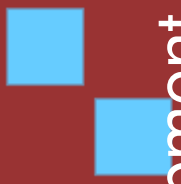


B.Sc. (Hons) in Software Development

Ollscoil Teicneolaíochta an Atlantaigh

Atlantic Technological University

StockWise – An Intelligent Inventory System

By
Ross Hannon

April 27, 2025

Minor Dissertation

Department of Computer Science & Applied Physics,
School of Science & Computing,
Atlantic Technological University (ATU), Galway.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This dissertation presents a detailed description of the development process from start to finish for an inventory management system called StockWise. It is built for small businesses that are looking for an easy way to keep track of their stock and enhance efficiency by reducing time consumption. While all the time keeping the traditional ERP system feel to StockWise, there is one enhanced feature that makes StockWise stand out, and that is the ability to forecast stock depletion through usage analytics. Real-world business logic was used to ensure that the system matched the usual flow of an inventory tracking system. The main framework used is React with Ionic as the front end and Firebase services as the back end, along with a dedicated FastAPI microservice which is integrated into the system to give machine learning features that provide a scalable app that pushes the boundaries with user interactions. There are six chapters in this dissertation, each giving a detailed and in-depth analysis of how StockWise was created. This introduction chapter will provide an overview of the main technologies, rationale and design thinking behind StockWise.

1.1 Dissertation Overview

- **Chapter 1 – Introduction:** Breakdown of the overall project and the rationale for why it was chosen.
- **Chapter 2 – Methodology:** What approach was taken for the development stage and what methodologies were used.
- **Chapter 3 – Technology Review:** Which technologies and frameworks were used and why.
- **Chapter 4 – System Design and Architecture:** How the system design and architecture perform.
- **Chapter 5 – Evaluation:** Assessing the achievement of the goals established previously.
- **Chapter 6 – Conclusion:** Reflects on the completion of the project and future plans.

1.2 Project Context: Challenges in Inventory Management

Inventory management in small businesses often falls short when it comes to having the right resources to use a high-end inventory management system; it often relies on doing a stock check manually. This can cause a substantial amount of time consumption and be very error prone. A survey found that “60% of small businesses have severe difficulty when it comes to high consumer demand and aligning with inventory storage."[1] StockWise aims to address these challenges by creating a low-cost, effective inventory management system which is tailored for small businesses like a local carpentry business. Important features such as detailed reports, stock categories, low stock alerts, user admins and bulk orders from suppliers. StockWise is built to make all these features seamless and offer a customisable app which is enhanced by its easy-to-use interface.

1.3 The Role of React and Ionic in Cross-Platform Development

Cross-platform development is a critical part of maintaining a compatible and responsive application that can be used across many different platforms, such as mobile and desktop. React and Ionic work together very well in scalability while allowing a user-friendly interface to be customisable. Ionic uses prebuilt components that can be used across all platforms, such as web, iOS and Android. “One disadvantage of applications built on Ionic is their low performance, especially for applications with heavy graphics.” [2] Ionic is seamless with the use of device features like camera and file storage, which is done with Capacitor. React Router is easy to use for a customisable interface across many pages while all the time keeping data consistent. React with Ionic is overall a successful combination for a robust inventory management app that can be easily run on different devices effortlessly. This has made time management a lot more reasonable, where greater focus can go on other development aspects.

1.4 Project Significance

StockWise holds a significance as many people struggled for years with a basic stock check. User feedback from local businesses shows that there is a need for an inventory management system app that reaches industry standards, with easy-to-use stock checking, restocking and detailed reports for decision-making.

1.5 Project Objectives

The project objectives for this inventory management system are to have a robust stock management system to handle a huge number of different stocks, provide detailed stock reports, handle orders to suppliers efficiently and effortlessly, have

different user roles for data security and, with the use of machine learning, be able to get an estimated restock weekly through data gathered over time.

1.5.1 Project Success Metrics

- **Implementing CRUD functionality for inventory items, suppliers, and orders:** CRUD is a necessity for StockWise with having to deal with many different stock items. Firestore, being a powerful backend, will handle the database, storing many different collections connected to inventory, suppliers, and orders pages.
- **Google Maps API, Sinch API, Barcode Scanner API:** Having extra functionality in StockWise was always going to be a significant enhancement. These 3 APIs bring a whole new professionalism to the app by minimising the need for users to perform tasks outside the app. Google Maps provides a quick way to find nearby suppliers when needed, Sinch is for low-stock alert messages for efficient stock management, and Barcode Scanner makes scanning in new items effortless.
- **User-Friendly UI:** The UI must be easy to use for user dynamics such as those with limited digital literacy or older age demographics who would not be up to date with computers. It must be visually appeasing and keep the same look throughout. A professional look mixed with the user-in-mind's preference is the main goal. Easy to navigate if the user manual is followed correctly.
- **Admin Roles:** To achieve a security level for an app that is fit for the market, an admin role had to be considered. With one admin overseeing all employees, there is no way for a new user to take over admin permissions. Admin has access to all the application features, while employees have access to only the CRUD features.
- **Machine learning for predictive restocking:** Machine learning is going to help see trends and usage patterns in customer items on a weekly, monthly, and yearly basis. There will be features such as predicting when a stock is going to run out and how much that stock needs to be updated for future needs. The feature of providing data for consumption patterns is going to greatly benefit a company that can keep track of their stock with minimal effort required. Findings from "A Smart Choice For Efficiency And Process Automation" [3] insisted that small datasets can still give positive insights through regression-based models, which is beneficial as data can be used quickly when added to StockWise.

1.6 Project Scope

StockWise offers a user-friendly UI which is designed for user experience. Sinch API will offer an out-of-work handle on the management of the stock, where the barcode scanner will be nothing but time efficient when it comes to updating the inventory quickly. The blend of machine learning for automated restocks and complex CRUD functionality makes this app cost-effective and improves customer satisfaction.

1.6.1 Relevance

StockWise addresses a new change in the old and outdated inventory management systems that small businesses use. Many small businesses use a manual stock check with storing hardcopy files of each stock check. This is a hindrance in time and is very error prone. AI machine learning is not very prevalent in any of the inventory management systems on the market at the moment, so there is an opportunity to incorporate it into the restocking section where data is significant in the decision-making of what stock needs to be ordered when low.

1.7 Project Management

Project management was a vital part in the development of StockWise, where skills acquired through college and internship experience were applied. Jira is used for user stories and to keep track of the timeline from start to finish. The use of sprints was a significant benefit, along with an agile approach to the development process. GitHub as a version control to keep a history log of commits and a backup for the project was crucial with such a broad project.

1.7.1 Jira and Scrum Methodology

- **Jira:** Jira was used as the main management platform, as past experiences made the process easier where it was greatly beneficial, and it is used in most software companies, as it is a great management tool for progress tracking, managing tasks and user stories. A big feature of Jira is Kanban boards and Gantt charts, which are great for visualising the future of the project and how easy it is to manage user stories throughout the development of StockWise. Keeping track of each sprint was simple with the use of labelling each user story to a sprint. The many organisational tools that Jira offers make it a fantastic management system.

- **Scrum Methodology:** Scrum methodology was used in this project, as it offers a well-defined workflow using sprints where the objectives are clear at the end of each sprint. The project included three sprints:

- **Scrum Sprints:**

  - Sprint 1: Structural Logic (Setting up the system foundation)
  - Sprint 2: Data-Handling Logic (Expanding inventory features and relationships)
  - Sprint 3: Predictive & Automation Logic (Machine learning and process automation)

Type	Key	Summary	Status	Assignee	Priority	Labels
	IMF-13	Fix react router version to get useNaviagate working	IN PROGRESS	Ross Hannon	=	Sprint1
	IMF-11	Add component for adding items to firestore database	IN PROGRESS	Ross Hannon	=	Sprint1
	IMF-7	Add user login functionality with firebase authentication	IN PROGRESS	Ross Hannon	=	Sprint1
	IMF-14	Login, Signup and Add item displaying on console when suc...	IN PROGRESS	Ross Hannon	=	Sprint1
	IMF-8	Add user signup functionality with Firebase authentication	IN PROGRESS	Ross Hannon	=	Sprint1
	IMF-6	Firebase Authentication	IN PROGRESS	Ross Hannon	=	Sprint1
	IMF-12	Add Firestore error handling for adding items	IN PROGRESS	Ross Hannon	=	Sprint1
	IMF-9	Add protected route to restrict access to home page based o...	IN PROGRESS	Ross Hannon	=	Sprint1
	IMF-10	Add Auth.css to login and signup page	IN PROGRESS	Ross Hannon	=	Sprint1

Figure 1.1: Sprint 1 – Structural Logic Tasks in Jira

Type	Key	Summary	Status	Assignee	Priority	Labels
	IMF-15	Implement item quantity deduction when stock is used	IN PROGRESS	Ross Hannon	=	Sprint2
	IMF-16	Add categories and subcategories for inventory items	IN PROGRESS	Ross Hannon	=	Sprint2
	IMF-17	Improve inventory filtering and search functionality	IN PROGRESS	Ross Hannon	=	Sprint2
	IMF-18	Implement a Suppliers section in the app	IN PROGRESS	Ross Hannon	=	Sprint2
	IMF-19	Integrate supplier information with the orders page	IN PROGRESS	Ross Hannon	=	Sprint2
	IMF-20	Implement an order creation system	IN PROGRESS	Ross Hannon	=	Sprint2
	IMF-21	Improve Firebase authentication session handling	IN PROGRESS	Ross Hannon	=	Sprint2
	IMF-22	Implement user roles (admin and standard user)	IN PROGRESS	Ross Hannon	=	Sprint2
	IMF-23	Improve Firestore security rules	IN PROGRESS	Ross Hannon	=	Sprint2
	IMF-24	Design and implement a more detailed inventory reports page	IN PROGRESS	Ross Hannon	=	Sprint2
	IMF-25	Improve mobile responsiveness for key pages	IN PROGRESS	Ross Hannon	=	Sprint2

Figure 1.2: Sprint 2 – Data-Handling Logic Tasks in Jira

Type	Key	Summary	Status	Assignee	Priority	Labels
	IMF-26	Implement machine learning model for restocking sugg...	IN PROGRESS	Ross Hannon	=	Sprint3
	IMF-27	Integrate AI model with the restock page	IN PROGRESS	Ross Hannon	=	Sprint3
	IMF-28	Implement Email API for low-stock notifications	IN PROGRESS	Ross Hannon	=	Sprint3
	IMF-29	Implement Firebase Cloud Functions for automated em...	IN PROGRESS	Ross Hannon	=	Sprint3
	IMF-30	Enhance the home page with AI-powered insights	IN PROGRESS	Ross Hannon	=	Sprint3
	IMF-31	Implement supplier reliability tracking	IN PROGRESS	Ross Hannon	=	Sprint3
	IMF-32	Optimize Firebase Firestore queries to reduce read costs	IN PROGRESS	Ross Hannon	=	Sprint3
	IMF-33	Implement Google Maps API for supplier location tracki...	IN PROGRESS	Ross Hannon	=	Sprint3
	IMF-35	Add Inventory Adjustment Logging for Audit Trail	IN PROGRESS	Ross Hannon	=	Sprint3
	IMF-36	Implement Role-Based Dashboard Customization	IN PROGRESS	Ross Hannon	=	Sprint3

Figure 1.3: Sprint 3 – Predictive & Automation Logic Tasks in Jira

1.7.2 Software Testing Methodologies

StockWise is a very heavy data-driven project therefore, many different test strategies were used.

- **Integration Testing:** Postman was used for supplier info and order docket creations to make sure the right information was connected to each other.

