicated spaces for power systems, and ventilation slots to ensure effective integration with electronic components. Safety measures, such as rounded edges, load-bearing capacity, and fire-resistant materials, ensure compliance with industry standards. The result is a durable, scalable, and user-friendly design that meets the functional and aesthetic requirements of the project while supporting the efficient operation of the inventory management system.

2.6 Machine Interface (HMI) Design

The Human–Machine Interface (HMI) for the Smart Inventory Management System (SIMs) has been meticulously designed to ensure an intuitive, efficient, and user-friendly experience, prioritizing accessibility and seamless interaction for all users. Central to the HMI design are the principles of clarity, responsiveness, and consistency, which have been implemented through careful consideration of user needs and operational contexts in the restaurant industry. The interface enables users to navigate critical functionalities effortlessly while maintaining a clean, professional aesthetic that aligns with modern usability standards.

The interface begins with the Login and Sign-Up screens, serving as the system's gateway. These screens provide clear, minimalistic input fields for user credentials and incorporate validation mechanisms to guide users

through account creation or login processes. Upon successful login, users are directed to the Main Menu screen, a centralized navigation hub offering access to core functionalities such as transaction history and profile management. The design employs a straightforward layout, ensuring that users can quickly locate and access desired features without cognitive overload.

The Transaction History screen provides a dynamic, real-time display of inventory data, including item names, serial numbers, transaction types, dates, and storage locations. This data is fetched directly from Firebase and presented in an organized, tabular format, allowing users to sort and filter information as needed. Additionally, the Settings screen enables users to update personal profiles, including their username, password, and role, with user-friendly prompts and real-time feedback to confirm successful changes.

To ensure accessibility across multiple devices, the interface is designed for cross-platform compatibility, available as both a web-based application and an Android APK. This dual functionality allows users to manage inventory seamlessly on mobile and desktop devices, with responsive layouts adapting to varying screen sizes and resolutions. The interface's usability is further enhanced by employing visual cues, real-time feedback mechanisms, and error handling, ensuring users are guided effectively through their interactions with the system.

The HMI's development leveraged a combination of tools and frameworks, including HTML, CSS, and JavaScript, to build a responsive and interactive interface. Figma was employed during the prototyping phase to create wireframes and refine the visual hierarchy, while Firebase facilitated secure authentication and real-time data synchronization. Adhering to a mobile-first design philosophy, the interface ensures optimal usability for restaurant staff who often rely on mobile devices for operational tasks.

Overall, the HMI design exemplifies a harmonious blend of functionality and aesthetic simplicity, aligning with user-centric design principles and the operational demands of the restaurant industry. By addressing usability challenges and promoting seamless integration of hardware and software components, the HMI contributes significantly to the system's overall effectiveness, ensuring that users can manage inventory with ease and precision.

2.7 System Flow Chart

The system flow chart for the Smart Inventory Management System (SIMs) illustrates the comprehensive sequence of operations, seamlessly integrating user interactions, hardware processes, and backend functionalities. The process begins with user authentication through Firebase, where credentials are verified via the Login or Sign-Up screens. Upon successful authentication, users gain access to the system's main functionalities, including transaction management and profile updates. Inventory data is captured using NFC-enabled ESP32 microcontrollers, transmitting item details such as location, expiry date, manufacturing date, transaction type, and serial number to Firebase via a secure API. During check-in, an ESP32 camera module captures high-resolution images of items, which are analyzed by AI-driven Python scripts to detect potential defects. Processed data is stored in Google Sheets for real-time updates and validation through automated Google Apps Scripts, dynamically categorizing transactions as "Check In" or "Check Out." Users can view inventory details on a responsive, cross-platform interface, enabling seamless management across web and mobile devices. The system incorporates feedback loops, allowing users to update profiles or correct discrepancies, ensuring data integrity and operational efficiency. This streamlined workflow reflects a harmonious integration of hardware, software, and cloud technologies, enabling robust, real-time inventory management.

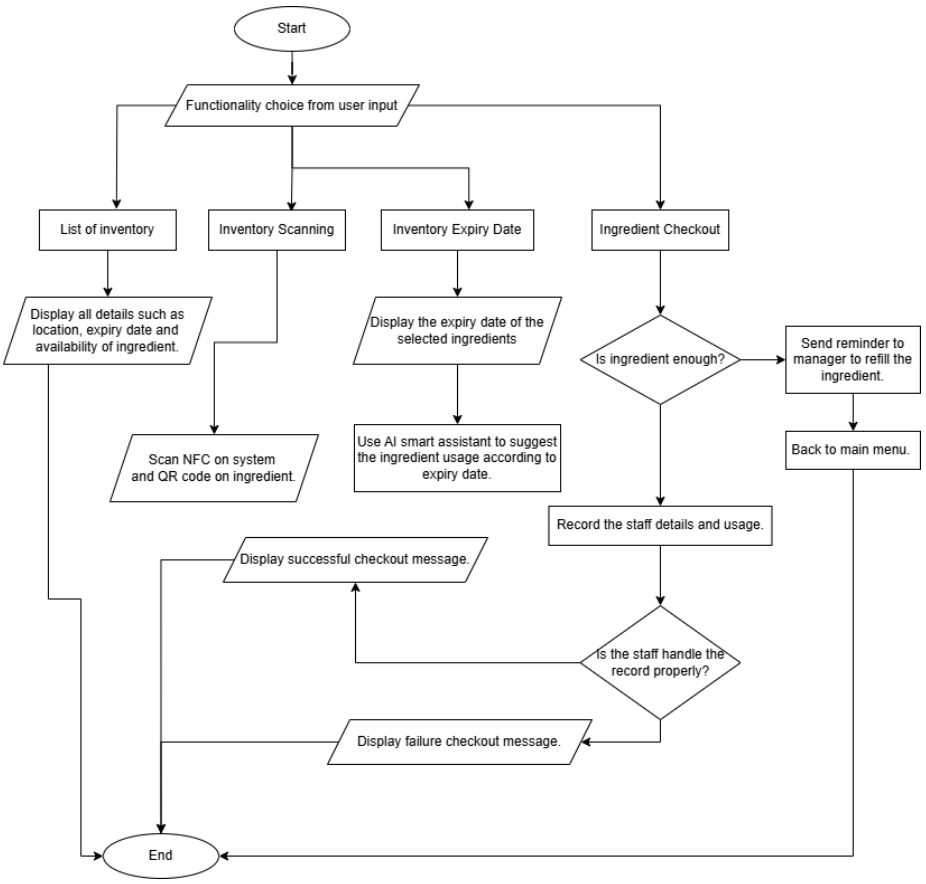

FIGURE 2.7: OVERALL SYSTEM FLOWCHART OF SIMS.

3. IMPLEMENTING

The implementing phase focuses on transforming the conceptual framework of the Smart Inventory Management System (SIMs) into a fully functional prototype. This stage marks the culmination of design, planning, and development efforts, bringing together hardware configurations, software integration, and physical fabrication to achieve a robust and scalable solution. The implementation process encompasses the creation of a responsive Human-Machine Interface (HMI), the integration of ESP32 microcontrollers for NFC and image capture, the development of custom scripts for inventory tracking and defect detection, and the construction of an acrylic enclosure to house the hardware components securely.

Each step in this phase required a methodical approach, leveraging advanced tools and techniques to ensure the system's performance and reliability. The HMI design prioritized user experience, offering a seamless interface for managing inventory across multiple platforms. Circuit implementation involved configuring the ESP32 microcontroller using Arduino IDE for efficient communication with Firebase and Google Sheets. The enclosure fabrication utilized Fusion 360 to design modular laser-cut components, ensuring durability and aesthetic appeal. Lastly, the coding efforts combined front-end development using HTML, CSS, and JavaScript with back-end integration via Firebase, enabling real-time data synchronization and AI-powered defect analysis.

This phase highlights the collaborative effort of the team, with each member contributing to specific tasks such as hardware setup, software development, and prototype design. The implementation process not only validated the project's feasibility but also ensured alignment with the operational requirements of restaurant inventory management. The following subsections detail the individual components of the implementation phase, elucidating the tools, methods, and challenges encountered during this critical stage.

3.1 Human–Machine Interface (HMI)

The Human–Machine Interface (HMI) serves as the primary point of interaction between users and the Smart Inventory Management System (SIMs), enabling seamless control and management of inventory operations. Designed with user-friendliness as a top priority, the HMI ensures that restaurant staff, regardless of their technical expertise, can effectively interact with the system to access real-time inventory data, manage profiles, and monitor transaction histories. Its purpose extends beyond functionality, focusing on providing an intuitive, accessible, and efficient platform to streamline inventory management tasks.

Design considerations were centered on simplicity and clarity, ensuring that users could easily navigate through the interface without confusion. The HMI's layout was developed using HTML, CSS, and JavaScript, offering a clean and responsive design optimized for both desktop and mobile devices. To accommodate a wide range of users, the interface adhered to accessibility standards, incorporating high-contrast color schemes, legible fonts, and touch-friendly controls for mobile users. Key functionalities, such as login, sign-up, viewing transactions, and managing profiles, were placed prominently to reduce cognitive load and improve task efficiency.

The design process was supported by tools such as Figma, which facilitated the creation of wireframes and mockups during the early stages of development. These prototypes were iteratively refined based on feedback gathered during user testing sessions. Testing involved restaurant staff and potential end-users who provided valuable insights into navigation flow, button placement, and overall usability. Issues identified during testing, such as overly complex workflows or unclear labels, were addressed promptly, resulting in a polished and intuitive interface.

To further enhance usability, the HMI was developed to function as a cross-platform solution. It is accessible both as a web-based application and as an Android APK, ensuring flexibility for users to manage inventory on their preferred devices. Firebase's real-time database integration allows for instantaneous updates and data synchronization, providing a seamless experience across platforms.

By prioritizing user experience and leveraging robust design principles, the HMI stands as a vital component of SIMs, bridging the gap between advanced technological features and end-user accessibility. This careful balance of functionality and simplicity ensures that the interface meets the operational demands of restaurant inventory management while remaining approachable and efficient for its users.

3.2 Circuit Implementation

In this project, stripboard is used for the circuit connection due to its ease of assembly. The ESP32, male connector headers and USB power supply are soldered to the stripboard, while the LCD, PN532 and ESP32 CAM are connected to the ESP32 through jumper wire. The design maximizes the stripboard layout, where the ESP32 is mounted on the stripboard and the male headers are in series with the pins of the ESP32. Power will be supplied via a USB connection with a 5V output.

In the beginning, a schematic is created using Tinkercad to design the components layout of the stripboard. The stripboard is cut into a smaller size and the copper traces are cut to isolate the circuit that wants to be separated such that the pins of the ESP32 will not share the same node. The ESP32 pins, male header connectors and USB power supply are soldered on the stripboard. After assembly, the circuit is tested with a multimeter, ensuring that there are no shorts circuits or bad soldering. The LCD, PN532 and ESP32 CAM are connected using female to female jumper wires to the headers which are in series with the PN532 pins. This approach is flexible for future modification and troubleshooting.

3.3 Enclosure or Casing Implementation

The enclosure or casing of the inventory system prototype serves the main purpose in protecting the internal components as well as ensuring the device had a professional look. Basically, its purpose is to protect certain delicate hardware: the ESP32 microcontroller, the NFC scanner and the camera from various environmental hazards such as dust, moisture, physical impact, etc. It also makes the overall aesthetic of the prototype much more presentable and user friendly.

Laser cut acrylic sheets were chosen as the primary material for enclosure for the final prototype. Due to its durability, lightweight properties and clean, modern look, acrylic was selected. These characteristics made it

perfect for making a nice looking yet sturdy casing. The team initially explored 3D printing the enclosure components using Fusion 360 as the design tool but due to limited access to high quality 3D printers and structural weakness in the printed parts, the team decided to pivot on to acrylic as a more reliable and practical solution.

Several steps were required in the process of fabrication for the enclosure. For the first step, the team modeled the casing in Fusion 360, with the detail in the design of the casing in the 3D models and related 2D schematics matching the hardware dimensions and assembly needs. In this phase a number of drafts and iterations were used in order that the design should meet the functional and the aesthetic standards. After finalizing the designs, they were exported to laser cut compatible software for the acrylic material that was chosen.

With the acrylic sheets, we measured their size and cut them out to precision with the laser cutter. The design was able to integrate the openings for the NFC scanner, power supply, and display screen without interrupting the main flow of the assembly line. The cut acrylic components were then polished and wrapped with wallpaper stickers for a professional finish before being assembled using adhesives and screws.

Finally, the final enclosure reached the balance between functionality and aesthetics and proved to effectively protect the inner components and facilitate the use and aesthetic appearance of the prototype. Laser cut acrylic was found to be a practical and successful solution, overcoming problems with 3D printing on the prototype.

3.4 Connector and Wires / Pipes / Tubes Selection

The selection of connectors and wires is a critical aspect of the system design to ensure functionality, durability, and overall performance. The process began by identifying specific requirements such as the size, type, and purpose of each connection. For electrical connections, male header connectors and jumper wires were selected due to their ability to efficiently transmit signals while maintaining electrical integrity. To simplify the power supply system, a USB power connector with two wires—power supply and ground—was chosen to eliminate unnecessary signal-transmitting wires not essential to the system's operation. Flexibility was a key consideration, especially for components requiring frequent movement or adjustments. Male header connectors offered the advantage of easy soldering and desoldering, enabling repositioning when needed. Jumper wires were selected for their ability to connect to any pin on the components, allowing for easy reconfiguration without compromising connectivity. Additionally, the length of the jumper wires was carefully chosen to ensure sufficient distance between components while maintaining a compact design. The final selection of connectors and wires was tailored to the specific needs of the project, and proper alignment and installation techniques were applied to minimize signal loss and maintain stable electrical connections throughout the system.

