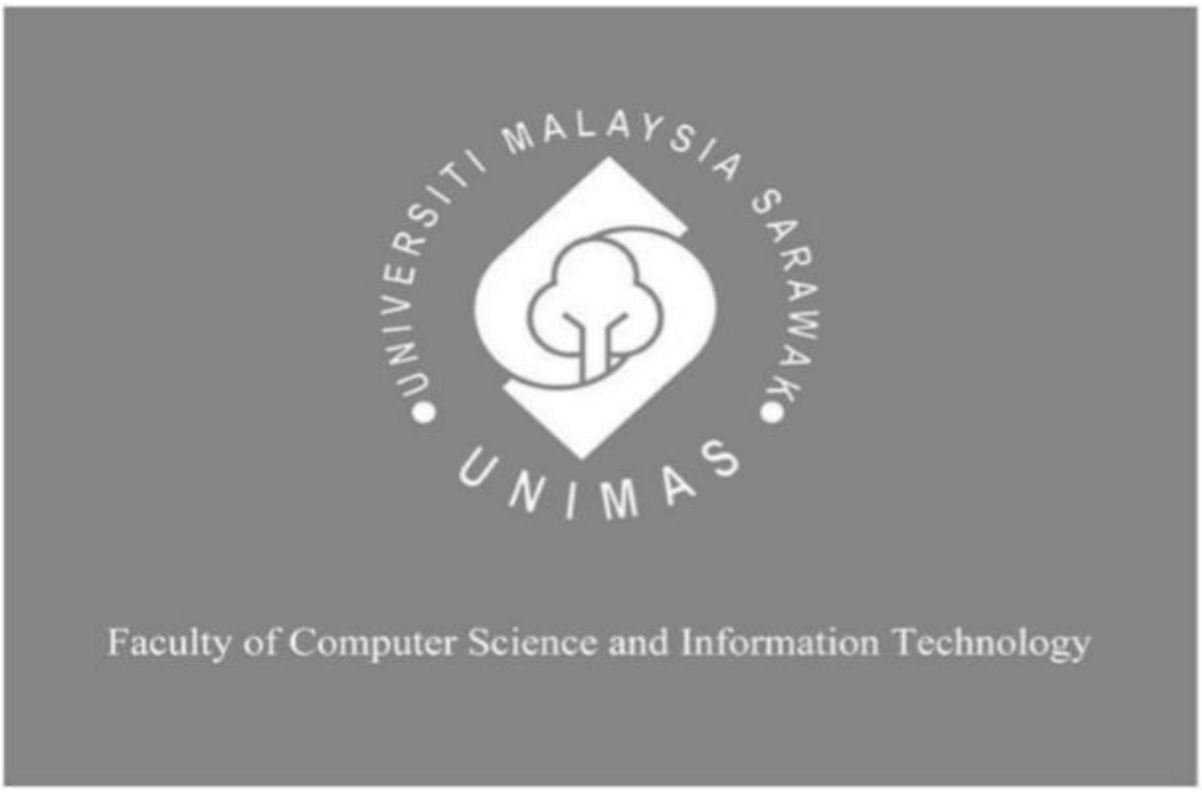

VENTO: VOICE-ASSISTED INVENTORY MANAGEMENT SYSTEM

AIMAN DANIAL JASNI

Bachelor of Computer Science (Multimedia Computing) with Honours

2024/2025

VOICE-ASSISTED INVENTORY MANAGEMENT SYSTEM

AIMAN DANIAL JASNI

The project is submitted in partial fulfilment of the

Requirements for the degree of

Bachelor of Computer Science with Honours

(Multimedia Computing)

Faculty of Computer Science and Information Technology

UNIVERSITI MALAYSIA SARAWAK

Year 4

UNIVERSITI MALAYSIA SARAWAK

THESIS STATUS ENDORSEMENT FORM

TITLE Vento: Voice-Assisted Inventory Management System

ACADEMIC SESSION: 2024/2025

(CAPITAL LETTERS)

hereby agree that this Thesis* shall be kept at the Centre for Academic Information Services, Universiti Malaysia Sarawak, subject to the following terms and conditions:

1. The Thesis is solely owned by Universiti Malaysia Sarawak
2. The Centre for Academic Information Services is given full rights to produce copies for educational purposes only
3. The Centre for Academic Information Services is given full rights to do digitization in order to develop local content database
4. The Centre for Academic Information Services is given full rights to produce copies of this Thesis as part of its exchange item program between Higher Learning Institutions [or for the purpose of interlibrary loan between HLI]
5. ** Please tick (√)

☐	CONFIDENTIAL	(Contains classified information bounded by the OFFICIAL SECRETS ACT 1972)
☐	RESTRICTED	(Contains restricted information as dictated by the body or organization where the research was conducted)
☑	UNRESTRICTED	

(AUTHOR'S SIGNATURE)

Permanent Address

Lot 51, Taman Wong Wo Lo Fasa
1, Jalan Tun Mustapha, 87000
Batu Arang, W.P.Labuan

Date: 22nd June 2025

Validated by

(SUPERVISOR'S SIGNATURE)

Date: 22nd June 2025

Note * Thesis refers to PhD, Master, and Bachelor Degree
** For Confidential or Restricted materials, please attach relevant documents from relevant organizations / authorities

DECLARATION

I hereby declare that this project is my original work. This project has been carried out as part of my academic requirements at University Malaysia Sarawak (UNIMAS), and it is the result of my own research, design, and development efforts. All sources of information, data, and assistance that were used or referenced during the development of this project have been properly acknowledged and cited.

Signed,

…….

(AIMAN DANIAL JASNI)

Matric No: 77760

Faculty of Computer Science and Information Technology

UNIVERSITI MALAYSIA SARAWAK

Date: 22nd June 2025

ACKNOWLEDGEMENT

First and foremost, I would like to express my deepest gratitude to Dr. Izzatul Nabila binti Sarbini for her invaluable guidance, continuous support, and constructive feedback throughout the development of this project. Her expertise and encouragement have been instrumental in shaping the Voice-Assisted Inventory Management System and ensuring its successful completion.

I would also like to extend my appreciation to Universiti Malaysia Sarawak (UNIMAS) for providing the necessary resources and facilities that made this project possible. The knowledge and experience I have gained during my time at UNIMAS have played a crucial role in my academic and personal growth.

A special thank you to all the respondents who took the time to evaluate and provide feedback on my questionnaire survey for my project. Your insights have greatly contributed to improving the system's functionality and usability.

Lastly, I am sincerely grateful to my family and friends for their unwavering support, patience, and encouragement. Your belief in me has been a constant source of motivation, helping me stay focused and determined throughout this journey.

This project is the result of the collective support and contributions from many individuals, and I deeply appreciate each and every one of you.

ABSTRACT

Effective inventory management is essential for ensuring operational efficiency within retail businesses. This project presents the development of *Vento: Voice-Assisted Inventory Management System*, a cross-platform solution designed using Flutter to support small and medium-sized enterprises. The system consists of a web-based application for administrators and an Android mobile application for retail staff, enabling seamless inventory control through both manual input and voice commands.

The system integrates Firebase services for real-time data synchronization, secure authentication, and cloud-based storage. A key feature of the system is the voice assistant module, which leverages speech recognition and basic natural language processing to facilitate hands-free execution of core inventory operations, including product addition, modification, deletion, and retrieval. Additional functionalities include QR code generation, stock tracking by size, and the presentation of sales data through visual reports to assist in business decision-making.

Development was guided by a personal Scrum methodology, incorporating iterative testing and user feedback to refine system features and usability. The system's primary objective is to enhance inventory management efficiency, minimize human error, and improve customer service through streamlined, voice-enabled operations. By combining artificial intelligence elements with real-time data handling, the project aims to modernize inventory processes and contribute to improved accuracy and productivity within retail environments.

ABSTRAK

Pengurusan inventori yang berkesan amat penting bagi memastikan kecekapan operasi perniagaan runcit. Projek ini membangunkan Vento: Sistem Pengurusan Inventori Berbantukan Suara, yang direka untuk menyokong perusahaan kecil dan sederhana melalui aplikasi merentas platform yang dibina menggunakan Flutter. Sistem ini merangkumi antara muka berasaskan web untuk pentadbir dan aplikasi mudah alih Android untuk kakitangan runcit, membolehkan kawalan inventori dilakukan dengan lancar melalui input manual atau arahan suara.

Sistem ini diintegrasikan dengan Firebase untuk penyelarasan data masa nyata, pengesahan pengguna yang selamat, dan penyimpanan berasaskan awan. Ciri utama aplikasi ini ialah modul pembantu suara yang menggunakan pengecaman pertuturan dan pemprosesan bahasa semula jadi (NLP) bagi membolehkan pelaksanaan tugas inventori utama tanpa menggunakan tangan seperti menambah, mengubah, memadam, dan menyemak produk. Fungsi tambahan termasuk penjanaan kod QR, penjejakan stok berdasarkan saiz, dan penjanaan laporan jualan visual bagi membantu dalam membuat keputusan perniagaan.

Untuk memastikan sistem ini mesra pengguna, projek ini menggunakan metodologi Personal Scrum, di mana setiap ciri akan diuji dan diperbaiki secara berperingkat berdasarkan maklum balas pengguna. Matlamat utama adalah untuk menjadikan pengurusan inventori lebih mudah, mengurangkan kesilapan, dan meningkatkan perkhidmatan pelanggan dengan membolehkan kawalan inventori secara bebas tangan.

Gabungan teknologi AI dan data pada masa ini diharap dapat memberi perubahan besar dalam cara pengurusan inventori dilakukan di sektor runcit, menjadikannya lebih cekap, tepat, dan berkesan.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

1.1 PROJECT BACKGROUND

Effective inventory management is essential for maintaining customer satisfaction and ensuring smooth business operations in today's fast-paced retail industry. An inventory management system is a structured approach used by businesses to track, manage, and optimize stock levels. It plays a critical role in preventing overstocking and stockouts, which can negatively impact sales and operations. Traditionally, inventory tracking and stock level verification have been labor-intensive processes, often requiring staff to manually check items in storerooms, leading to delays and potential human errors.

With advancements in technology, artificial intelligence (AI) and automation have revolutionized inventory management by improving efficiency and accuracy. AI-powered inventory systems analyze large datasets in real time, offering predictive analytics to optimize stock levels and minimize waste (Singh & Adhikari, 2023). These capabilities allow businesses to make data-driven decisions, reducing operational costs while enhancing service levels.

One of the emerging technologies transforming inventory management is voice-assisted technology. Voice assistants, powered by AI and Natural Language Processing (NLP), enable hands-free interaction with inventory systems, simplifying operations such as stock inquiries, updates, and report generation. According to Zwakman et al. (2021), a voice assistant is defined as "a hardware device or software agent powered by artificial intelligence that assists people with information searches, decision-making efforts, or executing certain tasks using natural language in a spoken format." This technology enhances efficiency by allowing employees to access inventory data instantly without needing to navigate complex interfaces or manually enter data.

Traditional inventory management methods often require employees to leave the sales floor to check stock availability, which can disrupt customer service and lead to inefficiencies. Delays in retrieving stock information can increase wait times and impact customer satisfaction. Additionally, manual inventory tracking is prone to human error, leading to discrepancies in stock levels and potential financial losses.

The primary objective of an inventory management system is to maintain optimal stock levels of raw materials, work-in-progress items, and finished goods (Muñoz Macas et al., 2021). By ensuring that supplies are available when needed, businesses can achieve high service levels while minimizing costs. AI-driven inventory management systems offer real-time monitoring, demand forecasting, and automated stock adjustments, reducing reliance on manual processes and improving overall accuracy.

Integrating voice-assisted technology into inventory management provides a modern approach to addressing these challenges. Voice assistants enable employees to interact with inventory systems seamlessly, reducing time spent on manual searches and improving workflow efficiency. With real-time voice commands, staff members can instantly check stock levels, update inventory records, and generate reports without leaving their designated work areas.

By leveraging AI and voice-assisted technologies, businesses can enhance productivity, reduce human error, and improve customer service. The adoption of these innovations in inventory management represents a significant step toward automation and digital transformation, ensuring smoother retail operations and better resource management.

1.2 PROBLEM STATEMENT

Managing inventory in small and medium-sized retail stores today can be slow and inefficient because employees often must leave the sales floor and search for items in the stockroom. This means that when customers ask if a specific product is available, the staff member has to leave them waiting, which can result in delays and poor customer experience. This method can also lead to mistakes, such as giving incorrect information about what is in stock. Additionally, it is difficult for store managers to get a clear view of sales trends using manual or outdated systems, making it hard to plan stock levels effectively.

The proposed project aims to solve these problems by creating a voice-assisted inventory management system tailored for SMEs, where employees can check stock levels, update records, and add new items using voice commands without leaving the sales floor. However, retail environments can be noisy, which may impact voice recognition accuracy. To address this, the system will integrate noise-filtering techniques and provide manual input options as a backup solution.

Additionally, the system will provide real-time data synchronization to ensure that inventory records remain accurate across multiple store locations. A key feature of this system is visual sales reports, including daily, weekly, and monthly graphs, to help store managers analyze sales trends and make informed restocking decisions.

1.3 OBJECTIVE

The main objectives of this project are:

- To design and develop a cross-platform inventory management system that will include a web-based admin panel and a voice-assisted Android app for staff.
- To use Firebase Firestore and Authentication to enable real-time data operations and secure user access.
- To implement voice recognition features that will allow staff to add, edit, delete, and search for product details using speech.
- To generate visual sales reports showing daily, weekly, and monthly trends to help with inventory planning.
- To provide staff management features in the admin panel so that administrators can add, update, or remove staff accounts.
- To test and validate the system to ensure it works accurately, reliably, and is easy to use in a real retail environment.

1.4 BRIEF METHODOLOGY

The project will be carried out using the Personal Scrum methodology through the following steps:

- **Research and Planning:** Initial research will be conducted on voice recognition tools, natural language processing (NLP) integration, and Firebase capabilities. A survey will also be distributed, particularly targeting youth, to gather opinions and expectations related to the system. These insights will help identify user needs and guide the prioritization of features during development planning.

- **Design Phase:** The user interface (UI) and user experience (UX) will be designed using Figma to create wireframes for both the Android app and the web-based dashboard.

- **Development:** The system will be developed using Flutter to build both the mobile application (for staff) and the web application (for admin use). Android's built-in speech-to-text service will be integrated for voice functionality, and Firebase Firestore and Authentication will be used for real-time data management and secure user login.

- **QR Code and Sales Report Implementation:** QR codes will be generated for each product, allowing users to scan and view product information. Sales data will be visualized through bar and line graphs, showing daily, weekly, monthly, and yearly trends to support inventory planning.

- **Testing:** System testing will be carried out to evaluate features such as voice commands, inventory operations, QR code functionality, and performance on different Android devices. Usability testing will also be conducted to gather user feedback and identify areas for improvement.

- **Deployment and Refinement:** The system will be deployed for actual use, followed by final rounds of feedback collection and refinement to improve overall functionality, performance, and user experience.

1.5 SCOPES

This project aims to develop Vento: A Voice-Assisted Inventory Management System, which includes two platforms: a web-based panel for the admin and an Android mobile app for staff. The goal is to make inventory management faster and easier using both manual and voice-assisted controls.

The web-based panel will let the admin manage products by adding, editing, deleting, and viewing inventory. It also includes features like searching products, generating QR codes for easier tracking, managing staff accounts, and updating the admin's profile and password. The admin can also perform checkout actions that adjust product quantities, and all sales will be recorded and shown in bar and line graphs with daily, weekly, monthly, and yearly filters.

The Android app will allow staff to view and manage inventory. Staff can use voice commands to check stock, add or update items, delete items with confirmation, and view product details. The app also supports QR code scanning for fast product lookup and lets users update their profile and password.

The system will use Firebase for real-time data handling and secure login. This project is mainly designed for small to medium-sized retail stores, with the possibility of expanding to larger businesses in the future.

1.6 SIGNIFICANCE OF PROJECT

This project is significant because it offers a modern and efficient way for retail stores to manage inventory using voice-assisted technology and real-time cloud integration. It is designed to reduce manual effort, speed up inventory tasks, and improve coordination between staff and the admin.

One of the main advantages is the voice command feature, which allows staff to manage inventory tasks hands-free. This saves time and reduces the chance of manual input errors. Staff can stay on the sales floor while performing inventory actions such as checking stock, adding items, or viewing product details. This improves multitasking and enables them to assist customers more quickly, which enhances overall customer experience.

The system also makes it easier for new staff to learn how to use it. Since they can interact with the app through simple voice commands, they don’t need to memorize complex steps. This helps increase productivity, especially during peak hours.

On the admin side, the system includes a visual sales report feature that displays product performance in daily, weekly, and monthly views. These insights help the admin make better decisions about stock levels, best-selling items, and restocking strategies.

In addition, the system supports QR code generation and scanning. The admin can generate and download QR codes for each product, while staff can scan them using the mobile app to instantly view product details. Customers can also scan these QR codes using their own devices to access product information without needing to ask staff, which improves convenience and customer satisfaction.

The use of Firebase ensures that inventory data is updated in real time for all users. This prevents communication issues, avoids stock discrepancies, and keeps the workflow smooth between the admin and staff.

Overall, this system improves inventory accuracy, supports faster operations, and helps retail stores make smarter business decisions. By combining voice control, QR code scanning, data visualization, and real-time cloud syncing, the system provides a practical and modern solution for inventory management.

1.7 PROJECT SCHEDULE

The project schedule outlines the timeline for key phases of Final Year Project (FYP) 1, including proposal preparation, chapter submissions, and meetings with the supervisor. The project follows a structured timeline to ensure steady progress.

Key milestones include:

- October 2024 - November 2024: Preparation and submission of the brief and full proposal.
- November 2024 - December 2024: Writing and submission of Chapter 1 and Chapter 2.
- December 2024 - January 2025: Development of Chapter 3, including requirement analysis and design.
- April 2025 - May 2025: Development and submission of Chapter 4.
- May - June 2025: Development and submission of Chapter 5 and Chapter 6.
- June 2025: Submission of Final Report.

A detailed breakdown of the project schedule is available in Appendix A, where the complete Gantt chart is provided.

1.8 EXPECTED OUTCOME

The expected outcomes of this project include:

- A working Android application that allows staff to manage inventory by adding, editing, deleting, and viewing product details using both voice commands and manual input.
- Successful integration with Firebase for real-time data syncing, secure login, and cloud-based inventory storage.
- A web-based admin dashboard that includes features for managing staff accounts, handling product inventory, and viewing sales reports. These reports will display daily, weekly, and monthly sales trends to help with inventory planning.
- A built-in QR code system that allows the admin to generate and download product QR codes, and enables both staff and customers to scan these codes to view product information instantly.
- Improved staff productivity and better customer support, made possible by hands-free voice interactions and faster access to inventory data.
- A reliable and easy-to-use system that helps reduce errors, supports better decision-making, and improves inventory handling in small to medium-sized retail stores.

1.9 PROJECT OUTLINES

- Chapter 1: Introduction

This chapter introduces the background and purpose of the project. It explains the common problems in traditional inventory management, such as delays, manual errors, and staff needing to leave the sales floor to check stock. As a solution, the project proposes a voice-assisted inventory system with an Android app for staff and a web-based dashboard for the admin. It also describes the integration of real-time data using Firebase and the use of sales graphs for trend analysis. The chapter includes the project's objectives, methodology, scope, significance, and expected outcomes.

- Chapter 2: Literature Review

This chapter reviews three existing systems or technologies related to inventory management. It compares their features, strengths, and weaknesses in terms of functionality, user

experience, and real-time performance. The findings highlight gaps in current systems and justify the need for the proposed solution, which combines voice interaction, real-time syncing, and data reporting to offer a more complete and efficient approach.

- Chapter 3: Requirement Analysis and Design

This chapter outlines the system's functional and non-functional requirements. It covers features such as voice-based CRUD operations, Firebase integration, sales reporting, and QR code support. The chapter also explains the system architecture, showing how the mobile and web apps connect to Firebase services. Wireframes and UI designs are presented to illustrate how staff and admin will interact with the system.

- Chapter 4: Implementation

This chapter describes the development process using Flutter, Firebase Firestore, and various packages for voice recognition, text-to-speech, and chart display. It explains how each feature was developed, including voice commands, real-time syncing, QR code generation, and staff management. Challenges such as noisy environments and command recognition issues are discussed, along with the solutions used to overcome them.

- Chapter 5: Testing

This chapter presents the testing and evaluation process. Functional testing is done on both the mobile and web apps to ensure all features work as expected. Usability testing is also carried out with 10 testers to get feedback on the voice commands, UI, QR scanning, and overall performance. The results are analyzed, bugs are fixed, and improvements are made to ensure a smooth user experience.

- Chapter 6: Conclusion and Future Work

The final chapter summarizes the project's success in improving inventory management through voice control, QR code integration, and real-time data updates. It also identifies current limitations, such as the lack of offline features and limited language support.

Suggestions for future work include expanding admin capabilities, adding more language options, and improving voice flexibility. The chapter ends by highlighting the project's overall contribution to modernizing retail inventory systems.

CHAPTER 2: LITERATURE REVIEW

2.1 INTRODUCTION

Inventory management has progressed significantly, moving away from manual, time-consuming, and error-prone methods to embrace advanced technologies that cater to the fast-paced demands of modern businesses. To address these challenges, innovative inventory management systems have been developed, leveraging cloud computing, real-time data synchronization, and artificial intelligence (AI) to improve efficiency and accuracy. These technologies have made inventory tracking more accurate, reduced human errors, and enabled automated data collection. (Madamidola et al., 2024)

This chapter reviews three leading inventory management systems, which are Zoho Inventory, Sortly, and On Shelf. Three of the apps mentioned have set industry benchmarks. By analyzing their features, strengths, and limitations, the review explores how these systems address common inventory management challenges. It also identifies areas for improvement, laying the groundwork for the proposed voice-assisted inventory management app.

The analysis highlights certain gaps in the current solutions, such as the lack of hands-free functionality, limited usability for small to medium-sized businesses, and the absence of quick product access methods. To bridge these gaps, the proposed system aims to integrate AI-powered voice assistance for seamless voice commands, real-time data synchronization using Firebase, visual sales reporting tools, and QR code functionality for fast product lookup and inventory tracking. These features are designed to create a more modern, efficient, and user-friendly inventory management experience tailored to today's retail needs.

2.2 REVIEW ON SIMILAR EXISTING SYSTEMS

This section reviews existing inventory management systems that are similar to the proposed one, examining their features, strengths, and weaknesses. By analyzing these systems, we can identify industry standards, common practices, and areas for improvement. This helps set a benchmark for evaluating the proposed system, highlights opportunities for innovation, and provides valuable insights to guide its design to meet user needs and expectations.

2.2.1 Zoho Inventory

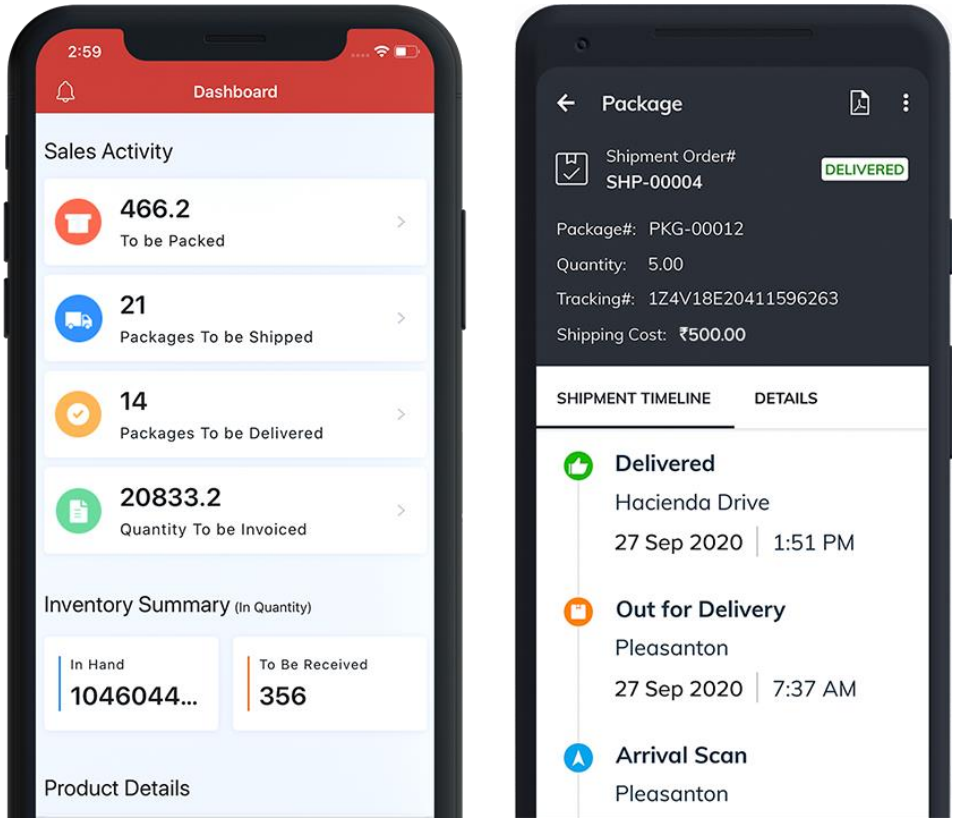

Figure 2.1 Zoho Inventory App

Zoho Inventory is cloud-based software designed to help small and medium-sized businesses manage their stock and orders more efficiently. It's part of the Zoho suite, which includes tools for customer relationship management (CRM), accounting, and e-commerce. Zoho Inventory offers real-time stock tracking, purchase order management, and automated reordering. It also integrates with platforms like Shopify, Amazon, and eBay, making it ideal for online stores. With advanced reporting and analytics, businesses can monitor sales trends and optimize inventory. However, its lack of voice-assisted features may make it less efficient for hands-free use in busy retail settings.

2.2.2 Sortly App

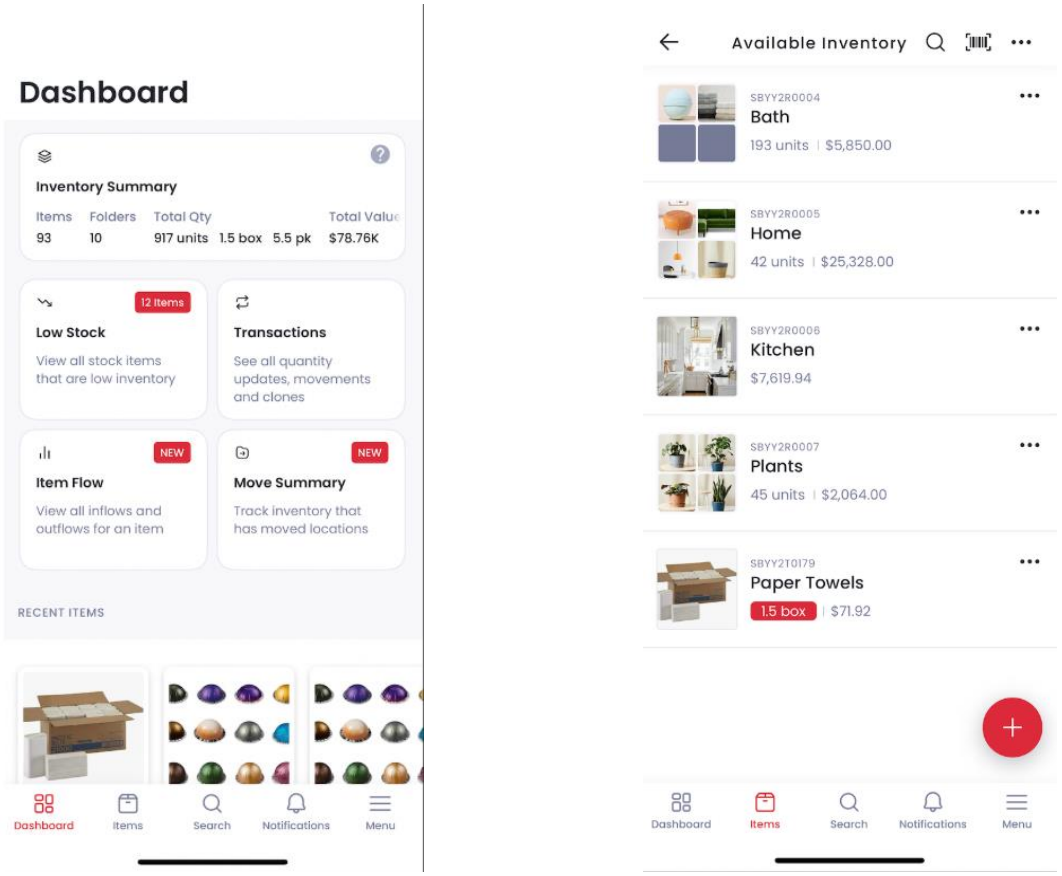

Figure 2.2 and Figure 2.3 show one of the interfaces in the Sortly App.

Sortly is a simple and user-friendly inventory management app designed to help individuals and small businesses easily organize, track, and manage their items. It offers a visual approach to inventory tracking by allowing users to upload photos of their items, making it easier to recognize and organize products. Sortly is also cloud-based and provides real-time data synchronization, which ensures that inventory updates are accessible from any device. It supports barcode scanning, custom categories, and advanced reporting tools to help users manage and analyze their inventory. Sortly's intuitive interface is particularly suited for businesses without technical expertise, as it focuses on ease of use and quick setup. Despite having a barcode scanning feature and advanced reporting tools, they are only available in the premium versions, which could be a barrier for small businesses with tight budgets. The free version only provides custom categories and basic reporting tools.

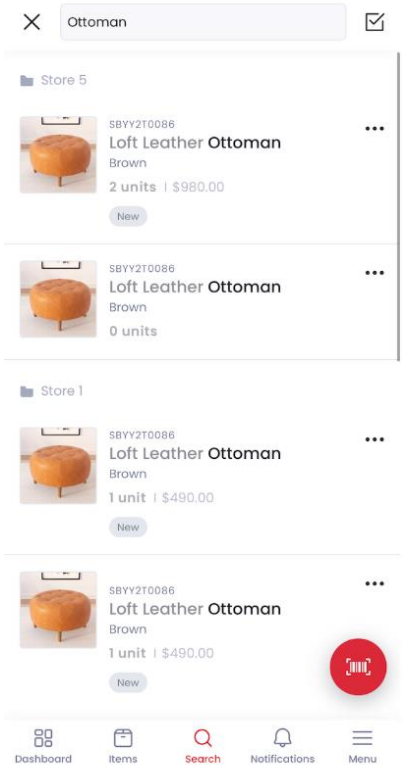

Figure 2.4 shows that you can search for items in the Sortly app.

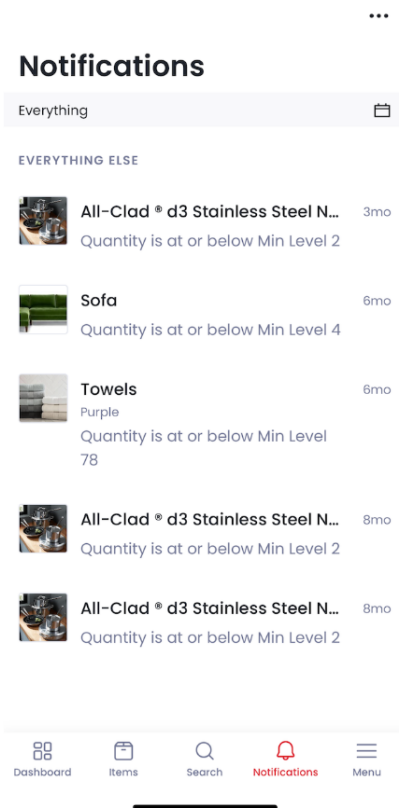

Figure 2.5 The app even provides a notification section where users can configure date and quantity alerts to ensure proactive inventory management and stay connected with push notifications for timely updates and reminders.

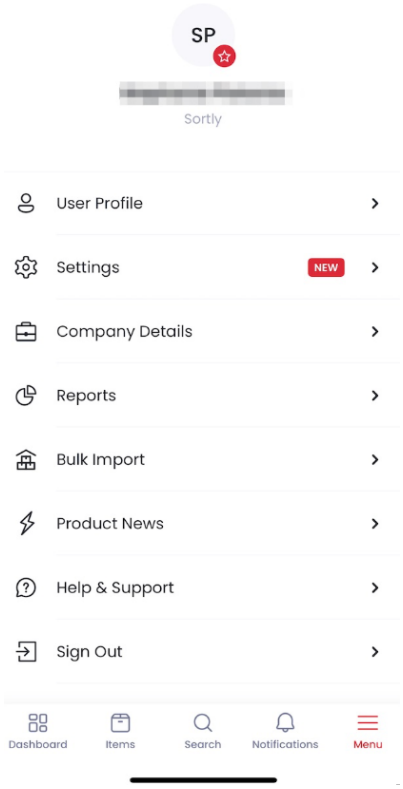

Figure 2.6 The menu section in the Sortly app.

2.2.3 On Shelf App

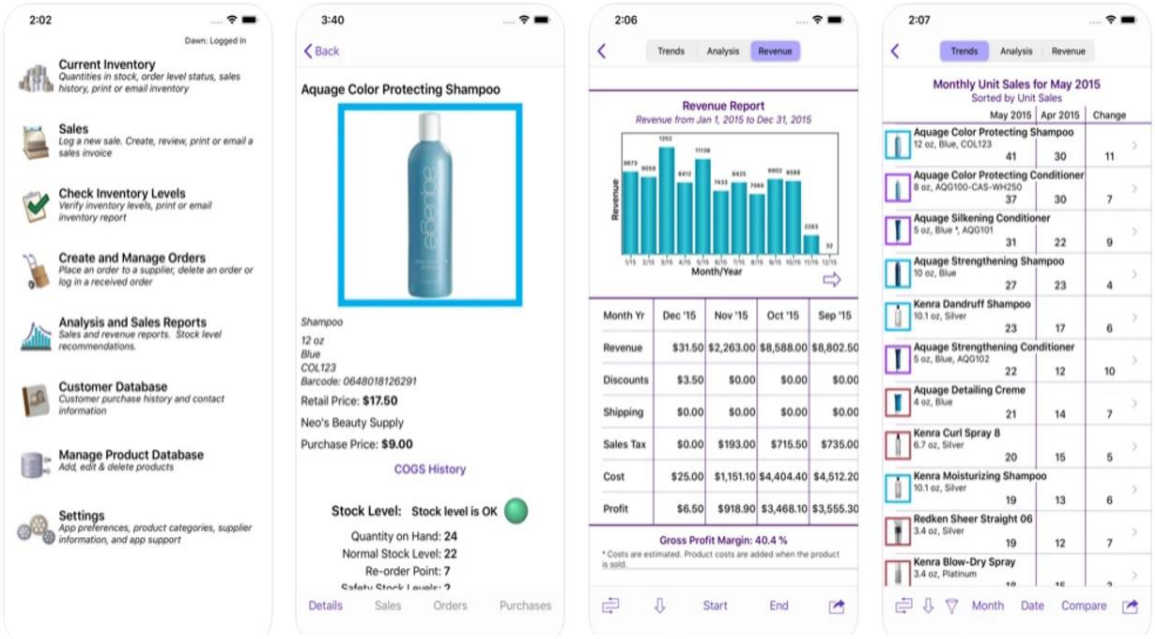

Figure 2.7 Interfaces in the On Shelf app.

On Shelf is a simple inventory management app designed for small businesses to track stock levels and manage products effectively for iOS users only. One of the main features of On-Shelf is reviewing the customer's purchase history. The app also focuses on helping users monitor their inventory by providing features such as stock tracking, low stock alerts, and the ability to add and update items manually. The app offers an easy-to-use interface, making it suitable for small business owners and individuals who need a straightforward solution for managing their products. However, it lacks advanced features like real-time cloud synchronization, detailed analytics, or integrations with other business systems, making it more suitable for basic inventory management needs.

2.2.4 Comparison of Features between Systems

Table 2.1 compares the features of three existing inventory management apps, which are Zoho Inventory, Sortly, and On Shelf, as well as the proposed application. The comparison focuses on important features like platform compatibility, voice assistance, real-time data updates, reporting tools, scalability, and so on. The proposed system enhances existing solutions by integrating AI-powered voice control, real-time synchronization via Firebase, advanced sales reporting, and QR code functionality, making it a more efficient and accessible tool for small to medium-sized retail stores.