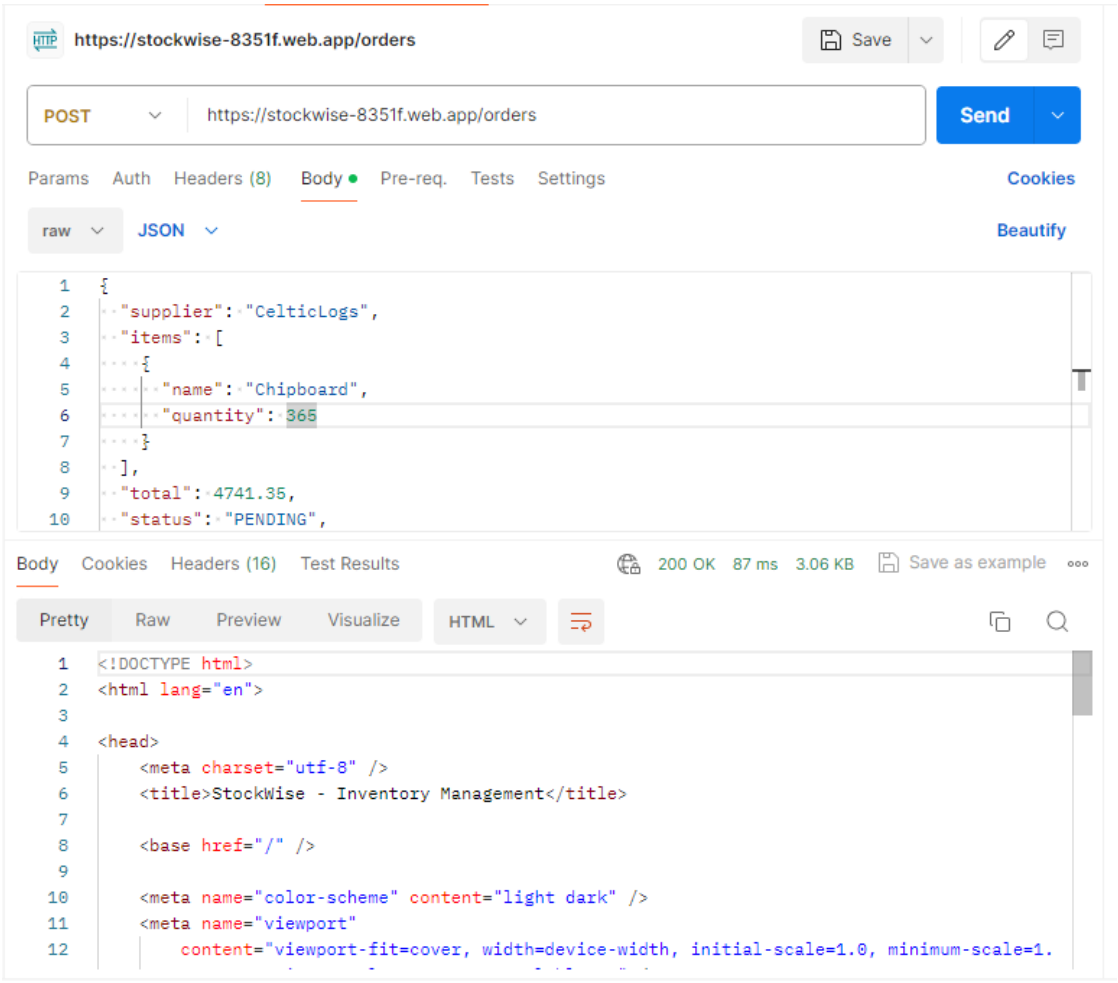


Figure 1.4: Postman test of the Order Creation API. The request sends a new order to the backend and receives a successful response.

- **Functional Testing:** Manual tests such as adding, updating, and deleting stock were done for the functional tests to make sure the system was doing the basic scenarios each time.
- **User Experience Testing:** Google Forms was used to create a questionnaire as a working prototype was sent to small local businesses. This questionnaire gathered valuable insight into necessary changes that needed to be made to StockWise.

1.8 Custom UI Components and Ionic Framework

StockWise was designed to have a user-centric flow to all the pages; therefore, the use of custom UI components was crucial to make the most of the extensive customisation. The restock suggestion section with ML data for each restock item, Chart.js using many different charts in the reports page to show the different trends, and how the stock items were displayed on the inventory page as horizontal and when edited a modal was used. These features are all important to showcase how customisable you can be with UI components.

1.9 Project GitHub Repository

StockWise and all relevant documentation related to the project were stored on GitHub and are accessible through this link: `https://github.com/rosshannon7677/Stockwise.git`

1.9.1 Frontend Code

The frontend code is structured in the `src/` directory, which contains all the React and Ionic components. This includes:

- `components/`: Reusable UI components such as modals, buttons, and data tables.
- `pages/`: Main application views including Inventory, Restock, Reports, Orders, and Suppliers.
- `services/`: Logic for Firebase interaction and API communication.
- `hooks/`: Custom React hooks for authentication and data handling.
- `contexts/`: Global state management using React Context API.
- `types/`: TypeScript interfaces for data consistency and safety.

Additional utilities include:

- `ProtectedRoute.tsx`: Route protection for pages requiring authentication.
- `App.tsx` and `main.tsx`: Entry points for the app initialisation and component rendering.

1.9.2 Backend Logic

The backend is developed using `FastAPI` and located in the `ml_service/` directory. It is responsible for:

- Receiving inventory data and usage history from Firestore.
- Running a trained regression model using `scikit-learn` to:
  - Predict days until stock depletion,
  - Calculate recommended restock quantities,
  - Assign a confidence score for each prediction.
- Returning predictions through RESTful JSON endpoints.

Deployment: The backend is containerised and deployed on `Google Cloud Run` for scalable, serverless execution. It supports automatic OpenAPI documentation via Swagger UI.

Additional Features:

- Chart generation via `Matplotlib`, encoded as base64 for inline image delivery.
- Stateless REST API for compatibility with the frontend.
- All backend logic is contained in `api.py`.

1.10 Introduction Conclusion

In this dissertation there will be an in-depth look into the inventory management system StockWise, providing insight into taking it from an idea to a fully developed, trialled and tested working app. It is backboned using React and Ionic along with Firebase for user authentication and Firestore for database management and finally using a trained machine learning model for predictive stock shortage. In these chapters there will be a look at what methodologies were used along with sprint planning and the different management technologies which enhanced the tracking and time management of the project. There will be a chapter on which technology was used and system design. The overview of all these chapters should give a detailed summary of how StockWise was created and the approach that was taken to bring it to where it is today.

Chapter 2

Project Methodologies

Given the scale of the project, there had to be a well-thought-out project plan and a decision on which methodologies would best suit the development of StockWise. Agile was chosen, as it is the most common methodology out there and is best suited for a large project like StockWise. What makes Agile so well trusted? The 2020 Scrum Guide says that its scrum methodologies with iterative progress help ensure that consistency is delivered even when the requirements change. [4] (Schwaber and Sutherland, 2020)

Many different platforms for research were used, such as Google Scholar for academic papers on inventory management, ScienceDirect for how ML can be used for efficiency, Firebase Docs for how Firestore is structured, TensorFlow Docs for ML models and GitHub for inventory management projects.

Agile development helped plan out a detailed guide which was divided into three sprint phases. UI development, data handling and machine learning. Having set out these phases, there was a clear path to be taken to improve the project as it goes on. The use of Jira as the main management tool was pivotal when setting user stories. Why Jira? The documentation on Atlassian shows how Jira supports agile with kanban boards, sprint planning and backlogs, which were all very important in the development of StockWise. [5] (Atlassian, 2024) There is also room for flexibility in Agile, as it is one of the main principles which was beneficial when there was a busy time around Christmas with other projects and exams.

2.1 Agile Methodology: Flexibility in Development

The development was very easily navigated, as short sprints are the main approach when it comes to writing the code in phases. Instead of writing the code all at once and not in order or phases, it can easily be misguided, and you may encounter unforeseen issues on the way that you did not plan on. Sprints are the most important part of Agile Development, as they show steady progress along the way. This approach encourages responsibility, and feedback is provided after every sprint. [6] (Teamhood, 2025). In each sprint there was a focus on one particular part of the project, and a number of issues and user stories were designed for that sprint.

Scrums alongside Jira to show a visual representation of the workflow from where the project development will start and finish. Gantt charts provide the visual aspect where you can have a start and end date for each task, which can be colour-

coded to show the difference in the tasks. How do Kanban boards help people using Jira? The Atlassian forums explain that “Kanban boards in Jira provide a visual management tool that helps teams track work as it progresses through various stages, enhancing workflow efficiency and transparency" [7] (Atlassian Community, 2025). Kanban boards are divided up into Backlog, In progress and Done. Here is where you deal with your Jira issues, from setting the date, assigning a worker, and adding labels to setting a priority for how important the issue is. The Kanban board handles each issue in a clear and concise way for how each issue is being addressed. These Jira tools offer well-defined charts that keep StockWise development continuous and handle the workload well.

2.1.1 Tools Supporting Agile

- GitHub for Version Control: Tracking commits and managing code changes made.
- Kanban Boards: Shows sprint progress and development status with visual graphs.
- Gantt Charts: Tracking milestones that are scheduled with precise dates.

These tools ensure that there is steady progress made through the scrum cycles all the time, allowing changes that were not planned for due to user testing and the evolution of the project requirements.

2.2 Research Methodology: Grounding Decisions in Data

When doing research for inventory management systems for small businesses, the main goal was to pick out common problems that small to medium-sized businesses had with using inventory management systems.

This research found that there was a major gap in systems that were tailored for small businesses that could not afford the costs of setting up the system and having to pay a monthly subscription to keep it. The main platforms used were Fishbowl Inventory and Zoho Inventory, which were extremely popular but were often highly sophisticated for a small business that could afford an IT worker to look after the system. These systems also had an onboarding process which could take some time along with training for the complexity of the system.

2.2.1 Affordability & Accessibility:

The cost to run a system for the main platforms out there exceeds 100 a month for a subscription, and along with the signup fee, this is a significant amount of money spent for a small business. The use of Firebase as the backend is one of the main reasons StockWise is going to have reduced costs, as it has no hosting costs.

2.2.2 UI Simplicity:

There are so many applications out there that the complexity is overwhelming, especially for small business owners who have little to no experience in IT. The workflow would need to flow seamlessly in StockWise for people with little IT knowledge to be able to work on their own.

2.2.3 Predictive Analytics for Restocking:

The main way machine learning is used in inventory applications is for demand forecasting, which predicts the future needs for the company by using its past data. In StockWise this same approach is used but in a more immediate way, where the user could leverage very little data to predict the future needs for a product and be given a confidence score of how certain this is.

2.3 Planning and Requirement Gathering

StockWise was never the first idea that came to mind. Questions had to be asked, and use past experience that was gathered through an internship at SAP where there was so much to learn about automated systems and small business management. Through many brainstorming sessions with old colleagues and the supervisor's advice, StockWise was created.

2.3.1 Brainstorming and Ideation

Several different ideas were considered when brainstorming, such as a web scraping app which scraped data from a website and repurposed that data in a new format and also a cancer prediction app that uses data mining for accurate predictions. These ideas were thoroughly discussed with many different people, and diagrams were done to gain a better understanding of how they would look.