3.5 Wires / Pipes / Tubes Routing

Efficient and organized routing of wires is essential to ensure the safety and optimal performance of the operating system. A well-planned routing strategy minimizes the risks of tangling, signal interference, and potential damage caused by accidental pulling or improper handling of wires. The pathways for wires were carefully planned at the initial stages, taking into account potential sources of interference and the flexibility or movement required for the wires during system operation. Cable ties were used to bundle and organize the wiring effectively, ensuring that the layout allowed sufficient space for future expansion or reconfiguration. Special care was taken to avoid routing wires too tightly, as this could cause strain, disconnections, or damage to the wires over time. Rigorous testing was conducted to verify that the routing design did not interfere with the overall performance of the system. Electrical tests ensured there was no signal degradation or interference caused by the physical layout, while safety checks confirmed there was no risk of short circuits due to improper wire placement. This organized and tested routing design ultimately improved the system's efficiency, reliability, and safety.

3.6 Mechanical Implementation

The mechanical implementation of the system focuses on the integration and assembly of structural components to support the functionality and durability of the prototype. The key mechanical elements include frames, panels, fasteners, and joints, which were carefully selected and assembled to ensure stability and

precision. The assembly process began with constructing the main framework using durable materials, ensuring that the design could support all electronic components and maintain structural integrity under operational conditions. Precise measurements and alignments were critical during the assembly to ensure proper placement and integration of all parts. Tools such as calipers, screwdrivers, and clamps were used to achieve accurate positioning and secure attachment of the components. Design software like Fusion360 was utilized during the planning stage to simulate the mechanical layout and test its compatibility with the electronic system. Proper pathways and mounting points were incorporated for wiring and electronic integration, ensuring a seamless connection between the mechanical and electronic systems. The final assembly process prioritized robustness and ease of access for maintenance, creating a reliable and efficient prototype tailored to the project's requirements.

FIGURE 3.1: JOINT OF THE PROTOTYPE.

3.7 Coding Implementation

The coding implementation for the Smart Inventory Management System (SIMs) forms the backbone of the project, seamlessly integrating hardware control, data processing, and human-machine interaction. The system's code was meticulously designed to ensure efficient operation across multiple components, including NFC-enabled inventory tracking, image capture for defect detection, and real-time data synchronization. The development leveraged a combination of programming languages—Python, C++ (Arduino), and JavaScript—each chosen for its specific strengths and suitability for the respective system components.

The Arduino IDE, using C++, was employed to program the ESP32 microcontroller for controlling hardware functionalities. The code enabled NFC communication for reading inventory data such as item name, location, expiry date, and serial number. Additionally, the ESP32 camera module was configured to capture high-resolution images during check-ins. These images were stored in Google Drive and later processed by an AI-powered Python script for defect detection. The use of Arduino provided an accessible and robust framework for interfacing with hardware components while maintaining low-level control over microcontroller operations.

On the software side, Python played a pivotal role in processing captured images and implementing defect detection algorithms. A MobileNetV2 deep learning model, fine-tuned for binary classification, was employed to categorize items as "Good" or "Defective." The Python script utilized TensorFlow to preprocess data, train the model, and evaluate its performance. Its extensive libraries and support for machine learning tasks made Python an ideal choice for handling the AI component of the project.

The Human-Machine Interface (HMI) was developed using HTML, CSS, and JavaScript to provide a responsive and interactive user experience. JavaScript's integration with Firebase enabled real-time database synchronization, ensuring that inventory data remained up-to-date and accessible across devices. The HMI facilitated user actions such as logging in, signing up, viewing inventory transactions, and managing profiles. It was designed to function seamlessly as a web-based platform and as an Android APK, enhancing accessibility for users.

The codebase was modularly structured to separate hardware control, data processing, and user interface logic, ensuring scalability and ease of maintenance. For example, Firebase functions were isolated from UI elements, and Python scripts for defect detection operated independently from the ESP32 code. This approach enabled parallel development and minimized interdependencies, streamlining debugging and testing.

Challenges during development included ensuring reliable communication between the ESP32 microcontroller and Firebase, optimizing image processing speed for defect detection, and achieving a consistent user experience across platforms. These challenges were addressed through iterative testing, optimization of network configurations, and refinements to the user interface.

Overall, the coding implementation was a cohesive effort, leveraging the strengths of multiple programming languages to create a system that is efficient, scalable, and user-friendly. This robust implementation ensures that SIMs can reliably perform its intended functions, addressing critical challenges in restaurant inventory management.

4. Operating

The operating phase of the Smart Inventory Management System (SIMs) focuses on testing the prototype under diverse conditions to evaluate its performance, reliability, and practical usability. This stage involved rigorous in-laboratory testing to validate hardware and software integration, followed by on-site evaluations at a restaurant setting to assess real-world functionality. The goal was to ensure that the system met its intended objectives, such as real-time inventory tracking, defect detection accuracy, and seamless user interaction, while identifying areas for refinement.

Performance metrics such as speed, accuracy, durability, and efficiency were prioritized during testing. The system's ability to process NFC-tagged inventory data and update real-time records was monitored under varying loads. The defect detection algorithm, implemented through an AI-powered Python script, was evaluated for its precision and responsiveness in identifying defective items. Additional testing focused on the HMI's usability across platforms, ensuring that the interface remained consistent and intuitive for both web-based and Android users.

Based on testing outcomes, iterative improvements were made to enhance system efficiency and address identified issues. For instance, network optimization improved communication between the ESP32 microcontroller and Firebase, while adjustments to the defect detection model increased its classification accuracy. On-site evaluations provided valuable feedback from restaurant staff, which informed refinements to the user interface and system workflows.

This phase concludes with a comprehensive analysis of the results, supported by quantitative and qualitative data. The findings demonstrate the system's robustness, scalability, and effectiveness in addressing key challenges in restaurant inventory management, aligning with the project's objectives and contributing to the advancement of Sustainable Development Goal 9 (SDG 9). The following subsections detail the specific testing procedures, performance assessments, and iterative improvements undertaken during this critical phase.

4.1 In-laboratory Testing

In-laboratory testing was performed to evaluate the performance of the smart inventory system which using PN532 for reading NTag and Mifare cards, ESP32 fr sending data to Google Sheets and ESP32 CAM for capturing images and uploading them to Google Drive.The purpose of this testing phase was to ensure that all components functioned as expected and effective communication between them.

The tests were done to measure the speed and accuracy of the data read by PN532, data transmission of ESP32 and images captured and uploaded by ESP32 CAM. These tests were conducted without any external disturbances such as signal interference or lightning. The testing results were great as PN532 successfully read data from NTag and Mifare cards, ESP32 can transmit data to Google Sheets, ESP32 CAM was able to capture and send images to Google Drive.

However, there were some limitations such as slight delays of data transmission when multiple requests were sent in quick succession. Besides, the NTag requires an optimum reading distance so that there will be no missing data when read by PN532.

Generally, the in-laboratory testing provided valuable insight into the individual performance for every single component of the system, ensuring that they functioned properly.

TABLE 4.1: In-laboratory testing result.

Metric	**Test Description**	**Expected Performance**	**In-laboratory Performance**
Speed	Time needed for data transmission	<1 second	2.7 seconds
Accuracy	Data extracted from the Ntag or Mifare card	100%	100%
Others	Image captured quality	High quality	High quality

4.2 On-site or Testbed Testing

On-site testing was conducted to evaluate the smart inventory system's performance in a real world environment. The full functionality of the prototype was tested where all the components interact with each other. Testing was carried out in a real environment with real world factors such as lightning and network connectivity.

On-site testing was necessary to determine how well the system performed in a real world problem such as WiFi signal strength, varying light levels and user handling of the NFC cards. During the test, PN532 faced difficulty in reading NTag and Mifare cards when an obstacle was placed between PN532 and NFC cards. The position of the cards were required to be placed properly so that the data read was correct. For ESP32, the Wi-Fi strength was affecting the data transmission rate causing higher delay. This was due to the prototype being further away from the WiFi module and the WiFi module was also connected with many devices. On the other hand, the images captured by ESP32 CAM were affected by the light intensity. The image quality was reduced in low light environments.

In conclusion, the on-site testing revealed valuable insights of the system's behavior in a real world scenario. These findings emphasize the importance of optimizing the system to handle environmental factors for consistent high performance.

TABLE 4.2: TESTBED TESTING RESULT.

Metric	**Test Description**	**Calculated / Required Performance**	**In-laboratory Performance**	**Actual Performance**
Speed	Time needed for data transmission	<1 seconds	2.7 seconds	3.2 seconds
Accuracy	Data extracted from the Ntag or Mifare card	100%	100%	78%
Others	Image captured quality	High quality	High quality	Low quality

4.3 Improvement with Final Testing

Based on the in-laboratory and on-site testing, several improvements were made for the prototype to address the challenges observed and improve overall performance. These improvements focused on optimizing both hardware and software aspects of the system. Improvements made:

1. NFC data read:
    - Issue: Missed readings of data when card is not positioned correctly.
    - Improvement: The code was updated to implement a retry mechanism to read the NFC card until accurate data was captured.
    - Result: 100% successful rate for the data capture.
2. Data transmission
    - Issue: There was a long delay for data transmitted to Google Sheets.
    - Improvement: Using mobile hotspot and sending the data into large chunks.
    - Result: The latency of data transmission was reduced.
3. Image captured quality
    - Issue: In real world testing, image quality varied depending on the lightning conditions, resulting in blurry images.
    - Improvement: The ESP32 CAM built in flash light was activated to increase the light levels during capturing images.
    - Result: Image quality improved significantly.

TABLE 4.3: FINAL TESTING RESULT.

Metric	**Pre-Improvement**	**Post-Improvement**	**Improvement (%)**
Speed	3.2 seconds	1.9 seconds	40.6%
Accuracy	78%	100%	28.2%
Others	Low quality	High quality	-

4.4 Result Analysis

The data collected from the in-laboratory and on-site testing provided valuable insights into the smart inventory system's performance. During in-laboratory testing, the system operated well with PN532 reading NFC cards' data accurately, ESP32 successfully transmitting data to Google Sheets but a slight delay and ESP32 CAM capturing high quality images in optimal lighting. The data transmission latency might be caused by the

data being sent individually leads to multiple API requests. At the on-site testing phase, the challenges were shown for the low NFC card reading accuracy, longer delay for data transmission and the degradation of image quality. The obstacle between the PN532 and NFC card caused missing read data and required optimum position. Lower Wi-Fi strength because of further distance and WiFi shared by multiple devices resulting in a longer time for data transmission. The low light level caused the images captured by ESP32 CAM to be blurry and low contrast. To address these issues, several improvements were made. A retry mechanism was implemented to reread the NFC card data until accurate data was read. The users did not need to worry the NFC card position will affect the data not being read correctly. Data transmission was optimized by using mobile hotspot to overcome the WiFi overload and sending data in larger chunks to avoid multiple API requests. The ESP32 CAM built in flashlight was activated to improve the image quality in low light environments. These changes significantly enhanced the system's performance.

4.5 Any other related activity

In addition to the core testing procedures, several related activities were conducted to ensure the system's reliability, usability, and effectiveness. User feedback sessions were organized during testing phases, where stakeholders and end-users interacted with the system and provided valuable insights on its functionality and user experience. These feedback sessions helped identify minor usability issues, such as interface adjustments and system response times, which were subsequently addressed to enhance the system's performance. Furthermore, stress tests were conducted to evaluate the system's behavior under high workloads, simulating real-world usage scenarios. This included extended operation hours and multiple simultaneous operations to ensure the system's stability and durability. Documentation reviews were also carried out to verify that all technical details, including assembly instructions and operational guidelines, were clear and accurate. These related activities provided a comprehensive evaluation of the system, contributing to its refinement and ensuring it met the project's objectives and user expectations.

5. Project Management

Effective project management was integral to the successful development of the Smart Inventory Management System (SIMs), ensuring that all phases, from conceptualization to deployment, were completed on time and within budget. The management strategy emphasized meticulous resource planning, timeline adherence, and adaptive problem-solving to navigate the complexities of the project. A well-structured approach allowed the team to align technical efforts with project objectives, ensuring a seamless workflow.

The project budget was meticulously planned to balance affordability and functionality. Major expenditures included essential hardware components such as the ESP32 microcontroller, ESP32 camera module, NFC tags, and materials for the laser-cut acrylic enclosures. To optimize costs, the team relied on free or cost-effective software solutions such as Firebase and Google Drive for data storage and management. Regular budget reviews were conducted to monitor spending, and potential cost overruns were mitigated by sourcing alternative materials or components without compromising system quality.

Scheduling was guided by a detailed timeline that outlined weekly deliverables and milestones. Key stages, including hardware integration, software development, prototype fabrication, and testing, were planned sequentially to ensure cohesive progress. The timeline was visualized using Gantt charts, which provided a clear overview of the project's status and facilitated adjustments when delays occurred. Each team member focused on a specific domain—hardware configuration, software and interface development, or prototype design—ensuring parallel progress and efficient resource utilization.

Challenges encountered during the project required innovative solutions to maintain momentum. Communication delays between the ESP32 microcontroller and Firebase initially caused data synchronization issues. This was addressed by optimizing network configurations and implementing robust error-handling mechanisms in the code. Similarly, the AI-powered defect detection model required extensive fine-tuning and additional training data to achieve reliable accuracy. Feedback from on-site evaluations revealed usability concerns in the Human-Machine Interface (HMI), prompting refinements in navigation flow, button placement, and layout design to improve user experience. Collaborative brainstorming sessions and iterative testing cycles were instrumental in overcoming these hurdles.

Team collaboration was facilitated through weekly meetings to discuss progress, identify roadblocks, and reallocate resources as needed. Online tools such as Google Workspace and GitHub were used to streamline communication and manage version control, ensuring that all team members remained aligned with project objectives. Each member's expertise was leveraged effectively, creating a cohesive workflow that integrated hardware, software, and user interface development.

The project management strategy ensured the successful realization of SIMs as a robust, scalable, and user-friendly inventory management solution. By adhering to structured planning and adaptive problem-solving, the team delivered a system that met both technical requirements and user expectations, demonstrating the value of interdisciplinary collaboration and efficient resource management in engineering projects.

5.1 Project Costing

The following is a detailed breakdown of all costs for materials, software and hardware during the project.