Table 2.1 Feature Comparison of Inventory Management Systems

App Features	**Zoho Inventory**	**Sortly**	**On Shelf**	**The Proposed App**
Platform	Web and Mobile App (Android and iOS)	Android and iOS	IOS only	Android (developed with Flutter)
Voice Assistance	No	No	No	Yes (add, edit, delete, view, query via voice commands)
QR Code Support	No	No	No	Yes (QR code generated for each product; viewable via scan)
Real-time Data Handling	Yes (cloud-based updates)	Yes (cloud-based synchronization)	No (manual updates; data stored locally)	Yes (Firebase-based real-time synchronization)

Sales Graphs/Analytics	Yes (detailed analytics and reports)	Basic (limited to inventory levels, no detailed graphs)	No (focus om basic stock-level tracking)	Yes (daily, weekly, and monthly sales trends)
Cloud Integration	Yes (integration with cloud platforms like Amazon S3)	Yes (cloud-based storage)	No (local storage only)	Yes (Firebase cloud storage)
Ease of Use	Comprehensive but can be overwhelming for small users	Simple and intuitive	Simple but lacks advanced features	User-friendly with voice and manual input options
Target Audience	Small to medium-sized businesses	Small businesses	Small business owners and home inventory users	Small to medium-sized retail stores
Inventory Updates	Manual input, barcode scanning and integrations	Manual input only	Manual input only	Manual input and voice commands
Reporting Capabilities	Comprehensive reporting tools	Basic reporting only	Limited to stock-level updates only	Customizable reports with filtering and formatting options
Scalability	Highly scalable	Limited scalability	Limited scalability	Designed for future scalability with role-based access support
Cost	Subscription-based	Free or subscription-based (more features)	One-time purchase cost	Free with all features

2.2.5 The Proposed App and Potential Improvements

The proposed app stands out by integrating voice assistance for hands-free inventory management, which is a feature that is not available in Zoho Inventory, Sortly, or On Shelf. It allows users to manage inventory using both voice commands and manual input, making the system flexible and efficient. In addition, it provides sales graphs that display daily, weekly, and monthly trends, offering better support for decision-making compared to the limited reporting options in Sortly and On Shelf.

This app also uses Firebase for real-time data synchronization and secure cloud storage, ensuring that inventory data is always accurate and accessible. Another feature that adds value is QR code generation and scanning, which allows users to instantly view product details by scanning product-specific QR codes. This improves speed and convenience, especially during stock checks or customer inquiries.

The app is built using Android Studio and is currently designed for Android devices. It includes all features for free, targeting small to medium-sized retail businesses. In the future, the app can be scaled for larger businesses depending on demand and performance needs.

To further enhance the system, one possible improvement is to introduce customizable reporting. This would allow users to filter and display sales data based on their specific needs, making analysis more effective. Preparing the system for scalability, such as optimizing the database and adding role-based access control, would also support its use in larger retail environments.

Overall, the proposed app delivers a modern, user-friendly, and effective solution for inventory management, with the potential to grow and adapt to future business needs.

2.3 SUMMARY

In summary, Chapter 2 explains how inventory management systems have evolved with the help of modern technologies such as cloud computing and artificial intelligence. It reviews three existing systems: Zoho Inventory, Sortly, and On Shelf, by examining their main features, strengths, and limitations. Zoho Inventory offers a complete solution for small to medium-sized businesses but does not support hands-free functionality. Sortly is easy to use and includes basic features in its free version, although many advanced tools require a paid

subscription. On Shelf provides a simple and straightforward interface but lacks cloud integration and more advanced capabilities.

These comparisons show areas where the proposed system can improve upon existing solutions. The new app is designed to include voice-assisted inventory control, real-time data synchronization using Firebase, and sales reports that display daily, weekly, and monthly trends. It also includes QR code scanning and generation, making it easier for users to access product details quickly.

Possible future improvements include customizable reporting options and the ability to scale the system for use in larger businesses. Overall, this chapter supports the development of a modern, efficient, and user-friendly inventory management system that better meets the needs of retail operations.

CHAPTER 3: REQUIREMENT ANALYSIS AND DESIGN

3.1 INTRODUCTION

Thorough planning is essential for the success of a voice-assisted inventory management system. This includes a clear understanding of the development methodology, as well as detailed requirement analysis and system design. Proper planning at this stage helps reduce the risk of defects, glitches, or the need for major rework later in the development process, which leads to a more stable system and better user experience (McGuire, 2024).

This chapter outlines the key requirements needed to build the proposed system, including both functional and non-functional requirements, as well as the necessary software and hardware. The system is designed to support two main user roles: the admin, who uses the web platform to manage inventory, staff accounts, and view sales reports; and the staff, who use the Android app to perform inventory tasks and interact with the voice assistant. Each role has access only to the features relevant to their responsibilities.

The chapter also presents the system's design components, such as the system architecture diagram, use case diagram, sequence diagrams, and entity relationship diagram. These elements help visualize how the system functions from both user perspectives. In addition, wireframe designs are included to illustrate the user interface and layout of the application.

3.2 PERSONAL SCRUM METHODOLOGY

Figure 3.1 Personal Scrum Methodology

As mentioned in Section 1.4, this project follows the Personal Scrum methodology, an agile development approach tailored for individual work rather than large teams. It emphasizes flexibility, continuous feedback, and incremental development. The project is divided into smaller development phases called sprints, with each sprint focused on building and refining specific features such as voice command functionality, real-time data synchronization, and sales graph visualization.

At the end of each sprint, progress is reviewed through self-assessment and feedback gathered from retail staff and store managers. This iterative approach allows for continuous improvement and ensures the app remains aligned with real user needs. Personal Scrum also enables quick adaptation to unexpected issues, helping maintain the system's usability, performance, and reliability in real-world retail environments.

The methodology follows these key stages:

1. Research and Planning

The project begins with exploring tools and technologies suitable for voice-assisted inventory management. This includes evaluating Android's built-in Speech-to-Text and Natural Language Processing (NLP) tools for voice command interpretation. Firebase is selected for managing real-time database operations, user authentication, and cloud storage. A survey is conducted targeting young users to gather feedback and prioritize features based on user expectations.

2. Design Phase

User interface layouts and wireframes are created using Figma, defining key screens such as login, inventory lists, voice assistant modules, and the sales dashboard. This helps visualize the app structure and interactions for both staff (mobile) and admin (web) users.

3. Development Phase

The system is developed using Flutter in Android Studio, supporting both Android and web platforms.

- Voice features are implemented using the speech_to_text and flutter_tts packages.
- Firebase Firestore manages inventory data, staff records, authentication, and sales logs.
- QR code generation and scanning are built using third-party QR libraries.
- Sales graphs are displayed using the fl_chart package to visualize daily, weekly, and monthly sales trends.

4. Testing Phase

In this phase, comprehensive testing is carried out to ensure all key features function correctly.

- Functional testing is done to verify voice commands, CRUD operations, QR scanning, and graph updates.
- Compatibility is tested across various Android devices and web browsers to ensure responsiveness.
- Users are invited to try the system and provide feedback on usability, performance, and design.
- Collected feedback is used to fix bugs, enhance functionality, and improve the overall user experience.

5. Deployment and Feedback

After successful testing and refinements, the final version of the app is deployed for real-world use. This version includes all core features tested for stability and reliability. It is designed to support daily inventory operations for small to medium-sized retail environments.

3.3 REQUIREMENTS

3.3.1 Survey

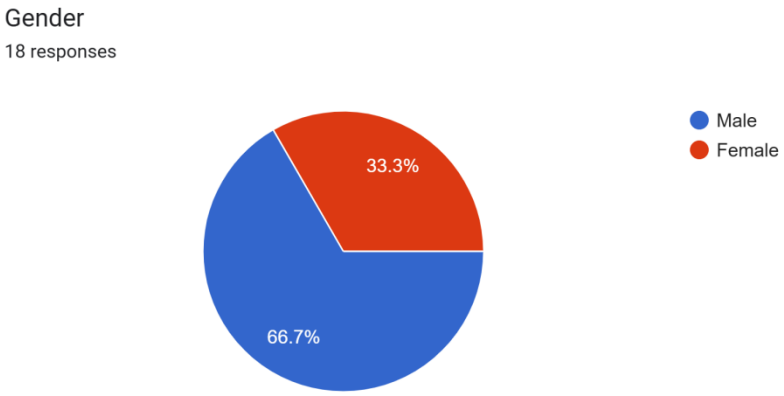

Figure 3.2 Gender

Figure 3.2 shows that out of 18 respondents, 66.7% of the respondents are male, and the other 33.3% are female.

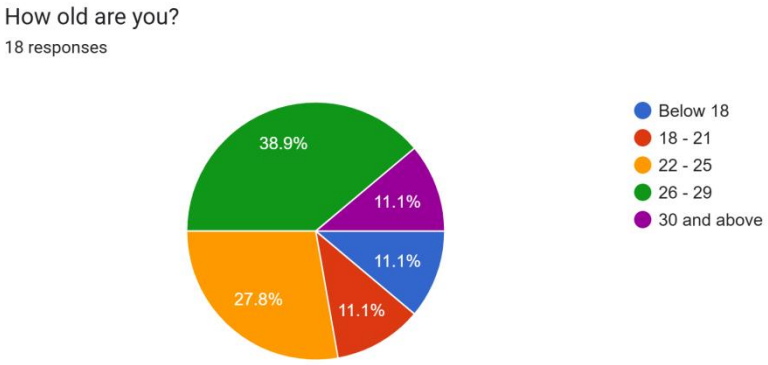

Figure 3.3 How old are you?

Figure 3.3 shows that out of 18 respondents, 38.9% are in the age range of 26-29, 27.8% are in the age range of 22-25, 11.1% are in the age range of below 18, 11.1% are in the age range of 30 and above, and the other 11.1% are in the age range of 18-21.

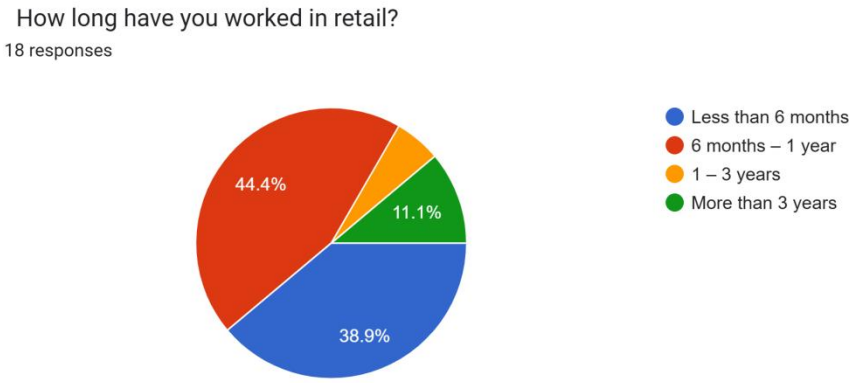

Figure 3.4 How long have you worked in retail?

Figure 3.4 shows that out of 18 respondents, 44.4% of them worked in retail for less than 6 months, another 38.9% have 6 months to 1 year of retail experience, 11.1% have more than 3 years of experience, and the minority of 5.6% have 1 year to 3 years of experience.

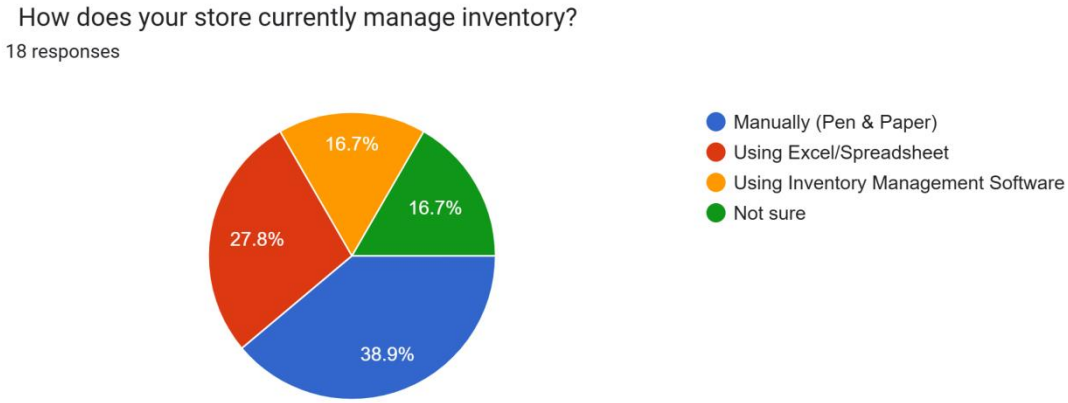

Figure 3.5 How does your store currently manage inventory?

Figure 3.5 shows that 38.9% of 18 respondents managing inventory manually, which is by pen and paper. The other 27.8% are using Microsoft Excel or Spreadsheet, 16.7% using inventory management software provided in retail store, and 16.7% are not sure.

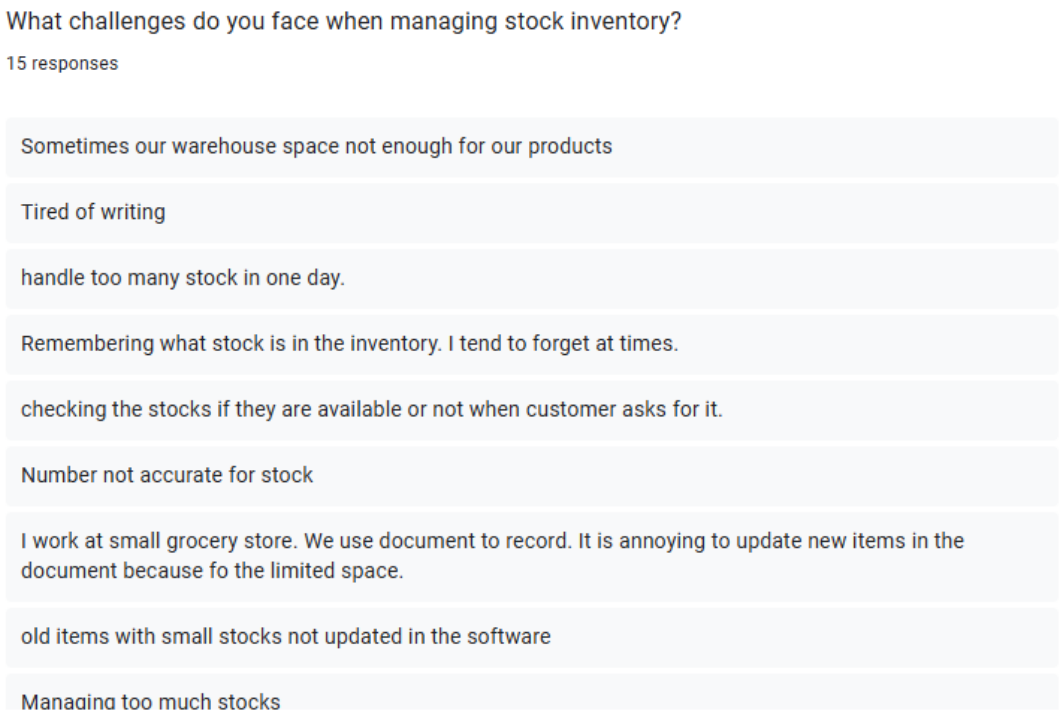

Figure 3.6 What challenges do you face when managing stock inventory?

Based on Figure 3.6, there are 15 responses out of 18 respondents. One of them experienced handling too much stock in one day, another had managed too much stocks, and also tired of writing manually using pen and paper. One of them that is working at a small grocery store used document to record and finds it annoying to update new items in the document because of the limited space. Another challenges mentioned are old items with small stocks are not updated in the software, numbers are not accurate for stocking, and many more.

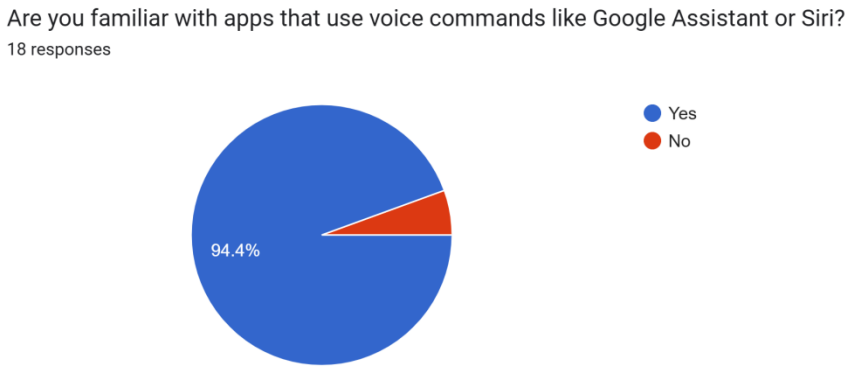

Figure 3.7 Are you familiar with apps that use voice commands like Google Assistant or Siri?

Figure 3.7 shows that a majority 94.4% of 18 respondents are familiar with apps that use voice commands such as Google Assistant or Siri. Meanwhile. The minority 5.6% are not familiar with such term.

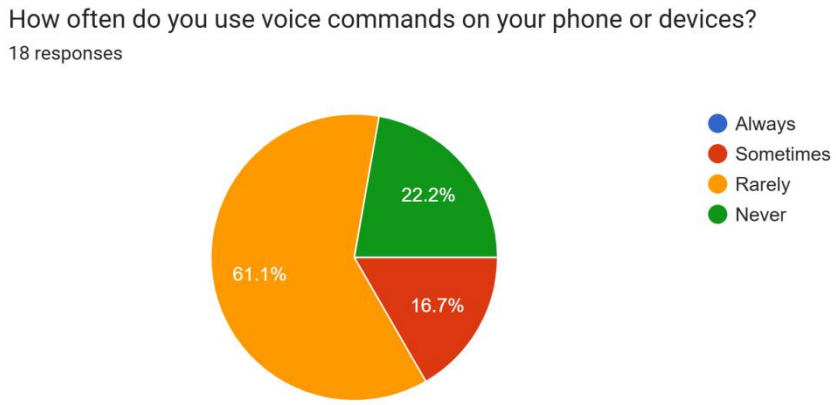

Figure 3.8 How often do you use voice commands on your phone or devices?

Out of 18 respondents, 61.1% of them rarely use voice commands on their phone or devices, while 22.2% never use voice commands ever. The minority 16.7% only using it sometimes. This shows that even though most of them are aware of voice commands in their phone, they are not using it regularly.

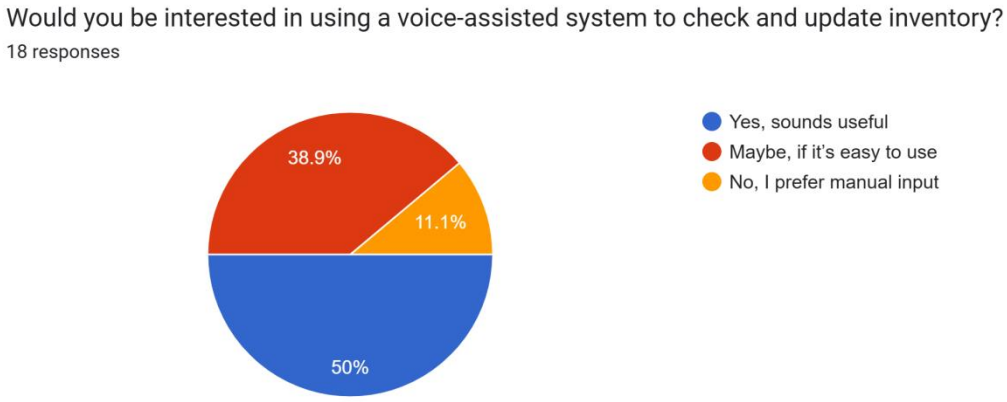

Figure 3.9 Would you be interested in using a voice-assisted system to check and update inventory?

Figure 3.9 shows that 50% of 18 respondents find voice-assisted system useful to check and update inventory for inventory management system. Meanwhile, 38.9% find that it may be useful if it is easy to use. A minority 11.1% of 18 respondents prefer manual input only. The voice-assisted inventory management system will provide both voice assistant and manual input.

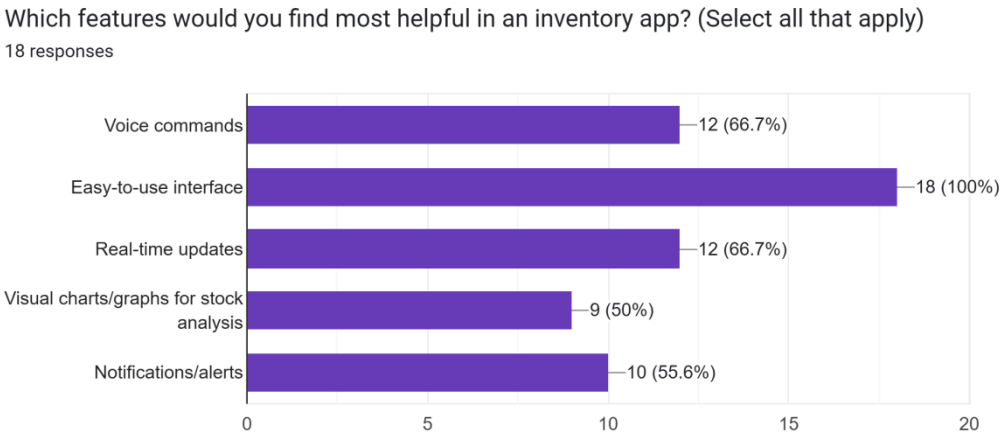

Figure 3.10 Which features would you find most helpful in an inventory app?

Figure 3.10 shows a bar chart of features that respondents find helpful in an inventory app. Based on 18 respondents, the most popular pick is easy-to-use interface feature with a total of 18 respondents, following with real-time updates and voice commands from 12 respondents, notifications/alerts from 10 respondents and the lowest vote is visual charts/graphs for stock analysis. This shows that a clear and simple interface is the most important aspect of building an app, especially for an inventory management system.

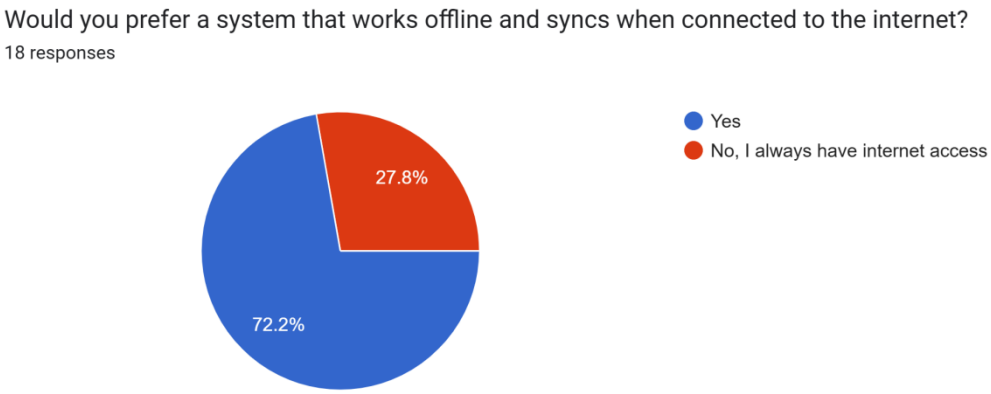

Figure 3.11 Would you prefer a system that works offline and syncs when connected to the internet?

Based on this pie chart in Figure 3.11, a majority 72.2% of 18 respondents prefer a system that works offline and syncs when connected to the internet, and the minority 27.8% disagree as they always have internet access. This means that the retail employees and managers can still add, update, view and delete inventory stocks even though there is no internet connection.

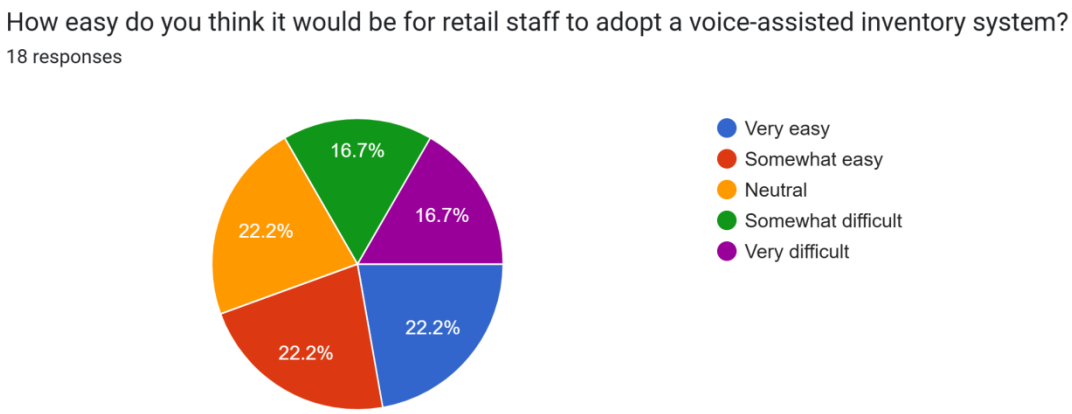

Figure 3.12 How easy do you think it would be for retail staff to adopt a voice-assisted inventory system?

Figure 3.12 shows the responses of 18 participants regarding the ease of using a voice-assisted inventory system. Among them, 22.2% find it very easy, 22.2% find it somewhat easy, 22.2% are neutral, 16.7% find it somewhat difficult, and 16.7% find it very difficult. This indicates that their opinions are divided on how easy it is to adopt a voice assistant for inventory management.

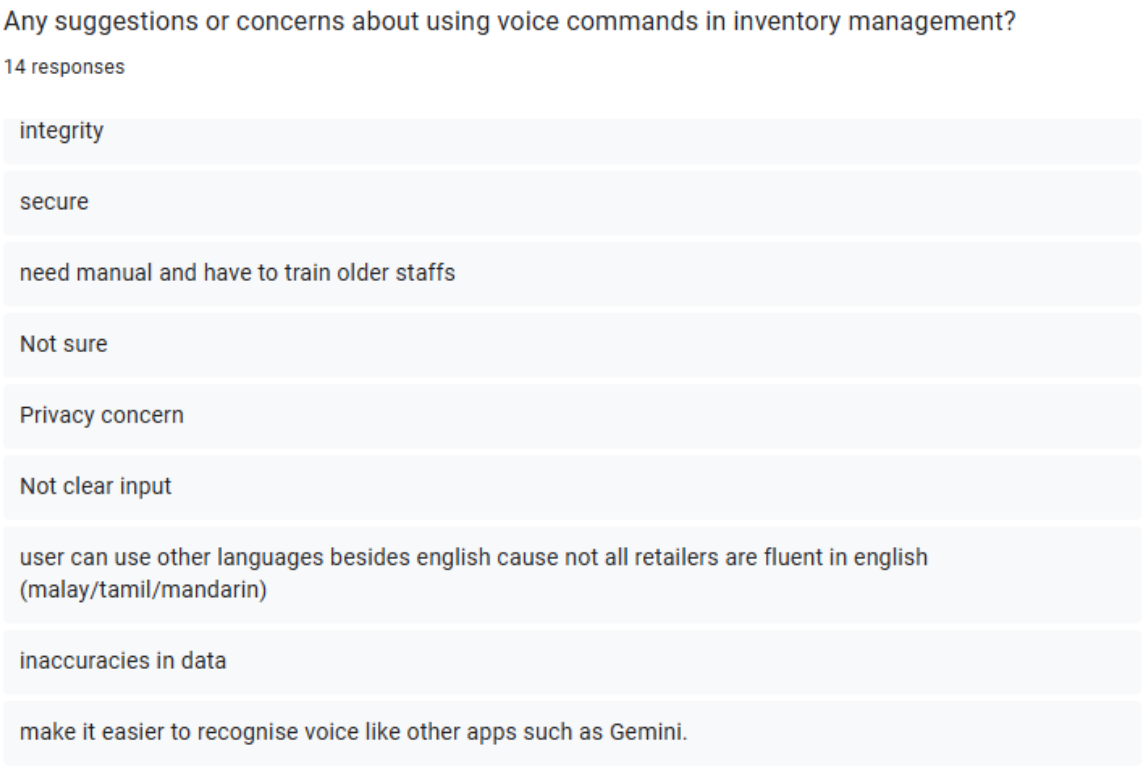

Figure 3.13 Any suggestions or concerns about using voice commands in inventory management?

Figure 3.13 shows an open-ended question about the respondents suggestions or concerns about using voice commands in inventory management. Based on 14 responses, one of them are talking about the privacy concern. One of them also suggests that user can use other languages besides English, such as Malay, Tamil, or Mandarin. Another suggestions such as including Gemini to recognize voice like other apps, needing manual input and have to train older staffs in using voice assistant, and so on. Some of them are having concern about not having clear input when using voice assistant, inaccuracies in data, and security. The point of this survey is for a better development of the app.

3.3.1 Functional Requirements

- Voice-Activated Commands (Staff Only)

Staff will be able to manage inventory using voice commands. Supported actions include adding, editing, deleting products, and checking or updating stock levels. This feature uses speech recognition along with basic natural language processing to understand staff input.

- Inventory Management

Both staff and admins will be able to manually add, edit, and delete product details such as name, category, price, gender, size, and stock quantity. These operations will update in real time using Firebase Firestore.

- QR Code Functionality

A unique QR code will be automatically generated for each product upon creation. Staff can scan these QR codes using the mobile app to instantly view product details. Customers can also scan the QR codes using their own devices to access read-only product information without needing to log in. Admins will have the ability to download and print the QR codes via the web dashboard.

- Staff Management (Admin Only)

Admins will be able to register, update, and delete staff accounts through the admin panel. Newly registered staff accounts will remain in a pending state until the staff verifies their email and logs into the Android app for the first time. Once verified, the staff status will automatically update in the admin dashboard. This process ensures secure onboarding and prevents unauthorized access.

- Sales Report Visualization (Admin Only)

Admins will have access to sales reports displayed as bar and line charts. These reports can be filtered by day, week, month, or year to support inventory planning and business analysis.

- Checkout Module (Admin Only)

Admins will be able to select products for checkout, adjust quantities based on size, and complete the transaction. Stock levels will be updated in real time, and each sale will be recorded for reporting.

- Admin Profile Management (Admin Only)

Admins can view and update their display name and password through the web dashboard. The email field is displayed as read-only for identification purposes.

- Staff Profile View (Staff Only)

Staff can view their name and email address within the mobile app. This screen is read-only, and all account updates must be managed by the admin.

- Login and Logout

Both staff and admins will be required to log in securely using Firebase Authentication. The system will support login and logout functionality for both roles.

3.3.2 Non-functional Requirements

- Performance

The system should respond quickly and run smoothly across both mobile and web platforms. The code will be optimized for speed, and performance will be tested under various usage scenarios to ensure reliability.

- Usability

The application must be easy to use for staff with different levels of technical experience. A clean interface, clear navigation, helpful prompts, and intuitive voice command integration will help enhance the overall user experience.

- Security

User data, including login credentials and inventory records, must be protected at all times. Firebase Authentication and Firestore security rules will be implemented to enforce access control. All data transmission will be secured using encryption to prevent unauthorized access.

- Scalability

The system should be capable of handling more users, products, and data as the business grows. This will be supported by Firebase's scalable cloud infrastructure and a modular system design that allows for future expansion.

- Accessibility

The app will include accessibility considerations such as voice interaction and touch-friendly controls, helping support users with different needs and ensuring ease of use on mobile devices.

3.3.3 Software Requirements

The proposed system will be developed using tools and frameworks that support cross-platform development, efficient performance, and ease of use. The mobile app is built using Flutter, an open-source UI toolkit developed by Google. Flutter enables fast and flexible development for Android, which is the main target platform for this project.

For backend services, the system uses Firebase, which offers several cloud-based features. Firebase Authentication handles secure login and logout processes, while Cloud Firestore manages real-time data storage and synchronization. These services ensure that inventory and user data are updated instantly across all devices.

To enable voice control, the app integrates Android's native Speech-to-Text functionality using the speech_to_text package. This allows staff to perform inventory tasks using natural voice commands. For voice responses, the system uses the flutter_tts (Text-to-Speech) package, which reads out product information or system messages, supporting hands-free interactions.

For sales reporting, the app uses the fl_chart library to display bar and line graphs. These visualizations help the admin view trends in daily, weekly, and monthly sales, assisting in inventory planning and decision-making.

3.3.4 Hardware Requirements

The mobile app is designed to run on standard Android smartphones or tablets used by staff in retail settings. These devices allow staff to manage inventory directly from the sales floor without needing to return to a computer. The app uses the device's built-in microphone to support voice commands, enabling hands-free operation.

To ensure smooth and real-time data synchronization with Firebase, the devices must have a stable internet connection, either through Wi-Fi or mobile data. Since these hardware components are commonly available and cost-effective, the system is practical and easy to deploy in most retail environments for staff use.

Admins will access the web dashboard through desktop or laptop computers, which are not covered under this mobile hardware requirement.

Table 3.1 Minimum specifications for an Android device to run the proposed app efficiently.

Component	Minimum Requirement
Operating System (OS)	Android 13 (Tiramisu) or newer
Processor	Octa-core processor (2.0 GHz or higher recommended)
RAM	4 GB
Storage	64 GB internal storage
Internet Connection	Wi-Fi or mobile data (required for real-time synchronization with Firebase)
Microphone	Built-in microphone (required for voice command input)
Screen	Full HD+ resolution (1080 × 2400 pixels) for optimal viewing of charts and UI elements

3.4 DESIGN

3.4.1 Overall System Architecture Diagram

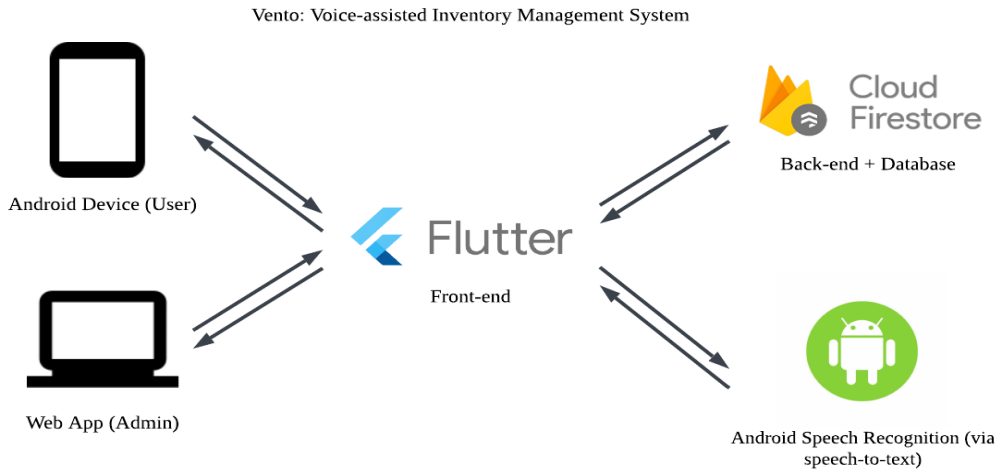

Figure 3.14 Overall System Architecture Diagram

Figure 3.14 shows the overall system architecture of the Voice-Assisted Inventory Management System, also known as Vento. The system consists of two main platforms: a mobile application for retail staff and a web-based dashboard for the admin. Both platforms are developed using Flutter, allowing for a consistent user interface and experience across devices.

The Android app, used by staff, enables inventory management through both manual input and voice commands. Voice commands are captured using Android's native speech recognition, integrated through the speech_to_text package. These commands are then converted to text and interpreted using rule-based logic to perform tasks such as adding, editing, deleting, or viewing product information.

The web-based dashboard, used by the admin, provides access to manage product inventory, staff accounts, QR code downloads, and sales reporting. The admin can perform tasks such as product checkout, adjusting stock levels, and monitoring user activity in real time.

Both platforms are connected to Firebase, which serves as the back-end of the system. Firebase Firestore handles real-time database operations, while Firebase Authentication manages secure login for both staff and admin accounts.

The system is designed to ensure that any changes made from either platform are instantly synchronized across all connected devices. This allows for a seamless, real-time inventory management experience that improves efficiency and accuracy for retail operations.