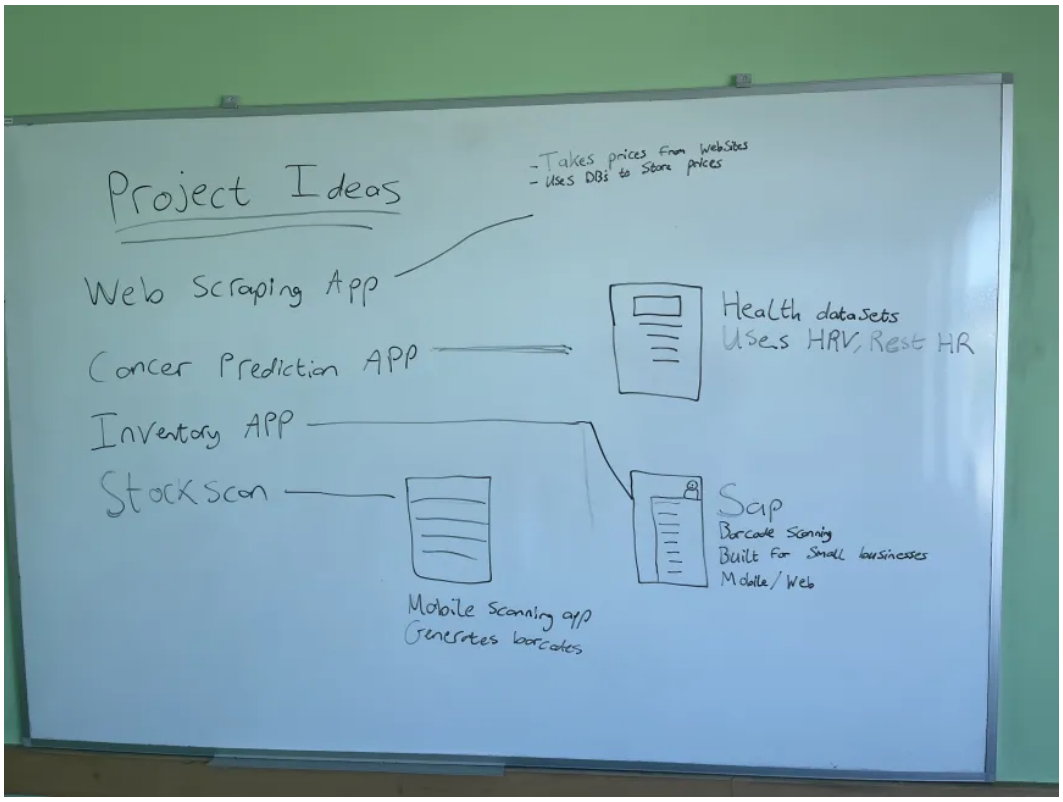


Figure 2.1: Project Ideas Diagram

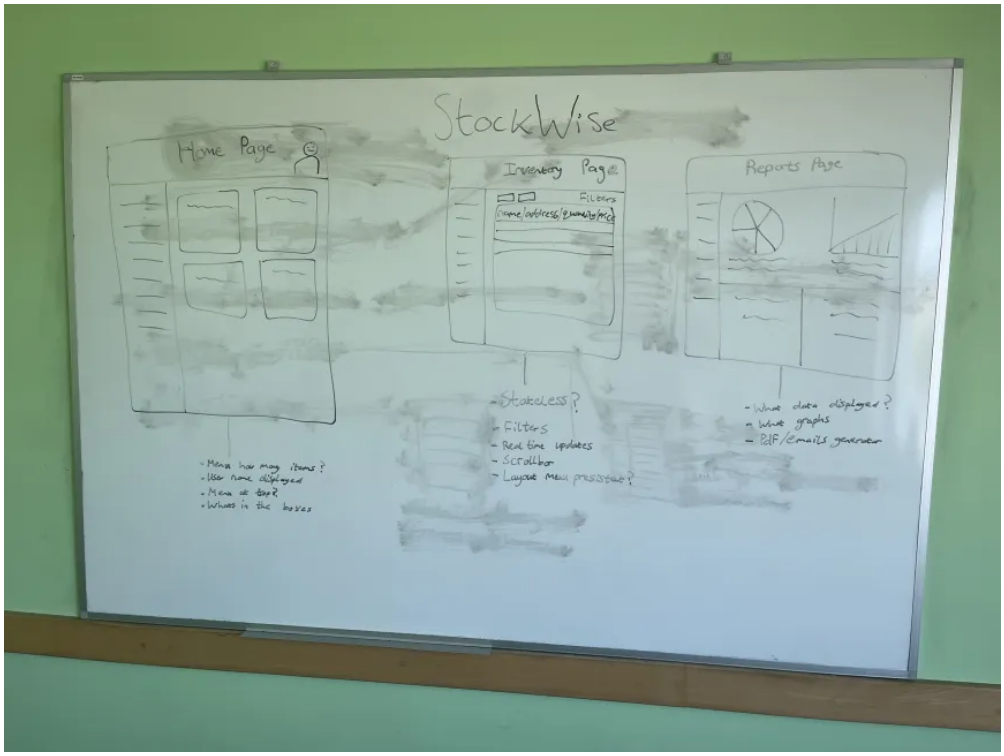


Figure 2.2: Storyboarding Wireframes for Home, Inventory and Reports Pages

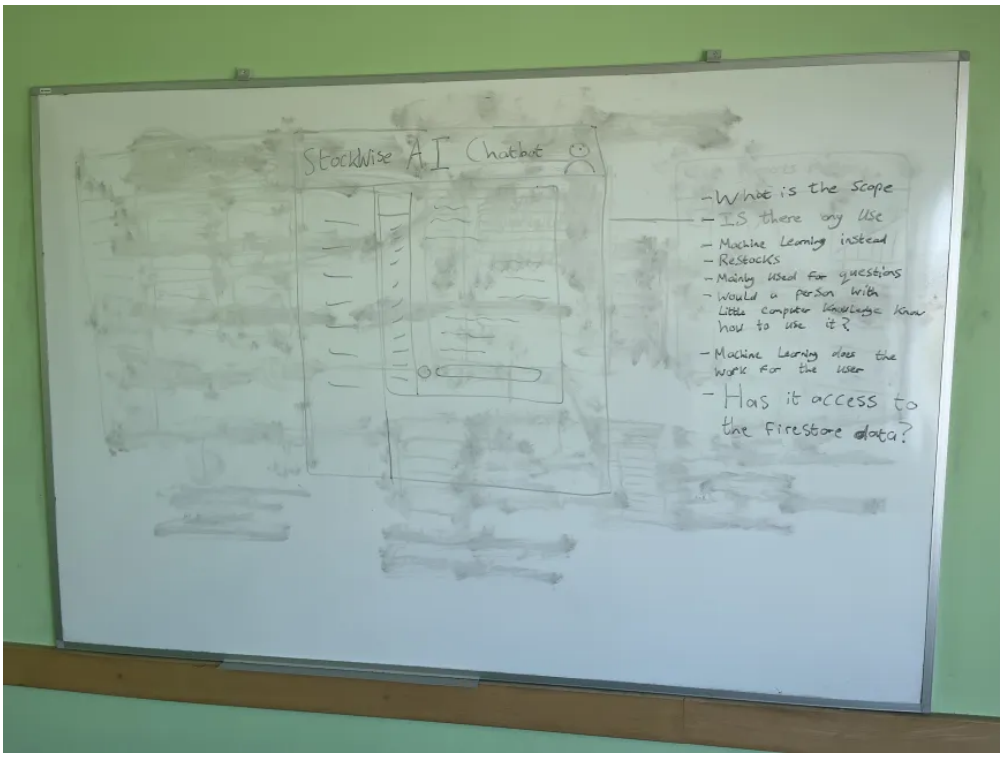


Figure 2.3: AI Chatbot Feature Brainstorming Session

In the end, the decision was to go with an inventory management system to showcase what was learnt in SAP with small business management systems.

2.3.2 Storyboarding and Wireframing

Wireframes and sketches were done for inventory items and restocking items. This made a clear picture of how the UI was going to look from an early standpoint.

The whiteboard session visualised how the main pages would interact with one another. There was a substantial amount of work put into how the add items/orders/suppliers would look, as it would be a reusable component across those three pages.

Storyboards were used to bring user stories to real life, such as what happens if a low stock alert happens, how a PDF would look if generated from the reports page and how the menu layout would be designed. This was a major help with both frontend and backend as problems were picked out at an early stage.

2.4 Technology Selection

The next step was to pick the technology stack to develop StockWise. To achieve a cross-platform, robust system that utilised machine learning for predictive analysis, the correct framework that would deliver on all fronts had to be chosen. Through the evaluation of trial and error, the best technology stack that best suited a system that was easy to use and could be very scalable for multiple functionalities.

2.4.1 Frontend Framework: React with Ionic

When doing research on which is the best frontend framework to use, React with Ionic was the clear winner. React offers an extensive component-based infrastructure, and its reusability is especially important with a large system where there are many different pages using the same components.

2.4.2 Backend and Database: Firebase with Firestore

For StockWise to be reliable, there needs to be a powerful database that would handle many large complex queries to handle real-time inventory system needs. Firebase was top of the list with its ease of use and how it offers a seamless architecture that can be scalable for many inventory items. Firestore was used for the database to pull inventory items, orders, and supplier information. Firebase also hosted the users which handles the authentication for each user when logging in.

2.4.3 Machine Learning: Python (scikit-learn)

Most systems rely on manually tracking stock to estimate how long until stock will be depleted. With the help of machine learning using Python-based FastAPI, it was possible to predict when stock would fall below a certain number and how many days it would take that stock to be fully depleted. This feature would be crucial for StockWise to help the user stay ahead of their business needs while handling other requirements.

2.4.4 Extra Technologies Used

- jsPDF: was used for generating reports on supplier and order information.
- Chart.js: for visualising what data has been collected and displaying it in various pie and bar charts on the reports page.
- Vite: was used for automatically detecting changes that were made to the code and only updating that file instead of doing a full reload on the project.

2.5 Weekly Supervisor Meetings

A weekly meeting with the project supervisor which lasted 30 minutes where the discussion was StockWise's development. In these meetings there was continuous oversight on what happened since last week's meeting, what challenges were encountered and what was planned to get done for next week's meeting. The format of the meetings was as follows:

1. Showcasing the latest work that had been done on StockWise (machine learning, UI improvement).
2. What challenges were encountered, and what way was gone about solving them?

3. What was planned to get done for next week's meeting.

The main focus of these meetings was around the business logic, as there could not be features implemented that did not match the requirements and user stories that needed to be completed. Instead, there had to be real-world scenarios that would be seen on an inventory management system. There was also an adjustment made to the sprint roadmap when the development of the machine learning took longer than expected with the training of the data needed for accurate readings.

2.5.1 Project Milestone Demonstrations

There were certain milestone demonstrations that were key moments in this development journey. Key features such as:

- Training the machine learning model to show accurate predictions for how many days until stock would run out by using the data gathered from stock usage.
- Google Maps API showing how long of a distance it is from the workshop to many stores around the country.
- Different users for access to the overall features of StockWise: Admin had full access, while manager and employee only had selected access.

These weekly meetings were also beneficial, as they were treated as a user testing scenario as the supervisor tested out the new features and how they reacted when data was entered. This helped with business logic flaws that may not have been seen during the sprint process when being so caught up in the development of the project and not looking at it from an outside perspective.

2.5.2 Project Objectives and Roadmap

StockWise's development was forecasted by using scrum methodology and by using Jira's Kanban boards to clearly see the tasks being outlined and the progress made to date. Scrum methodology is centred around having a strict sprint process, but there had to be some leeway with the last sprint, as the machine learning predictions took longer than expected, and therefore the sprint had to be pushed out by a week.

Sprints had to be fully completed for any working demonstration to be shown in the weekly meetings. Therefore, every task of a sprint had to be completed even when going over the time frame set out for that sprint.

In the final weeks of StockWise's development, the main focus of the weekly meetings was checking to see if the accuracy of the machine learning predictions was correct. This was achieved by performing calculations on the daily usage, predicted days until low stock, confidence score calculation and recommended restock date.

2.6 Testing and Validation

To ensure that StockWise would deliver on all aspects, there needed to be a lot of testing that would take place throughout the development of StockWise so that the business logic would stay consistent and the machine learning model's data was accurate. A user testing approach on most testing formats was used as business logic was prioritised to be the most important part of StockWise.

2.6.1 Testing Strategy

Unit testing was used to verify the functionalities of StockWise. For machine learning, there had to be a consistent check that the data was up-to-date and coordinated with the predictions that the model was making. Login authentication was a big security functionality that had to be tied down early, where many protocols were assessed to ensure authorised users only could log in. To keep the stock level consistent, there was a check that the inventory would update correctly when stock would be used.

Business logic testing made sure that all features in StockWise would be used in a real-world inventory management system and that the features would be aligned with one another and not be incorporated for no reason. This was tested by checking that the components would go in order, such as order -> inventory -> restock -> order.

2.6.2 Machine Learning Predictions

The data gathered from the stock used was then formatted to predict when a stock shortage would happen. The three main tests that were run for this were:

1. Comparing stock depletion data to real-world inventory trends to see if the accuracy of the machine learning predictions was correct.
2. Testing different formulas to predict how many days until stock would be depleted so that the most accurate readings would be gathered.
3. Training the ML model to gather enough data on an item to see how precise the confidence score would become.

These tests encountered minor mistakes that were made, such as how the daily usage of items was used and how the predictive stock depletion was using days between the last time of usage to predict the future stock depletion.

2.6.3 User Testing with Real Businesses

For StockWise to be tested fully, local businesses were asked to user test the finished product so there would be no errors when it was fully completed. The two businesses were Hannon's Kitchens in Caltra and Hughes SuperValu in Claregalway. This user testing trial lasted a total of one week, where StockWise was used as the main inventory management system, and they tested its machine learning capabilities, stock tracking features, report features and user authentication.

Once the user testing was completed, there was a questionnaire filled out by both businesses which captured the feedback of the main criteria tested. The five main sections looked at were:

- How easy StockWise was to navigate for people with high computer expertise compared to people with little to no experience with computers.
- The accuracy of stocktaking throughout the system, and was stock updated correctly when restocking was complete.
- Machine learning for future stock depletion with confidence scoring and estimating restock suggestions.
- The use of order tracking to navigate orders from being ordered to restocked.
- Business user authentication for having users accepted to StockWise by the admin.