1. Materials:
    - NFC Tags: RM5.95
    - Jumper Wires: Free
    - LCD Display: RM16.90
    - USB Power Supply: Free
    - Male Connectors Headers: RM0.60
    - Stripboard: Free
    - A2 Acrylic Sheet: RM 45.20
    - A3 & A2 Acrylic Sheet: RM67.83
2. Hardware:
    - ESP32: RM32.90
    - PN532 NFC Reader: RM19.90
    - ESP32-CAM Module: RM58.00
3. Software
    - Google Cloud API: Free
    - Arduino IDE: Free
    - Visual Studio: Free

The total cost is RM247.28 which is less than the expected cost. This was mainly due to no shipping fee being included for online purchase when more items were bought at once. Besides, some of the tools such as soldering iron did not need to be purchased as it can be borrowed from the UTM component store.

5.2 Timeline Analysis

The timeline for the Smart Inventory Management System (SIMs) project was meticulously structured to ensure methodical progress from conceptualization to a fully functional prototype within 12 weeks. The timeline was divided into specific phases, with each week dedicated to completing a single significant task. This structured approach allowed the team to focus on quality and detail, ensuring all aspects of the project were given sufficient attention while adhering closely to the planned schedule.

Weeks 1 to 4 served as the foundational phase, focusing on identifying user needs and understanding the problem domain. The team began by designing comprehensive survey questionnaires, which were divided into four sections: personal information and pre-satisfaction levels, challenges and limitations, implementation strategies, and future recommendations. These surveys were carefully distributed to a targeted group of restaurant customers, with each team member responsible for gathering feedback from specific respondents. Concurrently, the team conducted detailed site visits to restaurants and supplier locations to observe existing inventory management practices. Tasks during these visits included studying customer order flows, supplier inventory management, and purchase requisition processes. Data clustering was applied to the responses and observations to identify patterns and pain points, which were later analyzed to shape the system requirements. Insights from this phase were documented with visual aids such as clustered diagrams and slides, enabling the team to align their designs with real-world needs.

Weeks 5 to 8 marked the design and initial development phase, where the focus shifted towards conceptualizing and building the system's core components. The team created detailed 2D and 3D sketches of the proposed system using Fusion 360, emphasizing features like the integration of NFC scanning and ESP32 modules for inventory data transmission. The hardware team sourced essential components, including NFC modules, ESP32 microcontrollers, and supporting peripherals, ensuring all parts met the technical requirements. Parallelly, the software team developed the web and mobile application interfaces, implementing functionalities for user authentication, inventory tracking, and real-time data visualization. During this period, mock inventory data was generated to simulate real-world scenarios for testing the system's functionality. This phase also included creating a comprehensive system block diagram and program flowchart to document the system architecture and logic, which were later refined through internal reviews and feedback sessions.

Weeks 9 to 11 were critical for testing, debugging, and enhancement. The hardware team encountered issues with the ESP32 camera module and LCD display, where intermittent malfunctions required significant troubleshooting. They worked on stabilizing these components, ensuring smooth image capture and display functions. The software team integrated Firebase for inventory storage and retrieval, leveraging its scalability and real-time capabilities. Additionally, the machine learning algorithm for defect detection, built using Python and MobileNetV2, was refined during this phase. The team conducted extensive testing in simulated environments, creating scenarios that mimicked real-world restaurant operations to validate the system's accuracy and reliability. Debugging efforts were intensified during this phase to address issues identified during testing. Concurrently, preparations for the Electrical Engineering Capstone Showcase (EECS) were initiated, with tasks such as abstract writing, poster design using Canva, and demonstration rehearsals. The team aimed to present a polished, professional prototype during the showcase.

Week 12, the final phase, focused on consolidating all efforts for the EECS event. On-site testing was conducted at the demonstration venue to ensure the system operated seamlessly in a live environment. The team reviewed and finalized the project documentation, including the abstract, poster, and group report. The prototype was presented in the Communication category, where its features, such as NFC scanning, Firebase integration, and AI-powered defect detection, were highlighted. Rehearsals ensured a flawless demonstration, showcasing the system's real-world applicability and user-friendly interface.

Overall, the project timeline was executed with precision and diligence, with all tasks completed within their designated weeks. Minor adjustments during the debugging phase were promptly addressed, ensuring the project remained on track. The structured timeline facilitated the successful development of a comprehensive, innovative solution to restaurant inventory management challenges, culminating in a robust prototype ready for real-world deployment and competitive demonstration.

TABLE 4.4: GANTT CHART.

Task	**W 1**	**W 2**	**W 3**	**W 4**	**W 5**	**W 6**	**W 7**	**W 8**	**W 9**	**W 10**	**W 11**	**W 12**	**W 13**	**W 14**
Restaurant Inventory Management System	✓													
Customer Ordering and Queue Management System	✓													
Site Visit - Task 1 (Restaurant Operations)		✓												
Site Visit - Task 2 (Supplier Inventory Management)		✓												
Conceptual Sketch M1 with 2D & 3D Sketch			✓											
Program Flow Chart with Slide			✓											
S.T.E.E.P Analysis with Slide				✓										
Conceptual Prototype Demo Video (Video Editing)				✓										
Project Adjustment Based on Panel's Comment By M1					✓									
Develop System Hardware: NFC Tag Scanning with ESP32						✓								
Develop System Hardware: Inventory Data Transmission via						✓								

Google API														
Develop Software: Create HTML Structure for All Pages							✓							
Develop Software: Style Login, Sign-Up, and Dashboard with CSS								✓						
Develop System Hardware: Image Captured with ESP32 Camera								✓						
Develop System Hardware: Image uploaded to Google Drive using Google App Script								✓						
Firebase Integration for Inventory Storage									✓					
Testing and Debugging										✓	✓			
Draft Group Report											✓	✓		
Design of EECS Poster												✓		
Attend EECS													✓	
Submit Group Report														✓

5.3 Issues Encountered with Solutions

Throughout the project, several challenges were encountered, ranging from technical issues to logistical and managerial obstacles. A key technical challenge involved integrating the various system components, particularly ensuring seamless communication between the hardware and software. Initial attempts to synchronize the data from sensors with the database resulted in delays and inaccuracies. The solution involved reconfiguring the communication protocols and optimizing the software code to reduce latency and improve data accuracy. The team also conducted rigorous debugging sessions and implemented additional testing to ensure reliable performance under different conditions.

Another issue arose during the procurement of specific components, which led to delays in the project timeline. Certain materials and electronic parts were either unavailable locally or subject to long shipping times. To address this, alternative suppliers were identified, and substitute components with similar specifications were sourced to keep the project on track. This experience highlighted the importance of preemptive resource planning and having contingency plans in place for future projects.

On the managerial side, the team faced coordination challenges during the initial phases of the project due to varying skill levels and knowledge among team members. This was mitigated by organizing regular meetings and assigning tasks based on individual strengths, ensuring efficient use of resources and consistent progress. Collaborative tools, such as task trackers and shared documents, were introduced to enhance communication and streamline project management.

Finally, testing and validation presented another challenge. Early tests revealed inconsistencies in system performance under different environmental conditions. The team responded by refining the testing procedures, using controlled environments to isolate variables, and implementing adjustments to improve system robustness. This iterative process not only resolved the inconsistencies but also provided valuable insights into designing more resilient systems.

These challenges and their resolutions contributed to the team's growth, enhancing their problem-solving skills and project management capabilities. Lessons learned from overcoming these obstacles will be applied to future projects, including better initial planning, resource allocation, and adaptability to unforeseen issues.

6. Product Comparison

This section analyzes the comparison of the developed prototype, SIMs, with other similar products available in the market. The evaluation focuses on unique features, primary applications, and innovations, providing a comprehensive benchmark against existing solutions. The developed prototype's competitive edge is highlighted by integrating NFC-based inventory management, AI-driven defect detection, and real-time synchronization using Firebase. Each competitor is assessed based on its key functionalities, scalability, affordability, and relevance to the inventory management domain, ensuring an evidence-based analysis. References to official sources for each competitor have been included to substantiate the claims made.

TABLE 6.1: Comparison with similar products.

Features		**Competitor**			
	SmartPOS Barcode Scanner (Zebra DS2200 Series)	**RFID Inventory Tracker (Zebra FX9600)**	**POSlite (Square POS Lite)**	**WarehousePro Automation System (Oracle NetSuite)**	**Developed Prototype (SIMs)**
Primary Application	Inventory check-in/out and barcode scanning	Automated inventory tracking via RFID	POS with simple features	Warehouse automation and resource tracking	NFC-based restaurant inventory management system
May include other features	Scans multiple barcode formats, connects to POS systems	High-speed RFID tag processing	Low-cost cloud-based reporting	Advanced analytics, order management, and automation	NFC scanning, AI defect detection, cloud storage
May include other features	Plug-and-play setup, compatibility with Android/iOS	Durable for large-scale operations	User-friendly mobile app interface	Inventory and financial management integration	Multi-platform interface with Firebase and Google API
May include other features	Affordable for small businesses	Requires fixed installation and antenna setup	Requires stable internet connection	Expensive but highly scalable for enterprises	Android APK compatibility and on-site usability
Price	MYR 5,400 (USD 1,200)	MYR 24,750 (USD 5,500)	MYR 2,700 (USD 600)	MYR 67,500 (USD 15,000)	MYR 250 (Prototype cost)

7. Future Works

The Smart Inventory Management System (SIMs) has shown significant promise in addressing restaurant inventory management challenges, but several future enhancements could further refine its functionality and market competitiveness. One key area for improvement is scalability, where the integration of predictive analytics could optimize stock levels by forecasting inventory trends, enabling the system to cater to larger businesses with complex requirements. Additionally, enhancing the defect detection module through adaptive machine learning algorithms could improve its ability to identify diverse and evolving defect types. Durability of the hardware is another critical aspect, with future iterations potentially utilizing more robust materials, such as polycarbonate or aluminum, for the enclosures to withstand demanding operational environments. Optimizing communication protocols between the ESP32 microcontroller, Firebase, and other components can reduce latency and enhance data transmission reliability, ensuring faster real-time updates and a smoother user experience. Feature expansions, including voice-activated commands, multilingual support, and integration with third-party restaurant management tools, could increase the system's versatility and adoption. On-site testing in diverse, high-traffic restaurant settings would provide valuable insights into potential bottlenecks, allowing for targeted refinements. Enhanced analytics dashboards with customizable reports could also empower managers with deeper insights into inventory trends and operational efficiencies. By addressing these areas, SIMs can

evolve into a scalable, durable, and feature-rich solution that not only meets the immediate needs of the restaurant industry but also anticipates and adapts to future challenges, solidifying its position as a transformative tool for sustainable and efficient inventory management.

8. Conclusion

The Smart Inventory Management System (SIMs) successfully demonstrated its effectiveness in addressing key challenges in restaurant inventory management, integrating advanced technologies to streamline operations and enhance efficiency. During laboratory testing, the prototype performed reliably, showcasing its ability to process NFC-tagged inventory data, capture high-resolution images for defect detection, and synchronize records in real time using Firebase. On-site evaluations further validated the system's functionality, confirming its suitability for dynamic, real-world environments.

Several challenges arose during development, including communication delays between the ESP32 microcontroller and Firebase, as well as initial accuracy issues with the AI-powered defect detection model. These were resolved through network optimization, improved error handling, and iterative refinement of the machine learning model with an expanded dataset. Usability concerns identified during user testing, such as navigation flow and interface layout, were addressed through targeted improvements to the Human-Machine Interface (HMI).

The project successfully achieved its objectives, delivering a robust, user-friendly, and scalable solution tailored to the needs of restaurant inventory management. By combining innovative hardware, AI-powered software, and cross-platform accessibility, SIMs provides a practical and sustainable tool for improving operational efficiency. These accomplishments underscore the potential of SIMs to contribute to advancements in industry innovation and infrastructure, aligning with the goals of Sustainable Development Goal 9 (SDG 9), and laying a solid foundation for future enhancements and deployment at scale.

Acknowledgement

The authors would like to express their deepest gratitude to the Faculty of Electrical Engineering, Universiti Teknologi Malaysia, for their invaluable support and funding, which made this capstone project possible. Special thanks are extended to our dedicated teammates for their collaborative efforts and unwavering commitment throughout the development of the project. We also acknowledge the contributions and insights provided by our seniors and juniors, whose guidance and assistance were instrumental in refining various aspects of the project. Our heartfelt appreciation goes to The Cube Enterprise for granting us access to their facilities and providing real-world insights that enriched our understanding of inventory management challenges. We are profoundly grateful to our supervisor, Dr. Ahmad Shahidan bin Abdullah, for his expert guidance, constructive feedback, and continuous encouragement, which played a pivotal role in the successful completion of this project. Lastly, we thank all stakeholders and participants, including those who provided feedback during testing, for their invaluable contributions to the realization of this innovative solution.

References

[1] NFC Inventory Management: Bring Inventory at Your Fingertips. The NonStop Group. [Online]. Available: https://reliability.thenonstopgroup.com/nfc-inventory-management-bring-inventory-at-your-fingertips/. [Accessed: 04-Nov-2024].

[2] ESP32-CAM: A Comprehensive Guide. ThinkRobotics. [Online]. Available: https://thinkrobotics.com/blogs/learn/esp32-cam-a-comprehensive-guide. [Accessed: 08-Nov-2024].

[3] Using NFC Tags for Asset & Inventory Tracking. Comparesoft. [Online]. Available: https://comparesoft.com/assets-tracking-software/nfc/. [Accessed: 08-Nov-2024].

[4] ESP32-CAM Video Streaming Web Server (works with Home Assistant). Random Nerd Tutorials. [Online]. Available: https://randomnerdtutorials.com/esp32-cam-video-streaming-web-server-camera-home-assistant/. [Accessed: 20-Nov-2024].

[5] Using NFC for Asset Tracking & Inventory Management. AHG. [Online]. Available: https://www.ahg.com/business-mobile-apps-blog/nfc-for-asset-inventory-tracking.html. [Accessed: 04-Dec-2024].

[6] IoT CCTV Camera using ESP32 CAM & Blynk App – Live Streaming. Microcontrollers Lab. [Online]. Available: https://microcontrollerslab.com/iot-cctv-camera-esp32-cam-blynk-app-live-streaming/. [Accessed: 20-Nov-2024].

[7] Asset Tracking and Inventory Management with QR Codes and NFC. Ventipix. [Online]. Available: https://ventipix.com/asset-and-inventory-management-with-qr-codes-and-nfc.php. [Accessed: 04-Dec-2024].