3.4.2 Use Case Diagram

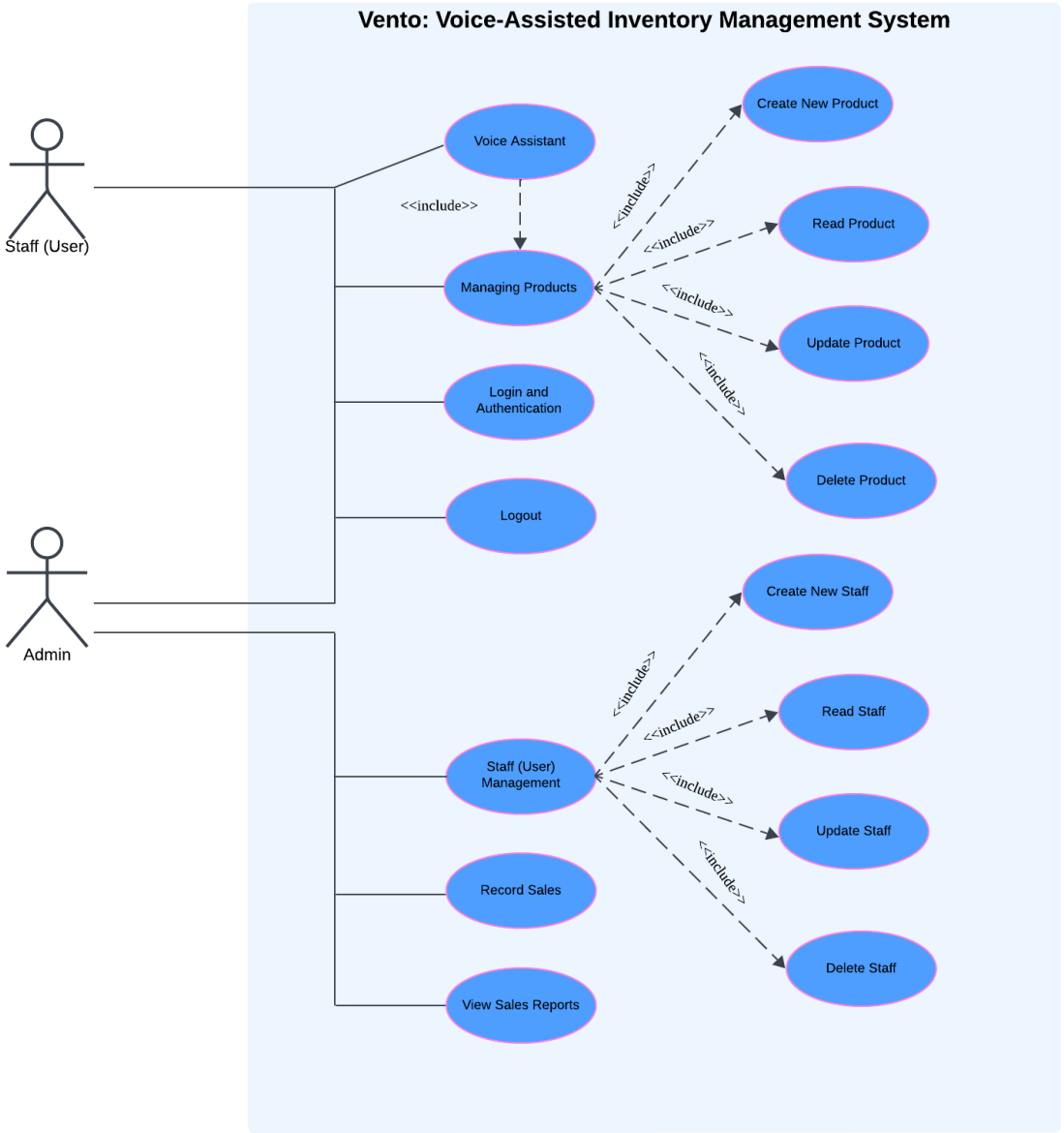

Figure 3.15 Use Case Diagram

Figure 3.15 shows the use case diagram for the voice-assisted inventory management system app. The diagram includes two primary actors: Staff and Admin. These actors represent the different roles or users who interact with the system.

Table 3.2 Use Case of Create a New Product

Use Case Name	Create a New Product
Description	This use case allows both Admin and Staff to add a new product to the inventory manually.
Primary Actor	Staff, Admin
Pre-condition	The actor must have the application or web open and be in the inventory management section.
Post-condition	A new product is added to the database with all required details.
Main Success Scenario	1. Actor selects the option to create a new product. 2. Actor enters the product details manually. 3. The system saves the new product information in the database. 4. Confirmation is displayed to the actor.
Alternative Scenario	1. If the entered product details are incomplete, the system prompts the actor to provide the missing information.

Table 3.3 Use Case of Read Product's Details

Use Case Name	Read the product's details
Description	This use case allows both Admin and Staff to view details of a specific product manually.
Primary Actor	Staff, Admin
Pre-condition	The actor must have the application or web open and be in the inventory management section.
Post-condition	The system displays the details of the requested product.
Main Success Scenario	1. Actor selects the option to read product details. 2. The system retrieves and displays the product details.
Alternative Scenario	1. If the product does not exist, the system will not show the product details.

Table 3.4 Use Case of Update Product's Details

Use Case Name	Update product's details
Description	This use case allows both Admin and Staff to modify details of an existing product manually.
Primary Actor	Staff, Admin
Pre-condition	The actor must have the application or web open and be in the inventory management section. The product must exist in the inventory.

Post-condition	The product details are updated in the database.
Main Success Scenario	1. Actor selects the option to update product details. 2. Actor specifies the product and the new details. 3. The actor updates the product information in the database. 4. Confirmation is displayed to the actor.
Alternative Scenario	1. If the product does not exist, the system notifies the actor. 2. If the entered details are incomplete, the system prompts the actor to provide the missing information.

Table 3.5 Use Case of Delete Existing Product

Use Case Name	Delete an existing product
Description	This use case allows both actors to remove a product from the inventory.
Primary Actor	Staff, Admin
Pre-condition	The actor must have the application or web open and be in the inventory management section. The product must exist in the inventory.
Post-condition	The product is removed from the inventory database.
Main Success Scenario	1. Actor selects the option to delete a product. 2. Actor specifies the product.

	3. The system removes the product from the database. 4. Confirmation is displayed to the actor.
Alternative Scenario	1. If the product does not exist, the system notifies the actor.

Table 3.6 Use Case of Voice Assistant

Use Case Name	Performing CRUD tasks via Voice Assistant
Description	This use case allows Staff to interact with the inventory system using voice commands to perform tasks such as adding, viewing, editing, or deleting products. The system replies using text-to-speech (TTS).
Primary Actor	Staff
Pre-condition	Staff must be logged into the Android app and have granted microphone permission.
Post-condition	The system processes the voice command and completes the corresponding task.
Main Success Scenario	1. Staff activates the voice assistant. 2. Staff gives a voice command (e.g., "Add product Adidas UltraBoost"). 3. System recognizes the command and performs the requested task. 4. System provides a confirmation or response using text-to-speech (TTS).
Alternative Scenario	1. If command is unclear or unsupported, the system guides the staff via TTS.

	2. If voice recognition fails, the system prompts the staff to repeat.

Table 3.7 Use Case of Record a Sale

Use Case Name	Record a Sale
Description	This use case allows the Admin to perform a checkout process by selecting products, adjusting quantities, and recording the transaction. Upon successful checkout, product stock is deducted and the sale is saved into the system.
Primary Actor	Admin
Pre-condition	Admin must be logged in and on the checkout page of the web dashboard.
Post-condition	The sale is recorded in the database, and inventory stock is updated accordingly.
Main Success Scenario	1. Admin adds products to the cart. 2. Admin completes checkout. 3. System processes the transaction, deducts the appropriate stock, and records the sale in the sales collection.
Alternative Scenario	1. If stock is insufficient for any product, the system notifies the Admin and prevents checkout. 2. If the transaction fails to save due to a system or network error, the system displays an appropriate error message.

Table 3.8 Use Case of View Sales Report

Use Case Name	View sales report
Description	This use case allows the Admin to view visualized sales data in the form of graphs and tables for daily, weekly, monthly, and yearly summaries. These graphs assist in inventory planning and business decision-making.
Primary Actor	Admin
Pre-condition	Admin must be logged in and navigate to the "Sales Report" section. Sales data must be available in the database.
Post-condition	The system displays sales data in chart and table formats based on the selected time range.
Main Success Scenario	1. Admin accesses the sales report module. 2. Admin selects a time filter (daily, weekly, monthly, or yearly). 3. System retrieves the relevant sales data. 4. Sales data is displayed using bar and line graphs with corresponding figures.
Alternative Scenario	1. If no sales data is available, the system displays a message indicating no data for the selected period. 2. If the graph fails to load, the system retries fetching the data and displays an error if unsuccessful.

Table 3.9 Use Case of Staff Management

Use Case Name	Staff management
Description	This use case allows the Admin to manage staff accounts by creating, viewing, updating, and deleting them. When a new staff account is created, it is initially marked as "Pending." The staff member must verify their email and log in through the Android app to change their status to "Verified."
Primary Actor	Admin
Pre-condition	Admin must be logged in and navigate to the staff management section on the web dashboard.
Post-condition	Staff records are updated in the system, including new registrations, modifications, deletions, and verification status updates.
Main Success Scenario	1. Admin selects a user management function: ● Create: Admin enters the new staff's details (email, display name). Account is added with "Pending" status. ● Read: Admin views the list of staff and their current statuses (Pending/Verified). ● Update: Admin edits staff display name or removes staff manually. ● Delete: Admin deletes a selected staff account. 2. The system processes the chosen action and updates the Firebase database accordingly. 3. For new staff, a verification email is sent. 4. When the staff verifies their email and logs into the Android app, the system automatically updates their status

	to “Verified.” 5. The admin panel reflects the updated verification status.
Alternative Scenario	1. If required information is missing during creation or update, the system prompts the admin to complete the data. 2. If the admin tries to delete a non-existent staff account, the system shows an error. 3. If network issues occur, the system retries or notifies the admin of the failure..

Table 3.10 Use Case of Login

Use Case Name	Login
Description	This use case allows both Admin and Staff actors to securely log into the system via the web dashboard (Admin) or the Android app (Staff). Newly created staff accounts must first verify their email before gaining access.
Primary Actor	Admin, Staff
Pre-condition	The actor must have a valid email and password. Staff must have verified their email address.
Post-condition	The actor is granted access to the appropriate platform based on their role (Admin → Web; Staff → Android).
Main Success Scenario	1. Actor opens the login screen. 2. Enters valid credentials. 3. System verifies credentials using Firebase Authentication.

	4. Actor is redirected to their respective dashboard.
Alternative Scenario	1. If credentials are invalid, the system displays an error message. 2. If staff account is unverified, access is denied until verification is complete. 3. If there is a network issue, the system notifies the actor.

Table 3.11 Use Case of Logout

Use Case Name	Logout
Description	This use case allows Admin and Staff actors to securely log out of the system and end their active session.
Primary Actor	Admin, Staff (User)
Pre-condition	The actor must be logged into the system.
Post-condition	Actor is logged out and session is terminated.
Main Success Scenario	1. Actor taps the logout option. 2. System ends the session. 3. Actor is returned to the login screen.
Alternative Scenario	1. If logout fails, system shows an error.

3.4.3 Sequence Diagram

3.4.3.1 Create a New Product

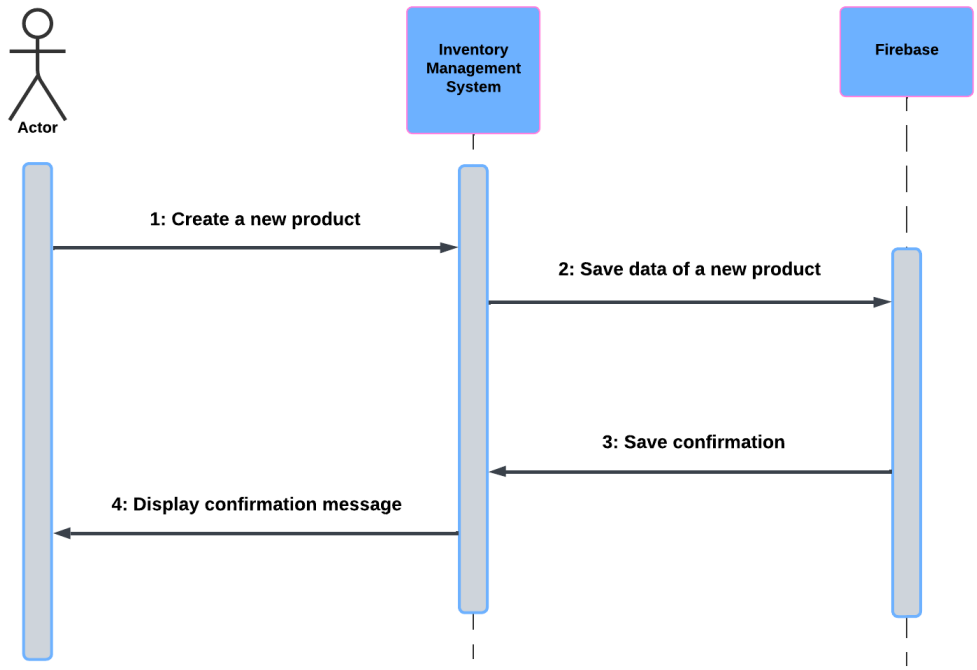

Figure 3.16 Create a New Product Sequence Diagram

Figure 3.16 shows the interaction between the actor (retail staff and admin), the Inventory Management System, and Firebase. The process begins when the actor initiates the addition of a new product through the inventory management system. Product details can be provided either manually or using voice commands. The inventory management system processes the input and sends the product information to Firebase for storage. After successful data storage, Firebase confirms the addition of the product, and the inventory management system displays a confirmation message to the actor.

3.4.3.2 Update the Product's Details

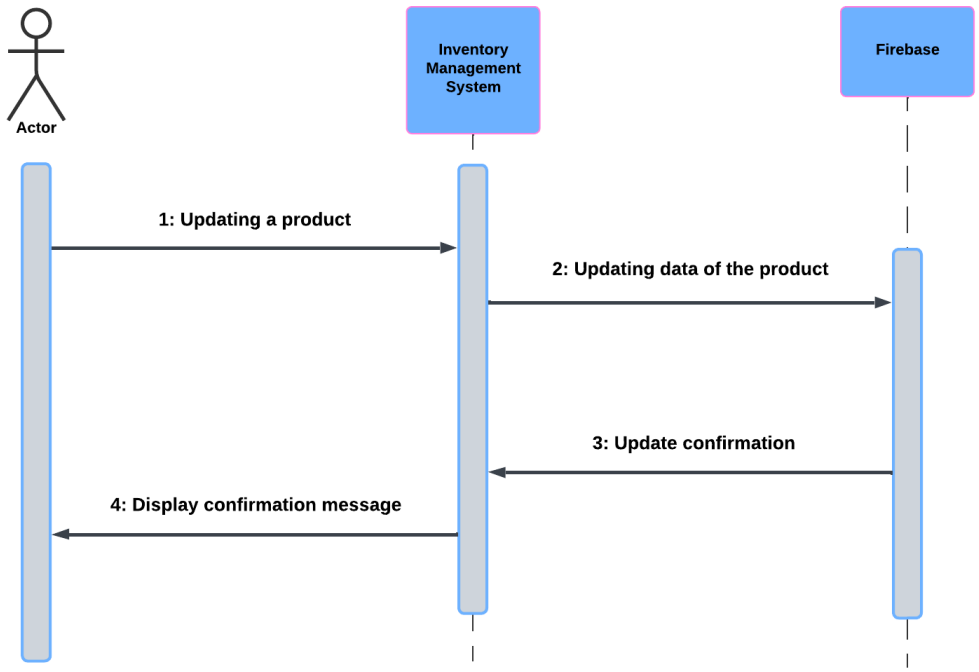

Figure 3.17 Update the Product's Details Sequence Diagram

Figure 3.17 shows the actor (Retail Staff and Admin) interacting with the Inventory Management System and Firebase. The process begins when the actor initiates an update for a product through the inventory management system. The actor can provide the updated information manually or using voice commands. Then, the inventory management system processes the input and sends the updated data to Firebase for storage. Once Firebase confirms the update, the inventory management system displays a confirmation message to the actor.

3.4.3.3 Read the Product's Details

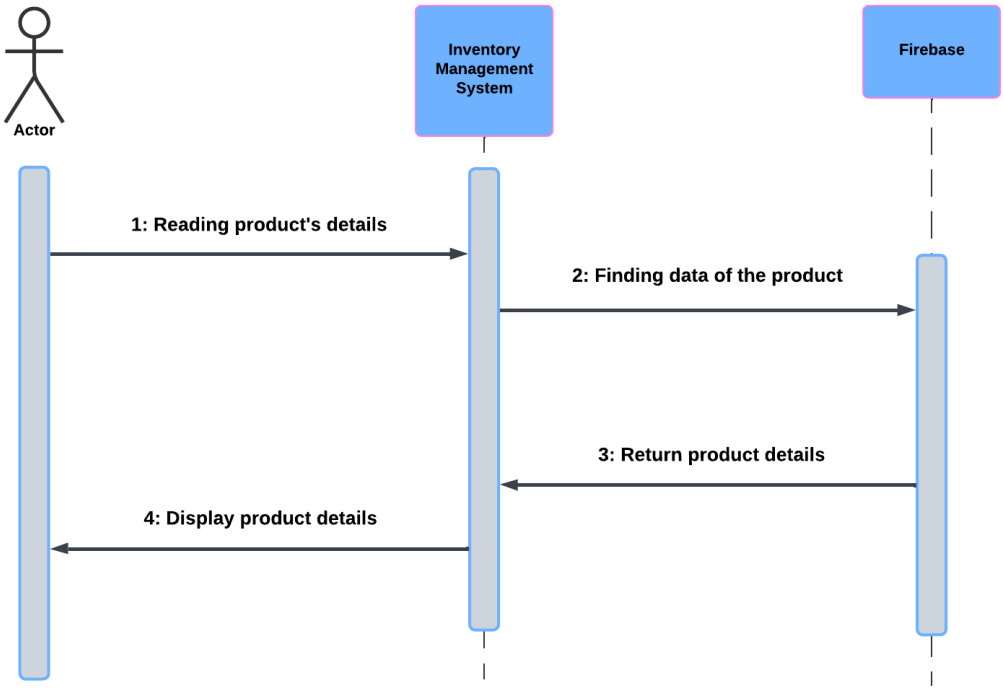

Figure 3.18 Read the Product's Details Sequence Diagram

Figure 3.18 shows the interaction between the actors (Retail Staff and Admin), the Inventory Management System, and Firebase. The process begins when the actor initiates a request to view product information through the inventory management system. The actor can specify the product manually or via voice commands. Then, the inventory management system processes the request and retrieves the product details from Firebase. Once Firebase provides the product information, the inventory management system displays it to the actor.

3.4.3.4 Delete an Existing Product

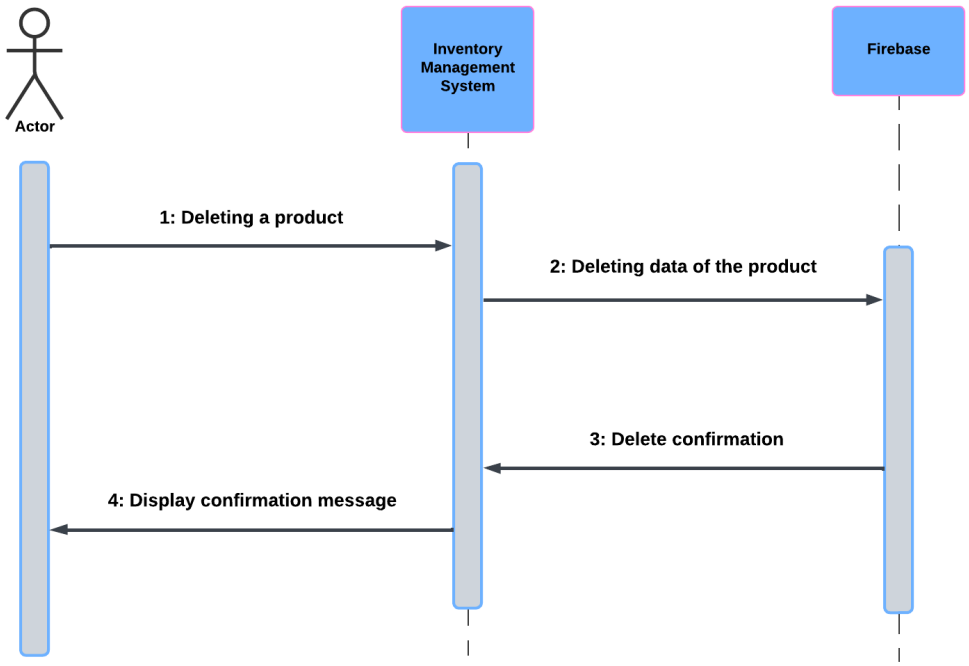

Figure 3.19 Delete an Existing Product Sequence Diagram

The sequence diagram for deleting an existing product shows the interaction between the actor (Retail Staff and Admin), the Inventory Management System, and Firebase. The actor starts the deletion process through the inventory management system, specifying the product either manually or via voice commands. Then, the inventory management system processes the input and sends a request to Firebase to remove the product. Once Firebase confirms the deletion, the inventory management system displays a confirmation message to the actor.

3.4.3.5 Retail Staff Management

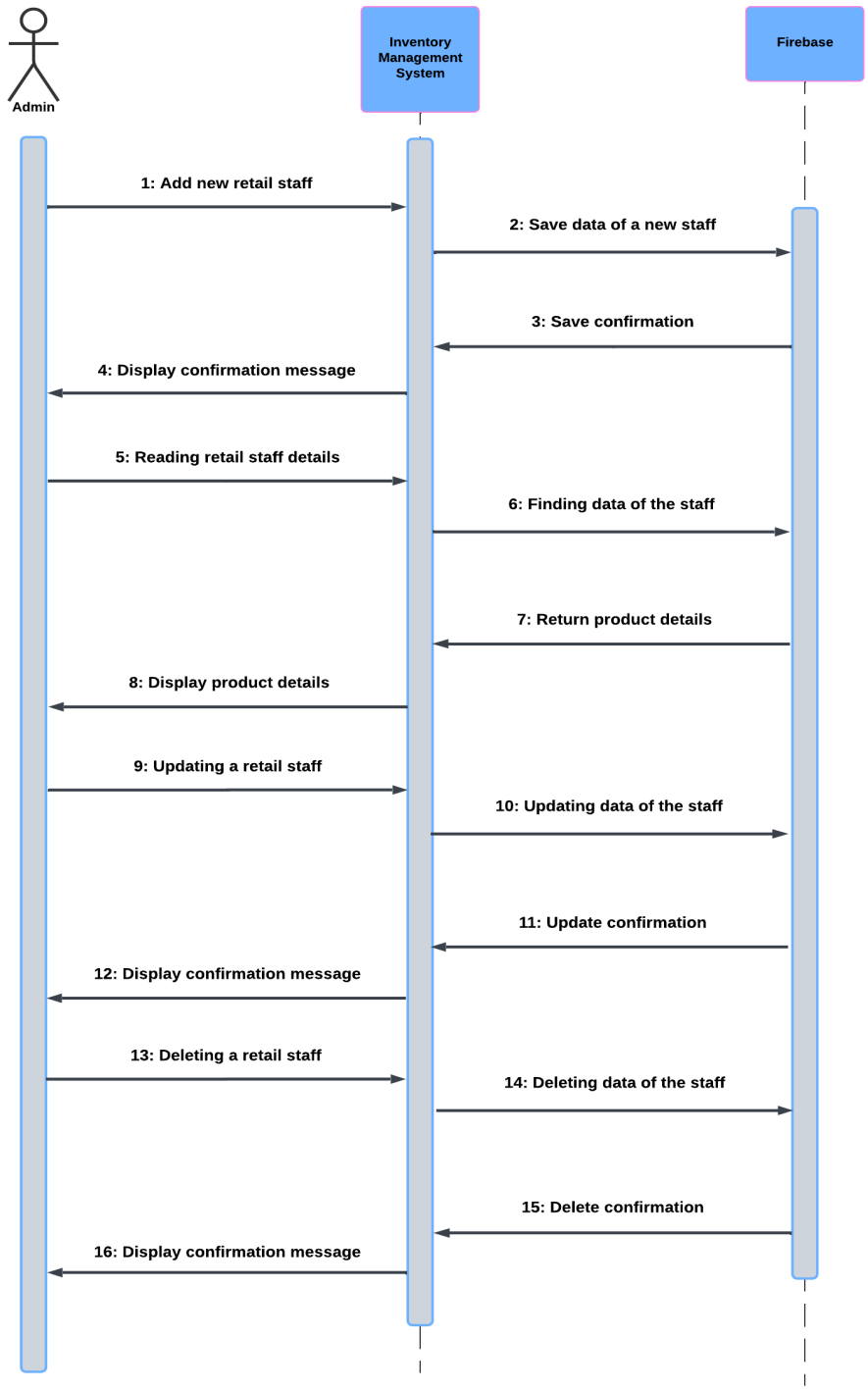

Figure 3.20 Retail Staff Management Sequence Diagram

Figure 3.21 illustrates the interaction between the admin, the inventory management system, and Firebase. The admin initiates actions such as adding, reading, updating, or deleting a retail staff member. The inventory management system processes these actions by interacting

with Firebase to manage staff data. Firebase performs the requested operation and sends back confirmations, which are then displayed to the admin as confirmation messages.

3.4.3.6 View Sales Reports

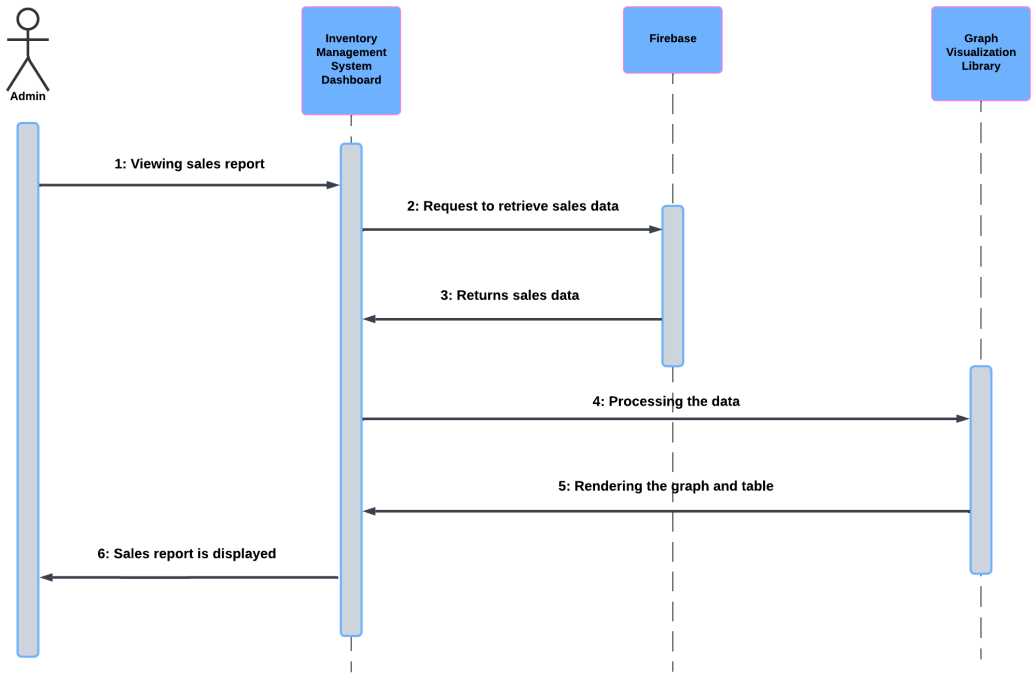

Figure 3.21 View Sales Reports Sequence Diagram

Figure 3.21 shows the interaction between the admin, the inventory management system dashboard, Firebase, and the graph visualization library. The process begins when the admin initiates a request to view the sales report through the inventory management system dashboard. The inventory management system sends a request to Firebase to retrieve the relevant sales data. Upon receiving the data, the inventory management system processes it and sends it to the Graph Visualization Library for rendering. Finally, the processed data is displayed to the admin in the form of a graph and table, completing the interaction.

3.4.4 Entity Relationship Diagram

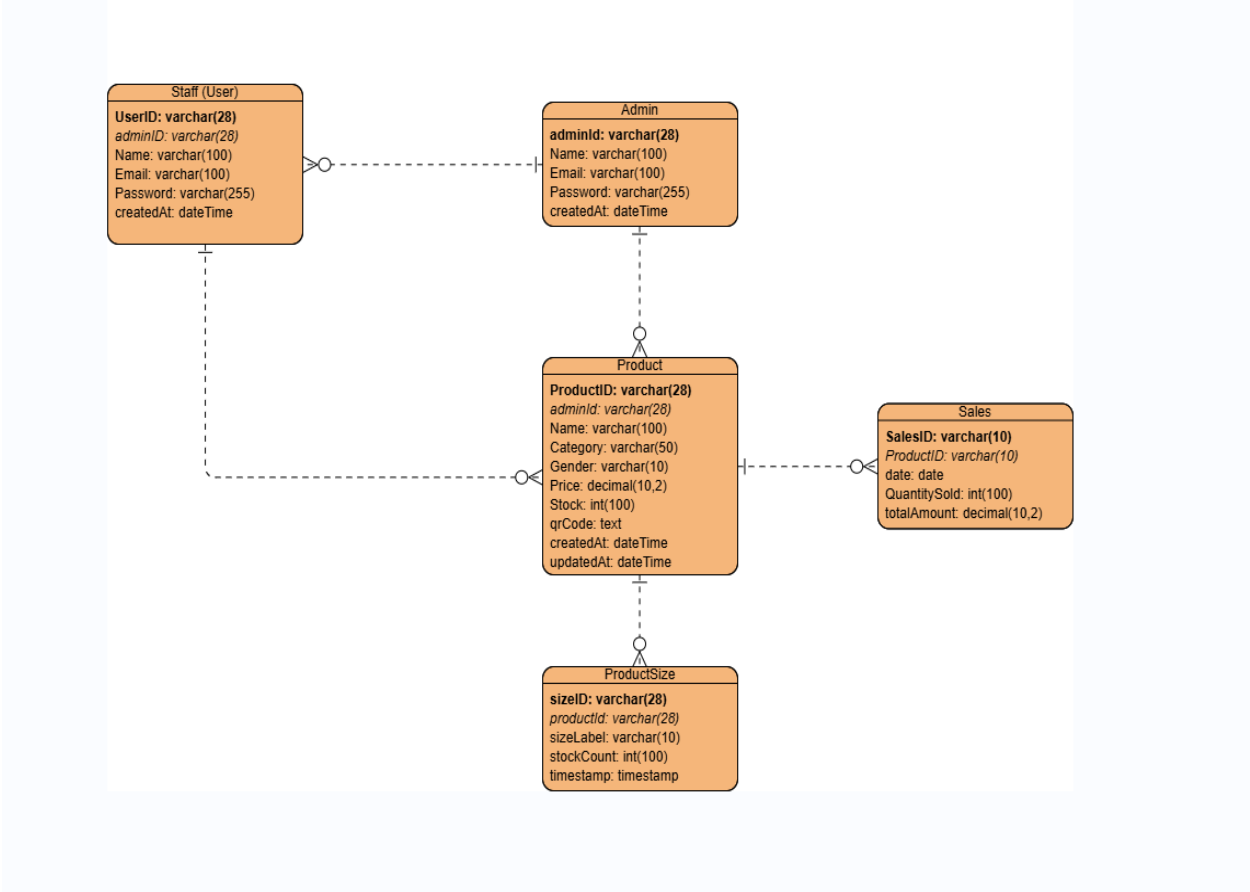

Figure 3.22 Entity Relationship Diagram

The Entity-Relationship Diagram (ERD) shows how the main parts of the system (called entities) are connected and how they work together.

There are five key entities in this system:

- Product

This refers to items in the inventory. Each product has important details like its name, category (e.g., shirt, backpack), target gender, price, stock quantity, QR code, and timestamps showing when it was added or last updated. A product can come in different sizes or colors, each with its own stock level.

- Retail Staff

These are the regular users who use the Android app to manage the inventory. Each staff member has a unique ID, name, username, and password. They can add, edit, or delete products, depending on their access.

- Admin

Admins have all the features of Retail Staff but with more control. They can manage staff accounts, perform full CRUD (Create, Read, Update, Delete) actions on any product, and view sales reports through the web dashboard.

- Sales

This entity records every transaction when a product is sold. It stores details such as which product was sold, how many units, the size (if applicable), the total sales amount, the date of the transaction, and who handled the sale.

- ProductSize

This table stores different size or color variations of a product (like "Red", "L", "XL", etc.) and how many units are available for each variation. This helps keep inventory tracking more accurate.

Table 3.7 Data Dictionary of the Entity Relationship Diagram (ERD)

Table	**Field Name**	**Data Type**	**Field Size**	**Description**	**Example**
Product	ProductID	varchar	10	Primary Key	P001
	Name	varchar	255	Product name	Adidas Adizero
	Category	varchar	50	Category the product belongs to	Shoes
	Gender	varchar	20	Target gender for the product	Unisex
	Price	float	10	Price per unit of the product	219.90
	Stock	integer	10	Total	42

				available quantity in stock	
	QRCode	varchar	255	QR code URL for the product	-
	AdminID	varchar	10	Foreign key linking to Admin	A001
	CreatedAt	timestamp	-	Timestamp of product creation	2025-06-08 02:35:20
	UpdatedAt	timestamp	-	Timestamp of last update of the product	2025-06-12 15:09:19
Staff (User)	userId	varchar	28	Primary Key	U001
	adminId	varchar	28	Foreign Key linking to Admin	A001
	Name	varchar	100	Name of the staff	Aiman Danial
	email	varchar	100	Staff email address	aimandanial@gmail.com
	Password	varchar	255	Login password using Firebase Authentication	aimandanial123
	createdAt	dateTime	-	When the staff account was created	2025-06-12 15:09:19
Admin	adminId	varchar	28	Primary Key	A001

	Name	varchar	100	Full name of the admin	Aiman Danial
	Email	varchar	100	Admin email address	aimandanial@gmail.com
	Password	varchar	255	Admin password using Firebase Authentication	aimandanial123
	createdAt	dateTime	-	Date and time when the admin account was created	2025-05-10 10:30:00
Sales	SalesID	varchar	28	Primary Key	S001
	ProductID	varchar	28	Foreign Key linking to Product	P001
	Date	date	-	Date of the sale	2024-10-13
	QuantitySold	int	100	Quantity of items sold	15
	totalAmount	decimal	10,2	Total price paid for the sold items	2500.00
ProductSize	sizeId	varchar	28	Primary Key	S001
	productId	varchar	28	Foreign Key linking to Product	P001
	sizeLabel	varchar	10	Label for size, color or	L, Red, 42

				variant	
	stockCount	int	100	Quantity available for this specific size	20

3.4.5 Wireframes

3.4.5.1 Wireframes for User App (Android)

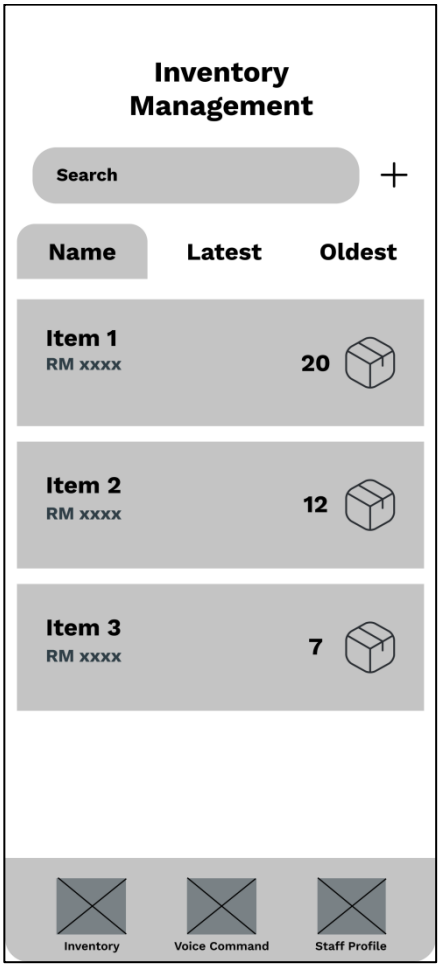

Figure 3.23

Figure 3.23 shows a wireframe of the inventory management interface after opening the Android app. The bottom navigation bar has three buttons: Inventory, Voice Command, and Staff Profile. There is a search bar to find items and sorting options like Name, Latest, and

Oldest. Each item displays its name, price, and quantity. An Add button next to the search bar allows the admin to add new products.

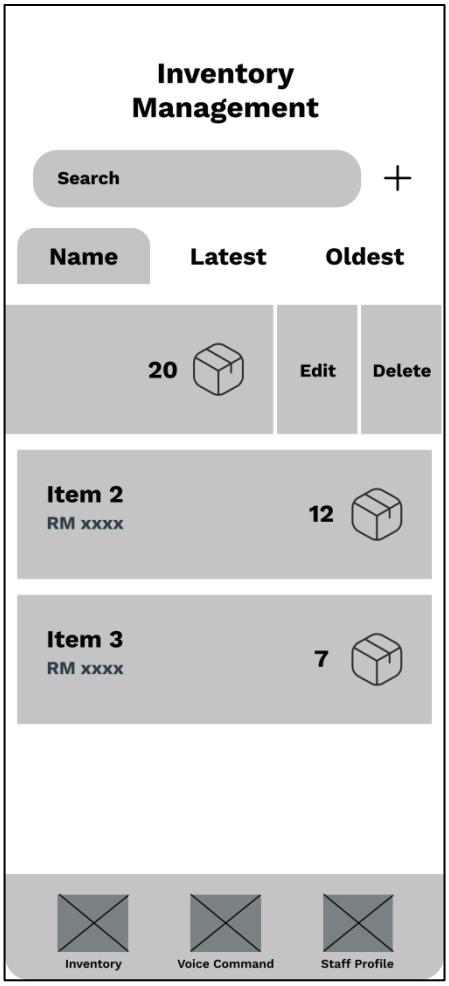

Figure 3.24

Figure 3.24 shows users can find the edit and delete product buttons by swiping an item to the left.