2.6.4 Summary of Feedback

The feedback gathered from the questionnaire showed a common theme:

1. “Very easy” to navigate was chosen by both businesses, who found the fluidity of StockWise to be very well documented. The users with little computer knowledge found that they were able to create a new account, add stock, create orders, generate PDFs, and understand the restock suggestions very easily.
2. The machine learning predictions were found to be useful for both businesses, where they both relied on a weekly manual stock count to predict future stock shortages, whereas with StockWise, they found this feature to save them time and hassle. There was a comment on a more in-depth investigation of each item to see which user updated said item.
3. Inventory tracking got a positive response where depletion trends could be looked at on a linear regression chart. It was pointed out that an item with irregular usage patterns was harder to read, with high spikes shown on the linear regression chart.
4. Generated emails for stock ordering were appreciated, as users found it to save a lot of time and also convenient not having to leave the application to generate an email.
5. The performance of StockWise was commented as being “instant” or ”fast” throughout. The database updated automatically without having to refresh the page after adding or updating an item, which ensured a very responsive system.

2.6.5 Features Added or Improved Based on Feedback

With the feedback gathered, there were multiple changes implemented to fix some of the concerns made by the users of StockWise.

- **Usage History Logging:** There was a concern with not being able to track who made an update to a stock item. There was a usage history feature created which is displayed on the homepage. This logs a message after every time an item is used, updated, or added. The user who made this change was displayed along with a timestamp on the message. All of this data is then fed into the machine learning for better predictions.
- **Role-Based Interface Improvements:** There was another concern with a clearly defined role between admin, manager, and employee. As a result of this feedback, StockWise now has distinct roles for admin with full access, manager with access to stock updates with little admin privileges, and employee with read, write, and edit to inventory and orders. This provides a more secure network within StockWise where admin has permissions to remove managers and employees.

2.6.6 Final Outcome

From this user testing phase, there were many fundamentals learnt with basic user requirements that may have gone unnoticed within the development phase but were spotted from an outside source. These additional features that were not in the planning phase and now greatly add a new dimension to StockWise for users, who can now track each item individually and its past history from the home page. The feedback was positive throughout, with multiple comments on the structure and machine learning feature for stock checks.

2.6.7 Error Handling

There were many simple error validations that had to be done in StockWise, such as making sure values never go below 0, that orders would not be placed for items unless it was approved by the owner and for days until low was calculated by using the machine learning data and not a fixed hardcoded data.

2.6.8 Challenges Encountered

This testing did bring up many challenges, such as adding stock manually when the order would have been delivered, stock usage not being calculated correctly, which then would lead to inaccurate days until low predictions, and automated testing not being used, so all errors had to be found through manually testing.

2.6.9 Areas of Improvement

If StockWise were to be started again, the testing methods that would be recommended would use automated testing to speed up the process for finding errors,

use more data for the machine learning to see how the ML model handles varying amounts of data for predictions, and have users test certain parts of StockWise to see if errors can be found.

2.7 Development Tools and Collaboration

What development tools that were used in this project were a major factor in getting the most effective workflow that would keep all sections of developing StockWise well structured? Through the use of GitHub and VS Code is where all of the coding and version control took place, and the project management features were kept track of with the use of GitHub issues and Jira. Firebase was also the main database which kept track of data syncing and handled authentication for all users.

2.7.1 GitHub and Git for Version Control

GitHub is the leading version control platform for creating, storing, and managing code and therefore was a first-choice pick when deciding which platform to use. GitHub is well-structured with an easy-to-navigate UI and is a safe and secure network to store your projects on. The use of GitHub issues was a plus where similar tasks could be created, such as on Jira, but were used in a more precise way.

The consistency in how each commit is formatted with time and date really helped with keeping track of how much work was done in each week of the development of StockWise. Following a format of detailed commit messages for each commit was beneficial when having to go back and see where a certain change was made to the code. An example of some commits:

- Order status added with distinct colourways for clarity
- Address for suppliers restricted to Ireland, layout made smaller for sign out button to appear

With doing a project on your own and not having to collaborate with another person, thankfully there would be no issues with merge conflicts. There were some conflicts due to some functionality sharing the same components, which would lead to debugging or reversing a recent commit.

2.7.2 Visual Studio Code as the Source Code Editor

Visual Studio Code is one of the leading development platforms out there to date, as it offers many features, such as highlighting code for uniqueness, many extensions, and many optional debugging tests. It also integrated many different languages in StockWise, such as Python, React and Ionic. The use of having GitHub source control built into VS was such an important feature which saved a lot of time and made the process of committing easier.

2.7.3 Project Management with Jira and GitHub Issues

Jira and GitHub issues were a massive help with setting out the goals of each sprint and having future goals to achieve for StockWise. Jira was mainly used for user stories where there could be a breakdown of each user story into tasks. Kanban boards were also a huge help when it came to seeing the development process. GitHub issues were used for small tasks of details that had to be fixed, such as UI design and tracking of inventory.

Chapter 3

Technology Review

In the Technology Review section, there will be an overview of the various technologies used that built the backend and frontend of StockWise, outlining how the architecture is structured and what frameworks were chosen to best help a large scalable project. Highlights deviations from the original proposal while in development and what roadblocks were encountered and overcome.

StockWise was built on the purpose of keeping to its core values, which is not having a complex system that non-technical users could not manage. Therefore, when choosing the correct technology stack, there is a big emphasis put on accessibility, scalability, and predictive functionality. These technical decisions were chosen to support complex and advanced features such as secure role-based access and low stock forecasting while all the time keeping a simple, user-friendly front end.

3.1 Evolution of Inventory Management Systems

The evolution of inventory management systems has been significant from when manual tracking was the primary method to track stock. In today's case, businesses mainly use systems that can handle multiuser capabilities and are scalable to help with efficiency. With StockWise, it has all of these features which can be user-friendly and available to small businesses.

Older methods of inventory tracking were done through static databases or large paper files. This was very prone to errors and time-consuming, and many times led to the stock count being incorrect. The help of cloud-based inventory management systems can now host large sets of data which can be updated in real time and have multiple users with separate roles.

A major decision to change from MongoDB to Firebase Firestore was made when the requirements for a more serverless database were decided.

3.1.1 Real-Time Inventory Tracking & Predictive Insights

Most inventory systems prioritise real-time updates to the database to make sure the stock is accurate, which provides an up-to-date stocktake for accurate readings.

The machine learning aspect of StockWise incorporates a whole new aspect for stock checking where the use of data which is gathered whenever a stock item

is used or updated is then forecasting a future depletion for that stock to run out. FastAPI is used for this predictive stock feature and helps with stock shortages, restock recommendations and the use of confidence scores with how much data has been gathered.

Chart.js helps with displaying how much stock is in each category and the value of each category.

3.1.2 Architectural Decisions & Excluded Features

To fully embrace the machine learning aspect of StockWise, there had to be some technology decisions made to benefit the overall development and business needs to help with the user requirements. These included:

- Not going through with barcode scanning, as the focus on having StockWise deployed on Android/iOS was turned to having it fully deployed in a web-based environment.
- CI/ID was considered and tested, but as it was late on in the development of StockWise, the decision to go with manual production to the localhost and deploy to Firebase afterwards was chosen.

3.2 Key Technologies in StockWise

The key technologies that are used in the development of StockWise were carefully chosen to build a technology stack that is unbreakable, scalable, and editable to ensure data synchronisation. The main technologies used are React with Ionic, Firebase, Firestore, FastAPI and Google Cloud services. These technologies were chosen as they provide an easy-to-use infrastructure designed to captivate a user-friendly interface that will host a secure backend, which is the main goal of StockWise.

3.2.1 React and Ionic for Frontend Development

React: React is fast becoming one of the leading JavaScript libraries with its component-based architectures, which were first created by Facebook, now known as Meta, back in 2013. The reason it is so popular is because of its virtual Document Object Model, which is a big-time saver. This is when a component is updated in the UI code and is directly updated on the UI straight away without having to do a full page reload, which helps with application responsiveness. React's performance and flexibility allow for an extensive UI where there can be many inventory listings which can be easily managed. To learn more about React's components, you can read *Learning React* by Alex Banks and Eve Porcello [8].

Ionic: Ionic Framework is also leading in its field when it comes to cross-platform compatibility. First released in 2013 by Drifty CO Ionic uses HTML, JavaScript and CSS. It has to work with a framework such as Vue, Angular or React. Ionic's pre-built components are a major advantage when it comes to visualising StockWise deployed on iOS/Android or on a web app. This saves a lot of time and effort when

not having to change CSS every time you make a change. StockWise was initially intended to be a cross-platform application, but due to the decision of not going ahead with the barcode scanner, there is no need to deploy it on mobile, so unfortunately these extra abilities Ionic offers will not be useful. For more information on Ionic, here is the official documentation [9].

3.2.2 Firebase for Backend Services

Bought by Google in 2014 Firebase is the leading backend-as-a-service out there that is known for its serverless database, which ensures fast and reliable transactions between frontend and backend. Firebase provides many different services for StockWise, from authentication to multiple users and the many different collections stored in the database, such as users, inventory, suppliers, etc. Firebase also has offline capabilities "Cloud Firestore supports offline data persistence" [10] which caches a copy of the data stored in Firestore and can be accessed when offline.

Firestore Real-Time Cloud Database: MongoDB was first selected for the database of StockWise, but as time went on, there was more of a lean towards Firestore, as its NoSQL cloud DB was so effortless and easy to set up, which would give more time to focus on other parts of the development of StockWise. Firestore uses collections, documents and fields to store its data, which is document-orientated, which makes the retrieval of the data seamless and allows data synchronisation across multiple users.

`onSnapshot` listeners are used to update the frontend when it receives a change to the Firestore database. This allows for there to be accurate real-time data synchronisation, which is crucial for stopping stock shortages. Every update to the StockWise database is stored in these collections: *Tracked Firestore Collections:*

`activityLogs`, `inventoryItems`,
`supplierOrders`, `suppliers`, `users`

Firestore's serverless architecture removes the time consumption which would have been taken up by querying the database and server management. Firestore's offline capabilities also ensure the ability to update the DB even when not connected to the internet, and when once connected again, the changes are automatically synced to Firestore, which ensures data retention and security.

Authentication & Role-Based Access Control (RBAC) Firebase also controls the user authentication to StockWise. Multiple security protocols are configured when creating a user and logging in as a user. Email and password authentication and Google Auth are being used for quick sign-in privileges with saved Google account details.

Firestore ensures that once authentication is passed, the signed-in user now has access to certain amounts of data depending on what permissions they have access to. This is role-based access control (RBAC), which is crucial for an employee user not having full access to all of the data provided by StockWise.

There are three levels of roles in StockWise:

- **Admin** – Only 1 admin is allowed, who has access to everything and controls all users permissions.

- **Manager** – Has access to all stock but nothing else and some small privileges to suppliers and orders but no access to roles.
- **Employee** – Read and write access for all data and no access to roles.

Firebase offers so much to StockWise with an easy-to-understand setup and impressive security measures. While its authentication protocols provided a safe and secure setup which ensured no foul play or any unauthorised access.

3.2.3 FastAPI and Machine Learning in StockWise

ML was integrated into StockWise to turn it from a reactive to a proactive inventory management system. The information gathered through countless hours of ML training is used to provide a data-driven prediction that helps the user to make decisions on how to manage future stock depletions.

FastAPI is used in StockWise as the backend framework which is hosting the ML service. In the article *FastAPI: The High-Performance Python Web Framework,* it highlights how it can handle REST APIs and is a high-performing framework [11]. FastAPI is a very popular lightweight API which was first found in 2018 and uses type hints to serialise and de-serialise data which then is displayed on the frontend, and the ML model is hosted on the cloud environment.

Predictive Stock Forecasting API

StockWise uses a predictive algorithm to calculate restock dates using daily consumption data retrieved from FastAPI. Here's a simplified example:

```
return predictions.map(pred => ({
  ...pred,
  recommended_quantity: Math.max(0, (pred.daily_consumption *
     10) - pred.current_quantity),
  predicted_days_until_low: Math.ceil((pred.current_quantity -
     10) / pred.daily_consumption)
}));
```

Public ML Endpoint Hosting

StockWise's predictions are served from a public ML endpoint hosted using FastAPI on Google Cloud Run:

- **Endpoint:** `https://ml-service-519269717450.europe-west1.run.app/predictions`
- **Returns:** A list of inventory items with fields including:
  - `name`, `current_quantity`, `predicted_days_until_low`
  - `confidence_score`, `recommended_restock_date`
  - `daily_consumption`, and `usage_history`
- **Format:** JSON format is returned and parsed in the frontend to display insights on inventory status and restocking urgency.