[8] A Deep Dive into the ESP32 CAM Module: Unleashing Potential. Full Skills. [Online]. Available: https://full-skills.com/iot/esp32-cam/. [Accessed: 04-Dec-2024].

[9] NFC Tags for Asset Tracking and Inventory Management. Xerafy. [Online]. Available: https://xerafy.com/nfc-series/. [Accessed: 09-Dec-2024].

[10] Live Video Stream Monitoring using ESP32-CAM and Ubidots. Ubidots Help Center. [Online]. Available: https://help.ubidots.com/en/articles/7156830-live-video-stream-monitoring-using-esp32-cam-and-ubidots. [Accessed: 09-Dec-2024].

[11] NFC Inventory Application. Small Business Inventory Management. [Online]. Available: https://small-business-inventory-management.com/nfc-for-inventory-management-asset-tracking.htm. [Accessed: 14-Dec-2024].

[12] Inventory Management Transformation with RFID vs. NFC Labels. Royal Label. [Online]. Available: https://www.royallabel.com/blog/smart-inventory-management-rfid-nfc/. [Accessed: 14-Dec-2024].

[13] Live Stream with ESP32 Camera Module over Internet with Face Recognition. Circuit Schools. [Online]. Available: https://www.circuitschools.com/live-stream-with-esp32-camera-module-over-internet-with-face-recognition/. [Accessed: 14-Dec-2024].

[14] NFC Tags for Inventory: Transforming Tracking Systems. The NonStop Group. [Online]. Available: https://reliability.thenonstopgroup.com/nfc-tags-for-inventory/. [Accessed: 19-Dec-2024].

[15] Using an ESP32-CAM Module to Create a Live Web Server for Video Streaming. GitHub - DiegoArmstrong/ESP-CAM. [Online]. Available: https://github.com/DiegoArmstrong/ESP-CAM. [Accessed: 20-Dec-2024].

[16] Upload an image directly from ESP32-CAM to Google Drive. GitHub - Gsampallo. [Online]. Available: https://github.com/gsampallo/esp32cam-gdrive. [Accessed 23-Dec-2024].

[17] ESP32 Datalogging to Google Sheets (using Google Service Account). Random Nerd Tutorials. [Online]. Available: https://randomnerdtutorials.com/esp32-datalogging-google-sheets/.

[18] Interfacing PN532 NFC RFID Module with Arduino. How to Electronics. [Online]. Available: https://how2electronics.com/interfacing-pn532-nfc-rfid-module-with-arduino/.

[19] ESP32 UART Communication (Serial): Set Pins, Interface, Send and Receive Data (Arduino IDE). Random Nerd Tutorial [Online]. Available: https://randomnerdtutorials.com/esp32-uart-communication-serial-arduino/.

[20] Change ESP32-CAM OV2640 Camera Settings: Brightness, Resolution, Quality, Contrast and More. Random Nerd Tutorials. [Online]. Available: https://randomnerdtutorials.com/esp32-cam-ov2640-camera-settings/.

[21] "Sliding Dovetail Lid for 3D Printed Box | Fusion 360 Tutorial," www.youtube.com. https://www.youtube.com/watch?v=HMtLqm5TkGE

[22] Christopher's Factory, "EASY 3D-Printed Dovetail Box w/ Lid Fusion 360 Tutorial (Free STL)," YouTube, Nov. 30, 2022. https://www.youtube.com/watch?v=Xj_sEO7u_IQ (accessed Jan. 19, 2025)

APPENDIX 1: INTERVIEW SCRIPT

Izwan, a staff member from The Cube Enterprise, located at Lot 6, Student Mall, Universiti Teknologi Malaysia, was interviewed to gather insights into current inventory management practices, challenges, and expectations for the proposed Smart Inventory Management System (SIMs). The following questions and answers detail the findings from the interview:

1. How do you currently manage your inventory?
   Answer: We use a combination of spreadsheets and manual record-keeping for managing inventory. The process involves entering stock details into Excel sheets, updating them whenever stock is received or dispatched. While this system has worked for years, it is cumbersome and error-prone, especially during peak business hours when multiple updates need to be logged simultaneously.

2. What challenges do you face with your current inventory system?
   Answer: There are several challenges. First, manual entry leads to frequent errors, which accumulate over time and result in significant discrepancies during stock-taking. Second, real-time updates are nearly impossible, as the process is entirely manual and often delayed. Lastly, there's a lack of automation for tasks like monitoring expiry dates or generating alerts for low-stock items.

3. How often do stock discrepancies occur, and what causes them?
   Answer: Stock discrepancies are fairly common, occurring at least once or twice a week. These are typically caused by human errors, such as forgetting to update the spreadsheet, duplicate entries, or miscounting items during stock verification. Sometimes, items are misplaced, and their locations are not properly logged, further complicating the issue.

4. What is the average time spent weekly on inventory-related tasks?
   Answer: On average, we spend around 10 to 15 hours per week on inventory tasks. This includes updating records, reconciling discrepancies, preparing orders for restocking, and conducting physical stock checks. During busy periods, the time required increases significantly.

5. How do you handle tracking the location and condition of inventory items?
   Answer: We manually track the location of items by designating specific storage areas for different categories of stock. However, this system is not foolproof, as items are often moved without updating the records. The condition of items is assessed visually during check-in, but there's no formal system for logging product conditions or reporting quality issues.

6. What features would you like to see in a modern inventory management system?
   Answer: A system that offers real-time tracking and automated updates would be incredibly valuable. Features like automated alerts for low-stock levels, expiry notifications, and a user-friendly interface for managing inventory on the go are high on our priority list. Additionally, a system that can analyze stock usage patterns and suggest optimal restocking schedules would greatly enhance efficiency.

7. How do you feel about implementing NFC technology for inventory tracking?
   Answer: NFC technology sounds promising, particularly for streamlining the check-in/check-out process. It could eliminate manual data entry and significantly reduce errors. However, the system must be easy to use and require minimal training for staff who are not very familiar with advanced technologies.

8. Have you used other inventory technologies, such as RFID or Barcode systems, in the past? If so, how effective were they?
   Answer: We've experimented with barcode systems in the past, but they required specialized scanners and additional equipment, which made them less convenient for our small-scale operations. The learning curve for using such systems was also a challenge, as training staff took time and effort. NFC seems like a simpler and more affordable alternative.

9. How do you currently track product quality during check-in?
   Answer: Currently, we rely on visual inspections by staff to assess the quality of items during check-in. While this helps identify obvious defects, there's no formal mechanism to document product conditions or ensure accountability. Having a system that can capture images of items during check-in would be a great improvement.

10. Would you find it helpful to include image capture as part of the check-in process?
    Answer: Absolutely. Capturing images of items during check-in would not only improve quality assurance but also provide a visual record that could be used to resolve disputes or verify product conditions at a later stage. It would be particularly useful for fragile or high-value items.

11. What are your preferences for accessing and managing inventory data?
    Answer: A mobile application would be ideal for on-the-go access, especially for tasks like stock-checking and item tracking. However, a desktop interface would be better suited for detailed reporting, analytics, and managing large datasets. Having both options would provide flexibility and cater to different use cases.

12. What kind of reports or analytics would be most useful for your operations?
    Answer: Reports on stock usage trends, inventory turnover rates, and cost analysis would be very useful for decision-making. We'd also like to see analytics that highlight slow-moving items and predict when specific items will run out based on historical data. These insights would help us optimize stock levels and reduce waste.

13. How would you ensure the system is easy to adopt for your staff?
    Answer: User-friendliness is key. The system should have an intuitive interface with simple instructions and minimal technical jargon. Providing a quick training session and an easy-to-follow user manual would also help ensure smooth adoption by staff.

14. Are there any specific challenges you foresee in implementing such a system?
    Answer: The primary challenge would be transitioning from our current system to a new one without disrupting daily operations. There might also be resistance from staff who are accustomed to manual processes. Additionally, ensuring that the system integrates seamlessly with our existing workflows will be critical.

15. What is your budget for adopting a new inventory management solution?
    Answer: Our budget is limited, so affordability is a key consideration. We're looking for a cost-effective solution that offers good value for money without compromising on essential features.

16. Would you recommend any additional features for the proposed system?
    Answer: It would be great to have features like role-based access control to ensure that only authorized personnel can modify critical data. Integration with accounting software to streamline financial reporting and invoicing would also be a valuable addition.

17. How do you currently handle expired or defective items in your inventory?
    Answer: Defective items are usually flagged during physical inspections and manually logged. However, expired items are often missed because we don't have an automated alert system. A feature that tracks expiry dates and sends reminders would be incredibly helpful.

18. How frequently do you conduct physical stock checks, and what are the challenges?
    Answer: Physical stock checks are conducted monthly. The main challenges are the time required and the likelihood of human errors. It's also difficult to coordinate stock checks without disrupting normal operations.

19. What kind of training would your staff need for using the new system?
    Answer: Basic training on how to use the mobile app, update inventory records, and troubleshoot minor issues would be sufficient. For managers, more advanced training on generating reports and analytics might be necessary.

20. How do you think this system will impact your operations in the long run?
    Answer: I believe this system will significantly reduce errors, save time, and improve inventory accuracy. Over time, it could lead to better decision-making, cost savings, and increased efficiency in managing our operations.

Figure A.1: Interview session with Izwan from The Cube Enterprise.

APPENDIX 2: OBSERVATION SCRIPT

The following observation script was used during on-site visits to The Cube Enterprise at Lot 6, Student Mall, UTM JB, and other selected restaurants to analyze their inventory management practices, operational workflows, and customer interaction processes. Observations were recorded to identify key pain points and areas for improvement. The script follows the POEMS (People, Objects, Environments, Messages, and Services) framework for structured data collection.

People

Objective:

Observe how staff interact with inventory systems and customers.

Monitor how staff handle inventory check-in and check-out processes.

Note whether staff encounter difficulties in recording inventory transactions.

Observe how staff allocate responsibilities for stock management and quality checks.

Examine how customers interact with the ordering and queuing system.

Record whether staff experience stress or confusion during peak business hours.

Objects

Objective:

Analyze tools and equipment used for inventory and order management.

Inspect the tools used for inventory tracking, such as spreadsheets or manual ledgers.

Check for the presence of NFC tags, barcodes, or other tracking mechanisms.

Observe the use of hardware such as tablets, scanners, or point-of-sale (POS) systems.

Evaluate the condition of storage units and their organization (e.g., labeled shelves).

Examine packaging and storage methods for maintaining product quality.

Environments

Objective:

Evaluate the physical space and conditions where inventory is managed.

Assess the organization of the inventory storage area.

Note the accessibility and safety of the storage environment.

Identify whether environmental factors, such as humidity or temperature, affect inventory.

Observe the workflow layout for tasks like stock checking and order preparation.

Evaluate if there is sufficient space for staff to work efficiently without overcrowding.

Messages

Objective:

Understand how communication is handled during inventory and customer management.

Observe how staff communicate inventory updates (e.g., verbally or through notes).

Check for the presence of digital or physical reminders for low-stock or expired items.

Monitor how instructions for daily tasks are disseminated among staff.

Record the clarity and accuracy of customer communication during order placement.

Identify any miscommunication that leads to inventory discrepancies or service delays.

Services

Objective:

Analyze the services provided to customers and their integration with inventory management.

Observe the flow of services, from customer order placement to product delivery.

Evaluate how inventory availability impacts the quality and speed of service.

Identify bottlenecks in the ordering, queuing, or payment processes.

Note whether customers report dissatisfaction due to missing or delayed items.

Examine how refunds or returns are handled when defective or incorrect items are provided.

APPENDIX 3: PERSONIFICATION AND DESIGN STATEMENT

Profile

- **Age**: 22 years old, a driven entrepreneur and student at the Faculty of Electrical Engineering, Universiti Teknologi Malaysia (UTM).
- **Role**: Founder and manager of The Cube, a restaurant located at Lot 6, Student Mall, UTM JB.
- **Attributes**: Enthusiastic about leveraging technology to streamline restaurant operations while excelling in academic endeavors.
- **Challenges**: Balances the dual roles of being a business owner and a full-time engineering student, managing time and resources effectively.

Pain Points

- Manual inventory tracking leading to frequent errors and operational inefficiencies.
- Difficulty ensuring product quality during check-in processes due to a lack of systematic monitoring tools.
- Limited ability to oversee real-time inventory status due to a demanding schedule.
- Challenges in preventing inventory wastage due to expired or overstocked items.
- Absence of a unified platform for seamless inventory management across devices.

Needs

- An NFC-based smart inventory management system to automate and simplify stock tracking processes.
- Real-time updates on inventory status, accessible on mobile and desktop platforms for greater flexibility.
- AI-powered defect detection integrated with an ESP32 camera to ensure product quality during check-ins.
- Automated notifications for low-stock or expiring items to enhance operational efficiency.
- A user-friendly, intuitive interface designed to save time and reduce workload, enabling Jin to focus on both business growth and academic performance.

Design Statement

How might we assist Jin, a young and ambitious student entrepreneur, in revolutionizing inventory management for The Cube through an NFC-integrated Smart Inventory Management System (SIMs), delivering real-time tracking, quality assurance, and operational excellence while enabling him to manage his academic and professional responsibilities effectively?