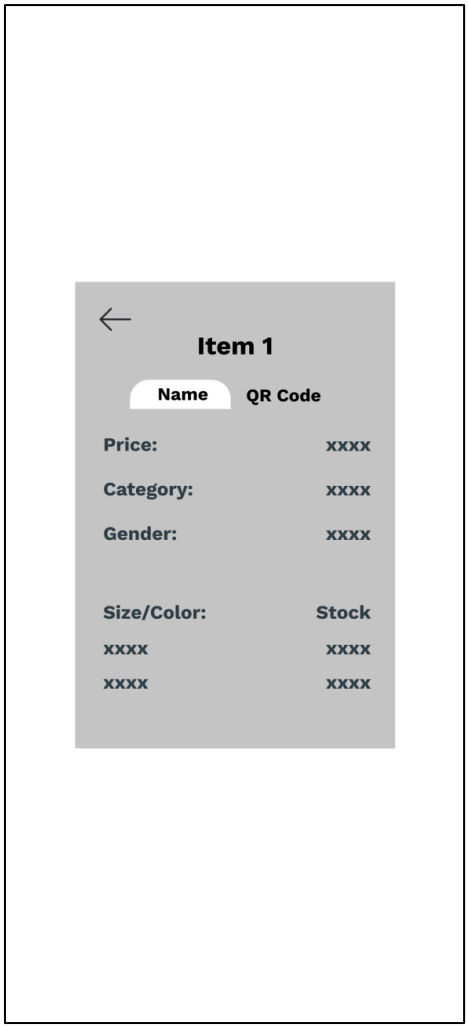

Figure 3.25

Figure 3.25 shows the wireframe of the product details interface that appears when you select an item. It includes a sorting option, product details, and a QR code.

Figure 3.26

Figure 3.26 shows the wireframe of the QR Code popup when pressing it in the sorting option. The QR code can be scanned using a phone camera and it will lead you to a URL link where it shows the particular item details.

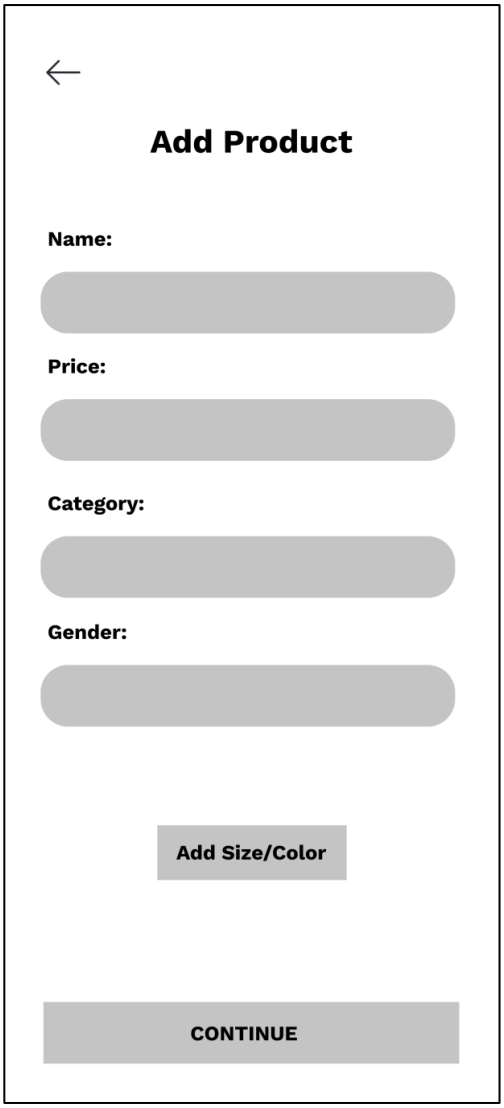

Figure 3.27

Figure 3.27 shows the wireframe for adding a new product after pressing the plus icon in the inventory list. There is an 'Add Size/Color' button that allows you to add sizes, colors, or other variants as appropriate for the product.

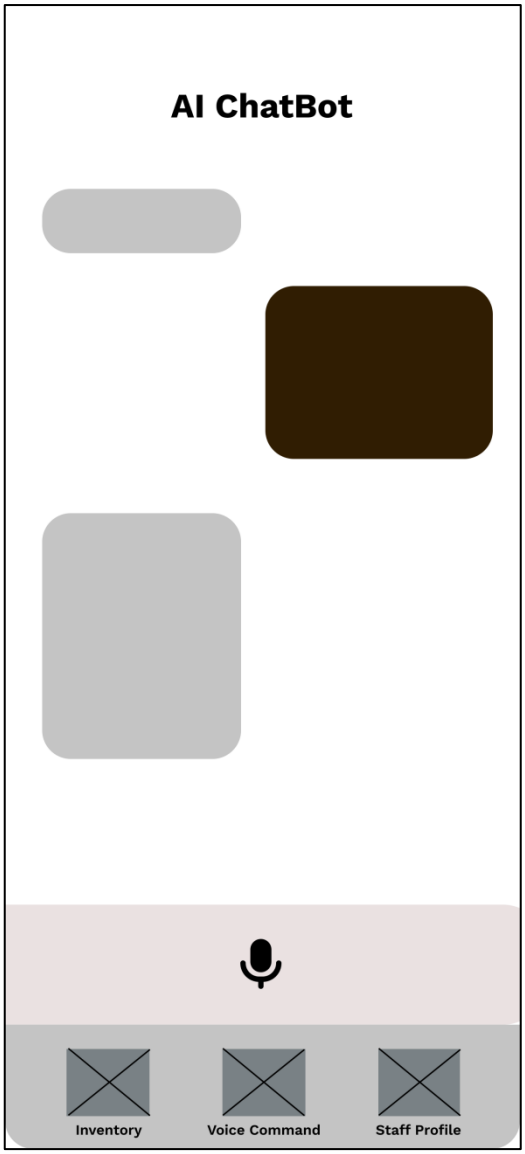

Figure 3.28

Figure 3.28 shows the wireframe of an AI Chatbot after pressing the voice command button. Users can perform CRUD operations for products using voice commands. The bottom navigation bar remains visible while using the voice command feature.

3.4.5.2 Wireframe for Web App

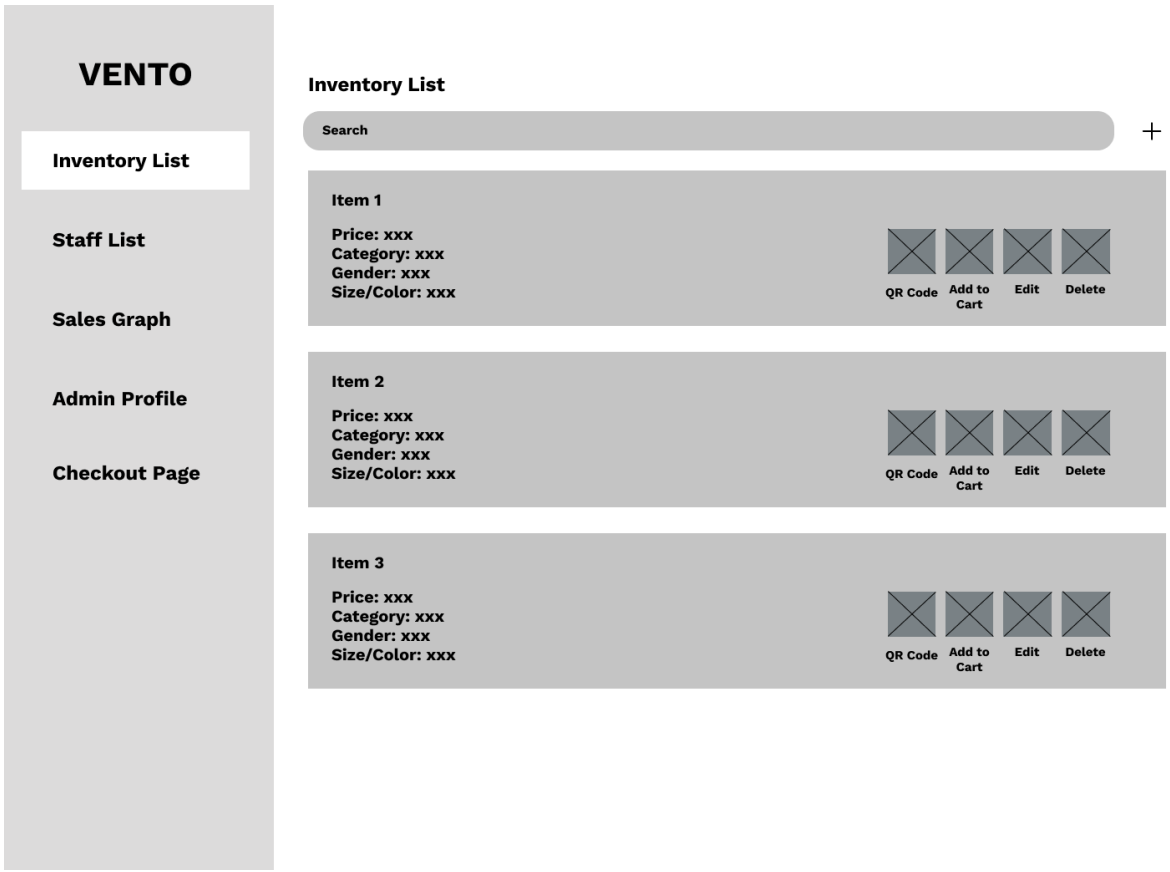

Figure 3.29

Figure 3.29 shows a wireframe of the inventory management interface in the Admin web app. The sidebar has five buttons: Inventory List, Staff List, Sales Graph, Admin Profile, and Checkout Page. There is a search bar to find items and also key points for category and gender. Each item displays its name, price, category, gender, size/color, and stock quantity. An Add button next to the search bar allows the admin to add new products.

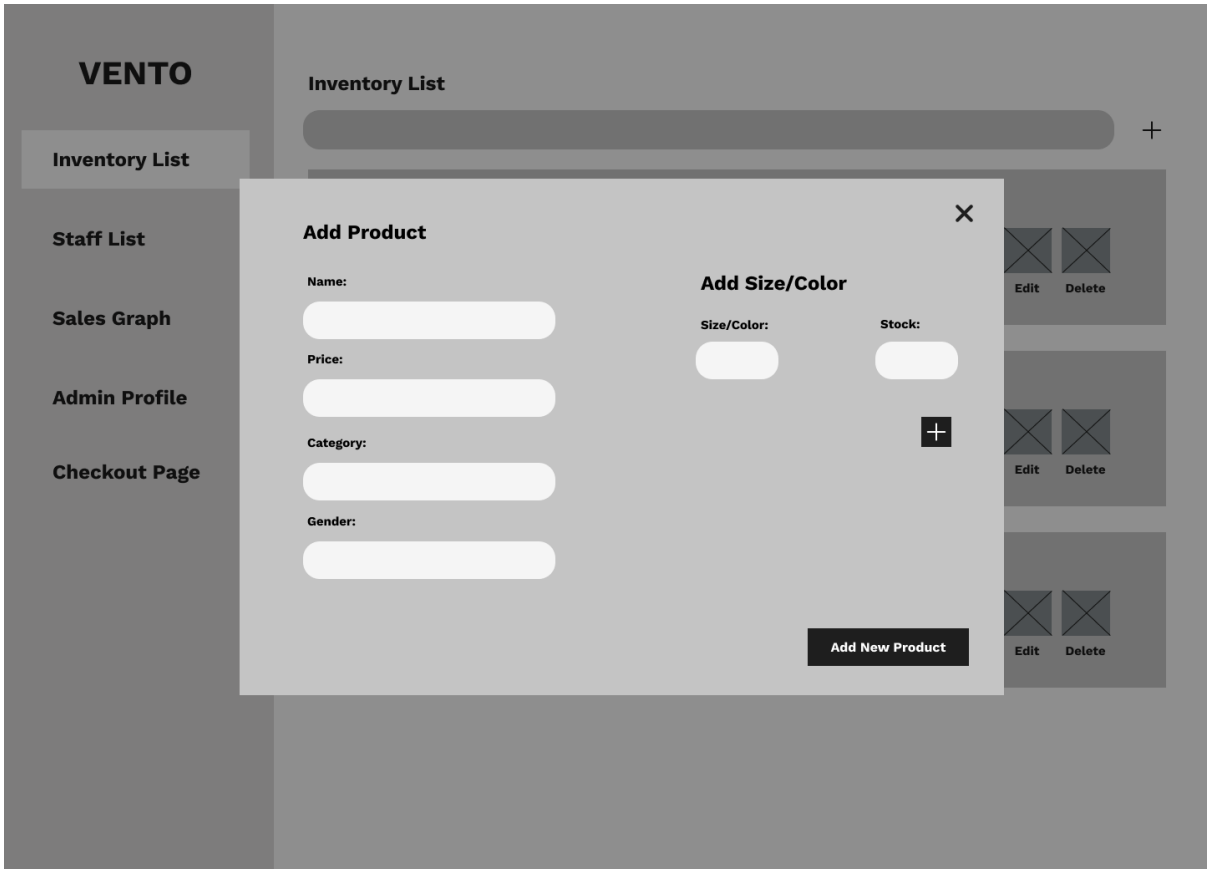

Figure 3.30

Figure 3.30 shows the wireframe for adding a new product after pressing the plus icon in the inventory list. There is a 'Add Size/Color' besides ‘Add Product’ that allows you to add sizes, colors, or other variants as appropriate for the product.

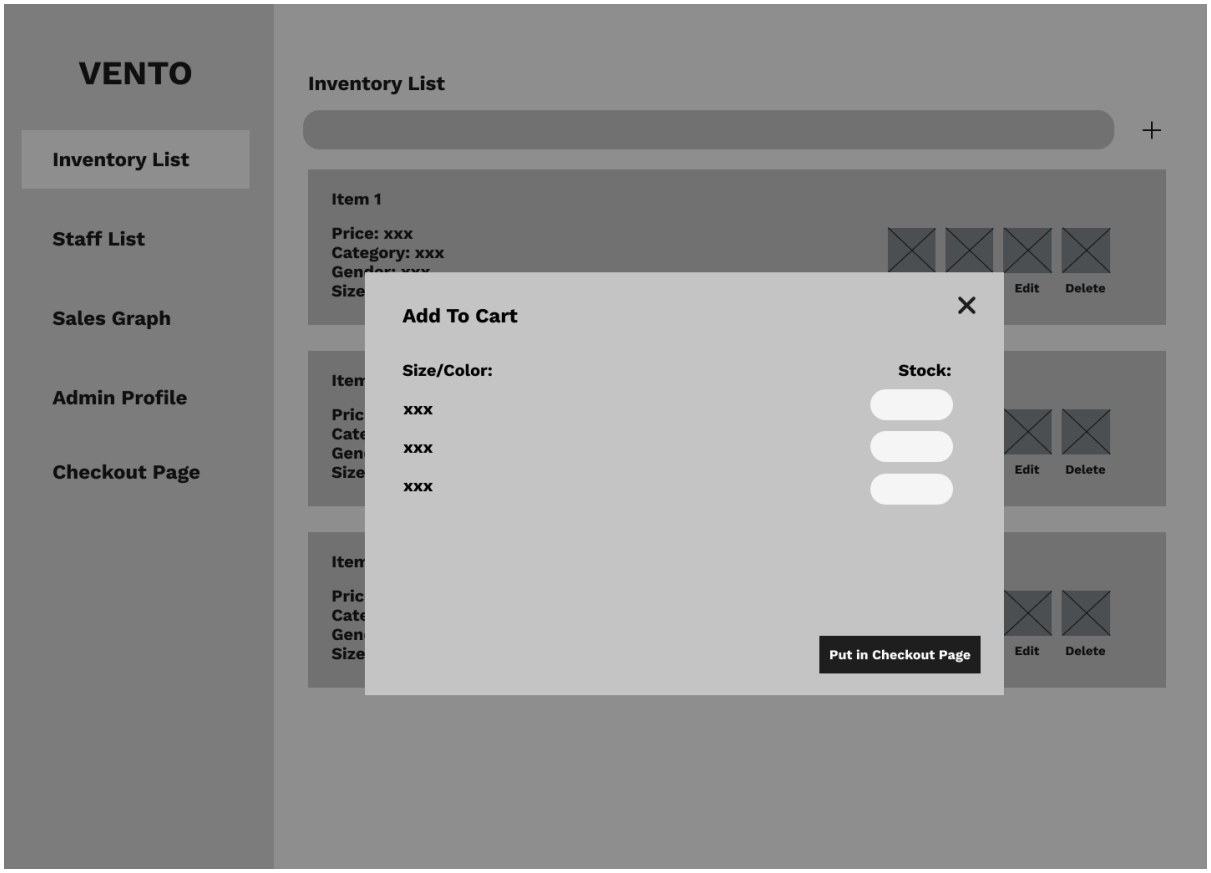

Figure 3.31

Figure 3.31 shows the wireframe of the 'Add to Cart' popup that appears after pressing the 'Add to Cart' button in the inventory list. It displays a list of registered sizes/colors along with the available stock for each. The admin can specify the quantity to add to the cart before proceeding to the checkout page.

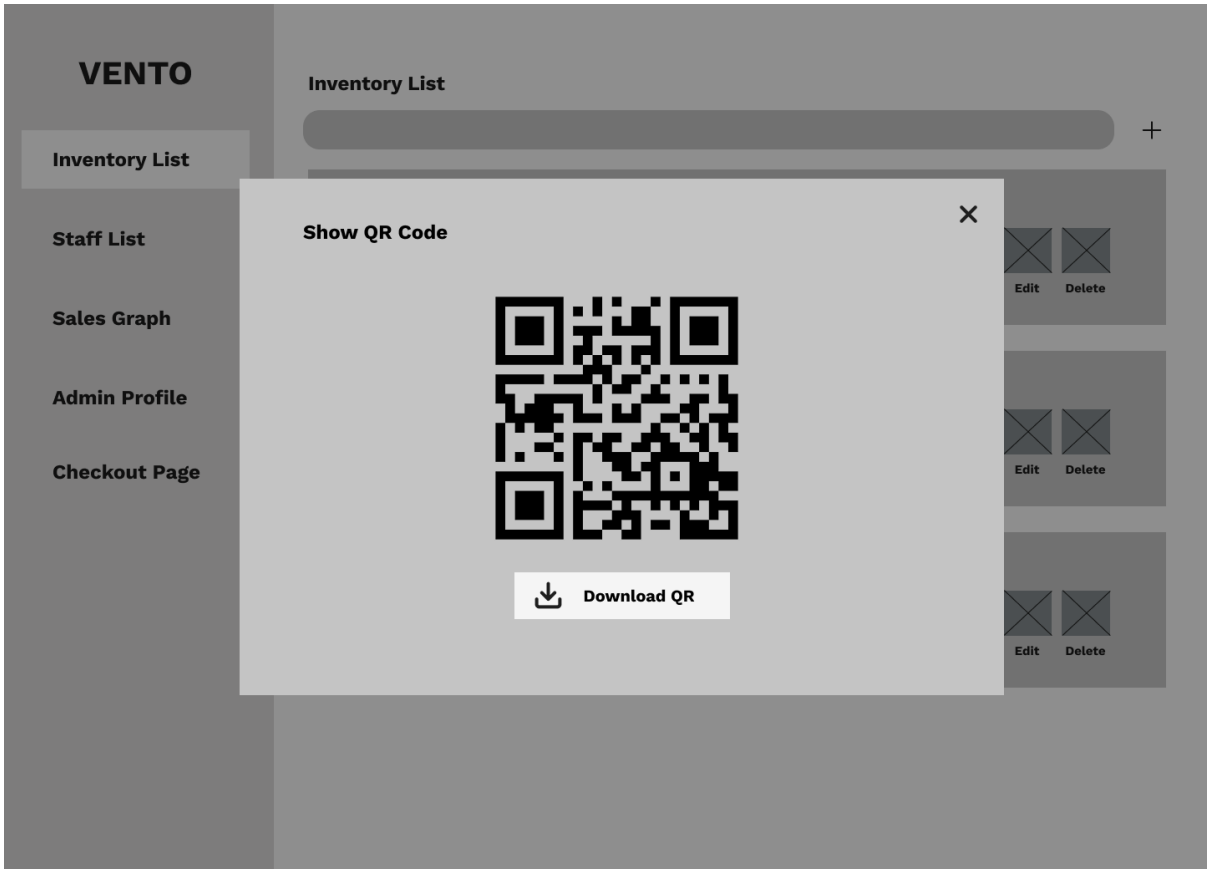

Figure 3.32

Figure 3.32 shows the wireframe of the 'Show QR Code' popup that appears after pressing the 'Show QR Code' button in the inventory list. The QR code can be scanned using a phone camera, which will direct the user to a URL displaying the specific product details.

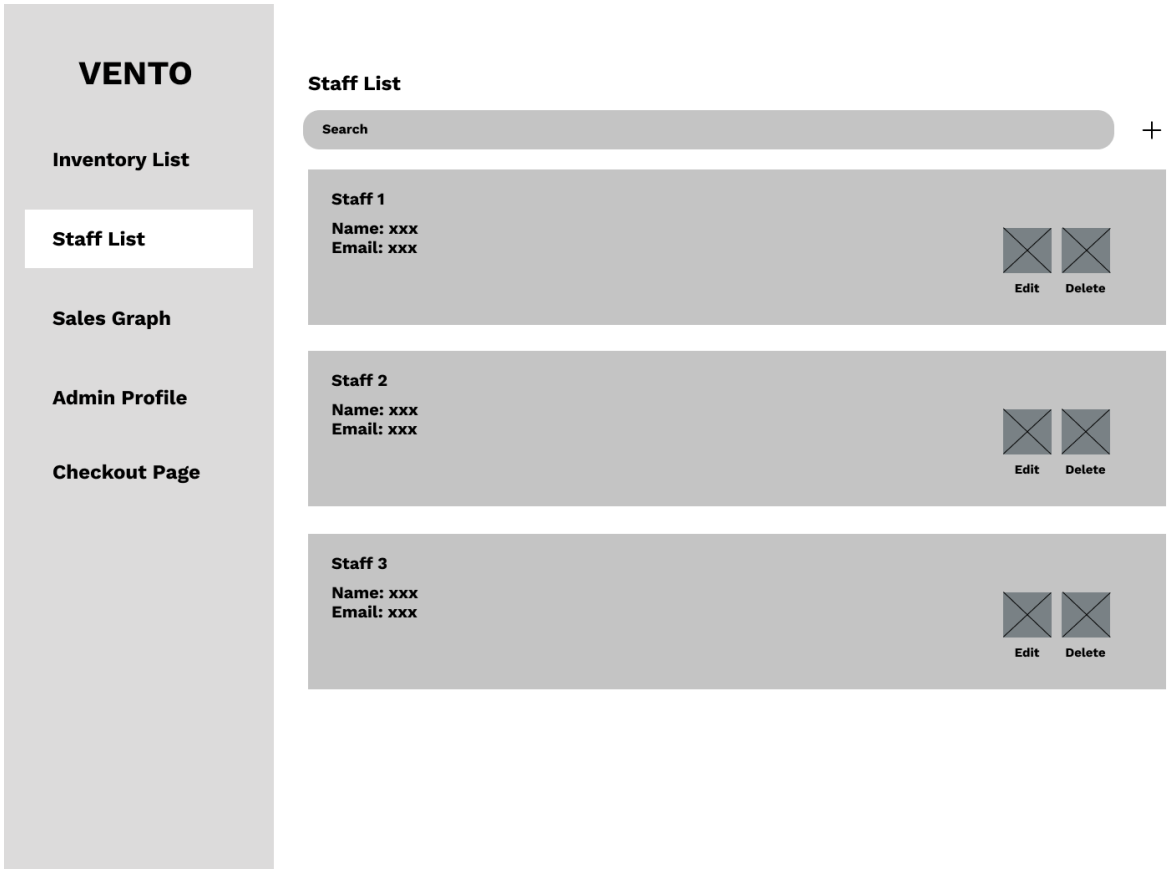

Figure 3.33

Figure 3.33 shows the wireframe of the staff list in the admin web app. The admin is responsible for managing staff (user) accounts and can create, read, update, and delete accounts under their supervision.

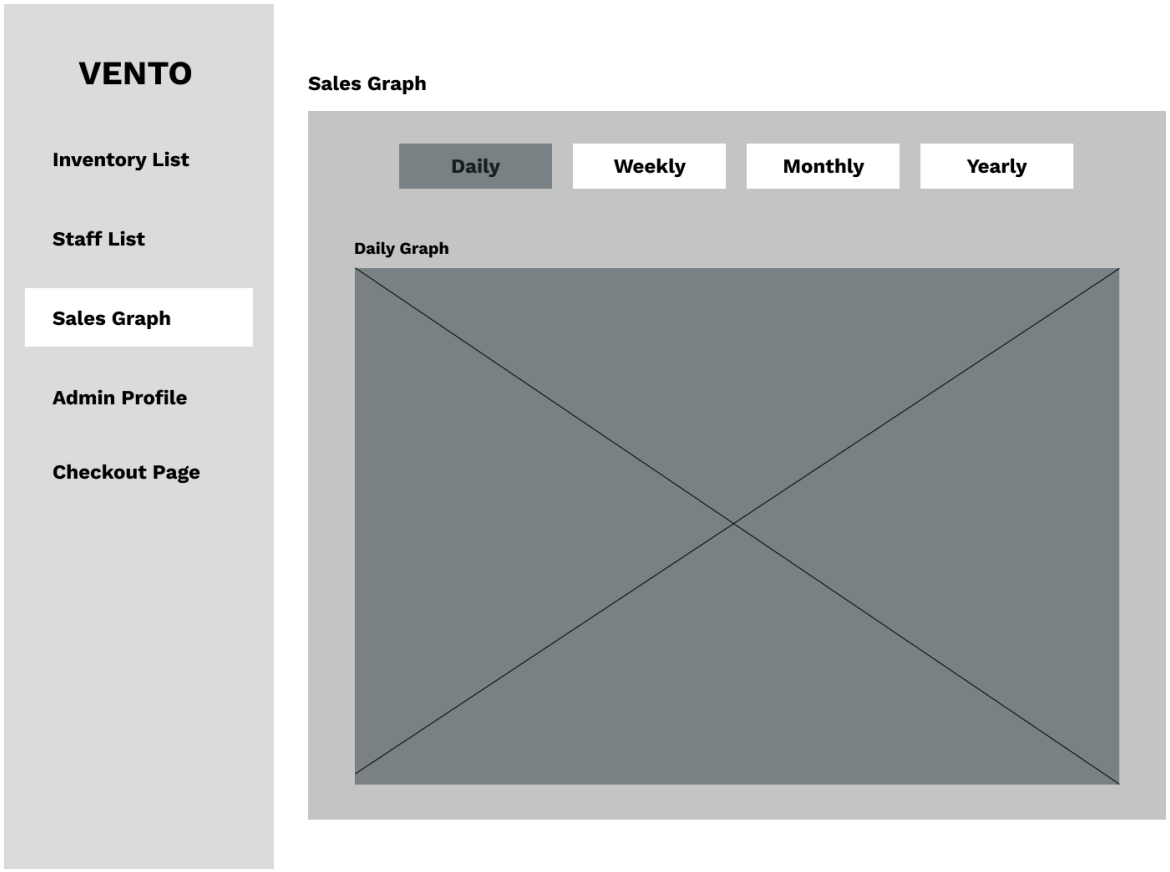

Figure 3.34

Figure 3.34 shows the wireframe of the sales graph. The admin can view a graph of completed sales, with a sorting option to display the data by day, week, month, or year.

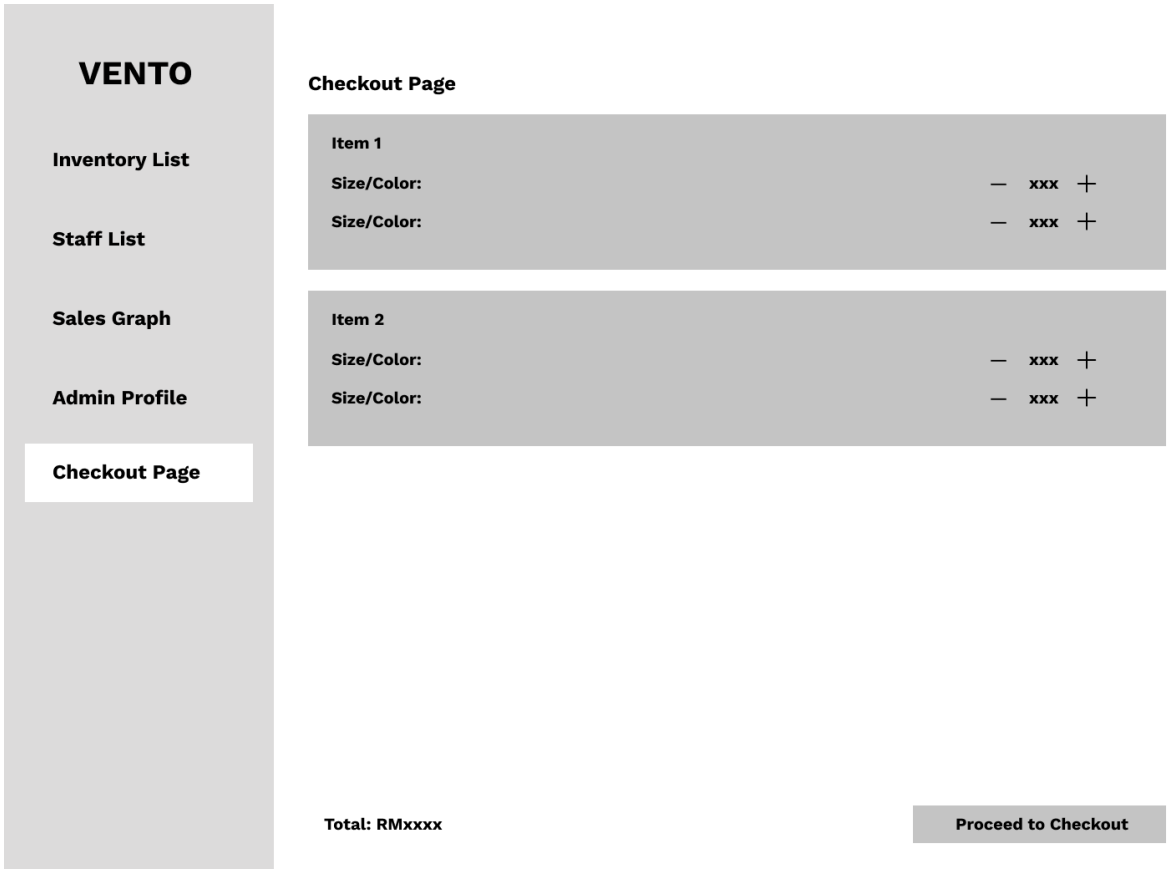

Figure 3.35

Figure 3.35 shows the wireframe of the checkout page. The items displayed are those selected for checkout after specifying the size/color stock quantities in the 'Add to Cart' popup shown in Figure 3.31. The admin can increase or decrease the quantity for each item's size/color. The total price is shown at the bottom, along with a 'Proceed to Checkout' button.

3.5 SUMMARY

This chapter reviews the personal scrum methodology as a key approach in the project. It also outlines the essential requirements for development, including a questionnaire survey, functional and non-functional requirements, as well as hardware and software specifications. Additionally, it presents the overall system architecture diagram, use case diagram with detailed tables, sequence diagrams, and entity relationship diagram, all of which are crucial for building a well-structured voice-assisted inventory management system app. Wireframe designs are also included to illustrate the app's layout, structure, and functionality.

CHAPTER 4: SYSTEM IMPLEMENTATION

4.1 INTRODUCTION

This chapter presents the implementation process of Vento: Voice-Assisted Inventory Management System, developed for small to medium-sized retail stores. It describes the development tools, frameworks, and environment used to build both the Android mobile application for retail staff and the web-based dashboard for the admin.

The implementation focuses on key system functionalities, including:

- Firebase integration for real-time cloud-based data storage and synchronization.
- Voice input features to support hands-free operations for retail staff.
- CRUD functionalities for product and staff management.
- QR code generation and scanning for efficient product lookup.
- Sales recording and reporting features available to admins via the web interface.

The system was built using Flutter to support cross-platform development, with Firebase providing backend services such as authentication, database, and hosting. The Android application is designed specifically for retail staff, allowing them to manage inventory using either manual or voice input. Meanwhile, the web application is developed for admins, who are responsible for managing product inventory, staff accounts, and overseeing sales performance through visual analytics.

The implementation prioritizes usability, performance, and accessibility, ensuring that store employees can perform inventory tasks quickly and efficiently while providing store managers with full control and oversight through a user-friendly admin dashboard.

4.2 FRONTEND DEVELOPMENT SETUP

The frontend of the Vento system was developed using Flutter, an open-source UI toolkit by Google that supports the creation of cross-platform applications from a single codebase. Flutter was chosen for its rapid development features, customizable widgets, and consistent design output across platforms. It supports the development of both the Android mobile application for staff (users) and the web dashboard for admins, ensuring consistency and reusability in the user interface components.

- The Android app is used by retail staff, who are responsible for day-to-day inventory operations. It supports both manual input and voice command features for managing products.
- The web-based dashboard is used by the admin, who holds elevated permissions, including product management, staff account control, and access to visual sales reports.

Both interfaces are intentionally designed to be simple, responsive, and user-friendly, enabling smooth navigation for users with varying levels of technical experience.

Below are the steps to set up Flutter in Android Studio:

Step 1: Install Flutter SDK

Download the Flutter SDK from the official website and extract it. Add its path to the system environment variables to allow command-line usage.

Step 2: Install Android Studio

Android Studio was selected as the main software development tool for its compatibility with Flutter and Android projects.

Step 3: Install Flutter and Dart Plugins

Navigate to Plugins in Android Studio's settings, then install both the Flutter and Dart plugins. These are essential for creating and maintaining Flutter projects.

Step 4: Create a New Flutter Project

From the welcome screen, select "New Flutter Project," configure the project name, path, and template.

Step 5: Enable Web Support

To develop and test the web dashboard for the admin, ensure Flutter web support is enabled. The app can be run on supported browsers such as Chrome.

Step 6: Organize the Project Structure

Maintain a clean and modular codebase by organizing files into the following folders:

- /screens – UI layouts for different pages
- /widgets – Reusable UI components
- /models – Data structures for products, users, etc.
- /services – Logic and API/database interaction

Step 7: Test the Application

Use Android emulators or physical devices to test the mobile app from the staff perspective. Use a browser to test the web admin dashboard. Confirm that layouts, navigation, and data synchronization work correctly on both platforms.

This setup ensures a consistent and scalable frontend architecture, while clearly supporting role-based functionalities. Staff members have access to inventory CRUD operations through the Android app, while admins manage products, view reports, and oversee staff accounts via the web dashboard. This separation of roles and permissions is maintained throughout the UI and enforced during login and task execution.

4.3 BACKEND DEVELOPMENT SETUP

The backend of the system was implemented using Firebase, a cloud-based platform that offers a scalable, secure, and real-time infrastructure for mobile and web applications. Firebase was selected due to its seamless integration with Flutter and its ability to handle critical backend tasks such as data storage, user authentication, and file hosting efficiently.

The system utilizes Cloud Firestore, Firebase Authentication, and Firebase Storage to power core functionalities:

- Cloud Firestore is used to store and manage structured data for products, inventory records, staff accounts, and sales transactions. It enables real-time data updates, ensuring

that any changes made by either admin or staff are immediately synchronized across all devices and platforms.

- Firebase Authentication secures the login process for both admin and staff. Role-based access control is enforced through user roles stored in Firestore. Admins have extended permissions such as managing staff records and viewing sales reports, while staff are limited to performing product-related operations.
- Firebase Storage is used to store media files such as auto-generated QR codes, which are linked to individual products. These files are securely stored and can be accessed from both the Android app (staff) and the web dashboard (admin).

Additionally, the system distinguishes between admin and staff roles during the login and registration process:

- Admins register new staff via the web dashboard. New staff accounts are initially marked as pending and must verify their email before accessing the system.
- Once verified and logged in via the Android app, the staff's status is updated to active, granting them access to inventory management features.

Below are the steps to set up Firebase:

Step 1: Create a Firebase Project

A new Firebase project was created through the Firebase Console. The project was named according to system requirements and served as the central backend environment for both mobile and web platforms.

Step 2: Register the Android App and Web App

Both the Android mobile app and the web dashboard were registered within the Firebase Console.