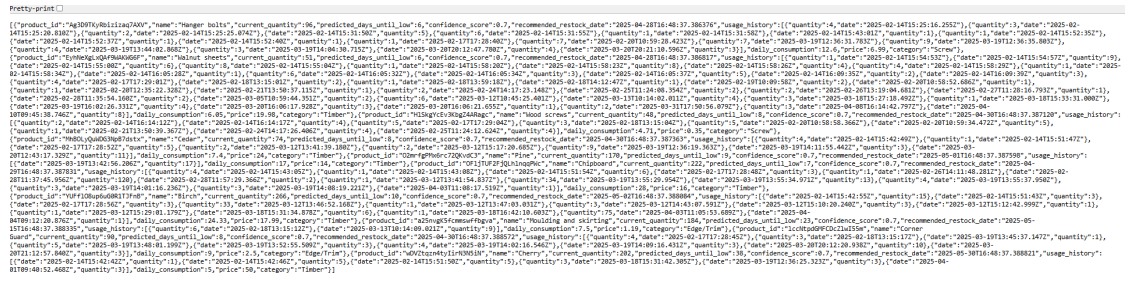


Figure 3.1: Sample JSON from FastAPI endpoint showing ML-powered inventory forecasts.

Why FastAPI?

- **High-Performance & Low Latency** – High performance is the main goal for FastAPI, as it promises to deliver up-to-date data whenever there is a change made to the machine learning. It is the fastest Python framework out there, which is needed for real-time data to be used for making accurate predictions. It is asynchronous, which helps with processing the data without having to be done automatically.
- **Native Compatibility with Python's ML Ecosystem** – The main libraries that are used in StockWise's FastAPI are Scikit-learn, Pandas, and NumPy, which are all used in the RESTful API, which helps because they are written in Python, which is lightweight, fast, and powerful.
- **Cloud Deployed via Google Cloud Run** – Google Cloud Run is hosting the machine learning, which is a very responsive and cost-efficient platform. Google Cloud Run is also serverless like Firestore, so it scales the data effortlessly and stores it in JSON format for easy querying for the frontend to display the data. Google Cloud Run can also support high workloads of data, which is crucial for an inventory management system.
- **Automatic API Documentation** – FastAPI uses Swagger and ReDoc, which provide documentation that is particularly useful for testing and debugging when working through errors with machine learning.

3.3 Machine Learning-Driven Enhancements in StockWise

Using data driven predictions StockWise utilises this data to help the business make better decisions on which stock is doing better than others and can minimise losses and costs.

3.3.1 Predictive Stock Management

StockWise's ML model continuously analyses inventory usage patterns to generate predictions that assist in maintaining optimal stock levels. This includes:

- Stock Depletion Forecasting

- Using the consumption trends, stock is predicted for when it is going to run out.
- Makes sure that the user is prompted with how many days till restock so there will not be a stock shortage.

```
days_until_low = (current_quantity - 10) / daily_consumption
```

- Restock Recommendations
- Gives an accurate prediction using this formula:

```
recommended_quantity = max ((daily_consumption * 10) -
    current_quantity, 0)
```

- Does not overstock or understock.
- Assists in determining optimal restocking dates using the sales velocity and usage.
- Confidence Scoring for Predictions
- The confidence scores are measured with consistent data being logged to the ML model where it can confidently predict when a stock is going to replenish.
- If not enough data is gathered, the confidence score is going to be low, which shows the ML model is not confident in its predictions.

Score Range	Interpretation
0.9 – 1.0	High confidence (strong usage history)
0.8 – 0.9	Good confidence (moderate stock + usage data)
0.7 – 0.8	Moderate confidence (high stock + moderate usage)
0.6 – 0.4	Low confidence (limited data or abnormal stock levels)

Table 3.1: Confidence Levels Used for Stock Predictions

- Consumption Plot Generation
- Matplotlib displays graphs which show the daily usage and the trend line for where the overall usage trend is heading.
- This helps businesses visually get a clear representation of how much stock is going out and see where there are spikes in the usage trends in all items. Matplotlib documentation was used for creating the consumption plots for StockWise. [12]

```
plt.plot(dates, quantities, 'b-o', label='Units Used')
plt.plot(trend_dates, y_trend, "r--", label='Trend Line')
```

The final plot is converted to a base64 string and rendered on the frontend via FastAPI.

3.3.2 How FastAPI Powers StockWise's AI-Driven Inventory Management

StockWise is built around FastAPI's capabilities to understand everything that is being tracked in the database. It is the engine that keeps running around the clock, keeping track of the data to gain insights into where the business needs are at all and what the future is for that business by constantly analysing its data.

- API requests are constantly forecasting data to the frontend so that the stock predictions can be made in real time and updated to the frontend for users.
- Having FastAPI as a service being hosted is greatly beneficial for not having to keep track of which service is on or off all the time. This offers high availability and does not require much maintenance once hosted efficiently.
- The recommendations use a very fair algorithm to not overestimate restocking, so this keeps the operational use high and results in very little stock waste.

FastAPI has turned StockWise into a fast intelligent system which uses its ML model to help the needs and requirements of the user first by optimising the data gathered over a certain period of time. This helps keep purchasing orders well in advance of stock depletions.

3.4 Data Visualization and Reporting

Data visualisation was a big part of StockWise, where showcasing the data gathered from an endless amount of inventory tracking was delivered on a simple graph for the user to see. Chart.js is mainly used on the reports page, which shows real-time data being represented in pie and bar charts. Chart.js official documentation was used for the implementation of the pie charts and rendering the car charts. [13]

3.4.1 Real-Time Data Representation

Real-time data is updated due to the Firestore listeners, which update automatically whenever there is a change in the Firestore database. This ensures the charts stay up-to-date, showcasing the newest information from the database, instead of there having to be a manual refresh each time.

The main visualisation features included in StockWise are:

3.4.1.1 Stock Overview Dashboard

The home page displays a summary of the main key metrics in a well-displayed format:

- Stock Location Map: Location indicator for all stock for quick access to said location, which saves a lot of time.
- Top Products Panel: Top products for different fields, such as high-quantity or high-value stock. Displayed with the quantity, name, and price of each item.

- Recent Activity Feed: Implemented for users to get a better understanding of who changed which item and when so they can track stock more accurately. Shows actions such as add, delete, and update.
- Usage Analysis Summary: ML predictions are being pulled from the restock page for quick access on the home page. Days until low and current stock of the item are displayed.

All of this data is displayed using basic Ionic components and JS along with React as a chart.js is not used on the home page for visualisation.

3.4.2 Restock Recommendations & Usage Forecasting

Machine learning restock recommendations are displayed in a card which shows information on each restock suggestion. These cards are displayed horizontally and go onto a new line when more than three are displayed together.

- Current Stock
- Predicted Daily Usage (units/day)
- Recommended Order Quantity
- Days Until Low
- Confidence Score

These cards show the forecast for the recommended suggestions that is calculated in the background by the data gathered by the ML model. Recent usage history is the main indicator for making these accurate predictions, which are backed up by a confidence score which is decided by how frequent the usage data gathered is.

Matplotlib, which is served by the FastAPI endpoint, creates a visual representation for each item on the restock page. This is a linear regression chart which shows trend lines and units used. The last 10 days of stock use are also detailed at the bottom, and the units used are detailed on the left, where it is plotted on the chart of how many units were used on which day. These visuals help the user understand the machine learning aspect of StockWise and how it is stored in the backend. Why is Matplotlib the best for generating graph visualisations? In the Matplotlib documentation, it gives extensive insight into how “Matplotlib is a comprehensive library for creating static, animated, and interactive visualisations in Python, [12], which is a significant advantage for how the user views the data.

3.4.3 Reports & Category Analysis

The reports page is the main page for data visualisation:

- Inventory Summary: KPIs for vital data such as Total Stock, Low Stock, Total Value, and Inventory Worth.
- Category Distribution: A pie chart which shows the number when you hover over it for each category throughout StockWise.

- Category Value Chart: A bar graph which shows the value of each category.
- Generating inventory and category reports: Create a detailed PDF of all inventory and categories which displays all relevant data for each item and each category.

Chart.js provides a very interactive and smooth layout for graphs where the data is clearly displayed and accessible. It helps managers with an easy-to-navigate interface when creating graphs.

These charts are implemented using Chart.js, providing smooth interactivity and responsive rendering. The layout simplifies report comprehension for managers who may not be familiar with data-heavy spreadsheets.

3.4.4 Supplier Location Mapping

Google Maps API incorporates a unique feature on the supplier page where there can be a real-time display of the supplier with the address entered into the search bar and uses predictive search for the address if the user does not fully know the address. The map is displayed on a modal with the route from the workshop to the supplier's address.

3.4.5 Summary of Technologies Used

Technology	Purpose
Chart.js	Visual reporting and analytics
Matplotlib (via FastAPI)	ML usage trend visualization
Firebase Firestore	Real-time database and syncing
FastAPI + Scikit-learn	ML backend
React + Ionic	Frontend UI and structure
Google Maps API	Location features

Table 3.2: Summary of Technologies Used

3.5 Technology Selection Process

StockWise technologies were chosen to enhance every aspect of this inventory management system with development efficiency and have an easy-to-understand business logic feel. Throughout the development process, there were several changes made to which technologies were to be used as requirements changed and StockWise moved more towards real-world implementation.

3.5.1 What Worked Well

These technologies proved to be the best choice throughout the development stage:

- React + Ionic Framework: React's reusability helped maintain a similar feel throughout the project, as it was easier to manage and design a UI.

- Firebase Firestore: Firestore's real-time capabilities made it extremely simple to update the database in real time and see changes instantly when testing. onSnapshot listeners really made the UI updates effortless, which meant there was no need for backend polling mechanisms.
- FastAPI + Python ML Stack: The most important part of StockWise was developed through FastAPI, Scikit-learn and Pandas. Through the combination of these, there was a data-driven interface displayed throughout StockWise, which was hosted by Google Cloud Run.
- Chart.js & Matplotlib: Visual representation was done through Chart.js and Matplotlib. Chart.js enhanced the data to be shown in a well-displayed bar and pie chart, while Matplotlib was served through FastAPI, which displayed the ML data on a graph.

3.5.2 What Did not Work or Required Reconsideration

There was a number of features that must be reconsidered or removed because of issues encountered.

- Gmail API Integration (Email Alerts): Low stock email alerts were working perfectly on localhost using Gmail API from Google Cloud Console and OAuth2, but once deployed, there was an issue that kept occurring with Cloud Run, which kept providing an authentication failure when testing. After multiple attempts at fixes, there was a decision to move on as time constraints were being exceeded. The likely reason for failure was because OAuth uses token flow to execute the email, but due to Google's stricter policies, the token was not being processed correctly.
- Barcode Scanner Support: Barcode scanning was once a big part of the plans for StockWise, but once the decision was made to move away from a mobile deployment to a web-only deployment, the barcode feature had to be dropped.
- CI/CD Pipeline (GitHub Actions): Continuous deployment was explored at a late stage in the development process but ultimately was not chosen, as a single developer working alone there was not a need to change up what was already working and stayed with localhost until deploying to Firebase in the end.

3.6 Limitations and Future Considerations

Although StockWise was an overall success, there would be some considerations on future enhancements to some limitations that were run into during development.

- ERP System Integration: While StockWise was based on SAP, from experience there are not actually any enterprise systems used in StockWise. The use of RESTful connectors to enhance it from a small business system to a medium- to large-sized business system.

- Supplier Intelligence: Using ML to track and learn about each supplier so it could give an estimate for delivery days and cost variance for deliveries for better insight into spending targets.

- Offline Support: Turning StockWise into a Progressive Web App would be another feature which would enhance the security and data persistence of StockWise.

3.7 Critical Evaluation of Technology Decisions

Several architectural and technology decisions were made early in the project which outlined the architectural and technical tools to be used in the final implementation for development.

3.7.1 Firestore vs MongoDB

In the proposal stage, MongoDB was picked but then went with Firestore DB, as it is renowned for its fast serverless real-time updates, and with the use of Firebase Authentication, it was easy to integrate both with one another. This kept the management of the backend a lot simpler, as the use of onSnapshot listeners was used mainly in keeping the DB up to date. Why choose Firestore over MongoDB? "Firebase Firestore provides excellent real-time performance. The database ensures instant synchronisation of data across all connected clients." [14] This decision did benefit the development phase, as there was more time to focus on other aspects of StockWise. The one downside is that with future updates to the database, it might not be able to handle a higher-scale enterprise.

3.7.2 FastAPI vs Flask

FastAPI was chosen over Flask as it is asynchronous and has OpenAPI documentation, which is beneficial with the initial setup. The integration with Python's ML, which was Scikit-learn, Pandas and NumPy, made it quite easy to deploy the machine learning aspect on StockWise. What makes FastAPI the best fit? "FastAPI takes a structured approach, using Python's type hints for validation, documentation, and a smoother development experience." [11]

3.7.3 Trade-offs in ML Simplicity vs Accuracy

Linear regression is used for stock predictions. While this is not the most popular model, it does offer a faster and more consistent score each time. Models like LSTM and ARIMA were considered, but their processing time was too high to be used for a fast and reliable deployment. The confidence score was added as a compensation for the odd occasional prediction that may not be so accurate.