Appendix 4: Programming Coding

HTML for Mobile App:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Smart Inventory Management System</title>
    <meta http-equiv="Cache-Control" content="no-store" />
    <link rel="stylesheet" href="style.css">
    <!-- Include Chart.js -->
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
</head>
<body>
    <div id="app">
        <!-- Login Screen -->
        <div id="login-screen" class="screen active">
            <h1>Login</h1>
            <input type="email" id="email-login" placeholder="Email" required>
            <input type="password" id="password-login" placeholder="Password" required>
            <button id="login-btn">Login</button>
            <button id="show-signup-btn">Sign Up</button>
        </div>

        <!-- Sign Up Screen -->
        <div id="signup-screen" class="screen">
            <h1>Sign Up</h1>
            <input type="text" id="username-signup" placeholder="Username" required>
            <input type="email" id="email-signup" placeholder="Email" required>
            <input type="password" id="password-signup" placeholder="Password" required>
            <button id="signup-btn">Create Account</button>
            <button id="back-to-login-btn">Back</button>
        </div>

        <!-- Main Menu Screen -->
        <div id="main-menu-screen" class="screen">
            <h1>Main Menu</h1>
            <button id="view-transactions-btn">Transactions</button>
            <button id="view-settings-btn">Settings</button>
            <button id="view-analysis-btn">Analysis</button>
        </div>

        <!-- Transactions Screen -->
        <div id="transactions-screen" class="screen">
            <h1>Transaction Details</h1>
            <div class="table-container">
                <table>
                    <thead>
                        <tr>
                            <th>Date</th>
                            <th>Location</th>
                            <th>Item</th>
                            <th>Transaction Type</th>
                            <th>Serial Number</th>
                            <th>Manufacture Date</th>
                            <th>Expiry Date</th>
                        </tr>
                    </thead>
                    <tbody id="transactions-table">
                        <!-- Transactions will be dynamically inserted here -->
                    </tbody>
                </table>
            </div>
            <button class="back-btn">Back</button>
        </div>

        <!-- Settings Screen -->
        <div id="settings-screen" class="screen">
            <h1>Settings</h1>
            <form id="profile-form">
                <input type="text" id="username" placeholder="Username" required>
                <input type="password" id="password" placeholder="New Password">
                <input type="text" id="role" placeholder="Role">
                <input type="text" id="country" placeholder="Country">
```

```html
                <button type="submit">Save Changes</button>
            </form>
            <button class="logout-btn">Logout</button>
            <button class="back-btn">Back</button>
        </div>

        <!-- Analysis Screen -->
        <div id="analysis-screen" class="screen">
            <h1>Analysis</h1>
            <div class="chart-container">
                <canvas id="transactionTypeDoughnutChart"></canvas>
            </div>
            <div class="chart-container">
                <canvas id="quantityLineChart"></canvas>
            </div>
            <div class="chart-container">
                <canvas id="accumulatedQuantityLineChart"></canvas>
            </div>
            <!-- New Bar Charts -->
            <div class="chart-container">
                <canvas id="quantityByItemBarChart"></canvas>
            </div>
            <div class="chart-container">
                <canvas id="quantityByLocationBarChart"></canvas>
            </div>
            <!-- New Charts for Check In/Check Out -->
            <div class="chart-container">
                <canvas id="quantityByItemWithTransactionType"></canvas>
            </div>
            <div class="chart-container">
                <canvas id="quantityByLocationWithTransactionType"></canvas>
            </div>
            <button class="back-btn">Back</button>
        </div>
    </div>

    <script src="script.js"></script>
</body>
</html>
```

FIGURE A.2: HTML CODE TO CREATE THE STRUCTURE OF SIMS'S WEB APPLICATION.

CSS for Mobile App:

```
/* General Styles */
body, html {
    height: 100%;
    margin: 0;
    font-family: 'Helvetica', sans-serif;
    display: flex;
    justify-content: center;
    align-items: center;
    background: linear-gradient(to right, #6a11cb, #2575fc);
}

#app {
    width: 90%;
    max-width: 350px;
    background: rgba(255, 255, 255, 0.8);
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
}

/* Screen Styles */
.screen {
    display: none;
    text-align: center;
}

.screen.active {
    display: block;
}

/* Login and Signup Screens */
#login-screen, #signup-screen, #main-menu-screen, #transactions-screen, #settings-screen,
#analysis-screen {
    height: 600px;
    width: 350px;
    overflow-y: auto;
}

/* Input and Button Styles */
input, button {
    width: 100%;
    padding: 12px 20px;
    margin: 8px 0;
    border-radius: 4px;
    border: 1px solid #ccc;
    box-sizing: border-box;
}

button {
    background-image: linear-gradient(to right, #ff7eb9, #ff65a3);
    color: white;
    border: none;
    border-radius: 10px;
    cursor: pointer;
    transition: background-color 0.3s ease, transform 0.3s ease;
}

button:hover {
    background-image: linear-gradient(to right, #ff65a3, #ff4b81);
    transform: scale(1.05);
}

/* Table Container for Horizontal Scrolling */
.table-container {
    overflow-x: auto; /* Enable horizontal scrolling */
    overflow-y: hidden; /* Prevent vertical scrolling inside the container */
    width: 100%; /* Ensure full width */
    margin-top: 20px;
    border: 1px solid #ddd; /* Optional: Add border for visibility */
    border-radius: 4px;
```

```
}

/* Vertical and horizontal scrolling for the screen */
#transactions-screen {
    overflow-y: auto; /* Enable vertical scrolling for the screen */
    overflow-x: hidden; /* Prevent horizontal scrolling at the screen level */
    height: 600px; /* Fixed height to enable vertical scrolling */
}

/* Table */
table {
    width: 100%;
    min-width: 800px; /* Minimum width to prevent column squeezing */
    border-collapse: collapse;
}

th, td {
    text-align: left;
    padding: 8px;
    border-bottom: 1px solid #ddd;
    white-space: nowrap; /* Prevent text wrapping */
}

/* Scrollbar Styling */
.table-container::-webkit-scrollbar {
    height: 8px; /* Horizontal scrollbar height */
}

.table-container::-webkit-scrollbar-thumb {
    background-color: #888;
    border-radius: 4px;
}

.table-container::-webkit-scrollbar-thumb:hover {
    background-color: #555;
}

/* Chart Container Styles */
.chart-container {
    width: 100%;
    height: 250px; /* Fixed height for the chart container */
    margin: 20px auto;
    position: relative; /* Ensure the chart stays within the container */
}

canvas {
    max-width: 100%; /* Ensure the canvas does not overflow */
    max-height: 100%; /* Ensure the canvas does not overflow */
}

/* Back Button Styles */
.back-btn {
    margin-top: 20px;
}

/* Logout Button Styles */
.logout-btn {
    margin-top: 10px;
    background-image: linear-gradient(to right, #ff4b81, #ff7eb9);
}

/* Form Styles */
#profile-form {
    display: flex;
    flex-direction: column;
    gap: 10px;
}

/* Analysis Screen Styles */
#analysis-screen h1 {
    margin-bottom: 20px;
}

/* Responsive Design */
@media (max-width: 480px) {
    #app {
        width: 100%;
```

```
        max-width: none;
        border-radius: 0;
    }

             #login-screen,   #signup-screen,   #main-menu-screen,   #transactions-screen,
#settings-screen, #analysis-screen {
        height: 100vh;
        width: 100%;
        border-radius: 0;
    }

    .chart-container {
        height: 200px; /* Adjust height for smaller screens */
    }

    .table-container {
        max-width: 100%; /* Ensure the container doesn't overflow */
    }

    table {
        font-size: 12px; /* Reduce font size for smaller screens */
    }

    th, td {
        padding: 6px; /* Adjust padding for smaller dimensions */
    }
}
```

FIGURE A.3: CSS SCRIPT FOR DESIGNING VISUAL APPEALING AND USER-FRIENDLY INTERFACE.

Javascript for Mobile App:

```
document.addEventListener('DOMContentLoaded', () => {
    const loginScreen = document.getElementById('login-screen');
    const signupScreen = document.getElementById('signup-screen');
    const mainMenuScreen = document.getElementById('main-menu-screen');
    const transactionsScreen = document.getElementById('transactions-screen');
    const settingsScreen = document.getElementById('settings-screen');
    const analysisScreen = document.getElementById('analysis-screen');
    const loginBtn = document.getElementById('login-btn');
    const signupBtn = document.getElementById('signup-btn');
    const viewTransactionsBtn = document.getElementById('view-transactions-btn');
    const viewSettingsBtn = document.getElementById('view-settings-btn');
    const viewAnalysisBtn = document.getElementById('view-analysis-btn');
    const backToLoginBtn = document.getElementById('back-to-login-btn');
    const transactionsTable = document.getElementById('transactions-table');
    const backButtons = document.querySelectorAll('.back-btn');
    const logoutButtons = document.querySelectorAll('.logout-btn');
    const profileForm = document.getElementById('profile-form');

    let users = []; // User storage
    let loggedInUser = null; // Currently logged-in user reference

    // Variables to store chart instances
    let transactionTypeDoughnutChartInstance = null;
    let quantityLineChartInstance = null;
    let accumulatedQuantityLineChartInstance = null;
    let quantityByItemBarChartInstance = null;
    let quantityByLocationBarChartInstance = null;
    let quantityByItemWithTransactionTypeInstance = null;
    let quantityByLocationWithTransactionTypeInstance = null;

    function showScreen(screen) {
        const screens = [loginScreen, signupScreen, mainMenuScreen, transactionsScreen,
settingsScreen, analysisScreen];
        screens.forEach(s => s.classList.remove('active'));
        screen.classList.add('active');
    }

    // Function to fetch data from Google Sheets
    function fetchTransactions() {
        const apiKey = 'AIzaSyAVm_K-H1nRU_Ve2VqwpqD13H4rQTaT3FU'; // Your API key
        const sheetId = '1k_TPTjTE1NPgLFCgsfsV1zjbSuNwh92qn3erodl_5bE'; // Your Google
Sheet ID
        const range = 'Transactions!A:G'; // Range includes columns A to G
        const timestamp = new Date().getTime(); // Prevent caching
        const url = `https://sheets.googleapis.com/v4/spreadsheets/${sheetId}/values/${range}?key=${apiKey}&t=${timestamp}`;

        return new Promise((resolve, reject) => {
            const xhr = new XMLHttpRequest();
            xhr.open('GET', url, true);
            xhr.onreadystatechange = function () {
                if (xhr.readyState === 4) {
                    if (xhr.status === 200) {
                        resolve(JSON.parse(xhr.responseText).values);
                    } else {
                        reject(new Error(`Error fetching data: ${xhr.statusText}`));
                    }
                }
            };
            xhr.send();
        });
    }

    // Function to populate transactions table
    async function populateTransactions() {
        try {
            const transactions = await fetchTransactions();
            transactionsTable.innerHTML = ''; // Clear the table

            if (transactions && transactions.length > 0) {
                transactions.slice(1).forEach(t => {
                    const row = document.createElement('tr');
                    row.innerHTML = `
```

```
                        <td>${t[0]}</td> <!-- Date -->
                        <td>${t[1]}</td> <!-- Location -->
                        <td>${t[2]}</td> <!-- Item -->
                        <td>${t[3]}</td> <!-- Transaction Type -->
                        <td>${t[4]}</td> <!-- Serial Number -->
                        <td>${t[5]}</td> <!-- Manufacture Date -->
                        <td>${t[6]}</td> <!-- Expiry Date -->
                    `;
                    transactionsTable.appendChild(row);
                });
            } else {
                const row = document.createElement('tr');
                row.innerHTML = `<td colspan="7">No transactions found.</td>`;
                transactionsTable.appendChild(row);
            }
        } catch (error) {
            alert('Error fetching transactions: ' + error.message);
        }
    }

    // Function to destroy existing charts
    function destroyCharts() {
        if (transactionTypeDoughnutChartInstance) {
            transactionTypeDoughnutChartInstance.destroy();
            transactionTypeDoughnutChartInstance = null;
        }
        if (quantityLineChartInstance) {
            quantityLineChartInstance.destroy();
            quantityLineChartInstance = null;
        }
        if (accumulatedQuantityLineChartInstance) {
            accumulatedQuantityLineChartInstance.destroy();
            accumulatedQuantityLineChartInstance = null;
        }
        if (quantityByItemBarChartInstance) {
            quantityByItemBarChartInstance.destroy();
            quantityByItemBarChartInstance = null;
        }
        if (quantityByLocationBarChartInstance) {
            quantityByLocationBarChartInstance.destroy();
            quantityByLocationBarChartInstance = null;
        }
        if (quantityByItemWithTransactionTypeInstance) {
            quantityByItemWithTransactionTypeInstance.destroy();
            quantityByItemWithTransactionTypeInstance = null;
        }
        if (quantityByLocationWithTransactionTypeInstance) {
            quantityByLocationWithTransactionTypeInstance.destroy();
            quantityByLocationWithTransactionTypeInstance = null;
        }
    }

    // Function to render doughnut chart with gradient colors
    async function renderTransactionTypeDoughnutChart() {
        const transactions = await fetchTransactions();
        const data = transactions.slice(1); // Remove header row

        const transactionTypes = data.map(t => t[3]); // Transaction Type column
        const typeCounts = {};
        transactionTypes.forEach(type => {
            typeCounts[type] = (typeCounts[type] || 0) + 1;
        });

                                                                const        ctx        =
document.getElementById('transactionTypeDoughnutChart').getContext('2d');

        // Create gradient colors
        const gradientColors = [
              createGradient(ctx, '#36A2EB', '#4BC0C0'), // Blue to Mint gradient for Check
In
             createGradient(ctx, '#FF6384', '#FF9F40')  // Red to Orange gradient for Check
Out
        ];

        // Store the chart instance
        transactionTypeDoughnutChartInstance = new Chart(ctx, {
            type: 'doughnut',
```

```
            data: {
                labels: Object.keys(typeCounts),
                datasets: [{
                    data: Object.values(typeCounts),
                    backgroundColor: gradientColors, // Use gradient colors
                    borderWidth: 1
                }]
            },
            options: {
                responsive: true,
                plugins: {
                    title: {
                        display: true,
                        text: 'Transaction Type Distribution'
                    }
                }
            }
        });
    }

    // Helper function to create gradient colors
    function createGradient(ctx, color1, color2) {
        const gradient = ctx.createLinearGradient(0, 0, 0, 400); // Vertical gradient
        gradient.addColorStop(0, color1); // Start color
        gradient.addColorStop(1, color2); // End color
        return gradient;
    }

    // Function to render the quantity line chart
    async function renderQuantityLineChart() {
        const transactions = await fetchTransactions();
        const data = transactions.slice(1); // Remove header row