- For Android, the google-services.json file was downloaded and placed in the android/app directory.
- For the web app, the firebase-config.js file was added to enable Firebase in the browser environment.

Step 3: Enable Required Firebase Services

The following services were enabled:

- Cloud Firestore: To manage real-time data storage.
- Firebase Authentication: To handle login and account management.
- Firebase Storage: To upload and retrieve QR code images.

Step 4: Add Firebase Dependencies to the Flutter Project

Firebase-related packages were added to the pubspec.yaml file:

- firebase_core
- cloud_firestore
- firebase_auth
- firebase_storage

Firebase was initialized in the main.dart file using Firebase.initializeApp().

Step 5: Integrate Firebase into the Application Code

Custom functions were written in Dart to:

- Perform CRUD operations on Firestore (products, staff, sales).
- Authenticate users and enforce role-based access.
- Upload and retrieve QR code images via Firebase Storage.

By following these steps, the system successfully set up a reliable backend that allows real-time inventory updates, supports voice-based actions, and ensures secure access for both Android app and web dashboard users.

4.4 MODULE IMPLEMENTATION

This section describes the detailed module implementation of Vento: Voice-Assisted Inventory Management System, developed using Flutter and integrated with Firebase Firestore. The system is divided into two main user sessions: the Admin (Web) and the User (Mobile Android App). Each session contains core modules designed for seamless interaction, efficient product and staff management, and real-time synchronization using Firebase.

a) User (Staff - Android App) Session

- Login
- View Inventory List
- Product Details Popup
- Size/Color Popup
- QR Code Popup
- Add Product Page
- Edit Product Page
- Delete Confirmation Dialog
- User Profile
- Logout
- Voice Assistant (Add/Edit/View/Delete Products Hands-Free)

b) Admin (Web) Session

- Login
- Inventory List
- Add Product
- Edit Product
- Product Details & Sizes View
- QR Code View & Download

- Checkout Page
- Staff Management
- Sales Report with Daily, Weekly, Monthly, Yearly Graphs
- Switch Between Bar and Line Chart
- Admin Profile
- Logout

4.4.1 User Session

The User Session module focuses on the features available to staff members using the Android application. These features help users manage inventory tasks easily and efficiently. The system supports both manual interaction and hands-free control using voice commands. This section explains each major feature in detail.

4.4.1.1 Login

The login feature allows staff to securely access the app using their email and password. This is an essential entry point for users to reach the dashboard and manage the inventory system.

Firebase Authentication is used to handle the login process. When a user enters their credentials and taps the Login button, the app checks if the credentials are valid using the signInWithEmailAndPassword method. If the email and password are correct and the account exists in the system, the user is directed to their main dashboard. If the login fails due to incorrect credentials or unregistered accounts, a clear error message is shown to inform the user.

The interface is designed to be clean and user-friendly, with validation checks for empty or invalid inputs such as improperly formatted emails or short passwords. The password field also includes a toggle to show or hide the password for easier input.

This login screen is shared by both regular users and admins, but the system checks whether the user belongs to the users or admins collection in Firestore to route them to the correct dashboard.

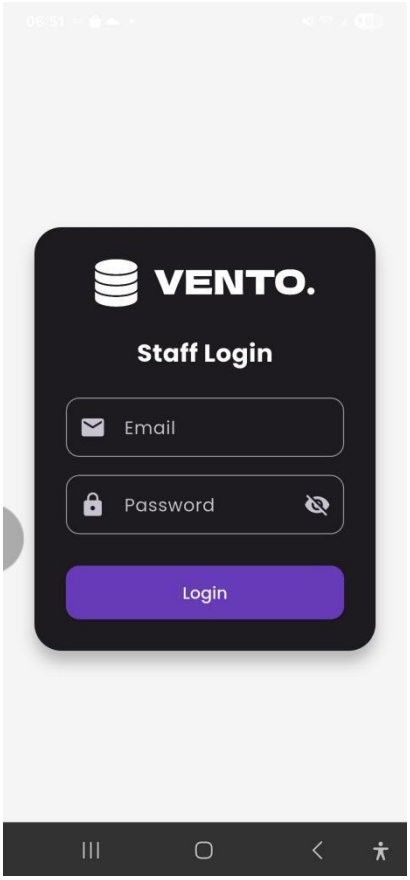

Figure 4.1 Staff (User) Login Interface

```
try {
  final credential = await FirebaseAuth.instance.signInWithEmailAndPassword(
    email: emailController.text.trim(),
    password: passwordController.text.trim(),
  );

  final uid = credential.user!.uid;

  final userDoc = await FirebaseFirestore.instance.collection('users').doc(uid).get();

  if (!userDoc.exists) {
    await FirebaseAuth.instance.signOut();
    _showError("Account not registered. Contact admin.");
    return;
  }

  if (mounted) {
    Navigator.pushReplacementNamed(
      context,
      '/userDashboard',
      arguments: {'userId': uid},
    );
  }
```

Figure 4.2 Code Snippet for Login using signInWithEmailAndPassword

4.4.1.2 View Inventory List

After successfully logging in, users are directed to the Inventory List screen. This page displays all the products that the user has added. Each product is shown with its name, price, stock level, sizes, and category. The list is connected to Firebase Firestore, allowing real-time updates whenever a product is added, edited, or deleted.

At the top of the screen, there is a search bar where users can type keywords to quickly filter products based on name, category, gender, or price. Additionally, users can sort the product list using sorting chips such as Latest, Name, Low Stock, or In Stock. This improves navigation and helps users quickly find the desired product.

The interface uses a dark-themed UI, and each product card includes a colored stock indicator, green for sufficient stock and red for low stock, which providing immediate visual feedback about inventory status.

This module plays a key role in helping users manage and monitor their inventory efficiently and intuitively.

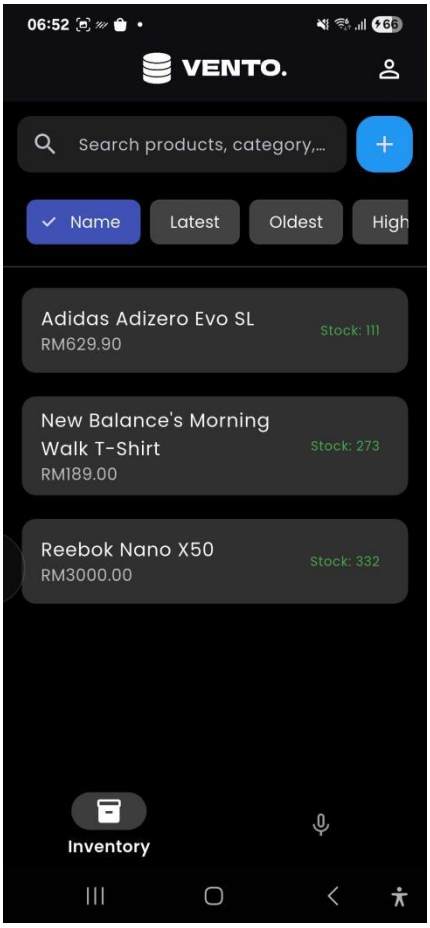

Figure 4.3 Inventory List Interface

```
      onSave: (name, price, sizes, gender, category, qrCode, imageUrl) async {
        final now = FieldValue.serverTimestamp();
        final totalStock = sizes.values.fold(0, (sum, qty) => sum + qty);
        final newDocRef = data == null ? productsCollection.doc() : productsCollection.doc(docId);
        final qrCodeUrl = 'https://vento-59b6e.web.app?id=${newDocRef.id}';
        final product = {
          'name': name,
          'price': price,
          'sizes': sizes,
          'stock': totalStock,
          'gender': gender,
          'category': category,
          'updated_at': now,
          if (data == null) ...{
            'adminId': widget.adminId,
            'created_at': now,
            'qrCode': qrCodeUrl
          }
        };

        if (data == null) {
          await newDocRef.set(product);
        } else {
          await newDocRef.update(product);
        }
      },
    ),
  );
}
```

Figure 4.4 Code Snippet for Fetching and Displaying Inventory from Firestore

4.4.1.3 Product Details Popup

When a product is tapped, a popup window shows full details including product name, price, stock, sizes, gender, and timestamps. This helps users check important info at a glance.

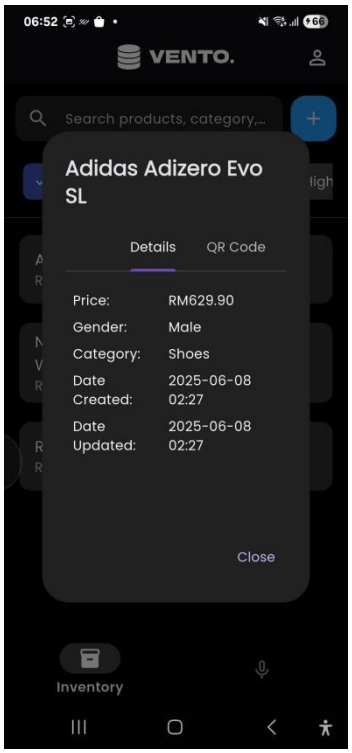

Figure 4.5 Product Details Popup Interface

```
void showProductDetail(Item item) => showDialog(
  context: context,
  builder: (_) => AlertDialog(
    titlePadding: const EdgeInsets.fromLTRB(24, 24, 24, 0),
    title: Text(
      item.name,
      style: const TextStyle(fontSize: 22, fontWeight: FontWeight.bold),
    ),  // Text
    content: Container(
      width: double.maxFinite,
      height: 300,
      child: DefaultTabController(
        length: 3,
        child: Column(
          children: [
            const TabBar(
              isScrollable: true,
              labelColor: Colors.white,
              unselectedLabelColor: Colors.white70,
              indicatorColor: Color(0xFF7349BD),
              indicatorWeight: 3,
              tabs: [
                Tab(text: 'Details'),
                Tab(text: 'QR Code'),
                Tab(text: 'Size/Color'),
              ],
            ),  // TabBar
```

Figure 4.6 Code Snippet for Displaying Product Data in Popup

4.4.1.4 Size/Color Popup

Each product can have multiple sizes or colors. Tapping on a product and navigating to the "Size/Color" tab in the popup displays all available variants along with their stock quantities. This allows users to easily check the availability of each size or color for the selected product.

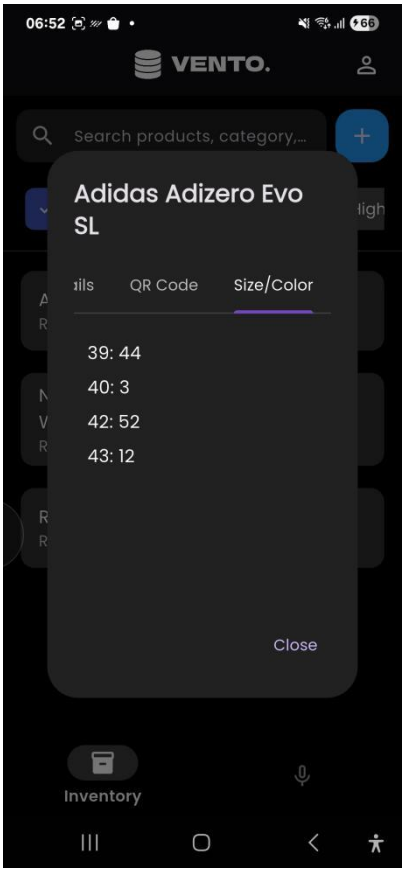

Figure 4.7 Size/Color Editing Interface

```
Padding(
  padding: const EdgeInsets.all(12),
  child: ListView(
    children: item.sizes.entries.map((e) {
      return Padding(
        padding: const EdgeInsets.symmetric(vertical: 4),
        child: Text(
          '${e.key}: ${e.value}',
          style: const TextStyle(fontSize: 16, color: Colors.white),
        ),  // Text
      );  // Padding
    }).toList(),
  ),  // ListView
),  // Padding
```

Figure 4.8 Code Snippet for Handling Size/Color Data as Map in Firestore

4.4.1.5 QR Code Popup

Each product has a unique QR code that links to its web page. Tapping the QR icon opens a popup displaying the code, which can be scanned for quick access or printed for labeling.

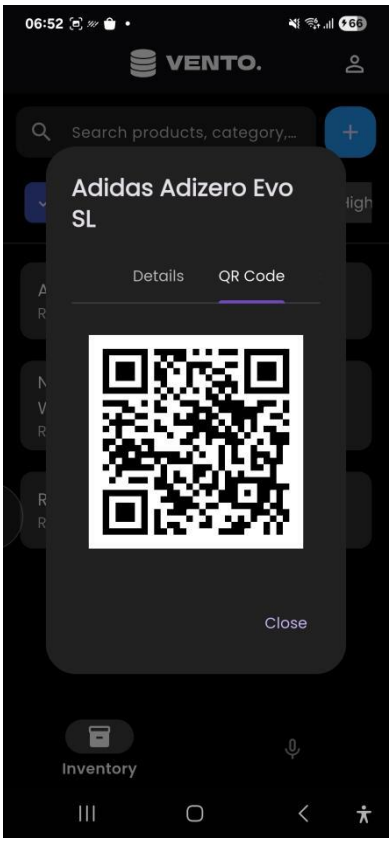

Figure 4.9 QR Code Popup Interface

```
Center(
  child: QrImageView(
    data: item.qrCode,
    version: QrVersions.auto,
    size: 200,
    backgroundColor: Colors.white,
  ),  // QrImageView
),  // Center
```

Figure 4.10 Code Snippet for Generating QR Code URL

4.4.1.6 Add Product Page

Users can add a new product by filling a form with product name, price, category, gender, and size/stock info. When saved, a new document is created in Firestore. The app also auto-generates a QR code for the product.

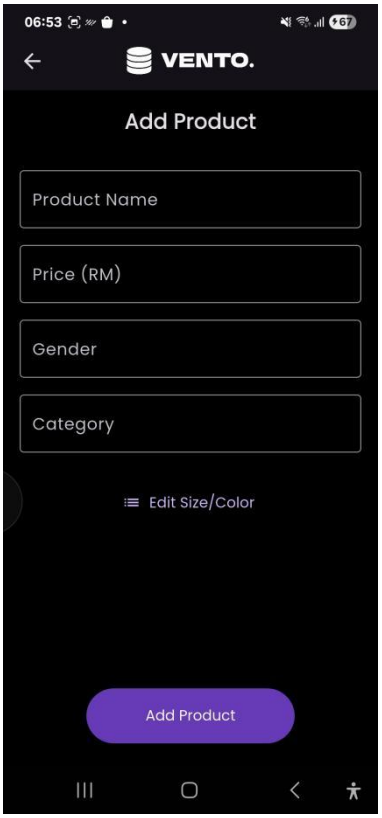

Figure 4.11 Add Product Form Interface

```
try {
  if (widget.existingItem == null) {
    final docRef = FirebaseFirestore.instance.collection('products').doc();
    final productId = docRef.id;
    final qrCodeUrl = 'https://vento-59b6e.web.app?id=$productId';

    await docRef.set({
      ...productData,
      'qrCode': qrCodeUrl,
    });
  } else {
```

Figure 4.12 Code Snippet for set() Operation in Firestore to Create Product

4.4.1.7 Edit Product Page

The edit feature allows users to update existing product information. When the user taps the edit icon, the form opens with existing data filled in. Changes are saved using Firestore's update() function.

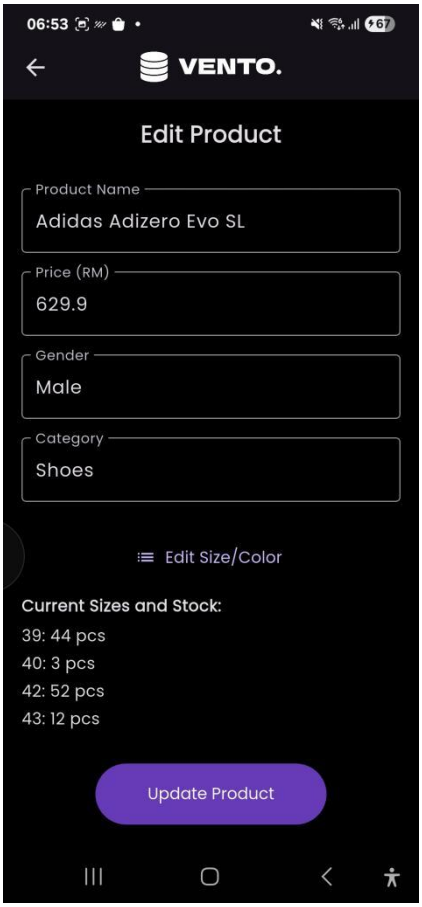

Figure 4.13 Edit Product Page Interface

```
} else {
  final productId = widget.existingItem!.id;
  final qrCodeUrl = widget.existingItem!.qrCode ?? 'https://vento-59b6e.web.app?id=$productId';

  await FirebaseFirestore.instance.collection('products').doc(productId).update({
    ...productData,
    'qrCode': qrCodeUrl,
  });
}
```

Figure 4.14 Code Snippet for Firestore update() Operation

4.4.1.8 Edit Size/Color Popup

Each product can have multiple sizes or color variants. Users can tap the "Edit Size/Color" button to open a popup dialog that allows them to manage stock for each variant. Inside the popup, users can add new size entries, update existing quantities, or remove unwanted entries. This helps ensure that size and stock data remain accurate and easy to maintain. Once confirmed, the size/stock values are updated in the form and later stored in Firestore.

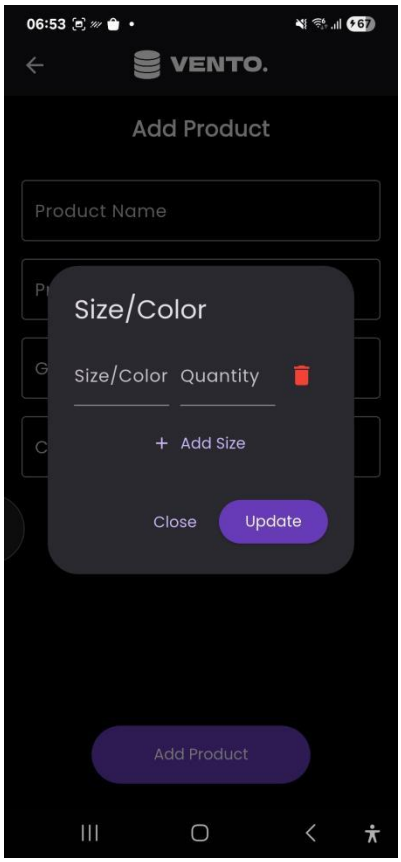

Figure 4.15 Edit Size/Color Popup Interface

```
TextButton.icon(
  onPressed: _showSizeStockDialog,
  icon: const Icon(Icons.list),
  label: const Text('Edit Size/Color'),
),  // TextButton.icon
```

Figure 4.16 Code Snippet for Size/Color Dialog Logic in Product Form

4.4.1.9 Delete Confirmation Dialog

Users can delete a product by tapping the delete icon. Before deletion, a confirmation dialog appears to prevent accidental removal. If confirmed, the product is deleted using Firestore's delete() function.

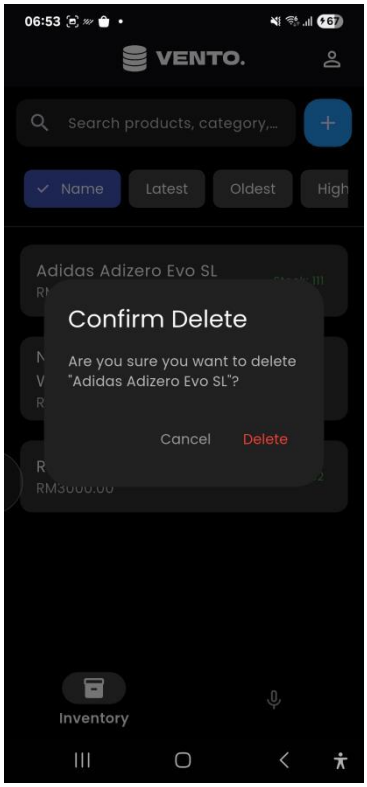

Figure 4.17 Delete Confirmation Dialog Interface

```
Future<void> confirmDelete(Item item) async {
  final confirmed = await showDialog<bool>(
    context: context,
    builder: (_) => AlertDialog(
      backgroundColor: Colors.grey[900],
      title: const Text('Confirm Delete', style: TextStyle(color: Colors.white)),
      content: Text(
        'Are you sure you want to delete "${item.name}"?',
        style: const TextStyle(color: Colors.white70),
      ),  // Text
      actions: [
        TextButton(
          child: const Text('Cancel', style: TextStyle(color: Colors.grey)),
          onPressed: () => Navigator.pop(context, false),
        ),  // TextButton
        TextButton(
          child: const Text('Delete', style: TextStyle(color: Colors.red)),
          onPressed: () => Navigator.pop(context, true),
        ),  // TextButton
      ],
    ),  // AlertDialog
  );
  if (confirmed == true) {
    await FirebaseFirestore.instance.collection('products').doc(item.id).delete();
    setState(() => items.removeWhere((e) => e.id == item.id));
  }
}
```

Figure 4.18 Code Snippet for delete() Operation

4.4.1.10 Staff Profile (View Only)

Each staff can access their profile page to view their personal information. This page displays the staff's name and email address, which are fetched from Firebase Authentication and Firestore. The purpose of this page is to provide basic account information in a clean and readable format.

Staff cannot make any changes to their profile in the current version. The fields are non-editable, and there is no button or form for updating their details. This is done to ensure consistency and maintain the integrity of user data.

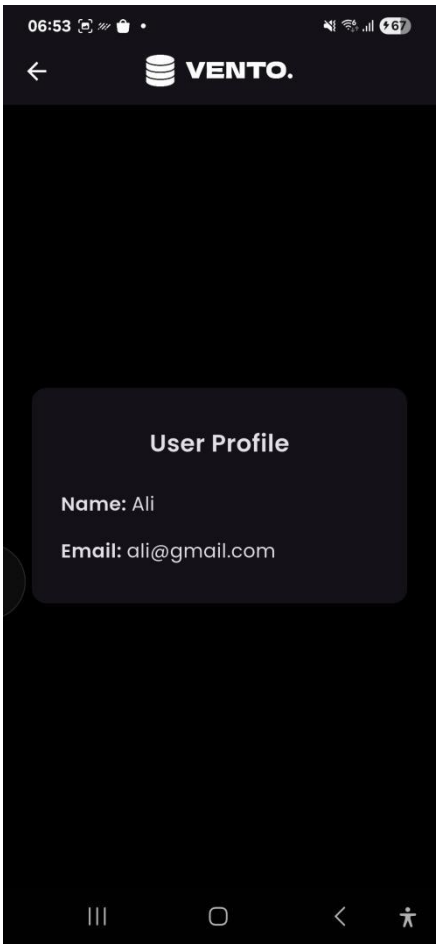

Figure 4.19 Staff Profile Page Interface (View Only)

```dart
Future<void> _fetchUserData() async {
  if (currentUser == null) {
    setState(() => isLoading = false);
    return;
  }

  try {
    DocumentSnapshot userDoc = await FirebaseFirestore.instance
        .collection('users')
        .doc(currentUser!.uid)
        .get();

    if (userDoc.exists) {
      final userData = userDoc.data() as Map<String, dynamic>?;

      name = userData?['name'] ?? currentUser!.displayName ?? 'No name found';
      email = userData?['email'] ?? currentUser!.email ?? 'No email found';
    } else {
      name = currentUser!.displayName ?? 'No name found';
      email = currentUser!.email ?? 'No email found';
    }
  } catch (e) {
    print('Error fetching user data: $e');
    setState(() {
      name = currentUser!.displayName ?? 'Error loading user data';
      email = currentUser!.email ?? '';
    });
  }

  setState(() => isLoading = false);
}
```

Figure 4.20 Code Snippet for Fetching and Displaying Staff Profile from Firebase

4.4.1.11 Logout

Users can securely log out of the application by tapping the logout icon in the top-right corner. This triggers a confirmation dialog to avoid accidental sign-outs. If confirmed, the app uses Firebase's signOut() function to end the session and navigates the user back to the login screen.

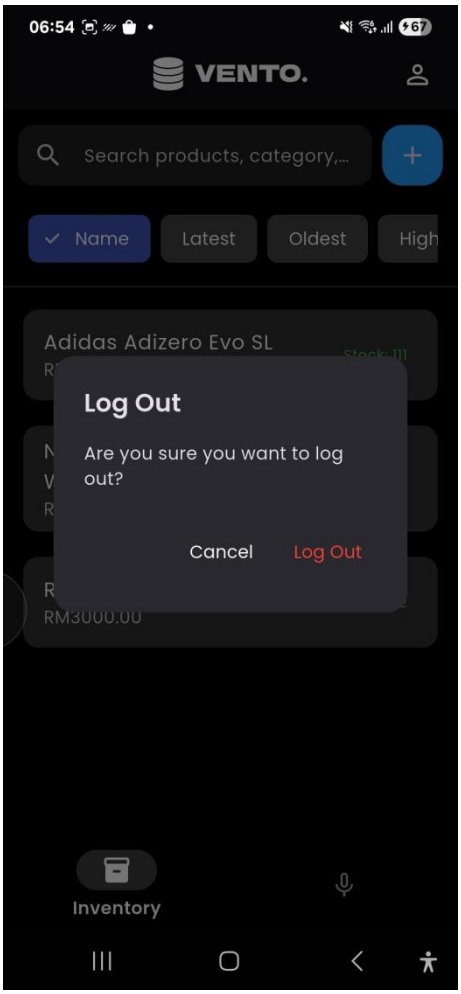

Figure 4.21 Logout Button Interface

```
if (shouldLogout == true) {
  await FirebaseAuth.instance.signOut();
  if (mounted) {
    Navigator.pushReplacementNamed(context, '/userAuth');
  }
}
}
```

Figure 4.22 Code Snippet for Firebase signOut() Function

4.4.1.12 Voice Assistant (Add/Edit/View/Delete Products Hands-Free)

The app includes a smart voice assistant that helps users perform inventory tasks using natural speech. It uses built-in Android's native Speech-to-Text to recognize commands and Flutter TTS to respond.

Users can say commands like:

- "Add new product named Adidas Bag"
- "Edit price of Adidas Bag to 120"
- "Delete product Nike Shirt"
- "Show stock of Large size in Adidas Bag"

The voice assistant handles follow-up questions when needed, ensuring full and correct product data is collected before saving to Firestore. It also includes a microphone button that allows users to turn the voice response (TTS) on or off, giving them the option to hear the assistant speak or remain silent.

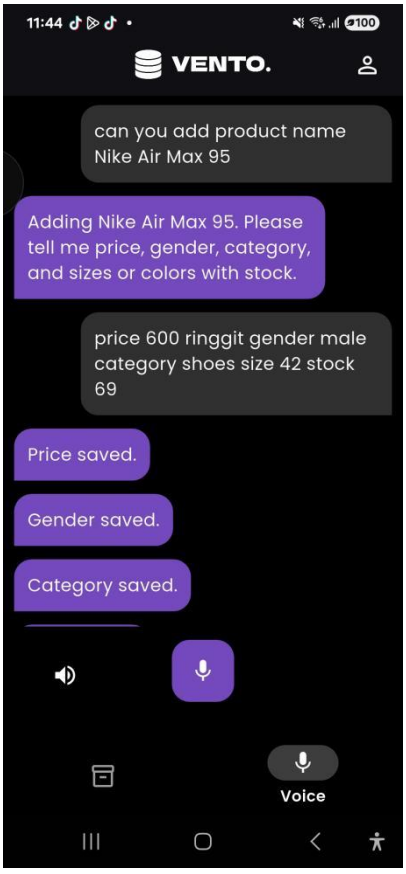

Figure 4.23 Voice Assistant for Adding Product Interface

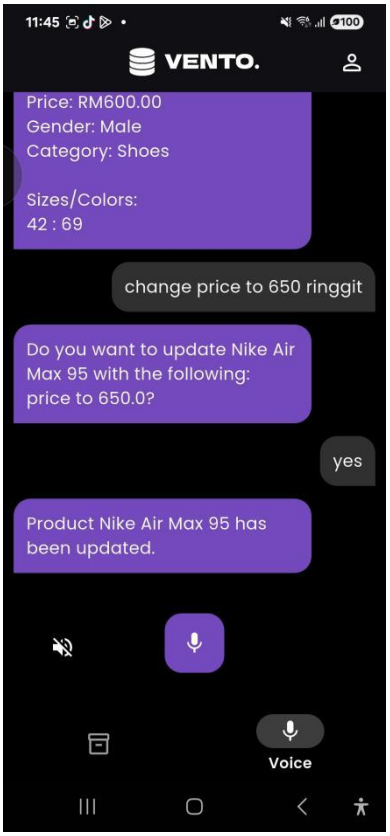

Figure 4.24 Voice Assistant for Updating Product Interface

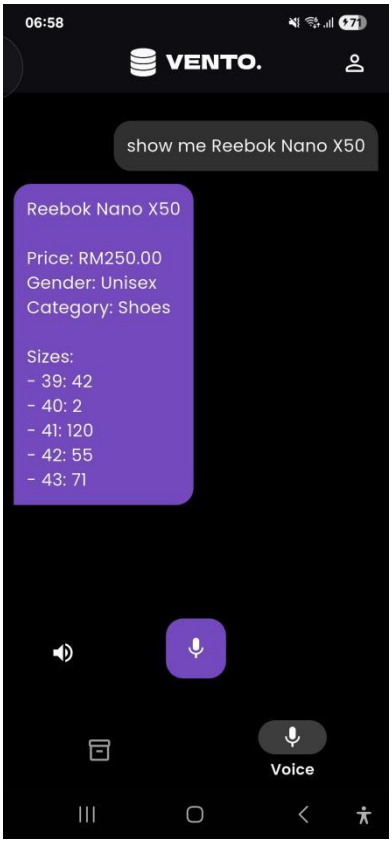

Figure 4.25 Voice Assistant for Viewing Product Interface

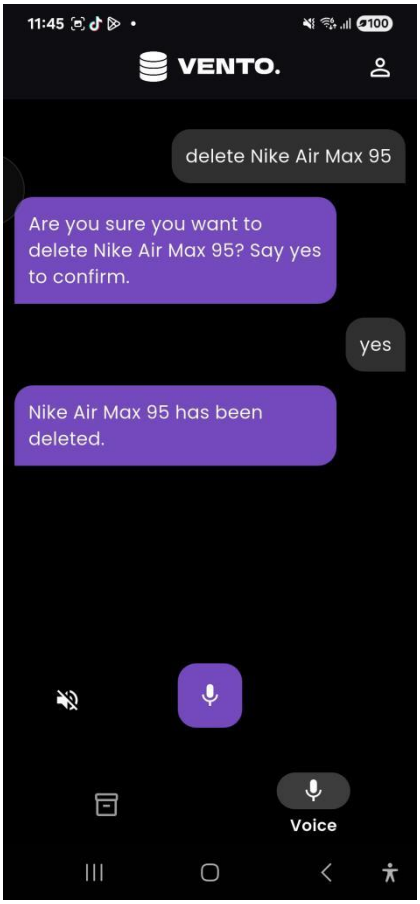

Figure 4.26 Voice Assistant for Deleting Product Interface

```dart
Future<void> _detectIntent(String text) async {
  final lowerText = text.toLowerCase();

  final offCommands = ['disable tts', 'turn off tts'];
  final onCommands = ['enable tts', 'turn on tts'];

  if (offCommands.any((phrase) => lowerText.contains(phrase))) {
    ttsEnabled.value = false;
    await _speak("Text to speech disabled.");
    _reset();
    return;
  } else if (onCommands.any((phrase) => lowerText.contains(phrase))) {
    ttsEnabled.value = true;
    await _speak("Text to speech enabled.");
    _reset();
    return;
  }
```

Figure 4.27 Code snippet for turning on/off text-to-speech via voice command.

```
if (_containsKeywords(text, ['add'])) {
  _currentProductName = _extractName(text);
  if (_currentProductName == null) {
    await _speak("Please say the product name again.");
  } else {
    _state = AssistantState.awaitingAddDetails;
    _pendingData = {};
    await _speak("Adding $_currentProductName. Please tell me price, gender, category, and sizes or colors with stock.");
  }
```

Figure 4.28 Code snippet for add function in voice command.

```
} else if (_containsKeywords(text, ['delete']) || _containsKeywords(text, ['remove'])) {
  _currentProductName = _extractName(text);
  if (_currentProductName == null) {
    await _speak("Please say the product name again to delete.");
  } else {
    _state = AssistantState.confirmingDelete;
    await _speak("Are you sure you want to delete $_currentProductName? Say yes to confirm.");
  }
```

Figure 4.29 Code snippet for delete function in voice command.

```
} else if (_containsKeywords(text, ['edit']) || _containsKeywords(text, ['update'])) {
  _currentProductName = _extractName(text);
  if (_currentProductName == null) {
    await _speak("Please say the product name again.");
  } else {
    final doc = await _findProductByName(_currentProductName!);
    if (doc == null) {
      await _speak("Product $_currentProductName not found.");
      _reset();
```

Figure 4.30 Code snippet for edit function in voice command.

```
  } else if (RegExp(r'\b(show|view|price|stock|sizes|colors|color|category|gender|details|available|tell me about)\b',
      caseSensitive: false).hasMatch(text)) {
    await handleView(text, _speak);
    _reset();
  } else {
    await _speak("Please say a command like add, edit, view, delete, or ask stock.");
  }
}
```

Figure 4.31 Code snippet for view function in voice command.

4.4.2 Admin Session

The Admin Session section explains the core features available to administrators through the web dashboard. These features are designed to help admins manage the inventory system, staff accounts, product data, sales reports, and profile settings effectively. All actions are performed through a modern Flutter Web interface connected to Firebase Firestore, with real-time data handling and clean user feedback.

4.4.2.1 Login

Admins access the system through a dedicated login page built for the web platform. This page is separate from the user login used in the Android app. When an admin logs in, their credentials are verified using Firebase Authentication. After successful authentication, the system checks whether the email exists in the admins collection in Firestore. If a match is found, the user is granted administrative access and redirected to the Admin Dashboard.

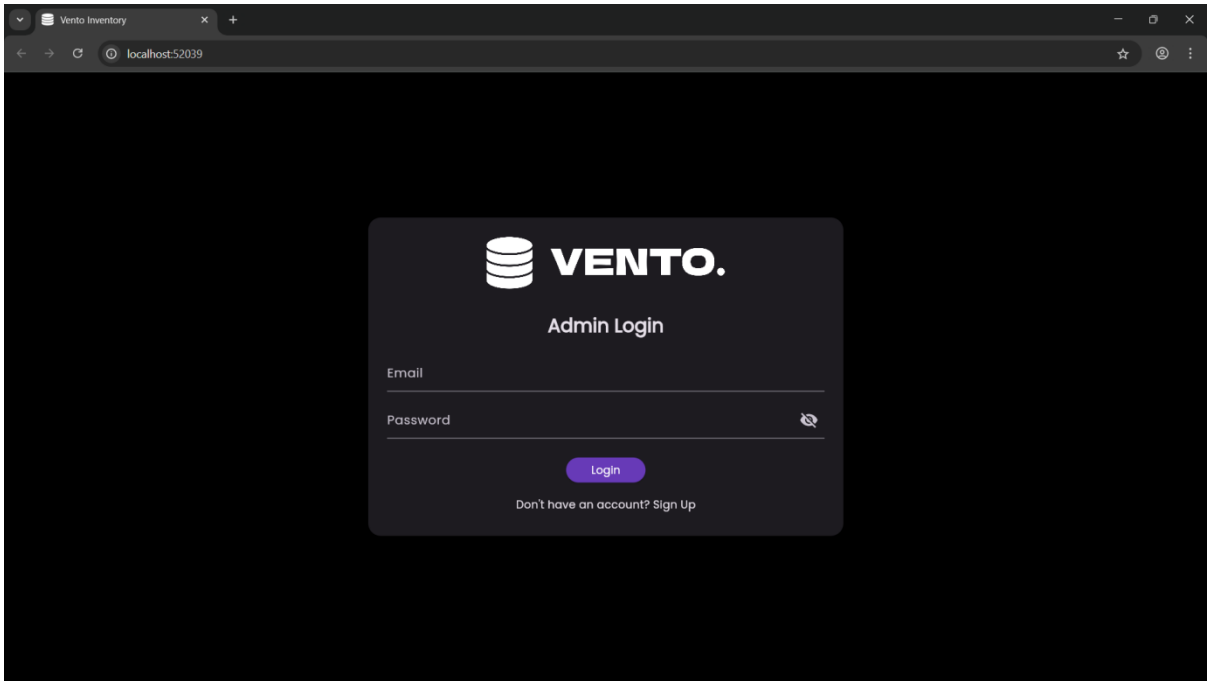

Figure 4.32 Admin Login Interface

```dart
final credential = await FirebaseAuth.instance.signInWithEmailAndPassword(
  email: emailController.text.trim(),
  password: passwordController.text.trim(),
);

debugPrint("Admin logged in: ${credential.user!.email}");

if (mounted) {
  Navigator.pushReplacementNamed(context, '/adminDashboard');
}
```

Figure 4.33 Code Snippet for Admin Role-Based Navigation

4.4.2.2 Signup Admin

Admins are allowed to register their own accounts through the system's sign-up page. However, to ensure account authenticity and system security, newly registered admins must verify their email address before they are able to log in. This helps prevent unauthorized access to sensitive features such as inventory data, staff management, and sales reports.