3.7.4 Role-Based Access Control (RBAC)

Initially, role-based access was only admin and user roles, but the implementation of additional roles and tiers to admin, manager and employee based on the user feedback that was gathered has given the users of StockWise more control over the data.

- Menu items and feature access are filtered based on the logged-in user's role.

```
const menuItems = allItems.filter(item =>
  item.roles.includes(userRole)
);
```

- User roles are fetched from Firebase like so:

```
const userRef = doc(db, "users", userId);
const userDoc = await getDoc(userRef);
return userDoc.exists() ? userDoc.data().role : 'employee';
```

Conclusion This technology stack has been chosen over many careful and well-researched decisions. The combination of Ionic, React, Firebase, FastAPI and ML libraries is a powerful system designed to deliver a powerful application. There were a couple of setbacks which were not planned for, such as email automation and web only. StockWise, however, still delivers a business-orientated application which is data-driven to inform users to make the correct decision.

Chapter 4

System Design

The system design process of StockWise was a major component to ensure that the technologies used would work in union with one another throughout to deliver a fault-tolerant and intelligent user-based inventory management system. The system was built around having machine learning capabilities, real-time tracking and making sure the database could handle large-scale data queries.

4.1 System Architecture Overview

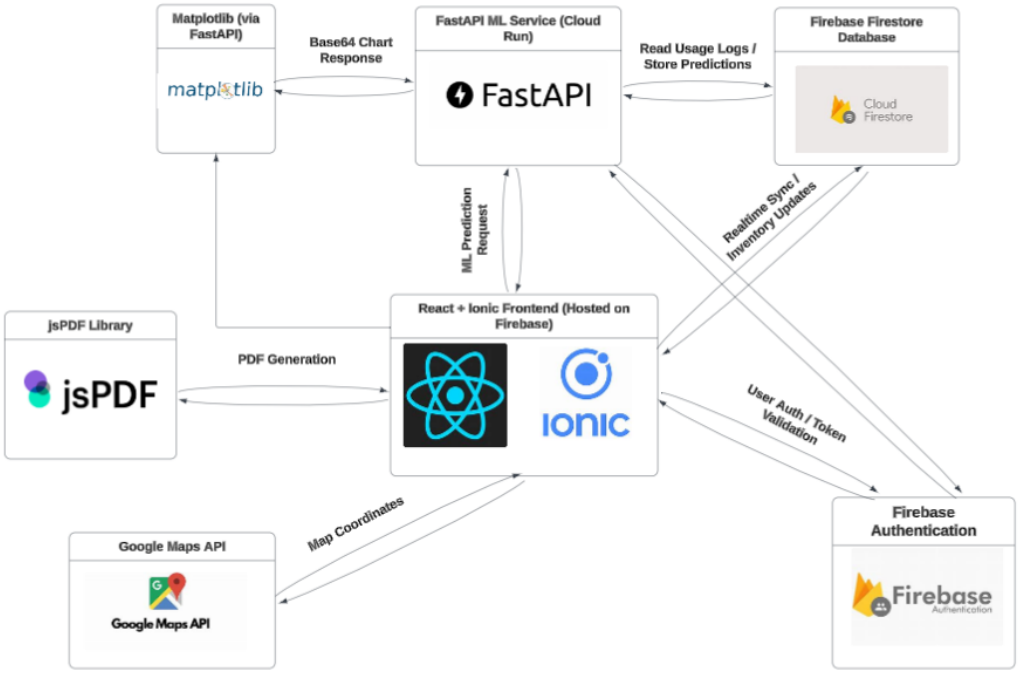


Figure 4.1: System Architecture of StockWise: Arrows labelled to show the flow of data.

As seen in Figure 4.1, the system architecture is showcasing how each technology interacts with one another. This diagram shows a unified technology stack which can

handle copious amounts of data through cloud-native tools and is being displayed on a responsive frontend.

4.2 Core Architectural Components:

React + Ionic Frontend
React, along with Ionic, serves as the UI for StockWise, which enables users to interact with the frontend as seamlessly as possible, all while providing a user-friendly, component-based interface. The frontend is hosted on Firebase, which integrates seamlessly with the backend, which is hosted on Google Cloud Run.

- **Role:** Shows real-time data pulled from Firestore. Handles ML updates and states of each order and displays inventory insights.
- **Integration:** firestoreService.ts communicates with Firestore, which makes API calls using REST and invokes the FastAPI backend, and displays the Matplotlib graphs. Firebase Auth is used to support access for users.

FastAPI ML Backend (Python)
The machine learning aspect is hosted by FasFastAPI, which is written in Python and is hosted on Google Cloud Run. It interacts with Firestore to receive user data, which is then processed using linear regression, and returns stock shortage predictions, recommended restocks and confidence scores.

- **Role:** Uses data inputted by the user to predict stock shortages and confidence scores and creates trend plots.
- **Integration:** Is deployed on Google Cloud Run and runs through Firestore, where it receives data through user input and uses that data to create predictions and graphs for the frontend.

Firebase Firestore (Realtime NoSQL DB)
Firestore is a real-time NoSQL DB which stores all of the data for StockWise. Such as inventory items, orders, suppliers, usage logs and user details. Firestore also manages role-based access and user authentication for all users who hold accounts with StockWise.

- **Role:** Use collections such as inventoryItems, suppliers, orders, activityLogs, and users which store data in them using the NoSQL DB provided by Firestore.
- **Integration:** Firestore rules decide many rules, such as access control per user role and data collections. These rules support onSnapshot listeners for which collections each user has access to.

Firebase & Google Cloud Hosting
Using a dual platform to host both frontend and backend, StockWise is built with security and robustness in mind. Firebase hosts the frontend, which serves as a static web application. Google Cloud Run hosts the backend, which is made up of API endpoints and machine learning. Cloud Run was easily chosen, as it offers high security protocols and can run FastAPI in a containerised environment while still offering quick responses to the frontend and maintaining low latency throughout.

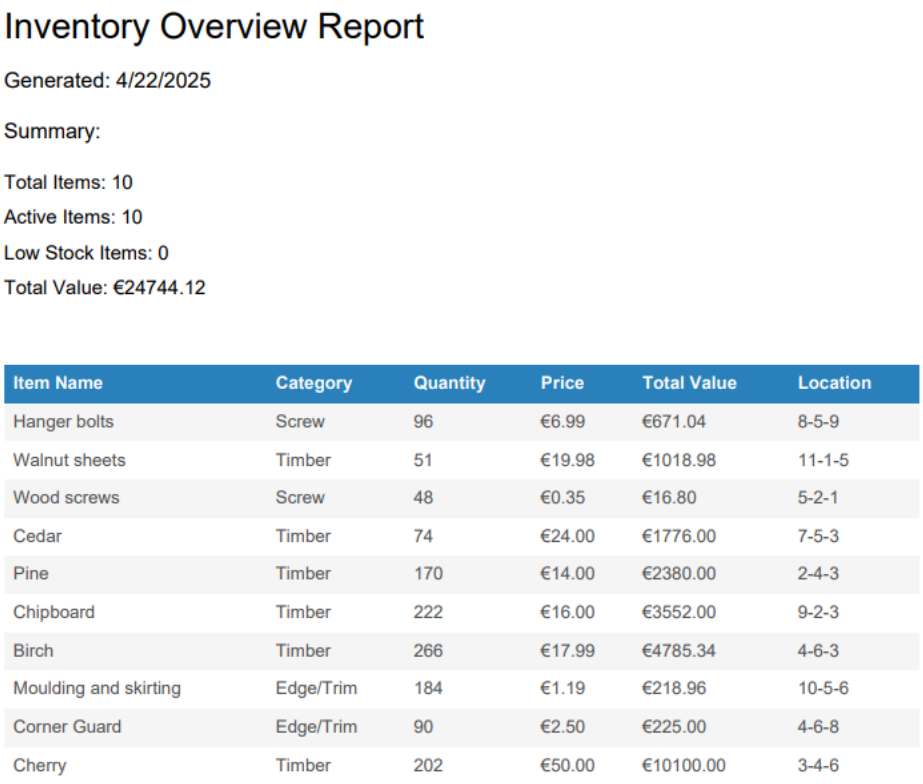
4.3 External Services

Google Maps API
Role: To display directions from the supplier address to hardcoded coordinates.
Integration: Uses predicted search and real-time mapping on the supplier page.
jsPDF
Used for generating downloadable reports with full inventory tables and metrics.

4.3.1 PDF Report Generation (jsPDF)

Reports are generated using `jsPDF` with detailed inventory tables and summary metrics.

```
doc.text('Inventory␣Overview␣Report', 14, 15);
doc.autoTable({
  head: [['Item', 'Qty', 'Price', 'Value']],
  body: inventoryData.map(item => [
    item.name, item.quantity,
    `EUR ${item.price.toFixed(2)}`,
    `EUR ${(item.quantity * item.price).toFixed(2)}`
  ])
});
doc.save('inventory-report.pdf');
```



Inventory Overview Report

Generated: 4/22/2025

Summary:

Total Items: 10
Active Items: 10
Low Stock Items: 0
Total Value: €24744.12

Item Name	Category	Quantity	Price	Total Value	Location
Hanger bolts	Screw	96	€6.99	€671.04	8-5-9
Walnut sheets	Timber	51	€19.98	€1018.98	11-1-5
Wood screws	Screw	48	€0.35	€16.80	5-2-1
Cedar	Timber	74	€24.00	€1776.00	7-5-3
Pine	Timber	170	€14.00	€2380.00	2-4-3
Chipboard	Timber	222	€16.00	€3552.00	9-2-3
Birch	Timber	266	€17.99	€4785.34	4-6-3
Moulding and skirting	Edge/Trim	184	€1.19	€218.96	10-5-6
Corner Guard	Edge/Trim	90	€2.50	€225.00	4-6-8
Cherry	Timber	202	€50.00	€10100.00	3-4-6

Figure 4.2: PDF Report: Generated using jsPDF with full inventory details including categories, quantities, and locations.

4.3.2 Usage Graphs (Matplotlib via FastAPI)

Role: Displays item usage trends to assist restocking decisions.
Integration: Base64-encoded images generated by FastAPI, embedded on the frontend.

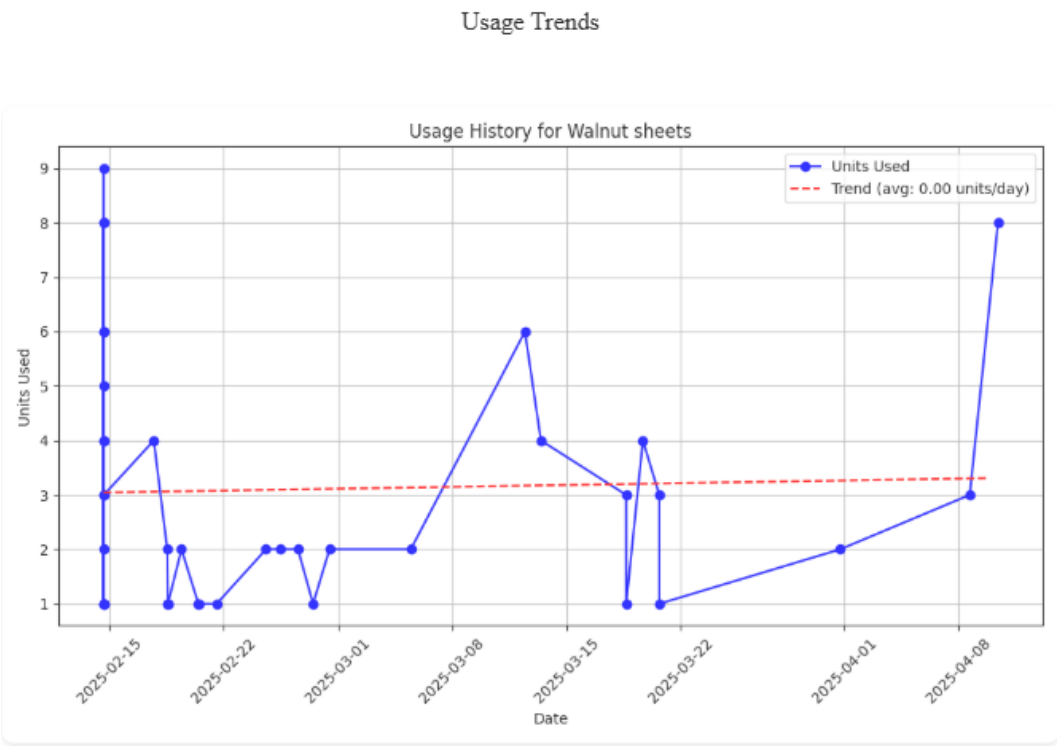


Figure 4.3: Matplotlib Graph: Item usage trends for smarter restocking, generated via FastAPI and rendered as base64.