        // Group data by Manufacture Date for Check In and Check Out
        const dateCheckInMap = {};
        const dateCheckOutMap = {};

        data.forEach(transaction => {
            const date = transaction[5]; // Manufacture Date column
            const transactionType = transaction[3]; // Transaction Type column

            if (!date) {
                console.warn('Missing Manufacture Date for transaction:', transaction);
                return; // Skip transactions with missing dates
            }

            if (transactionType === "Check In") {
                if (!dateCheckInMap[date]) {
                    dateCheckInMap[date] = 0;
                }
                dateCheckInMap[date] += 1; // Increment quantity for Check In
            } else if (transactionType === "Check Out") {
                if (!dateCheckOutMap[date]) {
                    dateCheckOutMap[date] = 0;
                }
                dateCheckOutMap[date] += 1; // Increment quantity for Check Out
            }
        });

        // Sort dates in ascending order
        const sortedDates = Object.keys({ ...dateCheckInMap, ...dateCheckOutMap }).sort((a,
b) => new Date(a) - new Date(b));

        // Prepare data for the chart
        const labels = sortedDates; // X-axis: Manufacture Dates
          const checkInQuantities = sortedDates.map(date => dateCheckInMap[date] || 0); //
Y-axis: Check In Quantities
         const checkOutQuantities = sortedDates.map(date => dateCheckOutMap[date] || 0); //
Y-axis: Check Out Quantities

        const ctx = document.getElementById('quantityLineChart').getContext('2d');

        // Store the chart instance
        quantityLineChartInstance = new Chart(ctx, {
            type: 'line',
            data: {
                labels: labels,
```

```
            datasets: [
                {
                    label: 'Quantity Checked In',
                    data: checkInQuantities,
                    borderColor: '#36A2EB', // Blue for Check In
                    backgroundColor: 'rgba(54, 162, 235, 0.2)', // Light blue fill
                    borderWidth: 2,
                    fill: true
                },
                {
                    label: 'Quantity Checked Out',
                    data: checkOutQuantities,
                    borderColor: '#FF6384', // Red for Check Out
                    backgroundColor: 'rgba(255, 99, 132, 0.2)', // Light red fill
                    borderWidth: 2,
                    fill: true
                }
            ]
        },
        options: {
            responsive: true,
            plugins: {
                title: {
                    display: true,
                    text: 'Quantity Checked In and Out Over Time'
                }
            },
            scales: {
                x: {
                    title: {
                        display: true,
                        text: 'Manufacture Date'
                    },
                    ticks: {
                        maxRotation: 45,
                        minRotation: 45,
                        autoSkip: false,
                        padding: 5
                    }
                },
                y: {
                    title: {
                        display: true,
                        text: 'Quantity'
                    },
                    beginAtZero: true
                }
            }
        }
    });
}

// Function to render the accumulated quantity line chart
async function renderAccumulatedQuantityLineChart() {
    const transactions = await fetchTransactions();
    const data = transactions.slice(1); // Remove header row

    // Group data by Date for Check In and Check Out
    const dateCheckInMap = {};
    const dateCheckOutMap = {};

    data.forEach(transaction => {
        const date = transaction[0]; // Date column (column A)
        const transactionType = transaction[3]; // Transaction Type column

        if (!date) {
            console.warn('Missing Date for transaction:', transaction);
            return; // Skip transactions with missing dates
        }

        if (transactionType === "Check In") {
            if (!dateCheckInMap[date]) {
                dateCheckInMap[date] = 0;
            }
            dateCheckInMap[date] += 1; // Increment quantity for Check In
        } else if (transactionType === "Check Out") {
            if (!dateCheckOutMap[date]) {
```

```
                    dateCheckOutMap[date] = 0;
                }
                dateCheckOutMap[date] += 1; // Increment quantity for Check Out
            }
        });

        // Sort dates in ascending order
        const sortedDates = Object.keys({ ...dateCheckInMap, ...dateCheckOutMap }).sort((a,
b) => new Date(a) - new Date(b));

        // Calculate accumulated quantities
        let accumulatedCheckIn = 0;
        let accumulatedCheckOut = 0;
        const accumulatedCheckInQuantities = [];
        const accumulatedCheckOutQuantities = [];

        sortedDates.forEach(date => {
            accumulatedCheckIn += dateCheckInMap[date] || 0;
            accumulatedCheckOut += dateCheckOutMap[date] || 0;
            accumulatedCheckInQuantities.push(accumulatedCheckIn);
            accumulatedCheckOutQuantities.push(accumulatedCheckOut);
        });

                                                                const        ctx        =
document.getElementById('accumulatedQuantityLineChart').getContext('2d');

        // Store the chart instance
        accumulatedQuantityLineChartInstance = new Chart(ctx, {
            type: 'line',
            data: {
                labels: Array.from({ length: sortedDates.length }, (_, i) => i + 1), // Use
index as labels (1, 2, 3, ...)
                datasets: [
                    {
                        label: 'Accumulated Check In',
                        data: accumulatedCheckInQuantities,
                        borderColor: '#4BC0C0', // Teal for Accumulated Check In
                        backgroundColor: 'rgba(75, 192, 192, 0.2)', // Light teal fill
                        borderWidth: 2,
                        fill: true
                    },
                    {
                        label: 'Accumulated Check Out',
                        data: accumulatedCheckOutQuantities,
                        borderColor: '#FF9F40', // Orange for Accumulated Check Out
                        backgroundColor: 'rgba(255, 159, 64, 0.2)', // Light orange fill
                        borderWidth: 2,
                        fill: true
                    }
                ]
            },
            options: {
                responsive: true,
                plugins: {
                    title: {
                        display: true,
                        text: 'Accumulated Quantity Checked In and Out Over Time'
                    }
                },
                scales: {
                    x: {
                        title: {
                            display: true,
                            text: 'Transaction Sequence'
                        },
                        ticks: {
                            display: false // Hide x-axis labels
                        }
                    },
                    y: {
                        title: {
                            display: true,
                            text: 'Accumulated Quantity'
                        },
                        beginAtZero: true
                    }
                }
```

```
        }
    });
}

// Function to render the quantity by item bar chart with unique colors
async function renderQuantityByItemBarChart() {
    const transactions = await fetchTransactions();
    const data = transactions.slice(1); // Remove header row

    // Group data by Item
    const itemCounts = {};
    data.forEach(transaction => {
        const item = transaction[2]; // Item column
        if (!item) return; // Skip if item is missing
        itemCounts[item] = (itemCounts[item] || 0) + 1; // Increment count for the item
    });

    const ctx = document.getElementById('quantityByItemBarChart').getContext('2d');

    // Define a color palette for items
    const itemColors = [
        '#36A2EB', // Blue
        '#FF6384', // Red
        '#4BC0C0', // Teal
        '#FF9F40', // Orange
        '#9966FF', // Purple
        '#00CC99', // Green
        '#FF66B2', // Pink
    ];

    // Assign a unique color to each item
    const backgroundColor = Object.keys(itemCounts).map((_, index) => {
        return itemColors[index % itemColors.length]; // Cycle through the palette
    });

    // Store the chart instance
    quantityByItemBarChartInstance = new Chart(ctx, {
        type: 'bar',
        data: {
            labels: Object.keys(itemCounts), // Item names
            datasets: [{
                label: 'Quantity by Item',
                data: Object.values(itemCounts), // Quantities
                backgroundColor: backgroundColor, // Use unique colors
                borderWidth: 1
            }]
        },
        options: {
            responsive: true,
            plugins: {
                title: {
                    display: true,
                    text: 'Quantity by Item'
                }
            },
            scales: {
                x: {
                    title: {
                        display: true,
                        text: 'Item'
                    }
                },
                y: {
                    title: {
                        display: true,
                        text: 'Quantity'
                    },
                    beginAtZero: true
                }
            }
        }
    });
}

// Function to render the quantity by location bar chart with unique colors
async function renderQuantityByLocationBarChart() {
    const transactions = await fetchTransactions();
```

```
        const data = transactions.slice(1); // Remove header row

        // Group data by Location
        const locationCounts = {};
        data.forEach(transaction => {
            const location = transaction[1]; // Location column
            if (!location) return; // Skip if location is missing
              locationCounts[location] = (locationCounts[location] || 0) + 1; // Increment
count for the location
        });

        const ctx = document.getElementById('quantityByLocationBarChart').getContext('2d');

        // Define a color palette for locations
        const locationColors = [
            '#00CC99', // Green
            '#9966FF', // Purple
            '#FF66B2', // Pink
            '#36A2EB', // Blue
            '#FF6384', // Red
            '#4BC0C0', // Teal
            '#FF9F40', // Orange
        ];

        // Assign a unique color to each location
        const backgroundColor = Object.keys(locationCounts).map((_, index) => {
                return locationColors[index % locationColors.length]; // Cycle through the
palette
        });

        // Store the chart instance
        quantityByLocationBarChartInstance = new Chart(ctx, {
            type: 'bar',
            data: {
                labels: Object.keys(locationCounts), // Location names
                datasets: [{
                    label: 'Quantity by Location',
                    data: Object.values(locationCounts), // Quantities
                    backgroundColor: backgroundColor, // Use unique colors
                    borderWidth: 1
                }]
            },
            options: {
                responsive: true,
                plugins: {
                    title: {
                        display: true,
                        text: 'Quantity by Location'
                    }
                },
                scales: {
                    x: {
                        title: {
                            display: true,
                            text: 'Location'
                        }
                    },
                    y: {
                        title: {
                            display: true,
                            text: 'Quantity'
                        },
                        beginAtZero: true
                    }
                }
            }
        });
    }

    // Function to render the quantity by item bar chart with Check In/Check Out split
    async function renderQuantityByItemWithTransactionType() {
        const transactions = await fetchTransactions();
        const data = transactions.slice(1); // Remove header row

        // Group data by Item and Transaction Type
        const itemCheckInCounts = {};
        const itemCheckOutCounts = {};
```

```
        data.forEach(transaction => {
            const item = transaction[2]; // Item column
            const transactionType = transaction[3]; // Transaction Type column

            if (!item) return; // Skip if item is missing

            if (transactionType === "Check In") {
                if (!itemCheckInCounts[item]) {
                    itemCheckInCounts[item] = 0;
                }
                itemCheckInCounts[item] += 1; // Increment count for Check In
            } else if (transactionType === "Check Out") {
                if (!itemCheckOutCounts[item]) {
                    itemCheckOutCounts[item] = 0;
                }
                itemCheckOutCounts[item] += 1; // Increment count for Check Out
            }
        });

                                                                const        ctx        =
document.getElementById('quantityByItemWithTransactionType').getContext('2d');

        // Define a color palette for Check In and Check Out
        const checkInColor = '#36A2EB'; // Blue for Check In
        const checkOutColor = '#FF6384'; // Red for Check Out

        // Get all unique items
        const items = Object.keys({ ...itemCheckInCounts, ...itemCheckOutCounts });

        // Store the chart instance
        quantityByItemWithTransactionTypeInstance = new Chart(ctx, {
            type: 'bar',
            data: {
                labels: items, // Item names
                datasets: [
                    {
                        label: 'Check In',
                                data: items.map(item => itemCheckInCounts[item] || 0), //
Quantities for Check In
                        backgroundColor: checkInColor,
                        borderWidth: 1
                    },
                    {
                        label: 'Check Out',
                               data: items.map(item => itemCheckOutCounts[item] || 0), //
Quantities for Check Out
                        backgroundColor: checkOutColor,
                        borderWidth: 1
                    }
                ]
            },
            options: {
                responsive: true,
                plugins: {
                    title: {
                        display: true,
                        text: 'Quantity by Item (Check In vs Check Out)'
                    }
                },
                scales: {
                    x: {
                        title: {
                            display: true,
                            text: 'Item'
                        }
                    },
                    y: {
                        title: {
                            display: true,
                            text: 'Quantity'
                        },
                        beginAtZero: true
                    }
                }
            }
        });
```

```
    }

    // Function to render the quantity by location bar chart with Check In/Check Out split
    async function renderQuantityByLocationWithTransactionType() {
        const transactions = await fetchTransactions();
        const data = transactions.slice(1); // Remove header row

        // Group data by Location and Transaction Type
        const locationCheckInCounts = {};
        const locationCheckOutCounts = {};

        data.forEach(transaction => {
            const location = transaction[1]; // Location column
            const transactionType = transaction[3]; // Transaction Type column

            if (!location) return; // Skip if location is missing

            if (transactionType === "Check In") {
                if (!locationCheckInCounts[location]) {
                    locationCheckInCounts[location] = 0;
                }
                locationCheckInCounts[location] += 1; // Increment count for Check In
            } else if (transactionType === "Check Out") {
                if (!locationCheckOutCounts[location]) {
                    locationCheckOutCounts[location] = 0;
                }
                locationCheckOutCounts[location] += 1; // Increment count for Check Out
            }
        });

                                                                const        ctx        =
document.getElementById('quantityByLocationWithTransactionType').getContext('2d');

        // Define a color palette for Check In and Check Out
        const checkInColor = '#36A2EB'; // Blue for Check In
        const checkOutColor = '#FF6384'; // Red for Check Out

        // Get all unique locations
        const locations = Object.keys({ ...locationCheckInCounts, ...locationCheckOutCounts
});

        // Store the chart instance
        quantityByLocationWithTransactionTypeInstance = new Chart(ctx, {
            type: 'bar',
            data: {
                labels: locations, // Location names
                datasets: [
                    {
                        label: 'Check In',
                         data: locations.map(location => locationCheckInCounts[location] ||
0), // Quantities for Check In
                        backgroundColor: checkInColor,
                        borderWidth: 1
                    },
                    {
                        label: 'Check Out',
                        data: locations.map(location => locationCheckOutCounts[location] ||
0), // Quantities for Check Out
                        backgroundColor: checkOutColor,
                        borderWidth: 1
                    }
                ]
            },
            options: {
                responsive: true,
                plugins: {
                    title: {
                        display: true,
                        text: 'Quantity by Location (Check In vs Check Out)'
                    }
                },
                scales: {
                    x: {
                        title: {
                            display: true,
                            text: 'Location'
                        }
```