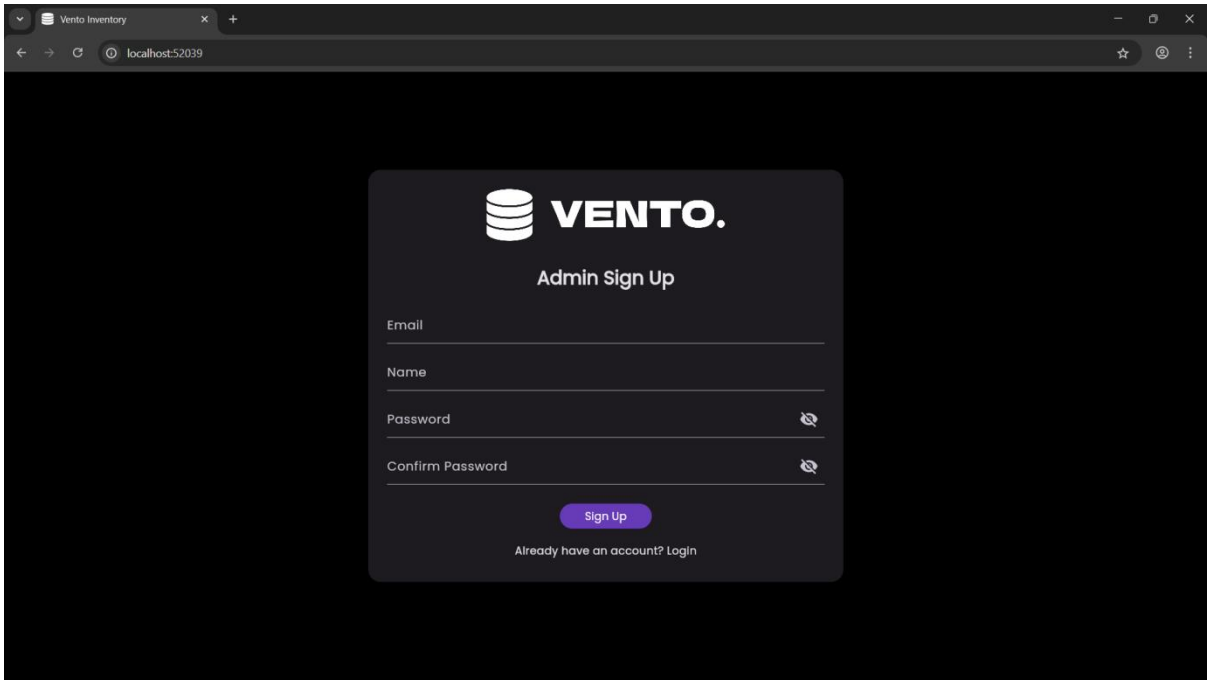

Figure 4.34 Admin Signup Form

```dart
final credential = await FirebaseAuth.instance.createUserWithEmailAndPassword(
  email: emailController.text.trim(),
  password: passwordController.text.trim(),
);

final uid = credential.user!.uid;

await FirebaseFirestore.instance.collection('admins').doc(uid).set({
  'uid': uid,
  'email': credential.user!.email,
  'name': nameController.text.trim(),
  'createdAt': Timestamp.now(),
});
```

Figure 4.35 Code Snippet for Admin Account Creation

4.4.2.3 Inventory List

The inventory list shows all products added by the admin. Each product displays its name, price, stock, size/color variants, category, gender, and timestamps. Admins can search, sort, and filter products for easier access.

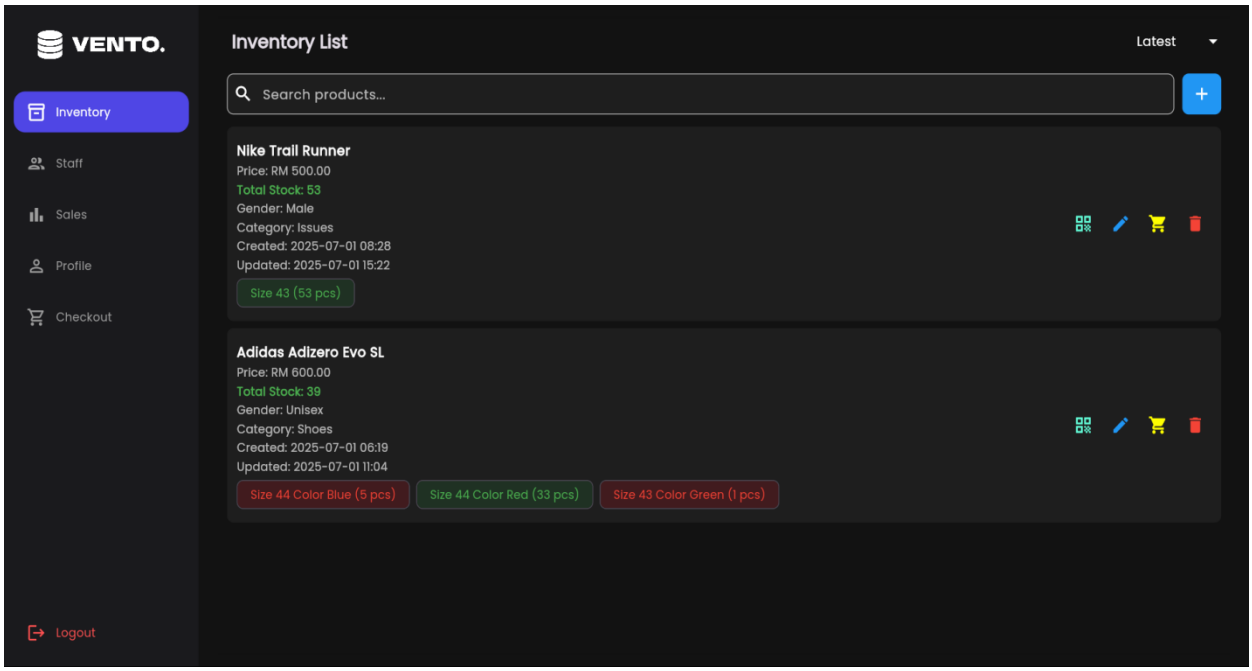

Figure 4.36 Inventory List Interface

```
        child: StreamBuilder<QuerySnapshot>(
          stream: productsCollection.where('adminId', isEqualTo: widget.adminId).snapshots(),
          builder: (context, snapshot) {
            if (snapshot.hasError) return const Center(child: Text('Error loading products', style: TextStyle(color: Colors.white)));
            if (!snapshot.hasData) return const Center(child: CircularProgressIndicator());
            final docs = applySearchAndSort(snapshot.data!.docs);
            if (docs.isEmpty) return const Center(child: Text('No products found', style: TextStyle(color: Colors.white)));
            return ListView.builder(itemCount: docs.length, itemBuilder: (_, i) => buildProductTile(docs[i]));
          },
        ),  // StreamBuilder
      ),  // Expanded
    ],
  ),  // Column
);  // Scaffold
```

Figure 4.37 Code Snippet for Firestore Read Operation with Filters

4.4.2.4 Add Product

Admins can add new products to the inventory using a form that includes fields for name, price, category, gender, and size/color stock. A unique QR code is automatically generated upon saving.

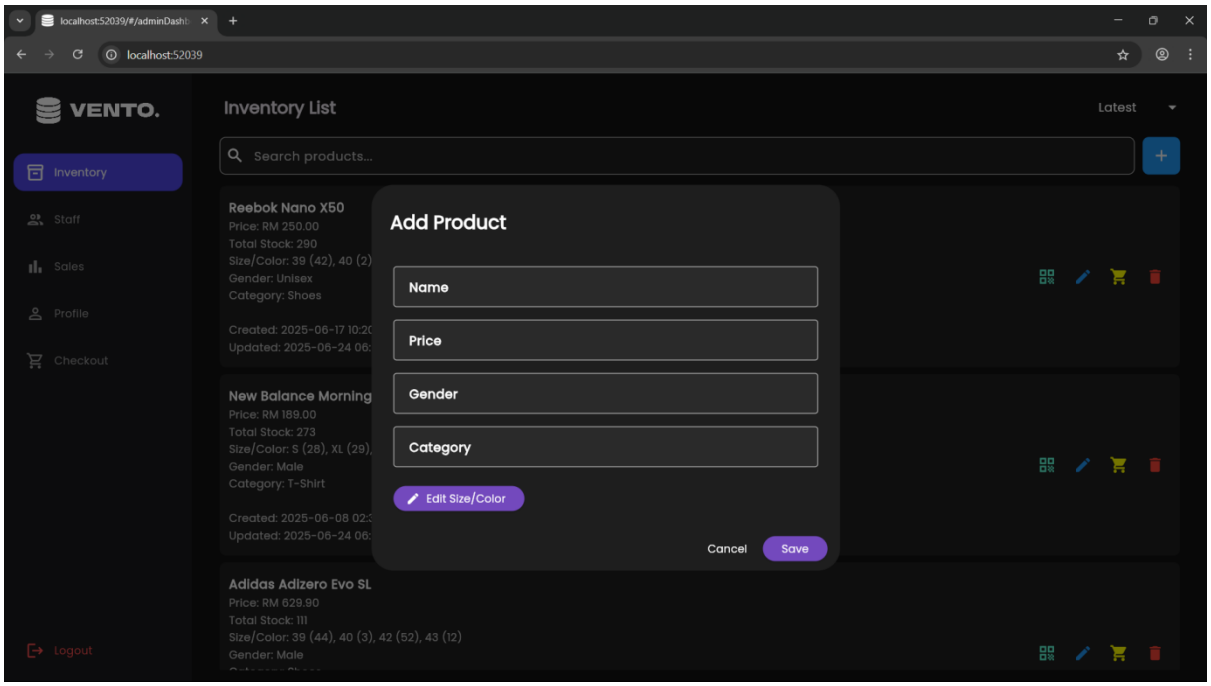

Figure 4.38 Add Product Page

```dart
final now = FieldValue.serverTimestamp();
final totalStock = sizes.values.fold(0, (sum, qty) => sum + qty);
final newDocRef = data == null ? productsCollection.doc() : productsCollection.doc(docId);
final qrCodeUrl = 'https://vento-59b6e.web.app?id=${newDocRef.id}';
final product = {
  'name': name,
  'price': price,
  'sizes': sizes,
  'stock': totalStock,
  'gender': gender,
  'category': category,
  'updated_at': now,
  if (data == null) ...{
    'adminId': widget.adminId,
    'created_at': now,
    'qrCode': qrCodeUrl
  }
};

if (data == null) {
  await newDocRef.set(product);
```

Figure 4.39 Code Snippet Using set() to Add Product to Firestore

4.4.2.5 Edit Product

Admins can update product details like price, category, or stock using an edit form. The update does not affect other fields in the document, thanks to Firestore's update() method.

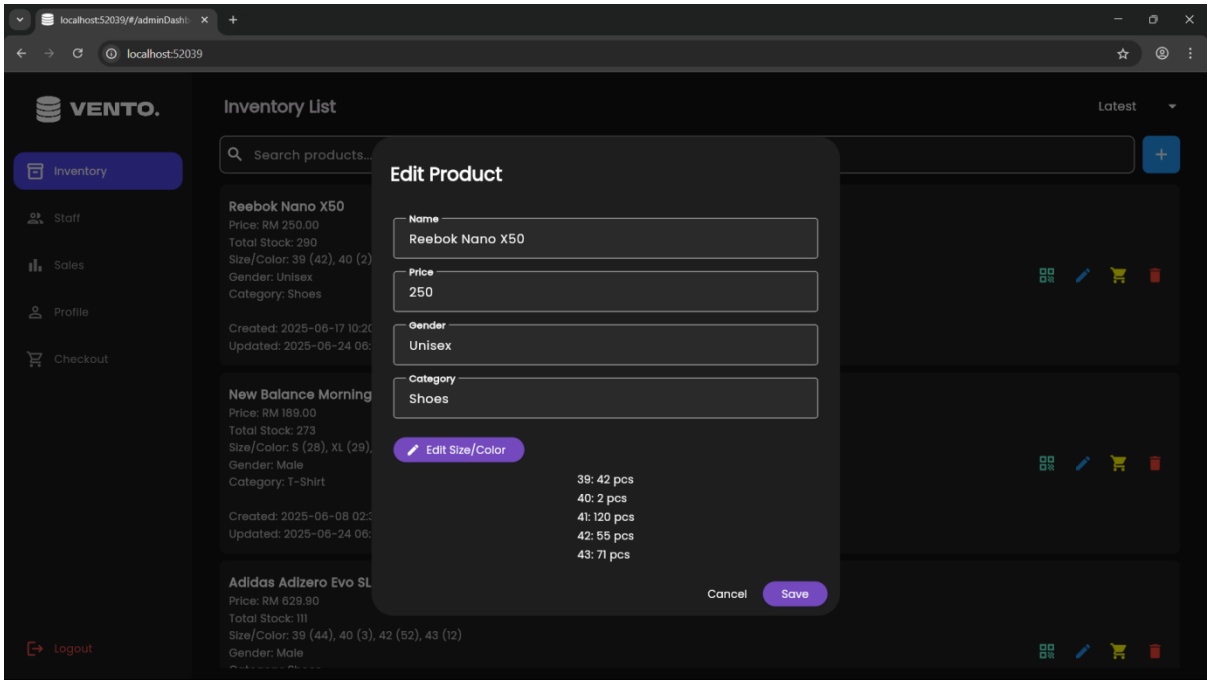

Figure 4.40 Edit Product Popup

```
if (data == null) {
  await newDocRef.set(product);
} else {
  await newDocRef.update(product);
}
```

Figure 4.41 Code Snippet for Firestore Update Operation

4.4.2.6 Add/Edit Size

Within the product form, admins can open a popup to manage product sizes and color variants. Each entry includes a label (e.g. “L” or “Red”) and its stock quantity.

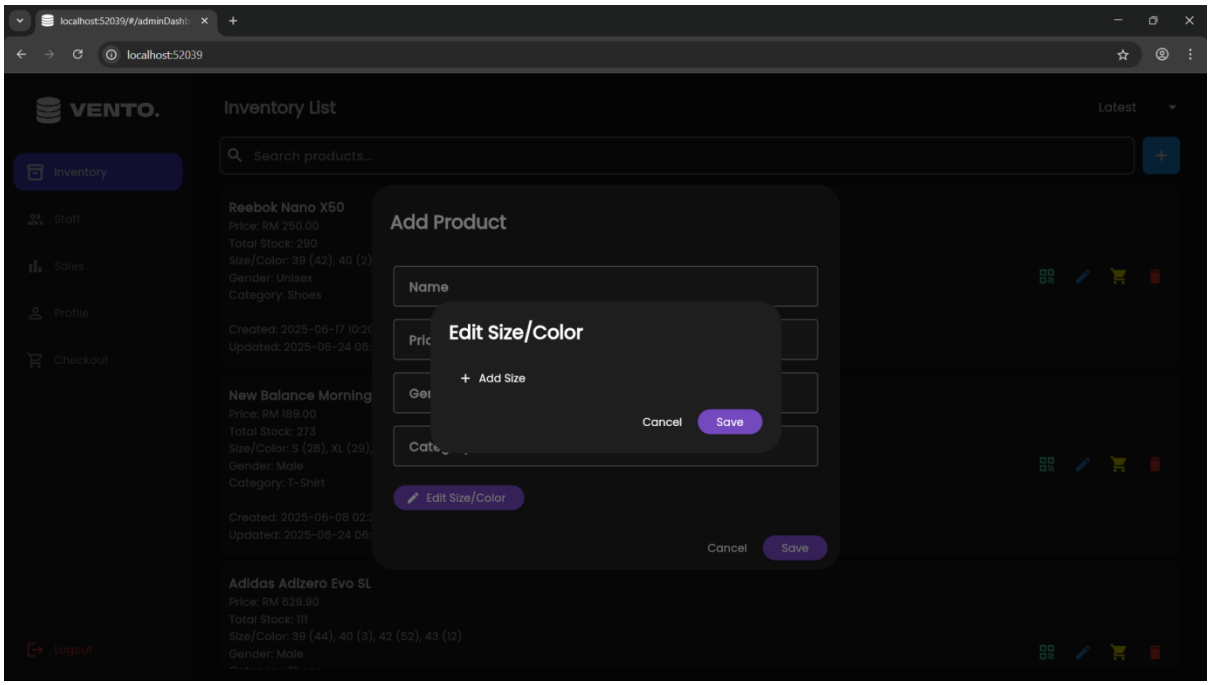

Figure 4.42 Size Editor Popup

```dart
void _saveSizes() {
  final Map<String, int> sizes = {};
  for (var entry in _sizeControllers) {
    final size = entry.key.trim();
    final qty = int.tryParse(entry.value.text.trim()) ?? 0;
    if (size.isNotEmpty && qty >= 0) {
      sizes[size] = qty;
    }
  }
  widget.onSave(sizes);
  Navigator.pop(context);
}
```

Figure 4.43 Code for Handling Size Map in Firestore

4.4.2.7 Delete Product

Products can be permanently removed from the system using the delete button. A confirmation dialog appears to avoid accidental deletions.

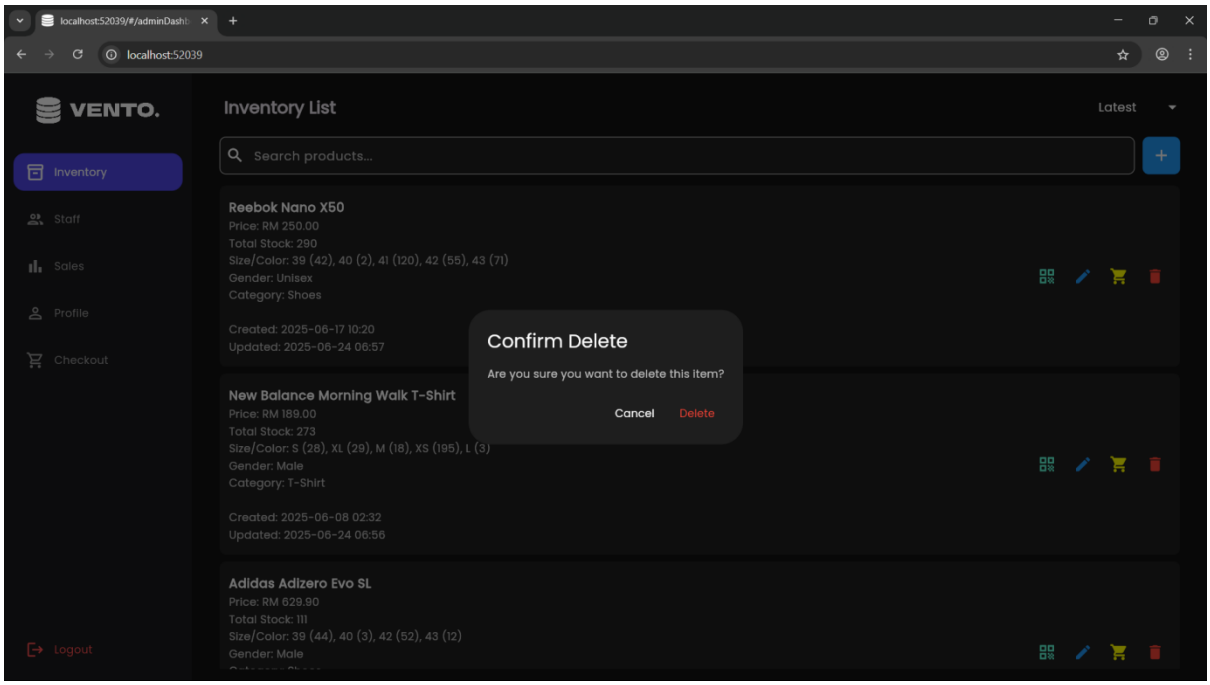

Figure 4.44 Delete Confirmation Dialog

```dart
Future<void> deleteItem(String id) async => await productsCollection.doc(id).delete();

Future<void> confirmDelete(String id) async {
  final shouldDelete = await showDialog<bool>(
    context: context,
    barrierDismissible: false,
    builder: (ctx) => AlertDialog(
      backgroundColor: const Color(0xFF1E1E1E),
      title: const Text('Confirm Delete', style: TextStyle(color: Colors.white)),
      content: const Text('Are you sure you want to delete this item?', style: TextStyle(color: Colors.white70)),
      actions: [
        TextButton(onPressed: () => Navigator.of(ctx).pop(false), child: const Text('Cancel', style: TextStyle(color: Colors.white))),
        TextButton(onPressed: () => Navigator.of(ctx).pop(true), child: const Text('Delete', style: TextStyle(color: Colors.red))),
      ],
    ),  // AlertDialog
  );
  if (shouldDelete == true) {
    await deleteItem(id);
    if (mounted) ScaffoldMessenger.of(context).showSnackBar(const SnackBar(content: Text('Item deleted')));
  }
}
```

Figure 4.45 Code Snippet Using Firestore delete() Function

4.4.2.8 Checkout Popup

Admins can simulate a checkout to temporarily deduct stock for a specific size. A popup allows selection of size and quantity to preview inventory changes. The stock is updated in Firestore immediately to reflect this deduction. However, this action does not log a sales record yet. The final sales report entry is only added upon checkout confirmation, which is handled later on the dedicated checkout page.

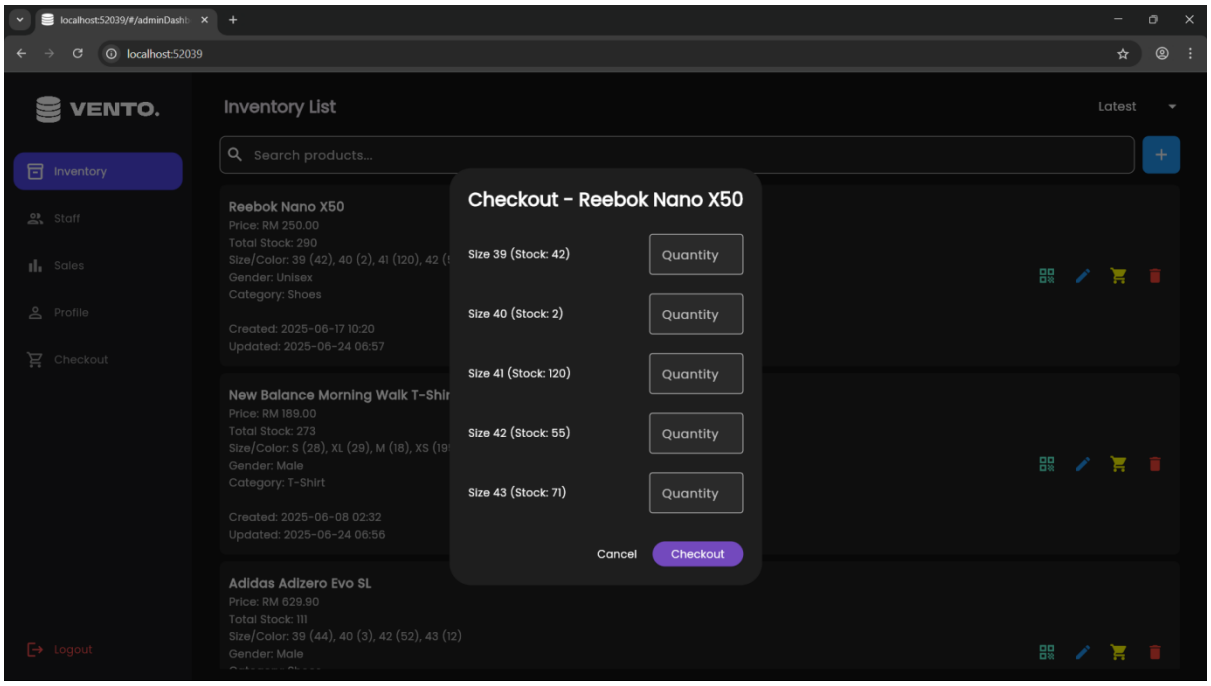

Figure 4.46 Checkout Dialog with Size Selection

```dart
void addToCart({
  required String id,
  required String name,
  required double price,
  required String adminId,
  required String size,
  required int quantity,
  required int availableStock,
}) {
  if (quantity <= 0 || quantity > availableStock) return;

  final index = _cartItems.indexWhere((item) => item['id'] == id);
  if (index >= 0) {
    final item = _cartItems[index];
    Map<String, int> sizes = Map<String, int>.from(item['sizes'] ?? {});
    int existingQty = sizes[size] ?? 0;
    int newQty = existingQty + quantity;

    if (newQty > availableStock) newQty = availableStock;

    sizes[size] = newQty;
    item['sizes'] = sizes;
  } else {
    _cartItems.add({
      'id': id,
      'name': name,
      'price': price,
      'adminId': adminId,
      'sizes': {size: quantity},
    });
  }
}
```

Figure 4.47 Code Snippet for Updating Stock and Logging Sale

4.4.2.9 QR Code View & Download

Each product has a unique QR code that links to its online product detail page. Admin can preview and download the QR code for printing or sharing.

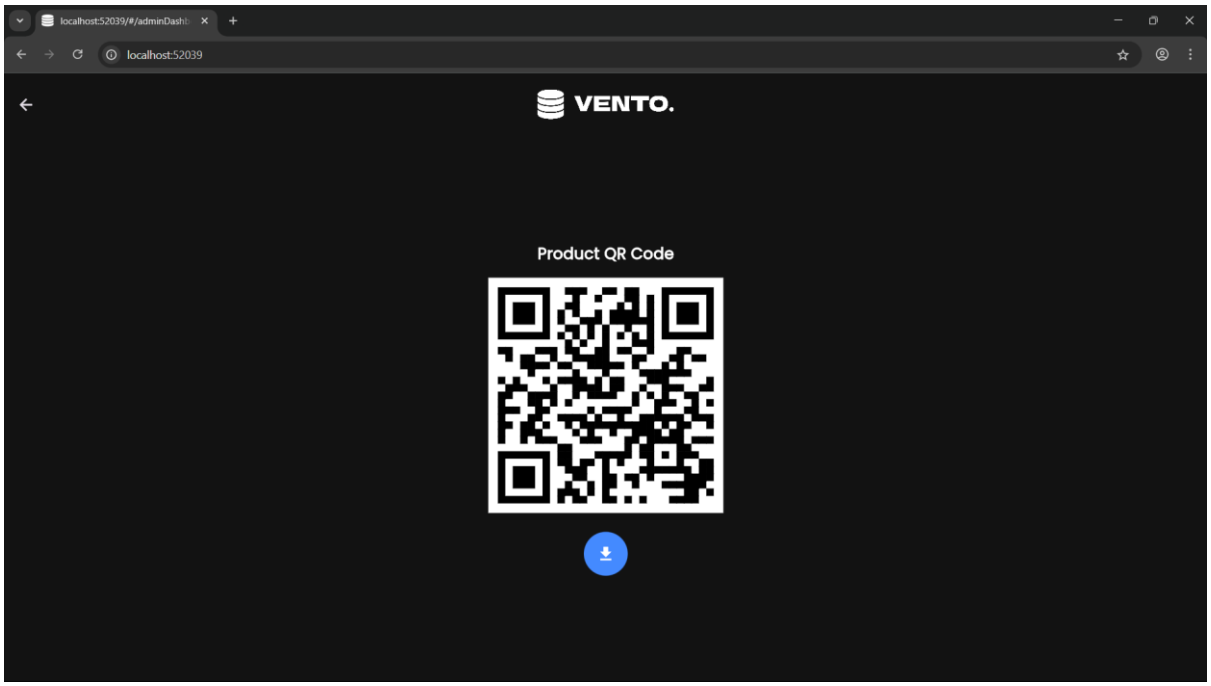

Figure 4.48 QR Code Page

```
RepaintBoundary(
  key: globalKey,
  child: QrImageView(
    data: widget.qrCode,
    version: QrVersions.auto,
    size: 300,
    backgroundColor: Colors.white,
  ),  // QrImageView
),  // RepaintBoundary
```

Figure 4.49 Code Snippet for QR Code Generation

```dart
Future<void> _saveQrToFile() async {
  try {
    final boundary = globalKey.currentContext?.findRenderObject() as RenderRepaintBoundary?;
    if (boundary == null) {
      _showMessage('Could not capture QR code.');
      return;
    }

    final image = await boundary.toImage(pixelRatio: 3.0);
    final byteData = await image.toByteData(format: ui.ImageByteFormat.png);
    final pngBytes = byteData!.buffer.asUint8List();

    final fileName = 'qr_code_${_sanitizeFileName(widget.productName)}.png';
    final result = await downloadQrCode(pngBytes, fileName);

    if (result != null) {
      _showMessage(result);
    } else {
      _showMessage('Download started');
    }
  } catch (e) {
    _showMessage('Error saving QR code: $e');
  }
}
```

Figure 4.50 Another Code Snippet for QR Code Generation

4.4.2.10 Staff Management

The admin can manage all registered staff members through a dedicated interface in the web dashboard. This module supports full CRUD operations, including adding new staff, editing staff details such as name and email, resetting passwords, and deleting staff records. A search bar is provided to help the admin quickly filter staff by name or email.

When a new staff account is added, it is initially marked as "pending verification" in Firestore. The staff member must verify their email and log in using the Android app. Once verified, their status is automatically updated to "verified", and they gain access to inventory features in the app. This ensures that only authenticated staff members can interact with the system.

All staff information is stored in Firestore and linked to the admin's unique adminId. Input validation ensures that required fields are not left empty, and a confirmation prompt is displayed before deleting any staff member to prevent accidental removals.

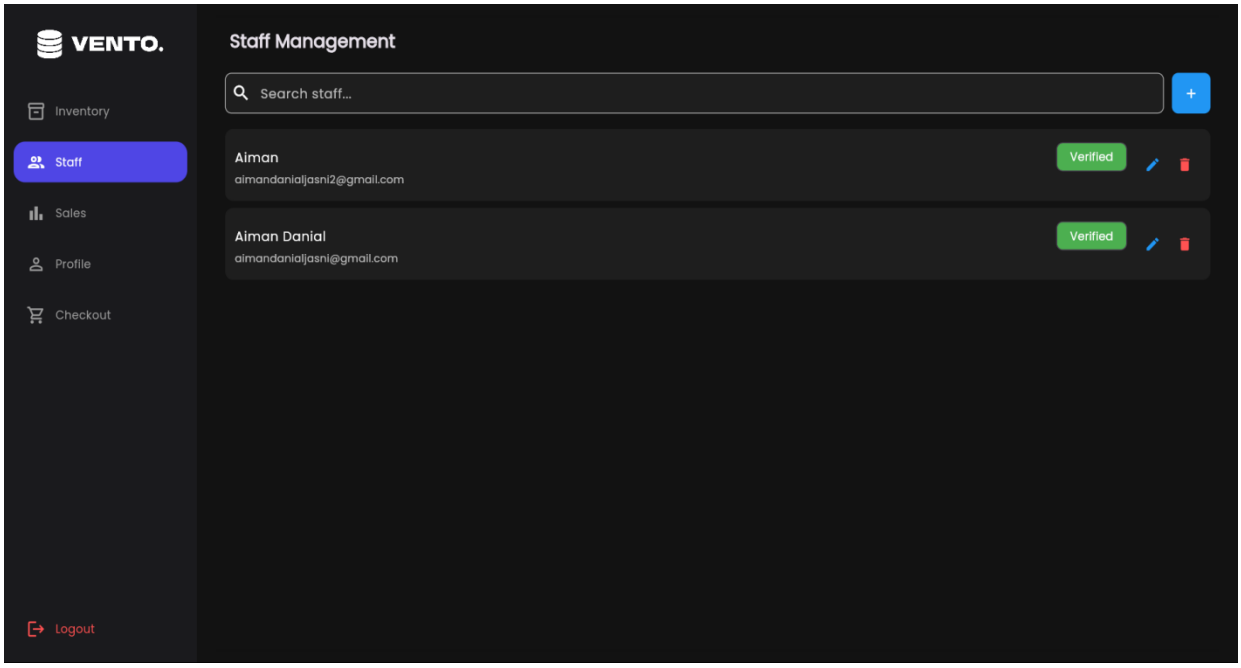

Figure 4.51 Staff List Interface

```dart
Future<void> _addStaff(String name, String email, String? password) async {
  if ([name, email, password].any((e) => e == null || e.trim().isEmpty)) {
    _showSnack('All fields are required', isError: true);
    return;
  }
  try {
    final existing = await staffCollection
        .where('adminId', isEqualTo: widget.adminId)
        .where('email', isEqualTo: email.trim())
        .get();
    if (existing.docs.isNotEmpty) {
      _showSnack('Email already exists', isWarning: true);
      return;
    }
    await staffCollection.add({
      'name': name.trim(),
      'email': email.trim(),
      'password': password!.trim(),
      'adminId': widget.adminId,
    });
    _showSnack('Staff added successfully');
  } catch (e) {
    _showSnack('Error: $e', isError: true);
  }
}
```

Figure 4.52 Code Snippets for Firestore Staff Management (Adding Staff)

```dart
Widget _buildStatusChip(String status) {
  return Chip(
    label: Text(
      status == 'verified' ? 'Verified' : 'Pending',
      style: const TextStyle(color: Colors.white),
    ),  // Text
    backgroundColor: status == 'verified' ? Colors.green : Colors.orange,
  );  // Chip
}
```

Figure 4.53 Code snippet for displaying staff verification status in the form of a colored chip. A green chip represents a verified staff member, while an orange chip indicates a pending (unverified) account.

```dart
Future<void> _updateStaffInfo(
    String staffId, String updatedName, String updatedEmail) async {
  if (updatedName.trim().isEmpty || updatedEmail.trim().isEmpty) {
    _showSnack('Name and Email cannot be empty', isError: true);
    return;
  }
  try {
    await staffCollection.doc(staffId).update({
      'name': updatedName.trim(),
      'email': updatedEmail.trim(),
    });
    _showSnack('Staff updated successfully');
  } catch (e) {
    _showSnack('Error: $e', isError: true);
  }
}
```

Figure 4.54 Code Snippets for Firestore Staff Management (Updating Staff)

```dart
Future<void> _deleteStaff(String staffId) async {
  try {
    await staffCollection.doc(staffId).delete();
    _showSnack('Staff deleted successfully');
  } catch (e) {
    _showSnack('Error deleting staff: $e', isError: true);
  }
}
```

Figure 4.55 Code Snippets for Firestore Staff Management (Deleting Staff)

4.4.2.11 Sales Report with Charts

The Sales Report page shows a visual summary of product sales over four different time periods: Daily, Weekly, Monthly, and Yearly. Admins can choose to view the data as either a Bar Chart or a Line Chart using a simple dropdown menu. They can also apply filters, such as "This Month", "Last 4 Weeks", or "This Year", depending on the selected time period. All sales data is retrieved from the Firestore 'sales' collection, and the system automatically groups and calculates the total sales for each selected time range.