4.4 UI Design

StockWise uses a minimalist design to help users who are not computer orientated. User efficiency is built around a real-time interface with machine learning precision insights. React's architecture is a key factor in how responsive the UI in StockWise is. Companies like Instagram, Airbnb, and Facebook all use React, as it supports real-time reloads and fast feedback and therefore is suitable for real-world applications. [8]

4.4.1 Key UI Components & Pages:

- **Home.tsx** Interactive dashboard which shows activity logs, stock locations, machine learning updates and usage trends. This provides users with a quick overview of inventory health, recent modifications, and forecasted depletion rates. It ensures timely decision-making based on real-time ML-driven predictions.

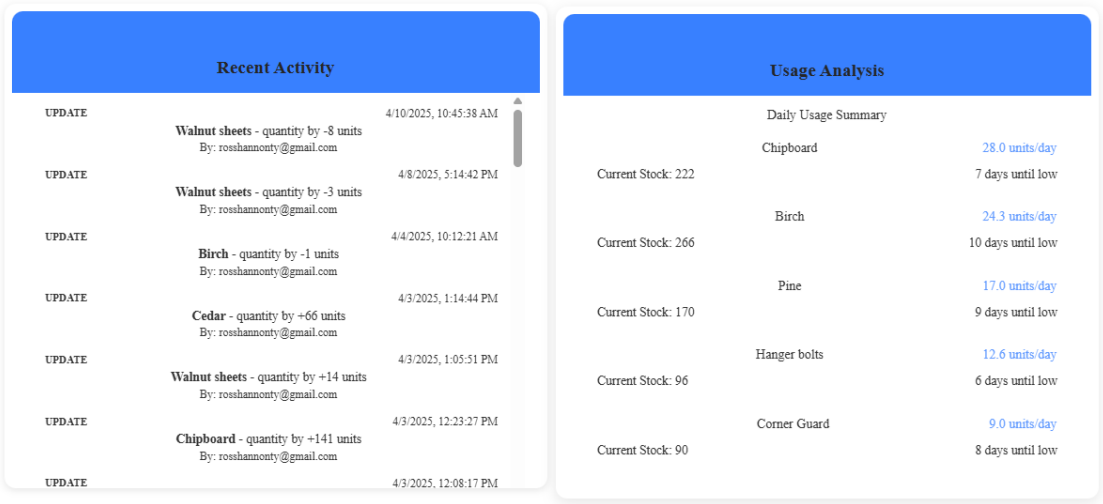


Figure 4.4: Home Page Analysis Dashboard showing Recent Activity and Usage Analysis panels.

- **Inventory.tsx:** Shows all inventory items in a horizontal format. Buttons for adding stock, editing, deleting, and using stock. Filtering system which is divided into categories, name, quantity and price. All changes to stock on the inventory page are then sent to the machine learning system to be processed.

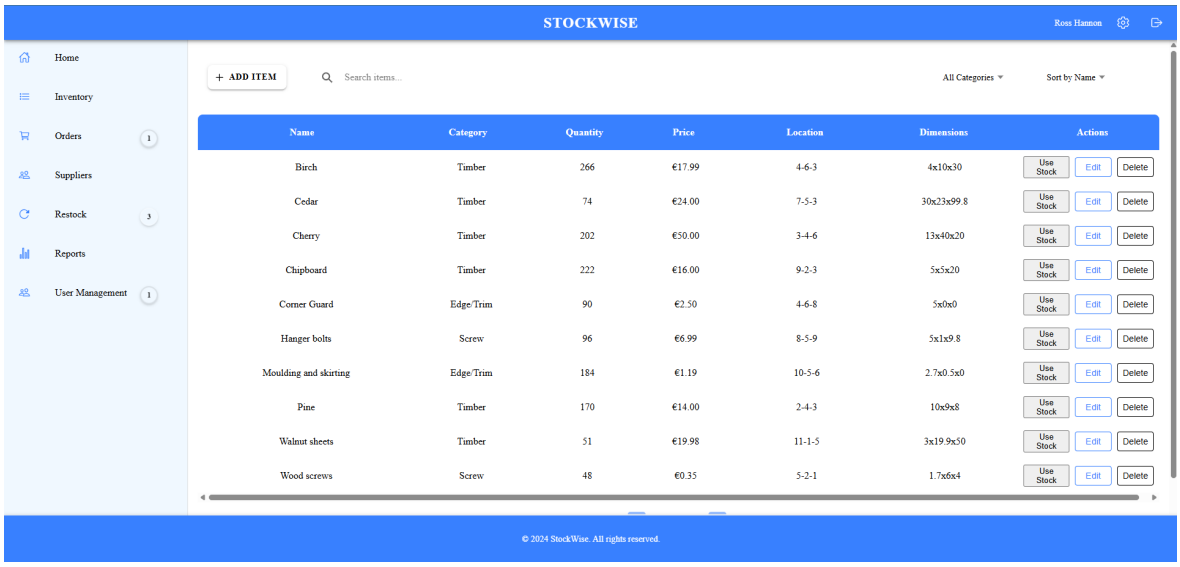


Figure 4.5: UI Screenshot of StockWise Inventory Page showing item listings, actions, and categories

- **Restock.tsx:** Shows two sections, ML Stock Insights & Usage History, which uses machine learning suggestions for restocking with all relevant data related to that item, and an order can be placed for that item. Restock Suggestions section which also shows all information related to every item that is on the inventory page and creates a usage history chart.

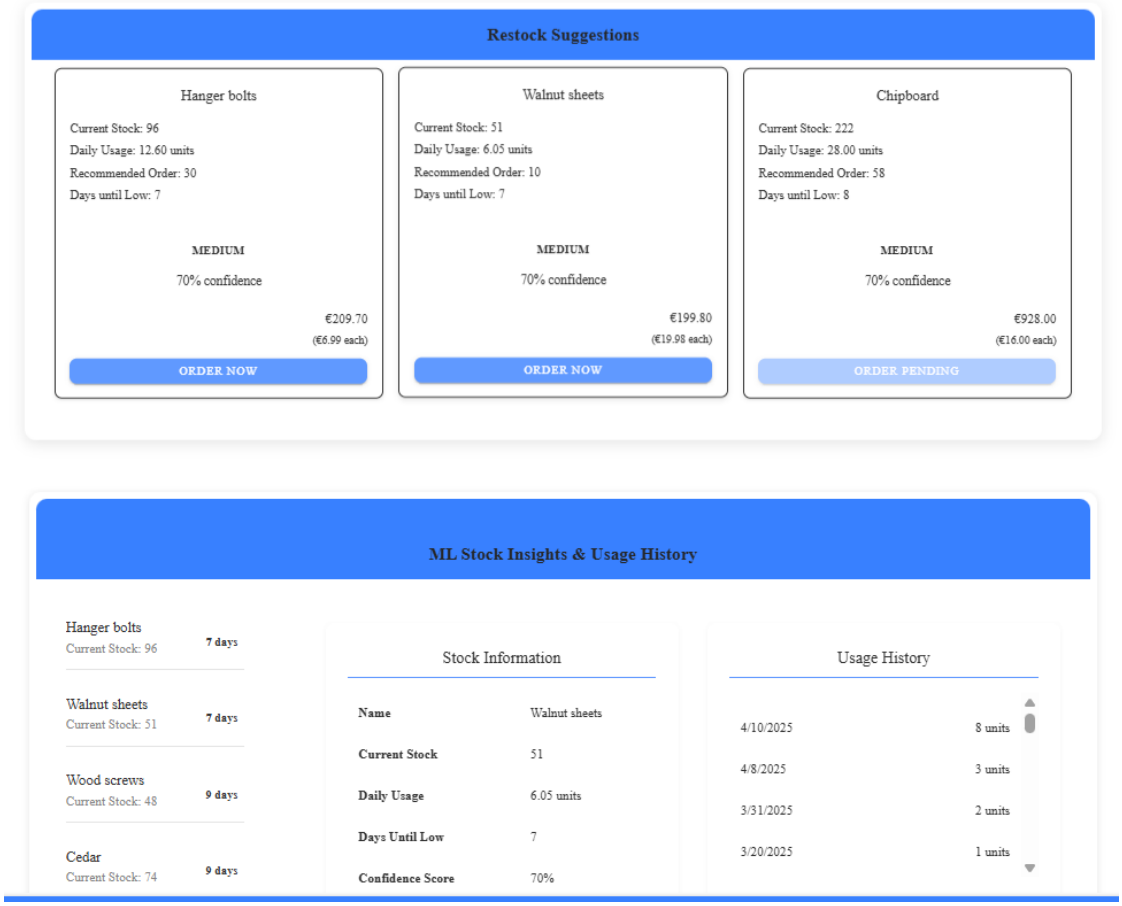


Figure 4.6: Restock.tsx Page: ML-powered suggestions and usage-based insights to predict future stock requirements.

- **Orders.tsx:** Holds all orders which are placed from the restock suggestions section. Holds state across all StockWise for each order. When order status is received, it is then hidden from the orders page, and the corresponding stock is updated on the inventory page.

- **Suppliers.tsx:** Supplier information is stored here in a horizontal format with buttons for add supplier, edit, and delete. Also tracks supplier history and links suppliers to their orders.

- **Reports.tsx:** Chart.JS displays basic information on all items and categories. jsPDF is used to generate downloadable PDFs.

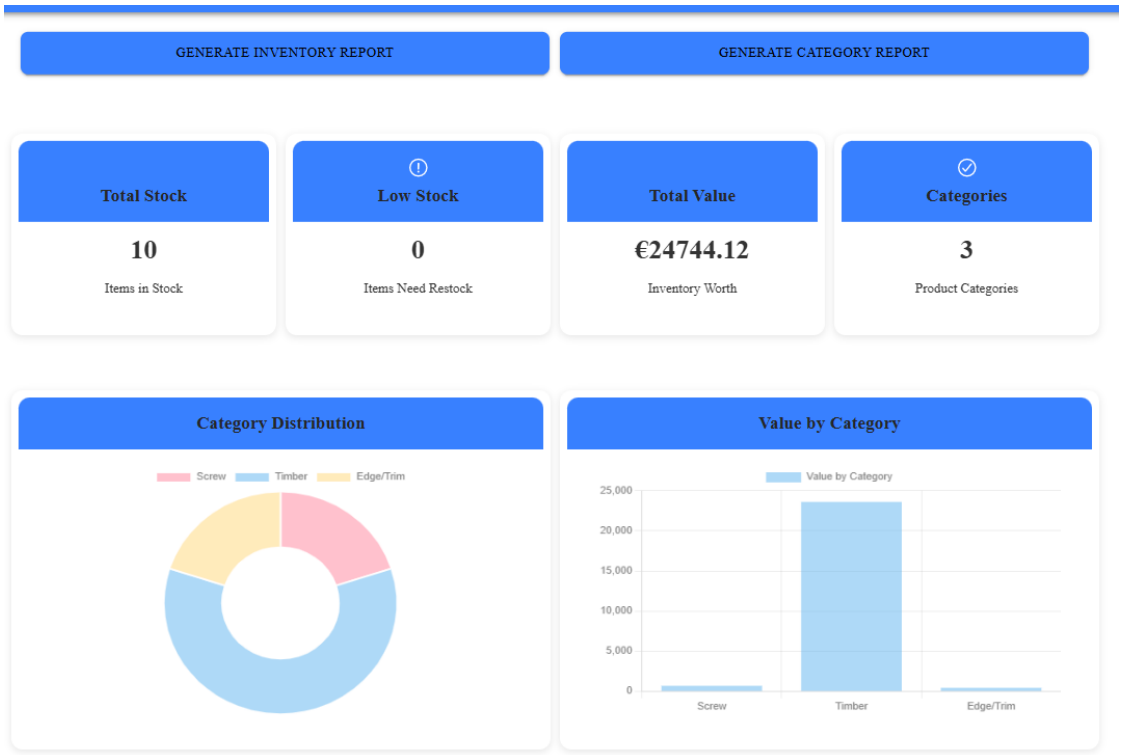


Figure 4.7: Reports Dashboard: Real-time stats on total stock, inventory worth, and category breakdown using Chart.js.