```
                    },
                    y: {
                        title: {
                            display: true,
                            text: 'Quantity'
                        },
                        beginAtZero: true
                    }
                }
            }
        });
    }

    loginBtn.addEventListener('click', () => {
        const email = document.getElementById('email-login').value.trim();
        const password = document.getElementById('password-login').value.trim();
        const user = users.find(u => u.email === email && u.password === password);
        if (user) {
            loggedInUser = user;
            showScreen(mainMenuScreen);
        } else {
            alert('Invalid login credentials. Please try again.');
        }
    });

             document.getElementById('show-signup-btn').addEventListener('click',    ()    =>
showScreen(signupScreen));
    signupBtn.addEventListener('click', () => {
        const username = document.getElementById('username-signup').value.trim();
        const email = document.getElementById('email-signup').value.trim();
        const password = document.getElementById('password-signup').value.trim();

        if (!username || !email || !password) {
            alert('All fields are required.');
            return;
        }

        if (users.some(u => u.email === email)) {
            alert('An account with this email already exists.');
        } else {
            users.push({ username, email, password });
            alert('Account created successfully!');
            showScreen(loginScreen);
        }
    });

    backToLoginBtn.addEventListener('click', () => showScreen(loginScreen));
    viewTransactionsBtn.addEventListener('click', () => {
        populateTransactions();
        showScreen(transactionsScreen);
    });

    viewSettingsBtn.addEventListener('click', () => {
        if (loggedInUser) {
            document.getElementById('username').value = loggedInUser.username;
            document.getElementById('password').value = loggedInUser.password;
            document.getElementById('role').value = loggedInUser.role || '';
            document.getElementById('country').value = loggedInUser.country || '';
        }
        showScreen(settingsScreen);
    });

    viewAnalysisBtn.addEventListener('click', () => {
        destroyCharts(); // Destroy existing charts
        renderTransactionTypeDoughnutChart();
        renderQuantityLineChart();
        renderAccumulatedQuantityLineChart();
        renderQuantityByItemBarChart();
        renderQuantityByLocationBarChart();
        renderQuantityByItemWithTransactionType(); // Render the new bar chart
        renderQuantityByLocationWithTransactionType(); // Render the new bar chart
        showScreen(analysisScreen);
    });

              backButtons.forEach(button    =>    button.addEventListener('click',    ()    =>
showScreen(mainMenuScreen)));
    logoutButtons.forEach(button => button.addEventListener('click', () => {
```

```
        alert('You have been logged out.');
        loggedInUser = null;
        showScreen(loginScreen);
    }));

    profileForm.addEventListener('submit', (event) => {
        event.preventDefault();
        if (loggedInUser) {
            loggedInUser.username = document.getElementById('username').value.trim();
            loggedInUser.password = document.getElementById('password').value.trim();
            loggedInUser.role = document.getElementById('role').value.trim();
            loggedInUser.country = document.getElementById('country').value.trim();
            alert('Profile updated successfully!');
        } else {
            alert('Error: No user is currently logged in.');
        }
    });
});
```

FIGURE A.4: JAVASCRIPT FOR HANDLING DYNAMIC INTERACTIONS BETWEEN FRONTEND AND BACKEND SYSTEMS.

Google AppScript for Transaction Identification:

```
function onEdit(e) {
  updateTransactionDetails();
}

function onOpen() {
  updateTransactionDetails();
}

function updateTransactionDetails() {
  var sheet = SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();
  var lastRow = sheet.getLastRow();
  var range = sheet.getRange(2, 4, lastRow - 1, 1); // Column D (Transaction detail)

  var serialCounts = {};

  for (var i = 2; i <= lastRow; i++) {
    var serialNumber = sheet.getRange(i, 5).getValue(); // Column E (Serial Number)
    if (serialNumber) {
      if (!serialCounts[serialNumber]) {
        serialCounts[serialNumber] = 0;
      }
      serialCounts[serialNumber]++;
      var transactionDetail = (serialCounts[serialNumber] % 2 === 0) ? "Check Out" : "Check
In";
      sheet.getRange(i, 4).setValue(transactionDetail); // Column D (Transaction detail)
      Logger.log("Updated row " + i + " with transaction detail: " + transactionDetail);
    } else {
      sheet.getRange(i, 4).setValue(""); // Column D (Transaction detail)
      Logger.log("Cleared transaction detail for row " + i);
    }
  }
}

function manualTrigger() {
  updateTransactionDetails();
}
```

FIGURE A.5: APPS SCRIPT FOR RECOGNIZING THE TRANSACTION TYPE OF THE INVENTORY.

Python for Machine Learning:

```
import tensorflow as tf
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
from tensorflow.keras.preprocessing import image

# Install Pillow if not already installed
try:
    from PIL import Image
    print("Pillow is already installed.")
except ImportError:
    print("Installing Pillow...")
    import subprocess
    subprocess.check_call(["python", '-m', 'pip', 'install', 'pillow'])
    from PIL import Image
    print("Pillow installed successfully.")

# Data augmentation and preprocessing
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2
)

train_generator = train_datagen.flow_from_directory(
    'dataset',
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    subset='training'
)

validation_generator = train_datagen.flow_from_directory(
    'dataset',
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    subset='validation'
)

# Load the pre-trained MobileNetV2 model
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Freeze the base model layers
for layer in base_model.layers:
    layer.trainable = False

# Add custom layers on top of the base model
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(1, activation='sigmoid')(x)  # Binary classification

# Create the final model
model = Model(inputs=base_model.input, outputs=predictions)

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001),             loss='binary_crossentropy',
metrics=['accuracy'])

# Train the model
model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // validation_generator.batch_size,
    epochs=10
)
```

```python
# Save the model
model.save('defect_detection_model.h5')

# Load the model
model = tf.keras.models.load_model('defect_detection_model.h5')

# Function to predict if an image is good or defective
def predict_image(img_path):
    img = image.load_img(img_path, target_size=(224, 224))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.0  # Normalize the image

    prediction = model.predict(img_array)
    if prediction < 0.5:
        print("Defective")
    else:
        print("Good")

# Test on a new image
predict_image('test_images/test_img1.jpg')
```

FIGURE A.6: PYTHON SCRIPT TO PERFORM MACHINE LEARNING TO DETECT INVENTORY DEFECTION.

Arduino C++ for NFC Protocol:

```
#include <WiFi.h>
#include <SPI.h>
#include <Adafruit_PN532.h>
#include <ESP_Google_Sheet_Client.h>
#include <stdlib.h>
#include "time.h"
#include <LiquidCrystal.h>
#include <GS_SDHelper.h>

#define WIFI_SSID "your Wi-Fi SSID"
#define WIFI_PASSWORD "your Wi-Fi password"

#define PROJECT_ID "your Project ID"
#define CLIENT_EMAIL "your Client Email"
#define USER_EMAIL "your User Email"

char numberArray[20];

// Service Account's private key
const char PRIVATE_KEY[] PROGMEM = "-----BEGIN PRIVATE KEY-----Your Private Key-----END
PRIVATE KEY-----\n";
// Define pins for the PN532 NFC module using I2C protocol
#define SDA_PIN 21
#define SCL_PIN 22
#define PN532_IRQ (2)
#define PN532_RESET (3)

// Initialize NFC reader
Adafruit_PN532 nfc(SDA_PIN, SCL_PIN);
// The ID of the spreadsheet to store the data
const char spreadsheetId[] = "your spreadsheet ID";

// Define the pins for the LCD
LiquidCrystal lcd(12, 14, 27, 26, 25, 33);

// Define pins for UART communication between ESP32 and ESP32 CAM
#define ESP32_TX_PIN 17  // ESP32 TX pin
#define ESP32_RX_PIN 16  // ESP32 RX pin

HardwareSerial mySerial(2);  // Use UART2 for ESP32

// Timer variables
unsigned long lastTime = 0;
unsigned long timerDelay = 30000;
// Token Callback function
void tokenStatusCallback(TokenInfo info);

// NTP server to request epoch time
const char* ntpServer = "pool.ntp.org";

// Function to check if PN532 is connected
bool isPN532Connected() {
  uint32_t versiondata = nfc.getFirmwareVersion();
  if (!versiondata) {
    return false;
  }
  return true;
}

// Function to initialize PN532
void initializePN532() {
  nfc.begin();
  uint32_t versiondata = 0;
  while (!versiondata) {
    versiondata = nfc.getFirmwareVersion();
    if (!versiondata) {
      delay(500);
    }
  }
  Serial.println("Found PN532 NFC Reader!");
}

void setup() {
  Serial.begin(115200);
```

```
  mySerial.begin(9600, SERIAL_8N1, ESP32_RX_PIN, ESP32_TX_PIN);  // Start UART2 with 9600
baud rate

  // Initialize PN532 NFC reader
  initializePN532();

  // Connect to Wi-Fi
  WiFi.setAutoReconnect(true);
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);

  Serial.print("Connecting to Wi-Fi");
  int connecting_process_timed_out = 10;
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(1000);
    if (connecting_process_timed_out-- == 0) {
      Serial.println("\nFailed to connect to Wi-Fi");
      ESP.restart();
    }
  }
  // Configure time
  const long gmtOffset_sec = 0;
  configTime(gmtOffset_sec, 0, ntpServer);

  GSheet.printf("ESP Google Sheet Client v%s\n\n", ESP_GOOGLE_SHEET_CLIENT_VERSION);

  // Set the callback for Google API access token generation status
  GSheet.setTokenCallback(tokenStatusCallback);

  // Set the seconds to refresh the auth token before expire
  GSheet.setPrerefreshSeconds(10 * 60);

  // Begin the access token generation for Google API authentication
  GSheet.begin(CLIENT_EMAIL, PROJECT_ID, PRIVATE_KEY);

  // Initialize the LCD
  lcd.begin(20, 4);  // 20 columns, 4 rows
  lcd.clear();
  delay(1000);
  mySerial.write('D');  //Send signal to ESP32 CAM in standby mode
}

void loop() {
  // Call ready() repeatedly in loop for authentication checking and processing
  bool ready = GSheet.ready();

  if (ready && millis() - lastTime > timerDelay) {
    lastTime = millis();

    FirebaseJson response;

    Serial.println("\nReading NFC card and sending data...");
    Serial.println("----------------------------");
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Place NFC card or");
    lcd.setCursor(0, 1);
    lcd.print("tag on the reader");

    FirebaseJsonData result;

    FirebaseJson valueRange;

    String location = "";
    String item = "";
    String ManufacturedDate = "";
    String ExpiryDate = "";
    String serialNumber = "";
    String blockData = "";
    String datastring = "";

    // Retry loop to read NFC card
    while (true) {
      // NFC card detection and UID read
      uint8_t success;
      uint8_t uid[] = { 0, 0, 0, 0, 0, 0, 0 };  // Buffer to store the UID
      uint8_t uidLength;  // Length of the UID (4 or 7 bytes depending on the card)
```

```
      uint8_t data[16];  // Array to store the block data
         uint8_t  keyA[6]  =  {  0xFF,  0xFF,  0xFF,  0xFF,  0xFF,  0xFF  };   // Universal key A
(replace with your actual key if needed)

      // Wait for an NFC tag (ISO14443A type, e.g., Mifare or NTAG)
      while (!success){
        success = nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A, uid, &uidLength, 15000);
        if(success){
          lcd.clear();
          lcd.setCursor(0, 0);
          lcd.print("Reading Data...");
        }
        if (!isPN532Connected()) {
          Serial.println("PN532 not found. Reinitializing...");
          lcd.clear();
          lcd.setCursor(0, 0);
          lcd.print("Reinitializing NFC reader");
          initializePN532();
          delay(1000); // Give some time for reinitialization
          lcd.clear();
          lcd.setCursor(0, 0);
          lcd.print("Place NFC card or");
          lcd.setCursor(0, 1);
          lcd.print("tag on the reader");
        }
      }

      // Convert the UID (byte array) to a string
      serialNumber = "";
      for (int i = 0; i < sizeof(uid); i++) {
        serialNumber += String(uid[i], HEX);  // Convert each byte to hexadecimal
      }

      if (success) {
        // Handle Mifare Cards (UID length 4 bytes or 7 bytes)
        if (uidLength == 4 || uidLength == 7) {
          // Handle Mifare Classic Card (UID length == 4)
          if (uidLength == 4) {
            for (uint8_t block = 4; block < 10; block++) {
              // Authenticate the block using Key A
                success = nfc.mifareclassic_AuthenticateBlock(uid, uidLength, block, 1,
keyA);

              if (success) {
                // Read the block data after successful authentication
                success = nfc.mifareclassic_ReadDataBlock(block, data);
                if (success) {
                  for (int i = 0; i < 16; i++) {
                       if ((47 <= data[i] && data[i] <= 57) || (65 <= data[i] && data[i] <=
90) || (97 <= data[i] && data[i] <= 122)) {  // Printable ASCII characters range
                         blockData += (char)data[i];
                     }
                  }
                }
              }
            }
            location = blockData.substring(2, 6);
            delay(500);
            ManufacturedDate = blockData.substring(7, 18);
            delay(500);
            ExpiryDate = blockData.substring(19, 30);
            delay(500);
            item = blockData.substring(32, 37);
            delay(500);
          }
          // Handle NTAG Cards (UID length == 7)
          else if (uidLength == 7) {
            uint8_t data[32];
            datastring = "";
            for (uint8_t i = 4; i < 18; i++) {
              success = nfc.ntag2xx_ReadPage(i, data);
              if (success) {
                for (int j = 0; j < 4; j++) {
                   if ((47 <= data[j] && data[j] <= 57) || (65 <= data[j] && data[j] <= 90)
|| (97 <= data[j] && data[j] <= 122)) {
                     datastring += (char)data[j];  // Only append printable ASCII characters
                  }
```

```
                }
              }
            }
            location = datastring.substring(2, 6);
            delay(500);
            ManufacturedDate = datastring.substring(7, 18);
            delay(500);
            ExpiryDate = datastring.substring(19, 30);
            delay(500);
            item = datastring.substring(32, 37);
            delay(500);
          }
        }
      }
      if(location.length() == 4 && item.length() == 5 && ManufacturedDate.length() == 11 &&
ExpiryDate.length() == 11){
        break;
      }
      else{
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Fail to read data");
        lcd.setCursor(0, 2);
        lcd.print("Try again");
        delay(1000);
      }
    }

    struct tm timeinfo;
    char dateStr[50];
    String asString;
    while (!getLocalTime(&timeinfo)) {
      Serial.println("Failed to obtain time");
    }
    timeinfo.tm_hour += 8;
    mktime(&timeinfo);
    strftime(dateStr, sizeof(dateStr), "%A,%d %B %Y,%H:%M:%S", &timeinfo);
    asString = dateStr;
    asString.replace(" ", "-");