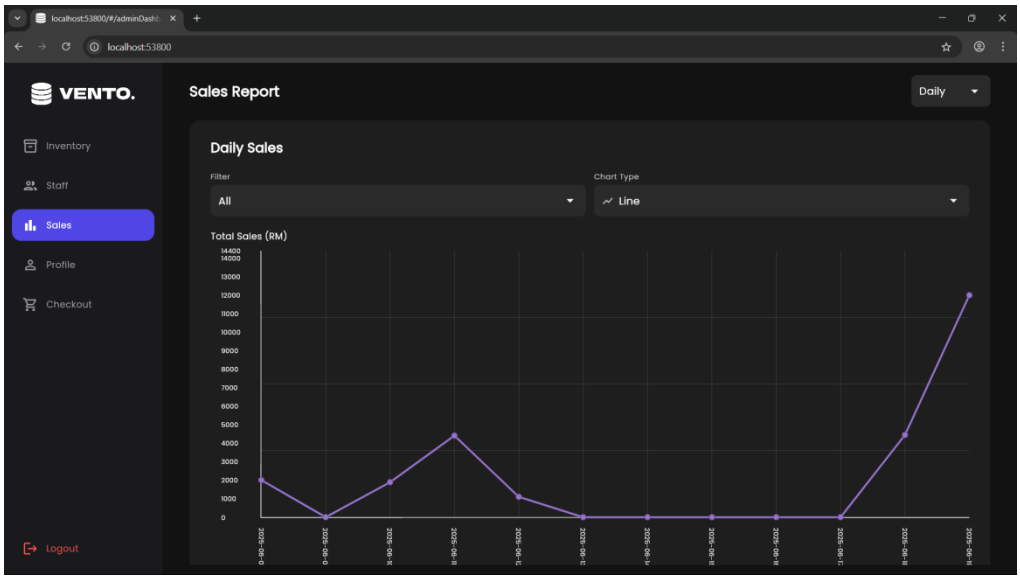

Figure 4.56 Sales Report with Time Filter and Chart Switcher

```
Future<void> loadSalesData() async {
  final snapshot = await salesCollection.get();

  Map<String, double> tempDaily = {}, tempWeekly = {}, tempMonthly = {}, tempYearly = {};
  DateTime? minDate, maxDate;

  for (var doc in snapshot.docs) {
    final data = doc.data();
    final date = (data['soldAt'] as Timestamp?)?.toDate();
    if (date == null) continue;

    final total = (data['totalPrice'] ?? 0).toDouble();
    minDate ??= date;
    if (maxDate == null || date.isAfter(maxDate)) maxDate = date;

    String d = formatter.format(date);
    String w = '${date.year}-W${((date.difference(DateTime(date.year)).inDays + DateTime(date.year).weekday) / 7).ceil()}';
    String m = '${date.year}-${date.month.toString().padLeft(2, '0')}';
    String y = '${date.year}';

    tempDaily[d] = (tempDaily[d] ?? 0) + total;
    tempWeekly[w] = (tempWeekly[w] ?? 0) + total;
    tempMonthly[m] = (tempMonthly[m] ?? 0) + total;
    tempYearly[y] = (tempYearly[y] ?? 0) + total;
  }

  if (minDate != null && maxDate != null) {
    for (DateTime d = minDate; !d.isAfter(maxDate); d = d.add(const Duration(days: 1))) {
      tempDaily.putIfAbsent(formatter.format(d), () => 0.0);
    }
  }
```

Figure 4.57 Code Snippet for Firestore Sales Query and Chart View

```
  setState(() {
    dailySales = Map.fromEntries(tempDaily.entries.toList()..sort((a, b) => a.key.compareTo(b.key)));
    weeklySales = Map.fromEntries(tempWeekly.entries.toList()..sort((a, b) => a.key.compareTo(b.key)));
    monthlySales = Map.fromEntries(tempMonthly.entries.toList()..sort((a, b) => a.key.compareTo(b.key)));
    yearlySales = Map.fromEntries(tempYearly.entries.toList()..sort((a, b) => a.key.compareTo(b.key)));
  });
}
```

Figure 4.58 Code Snippet for Setting Graph to Daily, Weekly, Monthly, and Yearly

```
        Row(
          children: [
            Expanded(child: buildLabeledDropdown(
                label: 'Filter',
                value: selectedSort,
                items: sortOptions[selectedView]!,
                onChanged: (v) => setState(() => selectedSort = v!)
              )
            ),  // Expanded
            const SizedBox(width: 12),
            Expanded(child: buildLabeledDropdown(
                label: 'Chart Type',
                value: selectedChart,
                items: ['Bar', 'Line'],
                onChanged: (v) => setState(() => selectedChart = v!), showIcons: true
              )
            ),  // Expanded
          ],
        ),  // Row
```

Figure 4.59 Code Snippet for Filtering Graph Type

4.4.2.12 Admin Profile (Update Name & Password)

Admins can view and update their display name and password through the web dashboard. The email address is displayed as read-only for identification and security purposes. When updating the password, the system uses Firebase Authentication to send a secure password reset link to the admin's registered email. All profile changes are synchronized with both Firebase Authentication and Firestore in real time.

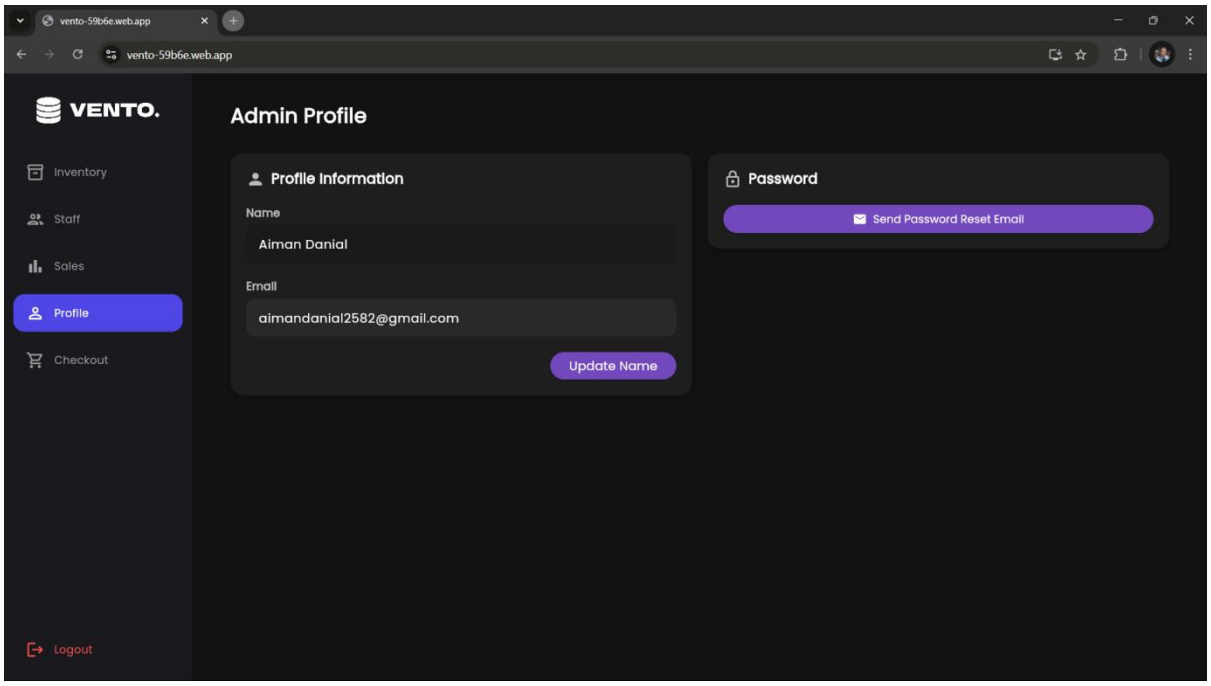

Figure 4.60 Admin Profile Page

```
onPressed: _isLoading
    ? null
    : () async {
  final result = await showUpdateAdminNameDialog(
    context: context,
    nameController: _nameController,
  );
  if (result) {
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(content: Text("Admin's name updated successfully")),
    );
    setState(() {});
  }
},
```

Figure 4.61 Code Snippet for Triggering Admin Name Update and Providing Confirmation Feedback

```dart
Future<void> _sendPasswordResetEmail() async {
  final user = _auth.currentUser;
  if (user == null || user.email == null) return;

  try {
    await _auth.sendPasswordResetEmail(email: user.email!);
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text("Password reset email sent to ${user.email}")),
    );
  } catch (e) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text("Failed to send reset email: $e")),
    );
  }
}
```

Figure 4.62 Code Snippet for Sending Password Reset Email via Firebase Authentication

4.4.2.13 Checkout Page

This page shows the products selected for checkout. Admins can adjust the quantity for each product size or remove items if needed. The total price updates automatically as changes are made. When checkout is confirmed, the system updates the product stock in Firestore and records the transaction in the sales report, including the date and the quantity sold for each size.

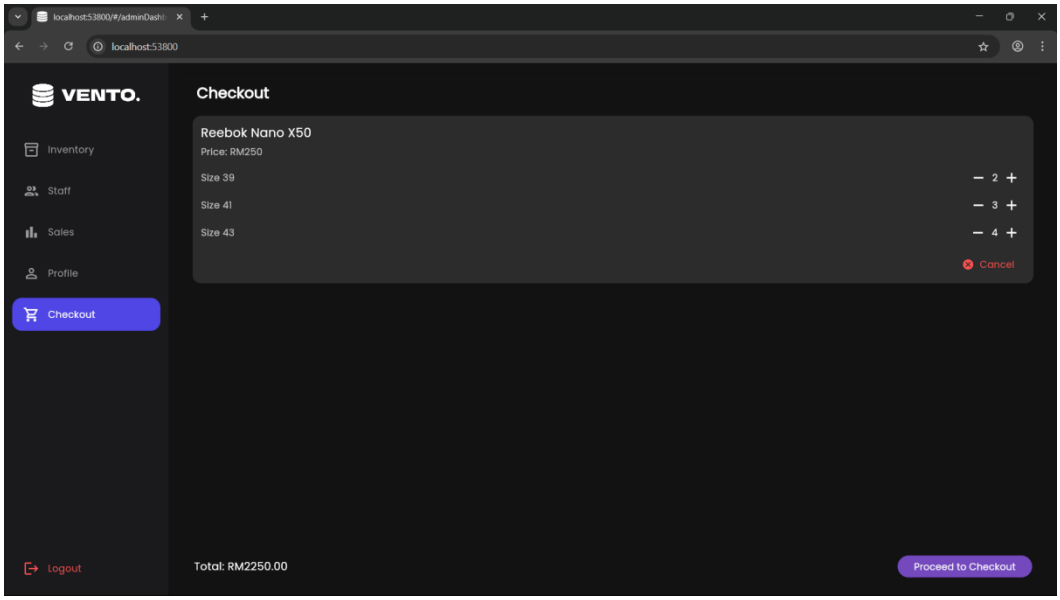

Figure 4.63 Checkout Page

```dart
Future<void> completeCheckout(List<Map<String, dynamic>> cartItems) async {
  for (var item in cartItems) {
    final id = item['id'];
    final name = item['name'];
    final price = (item['price'] ?? 0).toDouble();
    final adminId = item['adminId'];
    final sizes = Map<String, int>.from(item['sizes'] ?? {});
    final totalQuantity = sizes.values.fold(0, (a, b) => a + b);
    final totalPrice = totalQuantity * price;

    await salesCollection.add({
      'productId': id,
      'productName': name,
      'sizesSold': sizes,
      'quantitySold': totalQuantity,
      'totalPrice': totalPrice,
      'soldAt': FieldValue.serverTimestamp(),
      'adminId': adminId,
    });
  }
}
```

Figure 4.64 Code Snippet for Processing Checkout Logic

4.4.2.14 Logout

Admins can securely log out using the logout button in the navigation bar. It signs out from Firebase Authentication and redirects to the login page.

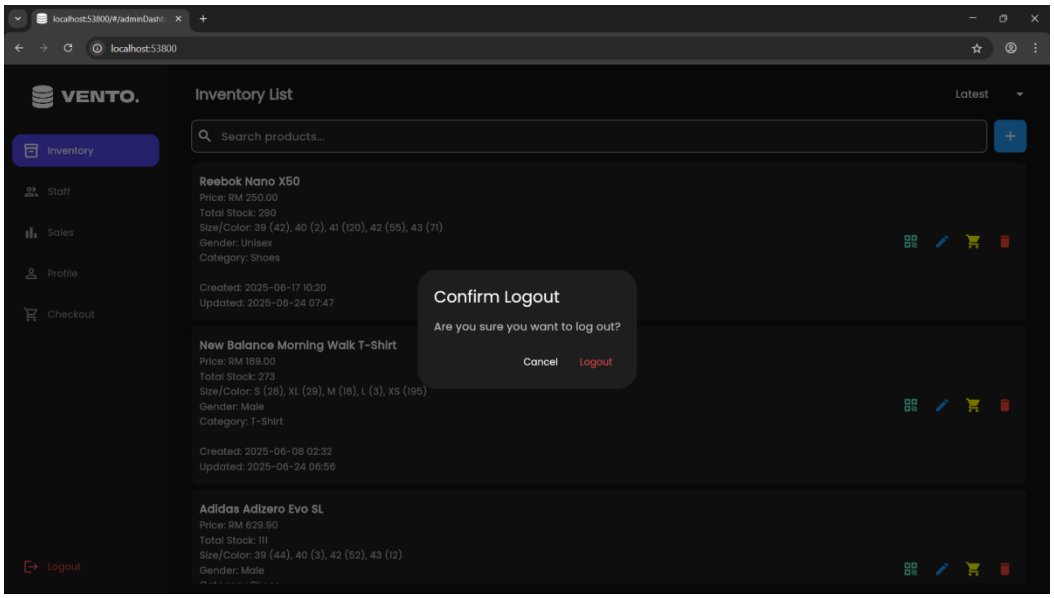

Figure 4.65 Logout Option in Navbar

```dart
Future<void> _confirmLogout() async {
  final confirmed = await showDialog<bool>(
    context: context,
    barrierDismissible: false,
    builder: (context) => AlertDialog(
      title: Text(
        'Confirm Logout',
        style: GoogleFonts.poppins(
          fontSize: 24,
          color: Colors.white,
        ),
      ),  // Text
      content: Text(
        'Are you sure you want to log out?',
        style: GoogleFonts.poppins(
          fontSize: 16,
          color: Colors.white70,
        ),
      ),  // Text
      backgroundColor: const Color(0xFF1E1E1E),
      actions: [
        TextButton(
          onPressed: () => Navigator.pop(context, false),
          child: Text('Cancel', style: GoogleFonts.poppins(color: Colors.white)),
        ),  // TextButton
        TextButton(
          onPressed: () => Navigator.pop(context, true),
          child: Text('Logout', style: GoogleFonts.poppins(color: Colors.redAccent)),
        ),  // TextButton
      ],
    ),  // AlertDialog
  );
```

Figure 4.66 Code Snippet for Firebase Sign Out Function

```dart
  if (confirmed == true) {
    await FirebaseAuth.instance.signOut();
    Navigator.pushReplacementNamed(context, '/');
  }
}
```

Figure 4.67 Continuation of The Code Snippet for Firebase Sign Out Function

4.5 SUMMARY

This chapter explained the full implementation of Vento: Voice-Assisted Inventory Management System. The system was developed using Flutter, which allowed it to run on both Android devices and web browsers, making it accessible across platforms. The Android app is specifically designed for retail staff to perform inventory tasks using either voice commands or manual input. Meanwhile, the web dashboard is used by admin users to manage inventory and staff records, as well as to view visualized sales reports.

Firebase served as the backend for real-time data synchronization, secure authentication, and cloud storage. Staff members are required to verify their accounts before gaining access through the Android app, while admins manage the system via the web platform. Role-based access is enforced throughout the system to ensure that staff can only perform product-level operations, whereas admin users are responsible for managing staff accounts and reviewing sales performance.

Key features such as voice command input, QR code generation and scanning, and CRUD functionality were successfully integrated. Customers can also scan the generated QR codes to view product information. The implementation also covered product size variations, voice assistant logic, and real-time updates through Firestore.

Overall, the system was implemented with a focus on usability, efficiency, and proper role separation, ensuring it meets the needs of both staff and admin users in a typical retail setting.

CHAPTER 5: TESTING AND EVALUATION

5.1 INTRODUCTION

This chapter presents the testing and evaluation procedures carried out for Vento: Voice-Assisted Inventory Management System, developed for both admin (web) and staff (mobile app) platforms. The main objective of testing was to verify that all core functionalities operate as intended, while also ensuring a user-friendly interface and smooth system performance. Testing included functional testing of CRUD operations, QR code usage, voice command accuracy, and system responsiveness, as well as usability evaluation to assess the overall user experience, particularly regarding the voice assistant and inventory management interface.

5.2 FUNCTIONAL TESTING

Functional testing was conducted to confirm that each feature of the system works according to the requirements. It focused on black-box testing, validating outputs based on user interactions without analyzing internal code logic. Test cases were designed based on expected user behavior and system specifications, covering all modules of both the admin and user sessions.

The functional testing verified correct behavior for inventory operations, voice command responses, user account management, checkout flow, and data storage in Firestore. Each module was tested independently and then integrated to ensure end-to-end functionality.

A) Admin Web System

1. Inventory Management Module

   - Add Product
   - Edit Product Details (name, price, category, size, gender)
   - Delete Product
   - View Product List with Search and Sorting
   - QR Code Generation and Download

- View Product Details via QR Code

2. Staff Management Module
    - Add Staff Member
    - Edit Staff Member Details
    - Delete Staff Member
    - View List of Registered Staff

3. Sales Report Module
    - View Daily, Weekly, Monthly, and Yearly Sales
    - Switch Between Bar and Line Chart Visualizations
    - Filter Reports Based on Time Ranges (e.g., "This Month", "Last 4 Weeks")

4. Checkout Module
    - Select Products for Checkout
    - Adjust Quantities Per Size
    - Cancel Items from Checkout
    - Finalize Checkout and Log Sale to Firestore

5. Admin Profile Module
    - □View and Update Display Name
    - □Send Password Reset Email
    - □Display Admin Email (read-only)
    - □Sync Profile Changes with Firebase Authentication and Firestore

6. Logout Functionality

    - Securely Log Out Admin and Redirect to Login Screen

B) Staff Mobile App System

1. Inventory Management (Mobile)

    - Add New Product Manually
    - Edit Product Information
    - Delete Product
    - View Inventory List with Product Details

2. Voice Assistant Module

    - Add Product via Voice Command
    - Edit Product via Voice Command (name, stock, size)
    - Delete Product via Voice Command (with confirmation)
    - View Product Details via Voice Query
    - Query Specific Product Info (e.g., stock of Size L)

3. Staff Profile Module

    - View Display Name (Read-only)
    - View Email Address (Read-only)

4. QR Code Functionality

    - Scan QR to View Product Info
    - Verify Real-Time Sync of Product Details

5. Logout Functionality

- Secure Log Out from Mobile App

5.2.1 Staff Session

This section explains the testing done for the user features in the Android mobile app of Vento: Voice-Assisted Inventory Management System. The goal was to check if important functions like signing up, logging in, logging out, and resetting passwords work properly. The tests also checked if the app gives the right feedback and is easy to use.

5.2.1.1 Login Module Testing

Table 5.1 Test Case for Login Module

Module Title:	Login				
Designed By:	Aiman Danial Jasni				
Test Objective:	To ensure the login functionality authenticates staff and provides proper error handling for invalid credentials.				
Test Case ID	**Test Case**	**Test Steps**	**Expected Result**	**Actual Result**	**Status**
TC-LO-01	Login with valid credentials	1. Open app 2. Enter registered email and password 3. Tap "Login"	Redirected to Inventory screen	Redirect successful	Pass
TC-LO-02	Invalid email format	1. Enter malformed email (e.g.,	Error message: "Invalid	Error displayed	Pass

		user@com) 2. Tap "Login”	email format"		
TC-LO-03	Incorrect password	1. Enter correct email but wrong password 2. Tap "Login"	Error message: "Incorrect password”	Error displayed	Pass
TC-LO-04	Forgot password navigation	1. Tap "Forgot Password"	Redirected to password reset screen	Navigation successful	Pass

5.2.1.2 Add New Product Manually

Table 5.2 Test Case for Add New Product Manually

Module Title:	Manual Product Addition				
Designed By:	Aiman Danial Jasni				
Test Objective:	To ensure staff can manually add new products with correct data validation and Firestore integration.				
Test Case ID	**Test Case**	**Test Steps**	**Expected Result**	**Actual Result**	**Status**
TC-MP-01	Add product with valid inputs	1. Navigate to Add Product page 2. Enter valid details (name,	Product added to Firestore and visible in list	Success	Pass

		category, price, sizes) 3. Tap "Save"			
TC-MP-02	Missing name field	1. Leave name empty 2. Tap "Save"	Error: "Product name is required"	Validation works	Pass

5.2.1.3 Edit Product Manually

Table 5.3 Test Case for Edit Product Manually

Module Title:	Edit Product Information				
Designed By:	Aiman Danial Jasni				
Test Objective:	To verify that staff can update product details and sync changes with Firestore.				
Test Case ID	**Test Case**	**Test Steps**	**Expected Result**	**Actual Result**	**Status**
TC-MP-03	Update price and stock	1. Tap product → Edit 2. Modify price and stock 3. Tap "Update"	Changes saved and reflected in UI	Updated correctly	Pass

5.2.1.4 Delete Product Manually

Table 5.4 Test Case for Delete Product Manually

Module Title:	Delete Product Manually				
Designed By:	Aiman Danial Jasni				
Test Objective:	To ensure users can delete products and the record is removed from Firestore.				
Test Case ID	**Test Case**	**Test Steps**	**Expected Result**	**Actual Result**	**Status**
TC-MP-04	Delete a product	1. Long press product 2. Tap "Delete" 3. Confirm deletion	Product is removed	Deleted successfully	Pass

5.2.1.5 View Inventory List

Table 5.5 Test Case for View Inventory List

Module Title:	View Inventory List with Product Details				
Designed By:	Aiman Danial Jasni				
Test Objective:	To verify that the product list displays all added products with accurate details and supports scroll and selection.				
Test Case ID	**Test Case**	**Test Steps**	**Expected**	**Actual**	**Status**

			Result	**Result**	
TC-IN-01	Load product list	1. Login 2. Navigate to Inventory	Product list is displayed	All products shown	Pass
TC-IN-02	View product details	1. Tap on a product 2. View full product info	Details popup or screen shown	Works as expected	Pass

5.2.1.6 Add Product via Voice Command

Table 5.6 Test Case for Add Product via Voice Command

Module Title:	Voice Add Product				
Designed By:	Aiman Danial Jasni				
Test Objective:	To confirm the voice assistant can capture product details and create a new product entry.				
Test Case ID	**Test Case**	**Test Steps**	**Expected Result**	**Actual Result**	**Status**
TC-VA-01	Add via voice	1. Tap mic icon 2. Say "Add product Nike shoes for RM250 in size L with stock 50"	Assistant confirms and adds product	Product added correctly	Pass
TC-VA-02	Incomplete command	1. Say "Add product named	Assistant asks for missing	Prompted correctly	Pass

		Adidas" only	fields		

5.2.1.7 Edit Product via Voice Command

Table 5.7 Test Case for Edit Product via Voice Command

Module Title:	Voice Edit Product				
Designed By:	Aiman Danial Jasni				
Test Objective:	To test if voice assistant can update product fields like name, size stock, and price.				
Test Case ID	**Test Case**	**Test Steps**	**Expected Result**	**Actual Result**	**Status**
TC-VE-01	Edit stock	Say "Edit Nike stock for size L to 100"	Assistant updates stock	Reflected correctly	Pass
TC-VE-02	Invalid product name	Say "Edit Banana stock to 30"	Assistant says "product not found"	Handled gracefully	Pass

5.2.1.8 Delete Product via Voice Command

Table 5.8 Test Case for Delete Product via Voice Command

Module Title:	Voice Delete Product
Designed By:	Aiman Danial Jasni
Test Objective:	To ensure assistant can delete product after confirmation.

Test Case ID	Test Case	Test Steps	Expected Result	Actual Result	Status
TC-VD-01	Delete command	Say "Delete product Nike" Confirm when asked	Product deleted	Works correctly	Pass

5.2.1.9 View Product Details via Voice

Table 5.9 Test Case for View Product Details via Voice

Module Title:	Voice View Product Details				
Designed By:	Aiman Danial Jasni				
Test Objective:	To test if the assistant can respond with full product info on request.				
Test Case ID	**Test Case**	**Test Steps**	**Expected Result**	**Actual Result**	**Status**
TC-VV-01	Ask product details	Say "Tell me about Adidas bag"	Voice responds with product info	Info read correctly	Pass

5.2.1.10 Query Specific Stock Info

Table 5.10 Test Case for Query Specific Stock Info

Module Title:	Voice Stock Query

Designed By:	Aiman Danial Jasni				
Test Objective:	To verify assistant can respond to size-specific stock queries.				
Test Case ID	**Test Case**	**Test Steps**	**Expected Result**	**Actual Result**	**Status**
TC-VS-01	Ask stock	Say "How many L size for Nike?"	Assistant says stock count	Accurate response	Pass

5.2.1.11 Scan QR to View Product Info

Table 5.11 Test Case for Scan QR to View Product Info

Module Title:	QR Code Scan				
Designed By:	Aiman Danial Jasni				
Test Objective:	To confirm QR scanning shows the correct product details from Firestore.				
Test Case ID	**Test Case**	**Test Steps**	**Expected Result**	**Actual Result**	**Status**
TC-QR-01	Scan product QR	Open phone camera → Scan QR	Product info shown	Synced & accurate	Pass

5.2.1.12 Real-Time Product Sync

Table 5.12 Test Case for Real-Time Product Sync

Module	QR Real-Time Sync

Title:					
Designed By:	Aiman Danial Jasni				
Test Objective:	To verify that changes in product info reflect instantly when QR is scanned again.				
Test Case ID	**Test Case**	**Test Steps**	**Expected Result**	**Actual Result**	**Status**
TC-QR-02	Edit then scan again	Edit product → Scan QR again	Updated info shown	Real-time verified	Pass

5.2.1.13 View Staff Profile Info

Table 5.13 Test Case for View Staff Profile Info

Module Title:	Staff Profile				
Designed By:	Aiman Danial Jasni				
Test Objective:	To confirm staff can update display name and view email.				
Test Case ID	**Test Case**	**Test Steps**	**Expected Result**	**Actual Result**	**Status**
TC-UP-01	View Profile Info	Go to Profile → Check display name and email address	Staff info (name and email) shown correctly	Info displayed as expected	Pass

5.2.1.14 Logout Functionality

Table 5.14 Test Case for Logout Functionality

Module Title:	Logout				
Designed By:	Aiman Danial Jasni				
Test Objective:	To verify staff can securely sign out and return to login screen.				
Test Case ID	**Test Case**	**Test Steps**	**Expected Result**	**Actual Result**	**Status**
TC-LO-05	Logout	Tap Logout button in profile or sidebar	App signs out and redirects to login	Works as expected	Pass

5.2.2 Admin Session

This section explains the testing carried out for the admin-side features of Vento's web-based Voice-Assisted Inventory Management System. Each feature was tested to make sure it worked correctly, allowed the admin to manage products, staff, sales reports, and profile settings smoothly, and gave accurate feedback during use.

5.2.2.1 Admin Login Module Testing

Table 5.15 Test Case for Admin Login Module

Module Title:	Admin Login
Designed	Aiman Danial Jasni

By:					
Test Objective:	To ensure the admin can log in with valid credentials and receives proper error feedback for invalid input.				
Test Case ID	**Test Case**	**Test Steps**	**Expected Result**	**Actual Result**	**Status**
TC-AL-01	Login with valid credentials	1. Go to login page 2. Enter valid email and password 3. Click Login	Admin dashboard should be displayed	Admin dashboard displayed	Pass
TC-AL-02	Invalid email format	Enter incorrect email format	Show email format error	Error displayed	Pass
TC-AL-03	Incorrect password	Enter valid email with wrong password	Show incorrect password error	Error displayed	Pass

5.2.2.2 Inventory Management Module Testing

Table 5.16 Test Case for Inventory Management Module

Module Title:	Inventory Management				
Designed By:	Aiman Danial Jasni				
Test Objective:	To verify product CRUD operations and QR code features function as intended.				
Test Case ID	**Test Case**	**Test Steps**	**Expected**	**Actual**	**Status**

			Result	**Result**	
TC-IM-01	Add Product	Fill form and click Add	Product appears in list	Product added successfully	Pass
TC-IM-02	Edit Product	Select product, modify fields, click Save	Product updates correctly	Product updated	Pass
TC-IM-03	Delete Product	Click Delete icon and confirm	Product removed	Product deleted	Pass
TC-IM-04	View Inventory List	Open inventory page	List displays all products with sorting/search	Products visible	Pass
TC-IM-05	Generate QR Code	Add product or view details	QR Code is generated	QR Code generated	Pass
TC-IM-06	View Product via QR	Scan QR from another device	Product detail page opens	Product info shown	Pass

5.2.2.3 Staff Management Module Testing

Table 5.17 Test Case for Staff Management Module

Module Title:	Staff Management
Designed By:	Aiman Danial Jasni

Test Objective:	To verify CRUD operations on staff records.				
Test Case ID	**Test Case**	**Test Steps**	**Expected Result**	**Actual Result**	**Status**
TC-SM-01	Add Staff	Fill form with name/email/role and click Save	Staff appears in list with "Pending" status	Staff added successfully	Pass
TC-SM-02	Edit Staff	Select staff and update info	Staff data updated	Staff edited	Pass
TC-SM-03	Delete Staff	Click Delete and confirm	Staff removed from list	Staff deleted	Pass
TC-SM-04	View Staff List	Open staff page	All registered staff visible	Staff listed	Pass
TC-SM-05	Verify Staff After Login	1. Staff receives registration email 2. Staff verifies email 3. Logs in via Android app 4. Admin refreshes staff list	Staff account status changes from Pending to Active/Verified	Status shown as Verified	Pass

5.2.2.4 Sales Report Module Testing

Table 5.18 Test Case for Sales Report Module

Module Title:	Sales Report

Designed By:	Aiman Danial Jasni				
Test Objective:	To ensure sales data is accurately displayed and chart features function correctly.				
Test Case ID	**Test Case**	**Test Steps**	**Expected Result**	**Actual Result**	**Status**
TC-SR-01	View Daily Report	Select "Daily" from toggle	Daily data chart shown	Data displayed	Pass
TC-SR-02	Switch Chart View	Select Bar or Line chart	Chart updates accordingly	Chart switched	Pass
TC-SR-03	Filter Date Range	Select range (e.g. This Month)	Data updates to filter	Data filtered	Pass

5.2.2.5 Checkout Module Testing

Table 5.19 Test Case for Checkout Module

Module Title:	Checkout				
Designed By:	Aiman Danial Jasni				
Test Objective:	To verify that checkout operations update stock and log sales correctly.				
Test Case ID	**Test Case**	**Test Steps**	**Expected Result**	**Actual Result**	**Status**
TC-CO-01	Select Products	Add products to checkout list	Products displayed	Products selected	Pass

TC-CO-02	Adjust Quantity	Increase/decrease per size	Quantities updated	Quantity changed	Pass
TC-CO-03	Cancel Item	Click cancel on item	Item removed from list	Item cancelled	Pass
TC-CO-04	Finalize Checkout	Click Proceed to Checkout	Stock reduced, sale logged	Checkout successful	Pass

5.2.2.6 Admin Profile Module Testing

Table 5.20 Test Case for Admin Profile Module

Module Title:	Admin Profile				
Designed By:	Aiman Danial Jasni				
Test Objective:	To confirm admin can update personal details and that changes are saved.				
Test Case ID	**Test Case**	**Test Steps**	**Expected Result**	**Actual Result**	**Status**
TC-AP-01	View Profile	Navigate to profile page	Show current name/email	Info displayed	Pass
TC-AP-02	Update Name	Edit name and click Save	Name updated in Firestore	Name updated	Pass
TC-AP-03	Reset Password	Click button to send reset email	Password reset email is sent	Email sent	Pass

TC-AP-04	Email Uneditable	Try to edit email field	Email is read-only	Cannot edit email	Pass

5.2.2.7 Admin Logout Functionality

Table 5.21 Test Case for Admin Logout Functionality

Module Title:	Admin Logout				
Designed By:	Aiman Danial Jasni				
Test Objective:	To ensure the logout button securely signs out and redirects to login.				
Test Case ID	**Test Case**	**Test Steps**	**Expected Result**	**Actual Result**	**Status**
TC-LO-01	Logout	Click on logout in sidebar	Signed out and redirected to login	Logout successful	Pass

5.3 USABILITY TESTING

Usability testing is a method of evaluating how easy and user-friendly a system is by observing real users as they interact with it. The purpose is to uncover any difficulties, gather feedback, and assess user satisfaction. In this project, usability testing plays a key role in ensuring that both the mobile app and web system meet user expectations and provide an intuitive experience for managing inventory using voice commands.

For this usability test, a total of 10 testers were selected, including both technical and non-technical users to simulate real-world scenarios. Testers interacted with key features of the system, such as product management, QR code scanning, sales reporting, and the voice assistant.

After using the system, each participant completed a detailed questionnaire divided into the following sections:

1. Website and App Navigation
2. Voice Assistant Capabilities
3. QR Code and Inventory Interaction
4. Overall System Performance and Satisfaction

The goal of this evaluation was to measure:

- The ease of navigating both the admin website and user mobile app.
- The accuracy and understanding of the voice assistant.
- The clarity and usefulness of the sales graph and QR code features.
- The overall design appeal and responsiveness of the system.
- Suggestions for future improvement.

The feedback obtained helped identify strengths and areas for enhancement, ensuring the system is practical, reliable, and accessible for inventory management purposes.

5.3.1 Usability Testing Results

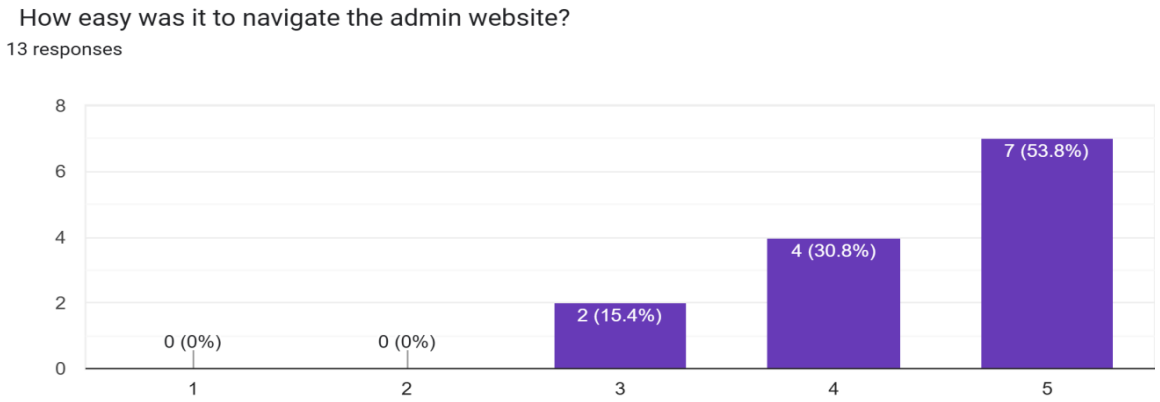

Figure 5.1 How easy was it to navigate the admin website?

Figure 5.1 shows user feedback on the ease of navigating the admin website. The majority of testers responded positively, with 7 out of 13 selecting 5 and 4 selecting 4, indicating that most users found the admin website easy to navigate. Only 2 users selected 3, and no users rated it below that, showing that the interface was generally intuitive and user-friendly. This suggests that the admin panel's layout and navigation flow were clear, accessible, and did not cause confusion during testing.

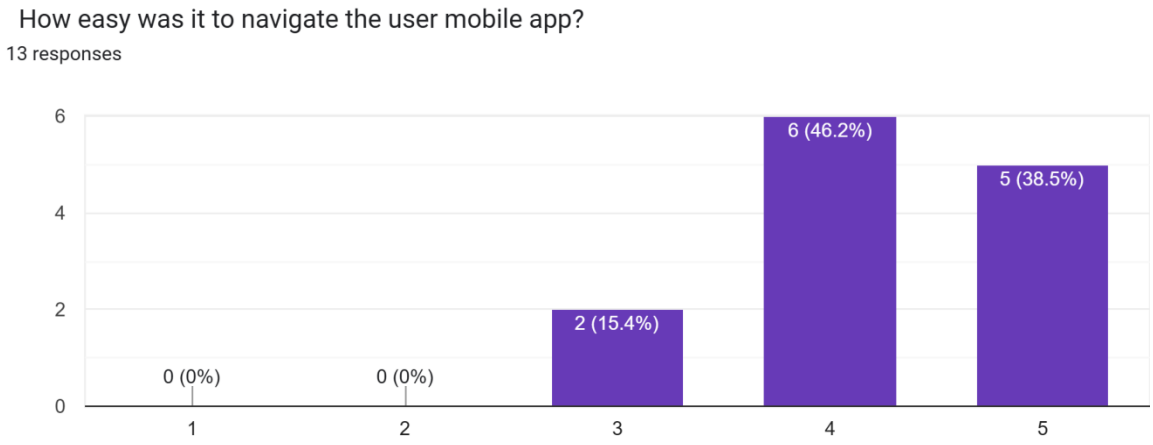

Figure 5.2 How easy was it to navigate the user website?

Figure 5.2 shows user feedback on how easy it was to navigate the user website. Out of 13 respondents, the majority rated the navigation experience positively, with 5 selecting 5 and 6 selecting 4. This indicates that most users found the user website clear and easy to use. Only 2 users selected 3, and no one rated it lower, suggesting that the website was generally user-friendly and simple to navigate for most testers.