    // Create an array for the row values
    FirebaseJsonArray rowValues;
    rowValues.add(asString.c_str());
    rowValues.add(location.c_str());
    rowValues.add(item.c_str());
    rowValues.add(""); // Empty string for column D
    rowValues.add(serialNumber.c_str());
    rowValues.add(ManufacturedDate.c_str());
    rowValues.add(ExpiryDate.c_str());

    // Create the "values" array containing this row
    FirebaseJsonArray valuesArray;
    valuesArray.add(rowValues);

    // Create the valueRange JSON object
    valueRange.set("majorDimension", "ROWS");
    valueRange.set("values", valuesArray);

    // Append the data to the sheet
     bool successSheet = GSheet.values.append(&response, spreadsheetId, "Transactions!A:G",
&valueRange);
    if (successSheet) {
      response.toString(Serial, true);
      valueRange.clear();
      delay(1000);
      Serial.print("\ndone");
      // Display location and item on the LCD
      lcd.clear();  // Clear the screen before displaying new data
      lcd.setCursor(0, 0);  // Set cursor to the first row, first column
      lcd.print("Location: " + location);  // Display the location on the first row
      lcd.setCursor(0, 1);  // Set cursor to the second row, first column
      lcd.print("Item: " + item);  // Display the item on the second row
      lcd.setCursor(0, 2);
      lcd.print("Data Sent");
      delay(2000);

      lcd.clear();
```

```
      lcd.setCursor(0, 0);
      lcd.print("Ready to");
      lcd.setCursor(0, 1);
      lcd.print("Capture Image");
      delay(5000);

      // Send the trigger command to ESP32-CAM to capture the photo
      mySerial.write('T');  // Send trigger signal to ESP32-CAM

      // Wait for a response from the ESP32-CAM (either failure 'F' or success 'S')
      while (true) {
        if (mySerial.available()) {
          char command = mySerial.read();  // Read the incoming UART signal from ESP32-CAM

          if (command == 'F') {  // If failure signal is received
            Serial.println("Capture failed. Sending retry signal to ESP32-CAM...");
            lcd.clear();
            lcd.setCursor(0, 0);
            lcd.print("Fail to capture");
            lcd.setCursor(0, 1);
            lcd.print("Retry");
            mySerial.write('R');  // Send retry signal to ESP32-CAM
          }

          if (command == 'S') {  // If success signal is received
            Serial.println("Capture successful. Stopping process.");
            mySerial.write('D');
            lcd.clear();
            lcd.setCursor(0, 0);
            lcd.print("Image is captured");
            break;  // Exit the loop, capture is successful
          }
        }
        else{
          delay(3000);
          mySerial.write('D');
          lcd.clear();
          lcd.setCursor(0, 0);
          lcd.print("Image is captured");
          break;  // Exit the loop, capture is successful
        }
      }
      delay(2000); // Give some delay before the next read
    } else {
      Serial.println(GSheet.errorReason());
    }

    Serial.println();
    Serial.println(ESP.getFreeHeap());
    delay(100);
  }
}
void tokenStatusCallback(TokenInfo info) {
  if (info.status == token_status_error) {
                GSheet.printf("Token    info:    type    =    %s,    status    =    %s\n",
GSheet.getTokenType(info).c_str(), GSheet.getTokenStatus(info).c_str());
    GSheet.printf("Token error: %s\n", GSheet.getTokenError(info).c_str());
  } else {
                GSheet.printf("Token    info:    type    =    %s,    status    =    %s\n",
GSheet.getTokenType(info).c_str(), GSheet.getTokenStatus(info).c_str());
  }
}
```

FIGURE A.7: ARDUINO IDE SCRIPT FOR ESP32 TO EXTRACT DATA FROM NFC READER AND SEND TO GOOGLE SHEET.

Arduino C++ for ESP32 Camera:

```
#include "esp_camera.h"
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include "soc/soc.h"
#include "soc/rtc_cntl_reg.h"
#include "Base64.h"

#define CAMERA_MODEL_AI_THINKER
#include "camera_pins.h"

// WiFi credentials
#define WIFI_SSID "your wifi id"
#define WIFI_PASSWORD "your wifi password"

String myDeploymentID = "your deployment id";
String myMainFolderName = "your folder name";

// Initialize WiFiClientSecure
WiFiClientSecure client;

#define FLASH_PIN 4     // Flash pin
#define ESP32_CAM_TX_PIN 1  // UART0 TX pin
#define ESP32_CAM_RX_PIN 3  // UART0 RX pin

HardwareSerial mySerial(0);  // Use UART0 of ESP32 CAM

bool photoCaptured = false;  // Flag to track if the photo has been captured

// Test connection to Google Apps Script
void Test_Con() {
  const char* host = "script.google.com";
  while(1) {
    Serial.println("-----------");
    Serial.println("Connection Test...");
    Serial.println("Connect to " + String(host));

    client.setInsecure();  // Disable SSL certificate verification

    if (client.connect(host, 443)) {
      client.stop();
      break; // Exit the loop
    } else {
      client.stop();
    }

    delay(1000);
  }
}

// Function to capture and send the photo to Google Drive
void SendCapturedPhotos() {
  Test_Con();

  // Capture a photo
  camera_fb_t * fb = NULL;
  fb = esp_camera_fb_get();
  if(!fb) {
    Serial.write('F'); // Send fail signal to ESP32
    delay(1000);
    return;
  }
  digitalWrite(FLASH_PIN, HIGH); // Turn on the flash LED
  delay(500);
  digitalWrite(FLASH_PIN, LOW); // Turn off the flash LED
  mySerial.write('S');  // Send success signal to ESP32

  // Send image to Google Drive
  String url = "/macros/s/" + myDeploymentID + "/exec?folder=" + myMainFolderName;
  client.println("POST " + url + " HTTP/1.1");
  client.println("Host: " + String(host));
  client.println("Transfer-Encoding: chunked");
  client.println();

  int fbLen = fb->len;
```

```
  char *input = (char *)fb->buf;
  int chunkSize = 3 * 1000;
  int chunkBase64Size = base64_enc_len(chunkSize);
  char output[chunkBase64Size + 1];

  int chunk = 0;
  for (int i = 0; i < fbLen; i += chunkSize) {
    int l = base64_encode(output, input, min(fbLen - i, chunkSize));
    client.print(l, HEX);
    client.print("\r\n");
    client.print(output);
    client.print("\r\n");
    delay(100);
    input += chunkSize;
    chunk++;
    if (chunk % 50 == 0) {
      Serial.println();
    }
  }
  client.print("0\r\n");
  client.print("\r\n");

  esp_camera_fb_return(fb); // Free the frame buffer after use

  // Wait for server response
  Serial.println("Waiting for response from Google Apps Script.");
  long int StartTime = millis();
  while (!client.available()) {
    if ((StartTime + 30 * 1000) < millis()) {
      break; // Exit the loop
    }
  }
  // Finished, disconnect the client
  client.stop();
}

// Camera initialization and setup
void configInitCamera() {
  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sccb_sda = SIOD_GPIO_NUM;
  config.pin_sccb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.frame_size = FRAMESIZE_UXGA;
  config.pixel_format = PIXFORMAT_JPEG;
  config.grab_mode = CAMERA_GRAB_WHEN_EMPTY;
  config.fb_location = CAMERA_FB_IN_PSRAM;
  config.jpeg_quality = 12;
  config.fb_count = 1;

  if (config.pixel_format == PIXFORMAT_JPEG) {
    if (psramFound()) {
      config.jpeg_quality = 10;
      config.fb_count = 2;
      config.grab_mode = CAMERA_GRAB_LATEST;
    } else {
      // Limit the frame size when PSRAM is not available
      config.frame_size = FRAMESIZE_SVGA;
      config.fb_location = CAMERA_FB_IN_DRAM;
    }
  }
```

```
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    delay(1000);
    ESP.restart(); // Restart ESP32 CAM
  }

  sensor_t *s = esp_camera_sensor_get();
  if (s->id.PID == OV3660_PID) { // Configure camera settings
    s->set_gain_ctrl(s, 1);
    s->set_exposure_ctrl(s, 1);
    s->set_awb_gain(s, 1);
    s->set_brightness(s, 1);
  }
}

void setup() {
  Serial.begin(115200);
   mySerial.begin(9600, SERIAL_8N1, ESP32_CAM_RX_PIN, ESP32_CAM_TX_PIN);   // Start UART0
with 9600 baud rate

  // Initialize the flashlight pin as an output
  pinMode(FLASH_PIN, OUTPUT);
  digitalWrite(FLASH_PIN, LOW);

  // Connect to Wi-Fi
  WiFi.mode(WIFI_STA);
  WiFi.setAutoReconnect(true);
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  int connecting_process_timed_out = 10;
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
     if (connecting_process_timed_out-- == 0) { // Manually attempt to reconnect after 10
seconds if disconnected
      WiFi.reconnect();
    }
  }

  configInitCamera(); // Initialize the camera

  Test_Con(); // Test connection to Google Apps Script
}

void loop() {
  // Check Wi-Fi connection
  if (WiFi.status() != WL_CONNECTED) {
    WiFi.reconnect();
    delay(5000);  // Wait a bit before trying again
  }
  if (mySerial.available()) {
    char command = mySerial.read();  // Read the UART command from ESP32
    if (command == 'T' && !photoCaptured) {  // If the trigger signal is received
      SendCapturedPhotos();  // Capture and send photo
      photoCaptured = true;  // Mark the photo as captured
    }
    else if (command == 'R' && !photoCaptured){ // If the retry signal is received
      SendCapturedPhotos();
      photoCaptured = true;
    }
    else if (command == 'D'){ // If the done signal is received
      photoCaptured = false;
    }
  }
}
```

FIGURE A.8: ARDUINO IDE SCRIPT FOR ESP32 CAM TO CAPTURE PHOTOS AND SEND TO GOOGLE DRIVE.

Google AppScript for Uploading Image:

```
function doPost(e) {
   const name = Utilities.formatDate(new Date(), 'GMT+8', 'yyyyMMdd-HHmmss') + '.jpg'   //
Change GMT+7 with GMT in your area.
  const subFolderName = Utilities.formatDate(new Date(), 'GMT+8', 'yyyyMMdd')
  const folderName = e.parameters.folder || 'ESP32-CAM'
  const data = Utilities.base64Decode(e.postData.contents)
  const blob = Utilities.newBlob(data, 'image/jpg', name)
  let folder
  const folders = DriveApp.getFoldersByName(folderName)
  if (folders.hasNext()) {
    folder = folders.next()
  } else {
    folder = DriveApp.createFolder(folderName)
  }
  let subFolder
  const subFolders = folder.getFoldersByName(subFolderName)
  if (subFolders.hasNext()) {
    subFolder = subFolders.next()
  } else {
    subFolder = folder.createFolder(subFolderName)
  }
  const file = subFolder.createFile(blob)
  return ContentService.createTextOutput('Done')
}
```

FIGURE A.9: APP SCRIPT TO HANDLE A REQUEST TO SAVE AN UPLOADED IMAGE TO GOOGLE DRIVE.

Appendix 5: Any Relevant Data, Diagram or Drawing

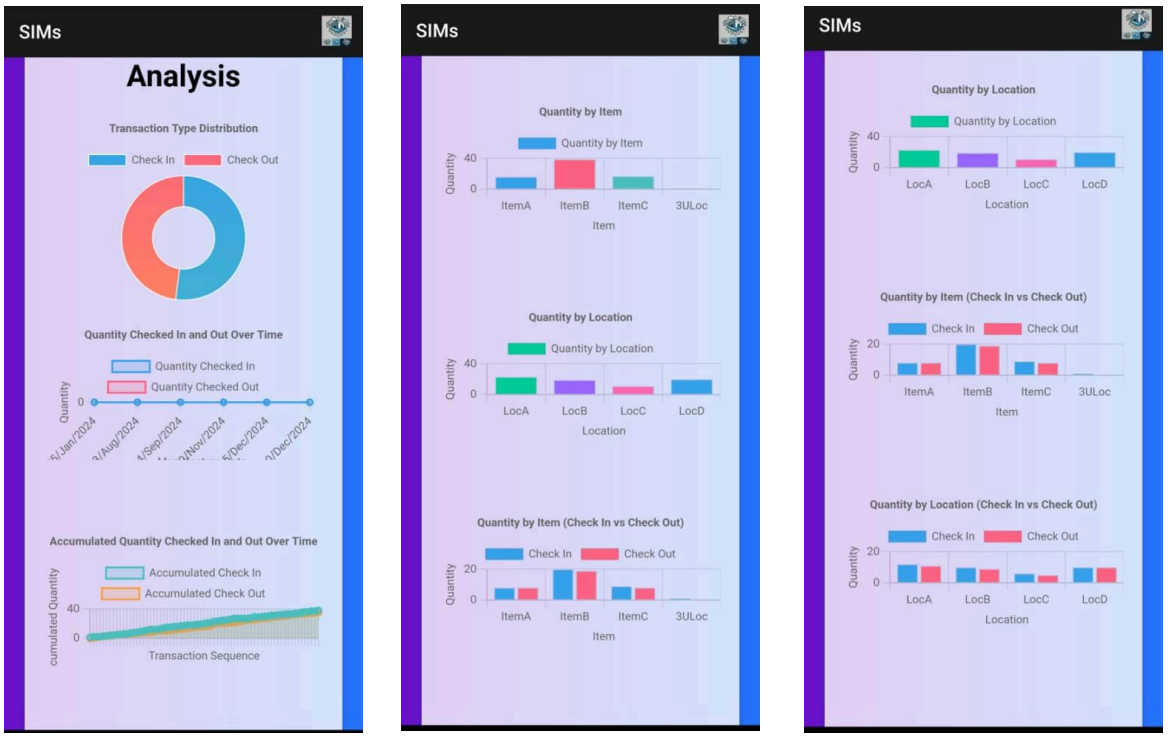

Figure A.10: Mobile application analysis page for inventory insight.

Figure A.11: Presentation session during 8th EECS event.