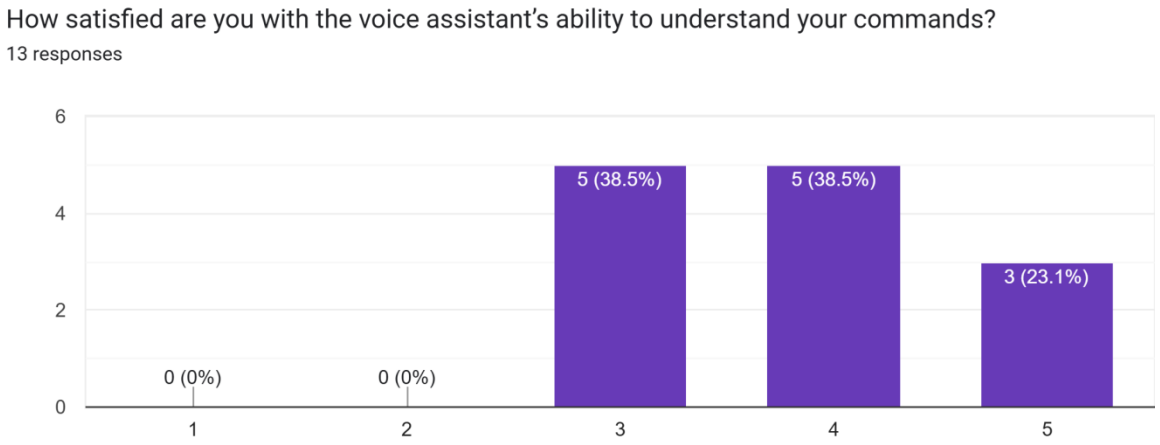

Figure 5.3 How satisfied are you with the voice assistant's ability to understand your commands?

Figure 5.3 shows the level of satisfaction with the voice assistant's ability to understand user commands. Out of 13 respondents, 5 users selected 3, 5 selected 4, and 3 selected 5. This indicates that while most users found the voice assistant fairly capable of understanding commands, there is still room for improvement in accuracy and responsiveness. The majority leaning toward 4 and 5 shows generally positive feedback, but the number of users selecting 3 suggests that the system may occasionally misinterpret input or require clearer articulation.

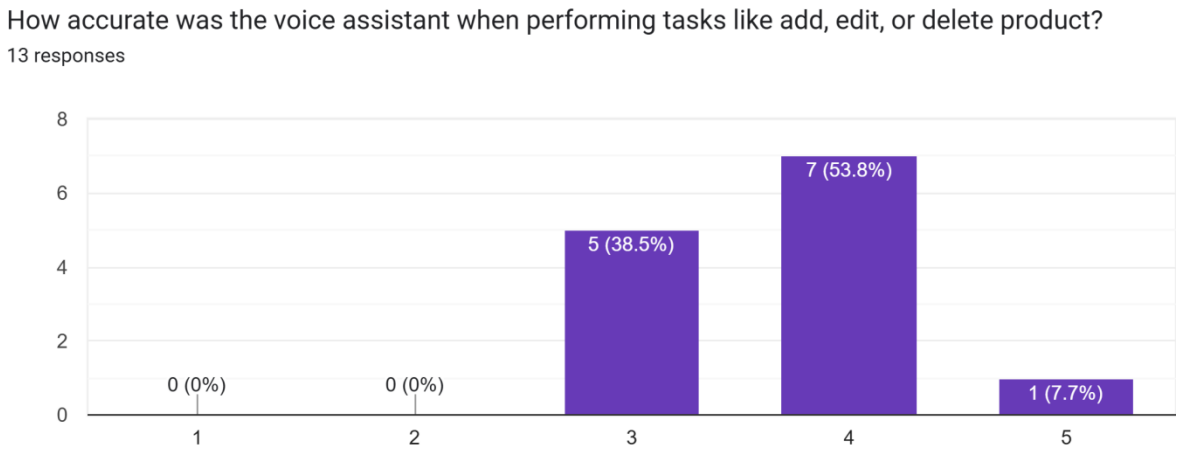

Figure 5.4 How accurate was the voice assistant when performing tasks like add, edit, or delete product?

Figure 5.4 illustrates the accuracy of the voice assistant in performing tasks such as adding, editing, or deleting products. Among 13 participants, 5 rated it a 3, 7 rated it a 4, and 1 rated it a 5. This indicates that most users found the voice assistant moderately accurate, with the majority leaning toward satisfactory performance. However, the fact that several users rated it a 3 suggests that occasional errors or misunderstandings may occur during task execution. Continued refinement of voice recognition and task mapping could help improve overall accuracy and user confidence.

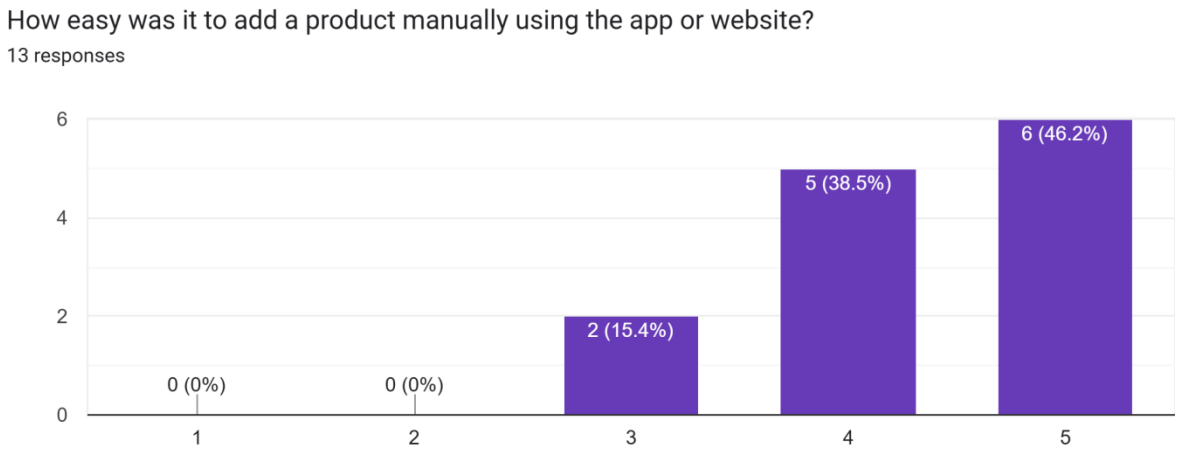

Figure 5.5 How easy was it to add a product manually using the app or website?

Figure 5.5 illustrates how easy it was for users to manually add a product using the app or website. From the 13 responses, 2 people chose 3, 5 people chose 4, and 6 people chose 5. This means most users found it easy to add a product manually. The results show that the steps were clear and the process was smooth for most people. However, a few users found it a bit harder, which suggests that small improvements, such as clearer instructions or better error messages could make the experience even better.

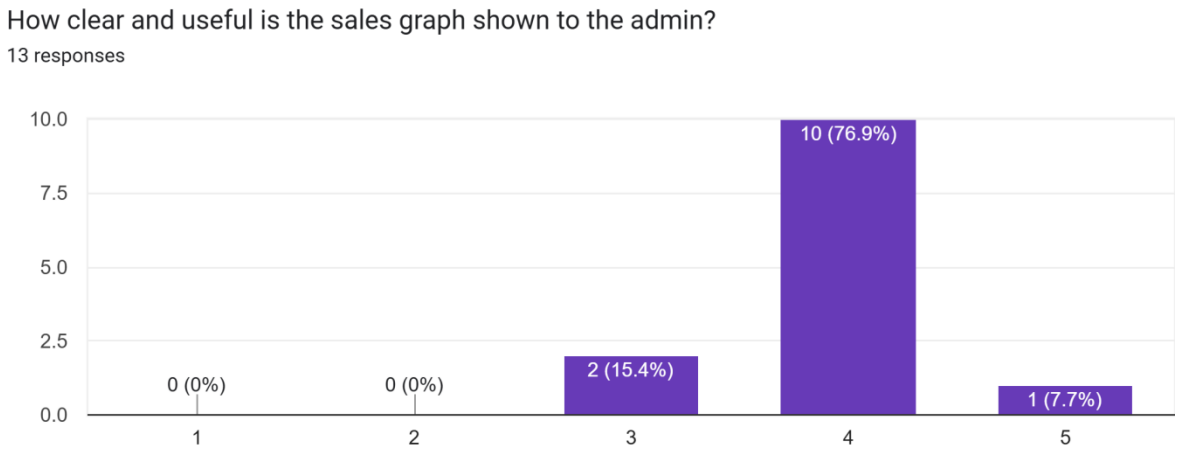

Figure 5.6 How clear and useful is the sales graph shown to the admin?

Figure 5.6 shows how clear and useful the sales graph was to the admin. From the 13 responses, 2 people selected 3, 10 selected 4, and 1 selected 5. This indicates that most users found the sales graph to be clear and useful, especially for tracking sales trends. Although only one person rated it as excellent, the majority felt it met their needs well. A small number of users found it slightly less clear, suggesting there may be room to improve the graph's labels, layout, or visual styling for even better clarity.

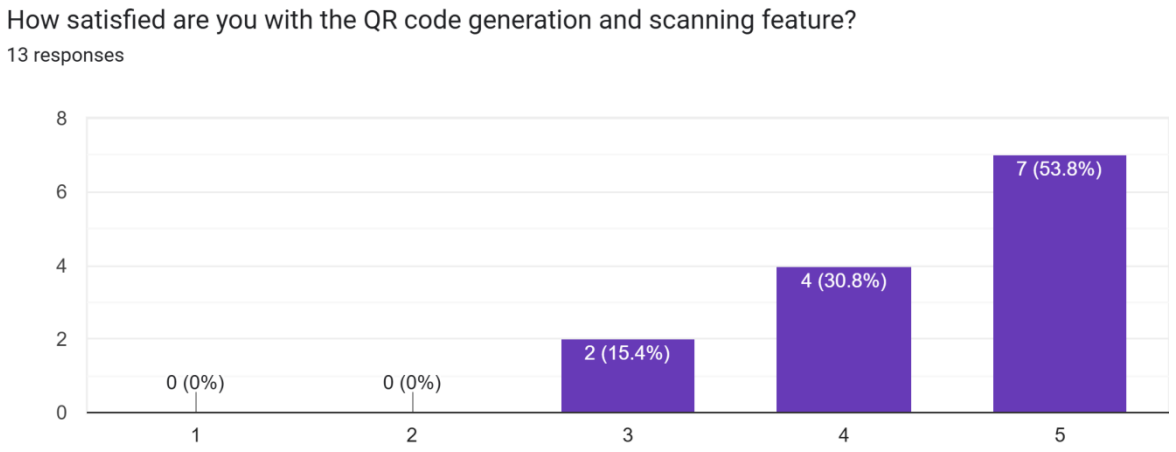

Figure 5.7 How satisfied are you with the QR code generation and scanning feature?

Figure 5.7 shows how satisfied users were with the QR code generation and scanning feature. Out of 13 responses, 2 users rated it a 3, 4 rated it a 4, and 7 rated it a 5. This indicates that the majority of users were highly satisfied with this feature, finding it easy and effective to generate and scan QR codes. A few users gave moderate ratings, which may suggest that minor usability improvements could be made, but overall, the feedback shows that the QR code functionality is performing well and is appreciated by users.

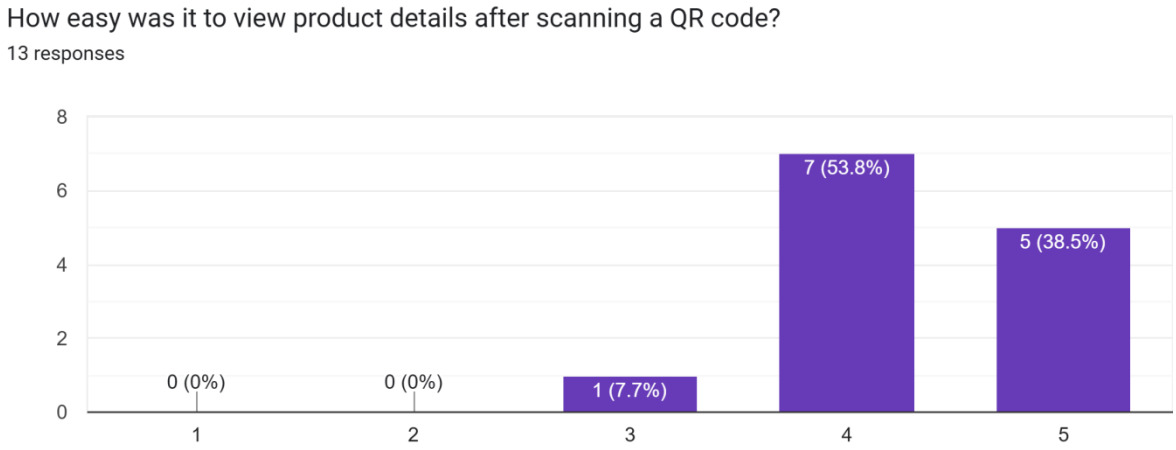

Figure 5.8 How easy was it to view product details after scanning a QR code?

Figure 5.8 illustrates how easy it was for users to view product details after scanning a QR code. Out of 13 responses, 1 user rated it a 3, 7 users rated it a 4, and 5 users rated it a 5. This

shows that the majority of users found it easy to access product details through QR code scanning. While one user found it moderately difficult, most users had a positive experience, indicating that the feature is functioning smoothly and intuitively for most people.

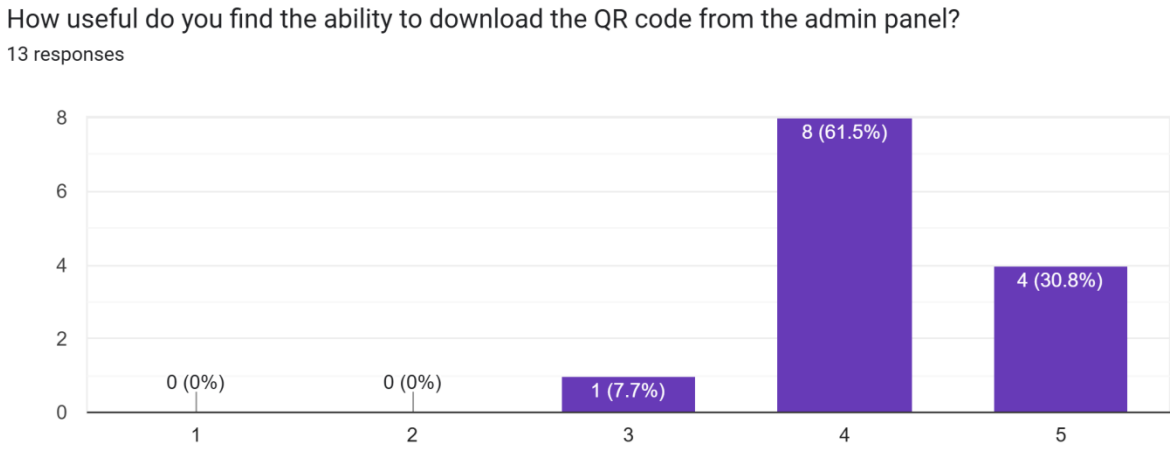

Figure 5.9 How useful do you find the ability to download the QR code from the admin panel?

Figure 5.9 shows how useful users found the ability to download the QR code from the admin panel. Out of 13 responses, 1 person rated it a 3, 8 people rated it a 4, and 4 people rated it a 5. This indicates that most users found the QR code download feature to be useful and convenient. The majority rating of 4 suggests users appreciated the functionality, while the 5 ratings show strong satisfaction. Only one user found it moderately useful, showing that this feature is generally well-received.

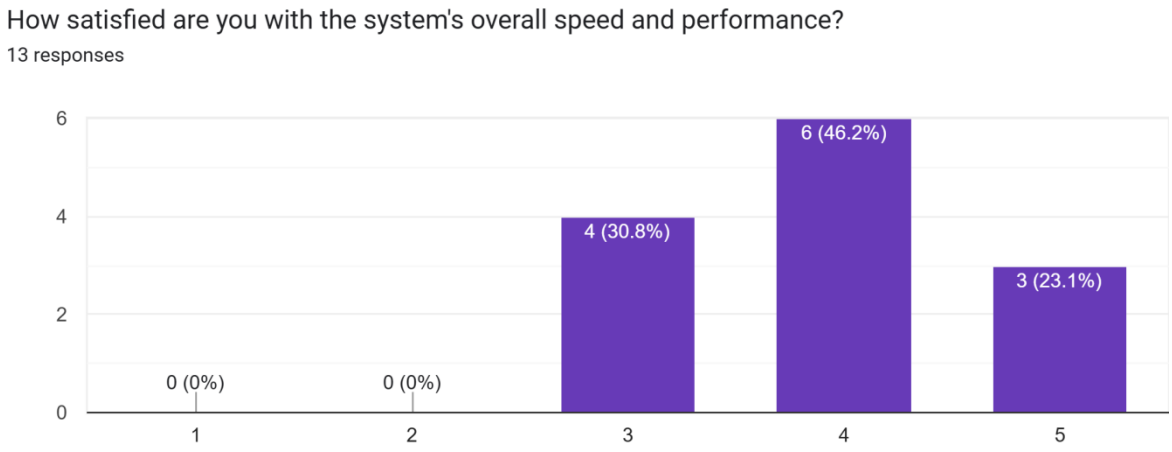

Figure 5.10 How satisfied are you with the system's overall speed and performance?

Figure 5.10 shows user satisfaction with the system's overall speed and performance. From the 13 responses, 4 users selected 3, 6 selected 4, and 3 selected 5. This suggests that most users were moderately to highly satisfied with the system's performance. While a few experienced average performance, the majority found the system to be responsive and smooth. The feedback indicates that overall speed is acceptable, but there may still be room for slight performance improvements to ensure a consistently fast experience for all users.

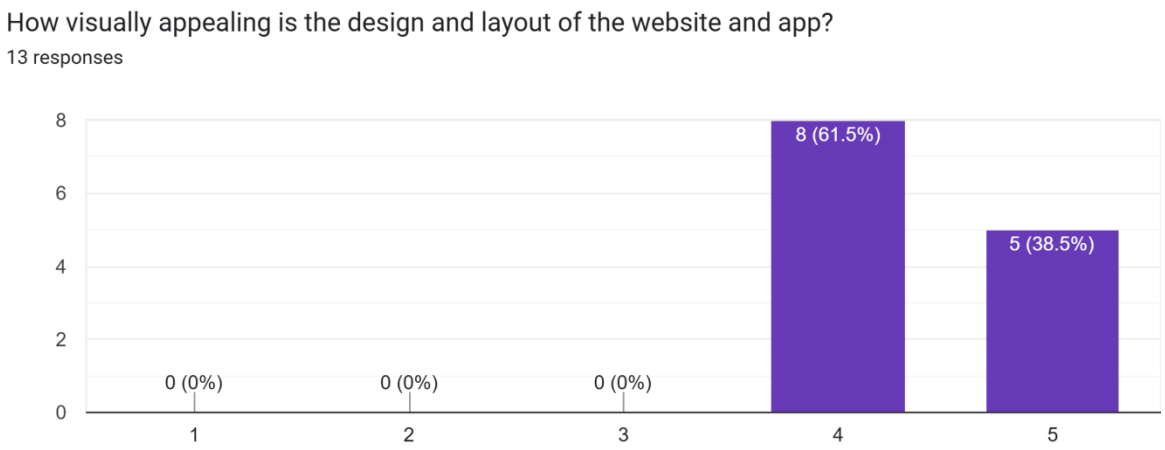

Figure 5.11 How visually appealing is the design and layout of the website and app?

Figure 5.11 illustrates user opinions on the visual appeal of the website and app's design and layout. Out of 13 respondents, 8 rated it a 4 and 5 rated it a 5, showing that users were highly satisfied with the interface. This indicates that the overall design is visually appealing, clean, and easy to navigate. The strong positive response reflects that the layout effectively supports a smooth user experience and leaves a good impression in terms of aesthetics.

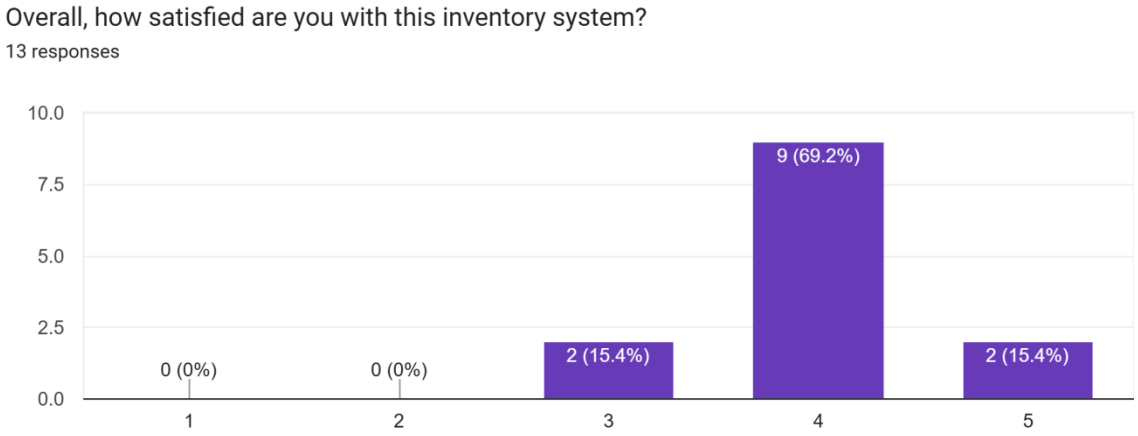

Figure 5.12 Overall, how satisfied are you with this inventory system?

Figure 5.12 presents the overall satisfaction level with the inventory system. Among the 13 respondents, 2 rated it a 3, 9 rated it a 4, and 2 rated it a 5. This shows that the majority of users are generally satisfied, with many giving a high rating. The responses suggest that the

system meets most user expectations in terms of functionality, ease of use, and performance, while still having minor areas for improvement to achieve full satisfaction.

5.4 SUMMARY

In summary, the testing and evaluation phase of Vento: Voice-Assisted Inventory Management System was important in confirming that the system functions correctly, provides accurate results, and is easy to use. Functional testing showed that the main features, such as voice commands, manual inventory management, QR code scanning, and sales report viewing, worked as expected.

The results also confirmed that the system's role-based access works effectively. Staff, using the Android mobile app, were able to manage products by using voice commands or entering data manually. Admins, using the web dashboard, could manage inventory, handle staff records, perform checkout, and view sales data through charts. Staff had access only to product-level tasks, while admins were given full control of system operations.

Feedback from both staff and admin testers helped highlight what the system does well and where it can be improved. Overall, the system proved to be reliable and easy to use. It supports both admin and staff in performing their roles efficiently, while keeping their permissions clearly separated and secure.

CHAPTER 6: CONCLUSION AND FUTURE WORK

6.1 INTRODUCTION

This chapter summarizes the overall outcome of the Vento: Voice-Assisted Inventory Management System project and outlines potential future improvements. It reflects on the system's key achievements, challenges faced, and opportunities for further development.

The project aimed to create an inventory management system that is efficient, user-friendly, and supports both manual and voice-assisted operations. Two main user roles were established: admins and staff. The web-based admin panel allows admins to manage inventory, staff accounts, and view sales reports. The Android mobile app enables staff to perform inventory-related tasks such as adding, editing, or deleting products using either manual input or voice commands.

This chapter highlights the project's success in fulfilling its core objectives, including the implementation of voice-based CRUD operations, QR code integration, and real-time sales reporting. It also identifies limitations found during development and testing, and proposes future enhancements to improve performance, flexibility, and scalability.

By reviewing the development process and testing outcomes, this chapter provides a clear overview of the current system capabilities and emphasizes the importance of clearly defined roles and permissions in maintaining system security and usability.

6.2 ACHIEVEMENTS

This section summarizes the key achievements of the project, showing how each objective outlined at the start was successfully met through the design, development, and testing phases. The table below presents the project's goals and what was accomplished for each one.

Table 6.1 Table of Project's Objectives and Accomplishments

Objectives	**Achievements**
To design and develop a cross-platform system, including a web-based admin panel and an Android app for voice-assisted	A complete system was built with two main components: a Flutter-based Android app for retail staff, allowing them to manage

inventory management.	products through voice or manual input; and a Flutter web admin panel, enabling admins to control inventory and manage staff accounts.
To integrate Firebase Firestore and Authentication for real-time database operations and secure user management.	Firebase Firestore was used for real-time storage of product and staff data, while Firebase Authentication ensures secure login and access control for admins and staff. All updates are synced instantly across platforms.
To implement voice recognition and NLP capabilities to support hands-free inventory tasks such as adding, editing, deleting, and querying product details.	The Android app integrates speech_to_text and custom natural language handling to support voice commands. Staff can add new products, update stock or size, delete items, or ask questions like “How many size L for Nike?” using voice.
To enhance decision-making by providing visual sales reports with daily, weekly, and monthly trends for better inventory planning.	The admin panel includes a dedicated sales report page with both bar and line graph views. Reports can be filtered by time range, allowing managers to analyze sales trends and make informed stock decisions.
To support staff management functionalities including adding, editing, and deleting staff members by the admin.	Admins can manage staff records via the Staff Management module on the web dashboard. Features include adding new staff, editing account details, and deleting staff records stored in Firestore.
To validate and test the system to ensure accuracy, reliability, and usability in real-world retail environments.	Functional and usability testing were completed with real testers. Both the app and admin panel were tested for correctness, ease of use, and performance. Voice recognition accuracy, inventory updates, and graph display were successfully validated.

6.3 LIMITATIONS

Although the main goals of Vento: Voice-Assisted Inventory Management System were successfully achieved, there are still some limitations found during development and testing. These areas can be improved in future updates to make the system better and more reliable.

A) Limited Voice Recognition Accuracy in Noisy Environments

The voice assistant works well even in noisy retail environments, making it useful for hands-free inventory tasks. However, it currently supports only basic actions such as adding, editing, deleting, and viewing product details. It does not yet handle more advanced tasks like showing a QR code for a specific product or answering detailed questions about inventory. Expanding the voice assistant's capabilities would improve efficiency, especially for staff working in fast-paced retail settings.

B) Lack of Granular Staff Permissions

Currently, the system defines two roles: admin and staff. However, all staff accounts share the same level of access and capabilities. There is no role-based differentiation (e.g., view-only staff, restricted-edit staff). This could pose security risks and limits flexibility in managing access rights within larger teams.

C) No Offline Mode Support

The system depends entirely on an active internet connection since Firebase handles real-time data operations. Both admins and staff are unable to access, view, or modify inventory if offline, which may disrupt store operations in areas with poor connectivity.

D) Limited Sales Graph Customization

While admins can view sales data through daily, weekly, and monthly charts, there is no feature to select custom date ranges or export report data (e.g., PDF or CSV). This limits analytical flexibility and can impact decision-making for retail managers.

E) No In-App Notification System

The system lacks notifications for key events such as low stock alerts, successful updates, or login attempts. This affects both staff and admins, who may miss important updates unless they are actively checking the app or dashboard.

F) QR Code Features Limited to Viewing

Currently, scanning a QR code only shows product details. There is no option for staff or admins to trigger specific actions such as editing stock or initiating checkout via QR. Expanding this functionality could streamline inventory operations even further.

G) No Support for iOS Devices

The app has been successfully tested on a range of Android phones and tablets, showing smooth performance and proper layout. However, it is currently only available for Android and has not been developed for iOS devices like iPhones or iPads. This limits the app's availability to Android users only.

H) Public Admin Sign-up Reduces Security

Right now, admin users can sign up on their own. This can be risky because anyone could create an admin account and get access to sensitive features like sales reports, inventory, and staff data. It would be safer if only a trusted system owner or higher-level admin could create admin accounts.

6.4 FUTURE WORKS

To make the voice-assisted inventory management system even more effective and user-friendly, several improvements can be considered for future development. These suggestions address current limitations and aim to enhance the system for both admin and staff.

A) iOS Compatibility

Currently, the system is available only for Android. In the future, developing an iOS version of the app would expand its usability and allow more users to benefit from voice-assisted inventory management across different platforms.

B) Role-Based Access Control

At present, the system distinguishes between admin and staff roles. Future development could introduce more granular permission levels. For example, store managers could have full access to staff management and reports, while regular staff members might be restricted to viewing or updating inventory only. This would improve security and make the system more adaptable for larger retail teams.

C) Improved NLP for Voice Commands

The voice assistant currently supports basic commands for inventory tasks. Enhancing the natural language processing (NLP) capability will allow it to understand more conversational and flexible speech. This would help retail staff interact with the app more naturally, even in noisy environments.

D) Multi-language Support

Adding support for additional languages beyond English would make the system more inclusive. Retail staff in multilingual environments could interact with the app in their preferred language, reducing confusion and training time.

E) Expanded Sales Analytics

While the current system includes daily, weekly, and monthly sales graphs, future enhancements could allow filtering by product category, visualizing best-selling items, and exporting data. This would give admins deeper insights into sales trends for better decision-making.

F) Admin Activity Logs

Implementing an activity log for admin users would allow tracking of actions such as adding or editing products and managing staff records. This would enhance transparency, accountability, and operational oversight for store managers.

G) Enhanced QR Code Features

Although QR code generation is already implemented, future updates could introduce batch QR generation, customizable QR styles, and better integration with barcode or scanning hardware. These features would speed up processes like stock-taking and checkout, improving efficiency for both staff and admins.

By addressing these areas, the voice-assisted inventory management system can become even more efficient, scalable, and accessible. These future enhancements will help businesses better manage their inventory, save time, reduce errors, and ultimately improve customer service.

H) Restrict Admin Registration to Higher Authorities

In future updates, the system can be improved by disabling public admin sign-up. Instead, a trusted system owner or super admin should be responsible for creating admin accounts. This change would improve access control and ensure that only authorized personnel can manage sensitive data and operations.

6.5 SUMMARY

In conclusion, the development of Vento: Voice-Assisted Inventory Management System successfully achieved its main objectives. The system provides a functional and user-friendly solution for managing inventory through both manual and voice commands. Key features such as real-time data synchronization using Firebase, QR code generation, staff account management, and visual sales reporting have been implemented effectively.

While the system works well overall, several limitations were identified, such as limited support for multiple user roles, lack of iOS compatibility, and the need for smarter voice recognition and more advanced analytics. These issues do not hinder core functionality but present opportunities for future enhancement.

By addressing these areas in upcoming versions, the system can become more scalable, accessible, and intelligent. Continued user feedback and technical improvements will ensure that the system remains reliable and efficient, ultimately helping retail businesses streamline operations and improve customer service.

REFERENCES

1. Kembro, J., & Norrman, A. (2019). Exploring trends, implications, and challenges for logistics information systems in omni-channels: Swedish retailers' perception. *International Journal of Retail and Distribution Management,* 47(4), 384-411.
2. Muñoz Macas, C. V., Espinoza Aguirre, J. A., Arcentales-Carrión, R., & Peña, M. (2021). Inventory management for retail companies: A literature review and current trends. *In 2021 Second International Conference on Information Systems and Software Technologies (ICI2ST)* (p. 71-78). IEEE. https://doi.org/10.1109/ICI2ST51859.2021.00018
3. Zwakman, D. S., Pal, D., & Arpnikanondt, C. (2021). Usability evaluation of artificial intelligence-based voice assistants: The case of Amazon Alexa. *SN Computer Science, 2,* 28. https://doi.org/10.1007/s42979-020-00424-4
4. Madamidola, O. A., Daramola, O. A., Akintola, K. G. & Adeboje, O.T. (2024). *A review of existing inventory management systems.* Retrieved from: https://www.researchgate.net/profile/Olugbenga-Madamidola/publication/383947700_A_Review_of_Existing_Inventory_Management_Systems/links/66e34077f84dd1716ce9ccee/A-Review-of-Existing-Inventory-Management-Systems.pdf
5. McGuire. (2024). *Requirements analysis for software development*. Pulsion Technology. Retrieved from: https://www.pulsion.co.uk/blog/requirements-analysis-for-software-development/

APPENDICES

Appendix A: Project Schedule Gantt Chart

ID	Task Name	Start	End
1	Chapter 1: Introduction	2024-10-07	2024-11-21
2	Prepare Brief Proposal	2024-10-07	2024-10-20
3	Submission of Brief Proposal	2024-10-20	2024-10-20
4	Meeting and Discussion with Supervisor	2024-10-24	2024-10-24
5	Prepare Full Proposal	2024-10-25	2024-11-14
6	Meeting and Discussion with Supervisor	2024-11-11	2024-11-14
7	Submission of Full Proposal	2024-11-14	2024-11-14
8	Prepare Chapter 1 FYP	2024-11-15	2024-11-21
9	Meeting and Discussion with Supervisor	2024-11-18	2024-11-21
10	Submission of Chapter 1 FYP	2024-11-21	2024-11-21
11	Chapter 2: Literature Review	2024-11-22	2024-12-13
12	Prepare Chapter 2 FYP	2024-11-22	2024-12-13
13	Meeting and Discussion with Supervisor	2024-12-09	2024-12-13
14	Submission of Chapter 2 FYP	2024-12-13	2024-12-13
15	Chapter 3: Requirement Analysis and Design	2024-12-14	2025-01-05
16	Prepare Chapter 3 FYP	2024-12-14	2025-01-05
17	Meeting and Discussion with Supervisor	2025-01-05	2025-01-05
18	Submission of Chapter 3 FYP	2025-01-05	2025-01-05
22	Preparing FYP Development	2025-03-01	2025-06-23
19	Chapter 4: System Implementation	2025-04-01	2025-05-15
20	Prepare Chapter 4 FYP	2025-04-01	2025-05-15
23	Meeting and Discussion with Supervisor	2025-05-05	2025-05-09
21	Submission of Chapter 4 FYP Draft	2025-05-15	2025-05-15
24	Chapter 5: Testing and Evaluation	2025-05-15	2025-05-30
25	Prepare Chapter 5 FYP	2025-05-15	2025-05-30
26	Meeting and Discussion with Supervisor	2025-05-26	2025-05-30
27	Submission of Chapter 5 FYP Draft	2025-05-30	2025-05-30
28	Chapter 6: Conclusion and Future Work	2025-05-15	2025-05-30
29	Prepare Chapter 6 FYP	2025-05-15	2025-05-30
30	Meeting and Discussion with Supervisor	2025-05-26	2025-05-30
31	Submission of Chapter 6 FYP Draft	2025-05-30	2025-05-30
32	Prepare FYP Paper	2025-05-15	2025-06-23
33	Submission of Full FYP Report	2025-06-23	2025-06-23
34	Submission of Full FYP Paper	2025-06-23	2025-06-23
35	Submission of Full FYP Development	2025-06-23	2025-06-23

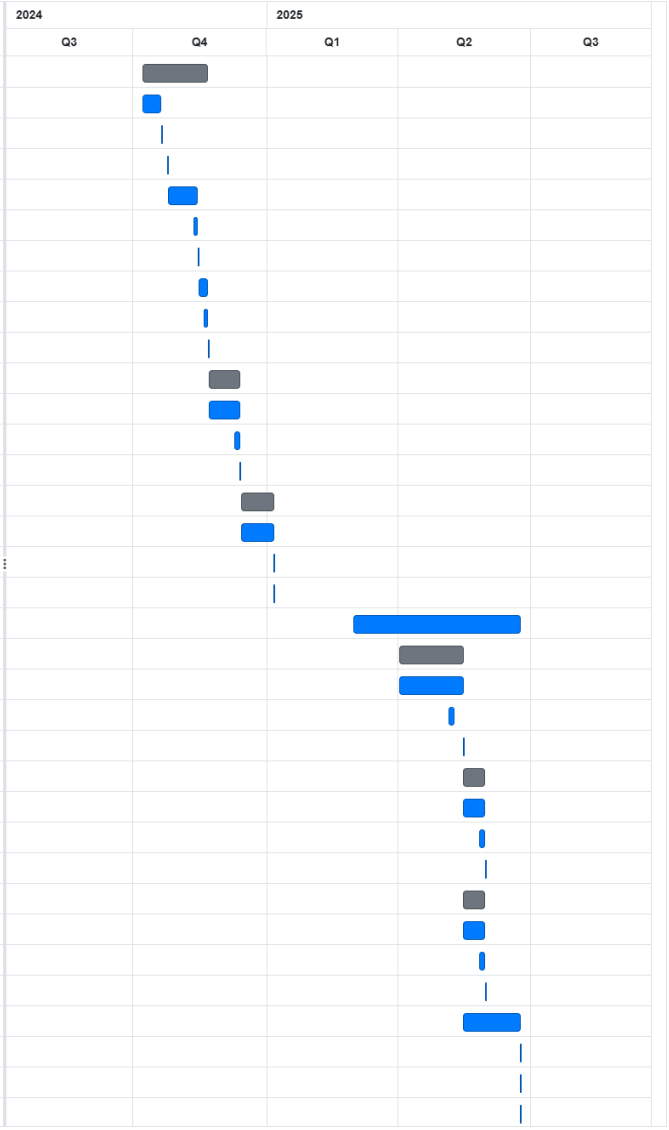

Figure A.1 Project Schedule Gantt Chart