4.4.2 Early UI Implementation

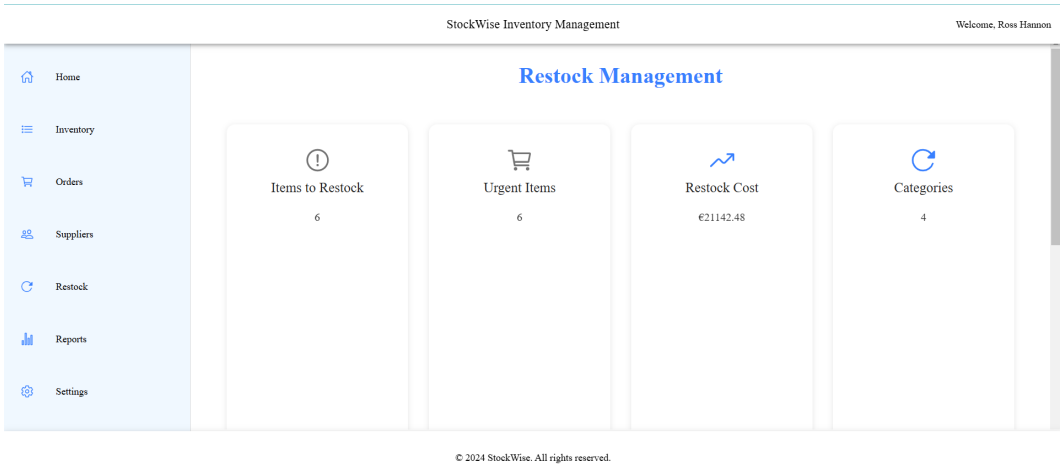


Figure 4.8: Early Development of StockWise Home Page

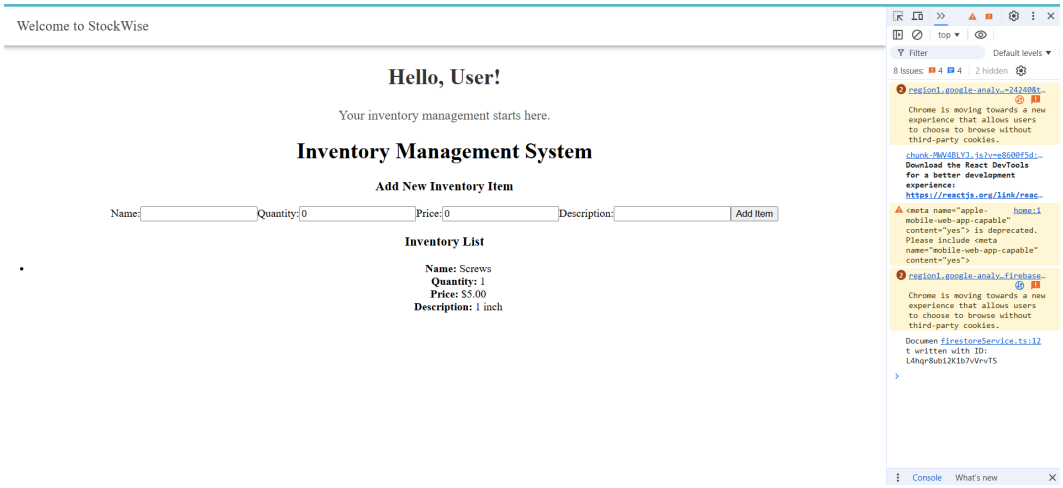


Figure 4.9: Early UI for Adding Inventory Items

Chapter 5

System Evaluation

Application evaluation is particularly important when looking back at what work has been completed and assessing whether it meets the criteria for success. Evaluation will be done across a number of key objectives, such as reactive and reusable architecture, innovative ML integration, UI coherence and accessibility, and compliance with industry standards. We will look at how they affect user experience, how they perform under pressure and if the development process met industry standards.

5.1 Reactive and Reusable Architecture

StockWise was built around having a reusable and scalable UI which is implemented consistently across all pages. Elements such as `<InventoryCard />`, `<RestockCard />`, and `<ModalForm />` are used throughout to maintain a consistent interface and reduce code duplication. Ionic components are used to hold a consistent cross-platform feel.

5.2 Innovative ML Integration

StockWise utilises machine learning powered by FastAPI and built through Scikit-learn. This combination delivers automated low-stock predictions using linear regression, which is trained through user data gathered through historical data consumption.

5.2.1 Machine Learning Forecasting Evaluation

The following table presents a snapshot of ML predictions made by StockWise for a selection of stock items. These predictions are based on live usage patterns and model inference hosted on FastAPI via Google Cloud Run.

Table 5.1: Predicted Stock Forecasts for Selected Inventory Items

Product Name	Category	Qty	Daily Use	Days Left	Restock Date	Conf. Score
Chipboard	Timber	380	38	9	2025-05-02	0.7
Hanger bolts	Screw	126	12.6	9	2025-05-02	0.7
Walnut sheets	Timber	61	6.05	8	2025-05-01	0.7
Wood screws	Screw	48	4.71	8	2025-05-01	0.7
Cedar	Timber	74	7.4	8	2025-05-01	0.7
Pine	Timber	170	17	9	2025-05-02	0.7
Birch	Timber	266	24.33	10	2025-05-03	0.7
Cherry	Timber	202	5	38	2025-05-31	0.7

The historical data is providing insight into restock suggestions which are evaluated through calculations handled in `firestoreService.ts`.

The extensive user testing conducted through the development process and local businesses provided valuable insight into how well these features worked. This reduced the need for manual stock checks, as the machine learning model kept count of every item that was used and provided a timestamp with each entry and introduced a new layer of intelligence which might not have been available to small businesses before.

5.3 UI Coherence and Accessibility

StockWise was designed with the needs of users with little computer expertise in mind while maintaining clarity throughout. As StockWise is focused on small businesses, there usually would not be a technical person working for that business, so the business owner would usually be the main inventory manager. There was a simple colour pattern used on all components to keep the visual aspects basic yet fluid. The styling for headers and footers remains consistent throughout, while all modals and forms have the same layout for adding and editing.

5.4 Compliance with Industry Standards

StockWise uses a wide range of frameworks and cloud platforms which all meet industry standards in today's development practices.

Today's trends are based on serverless DBs and modular software systems. Firebase uses a real-time backend which updates automatically without needing to refresh the page.

ML was developed by using FastAPI and Scikit-learn and deployed on Google Cloud Run, which is hosted on a containerised setup. This technology stack is what today's industry uses, as it supports high availability and low latency.

Role-based access control is a pivotal industry standard, as it provides security and safety for logins to Firebase's authentication setup. The StockWise admin holds full user privileges where it has full access and accepts new users.

5.5 Examples of StockWise in Action

The StockWise Lifecycle: From Usage to Restock

StockWise supports a complete end-to-end inventory lifecycle designed to help small businesses track and manage stock efficiently through machine learning, supplier integration, and real-time updates.

1. **Inventory Setup:** Stock is initially added to the system using the *Add New Inventory Item* form. The user provides item details such as name, quantity, price, category, and dimensions. The item is then visible on the main inventory page.

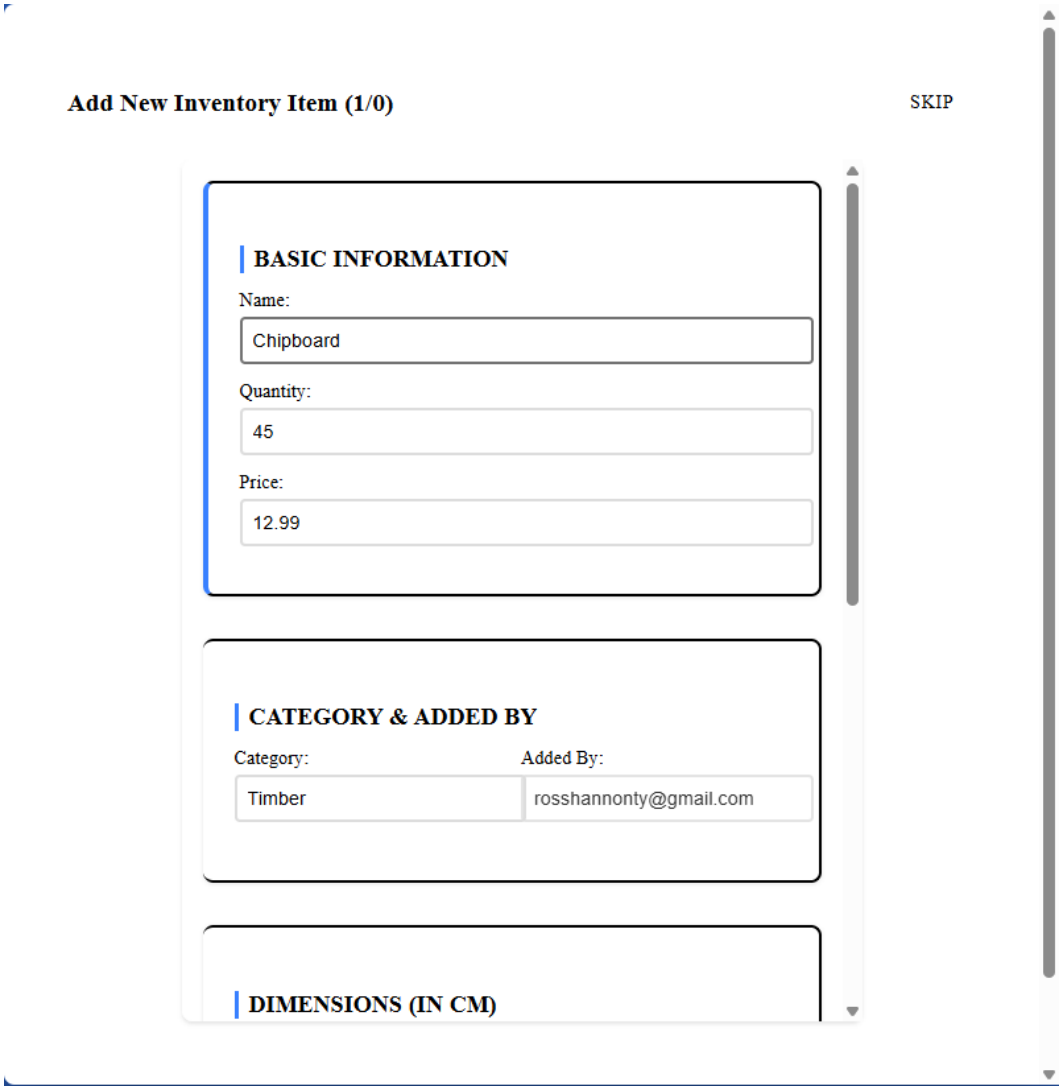


Figure 5.1: Adding a new inventory item through the inventory form.

2. **Stock Usage:** As materials are used, users log the quantity consumed using the *Use Stock* feature. This reduces the current quantity of the item in Firestore and records a usage timestamp.

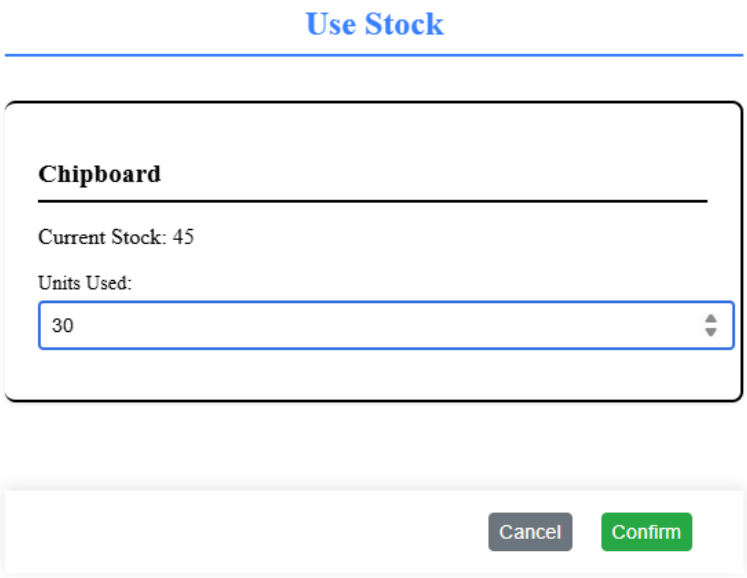


Figure 5.2: Logging usage of stock via the Use Stock feature.

3. **Restock Suggestion (ML Forecast):** Once usage patterns indicate low stock levels, StockWise generates a restock suggestion using its ML forecasting model. This model predicts the number of days until depletion, the recommended quantity to reorder, and a confidence score of how accurate the data gathered is.

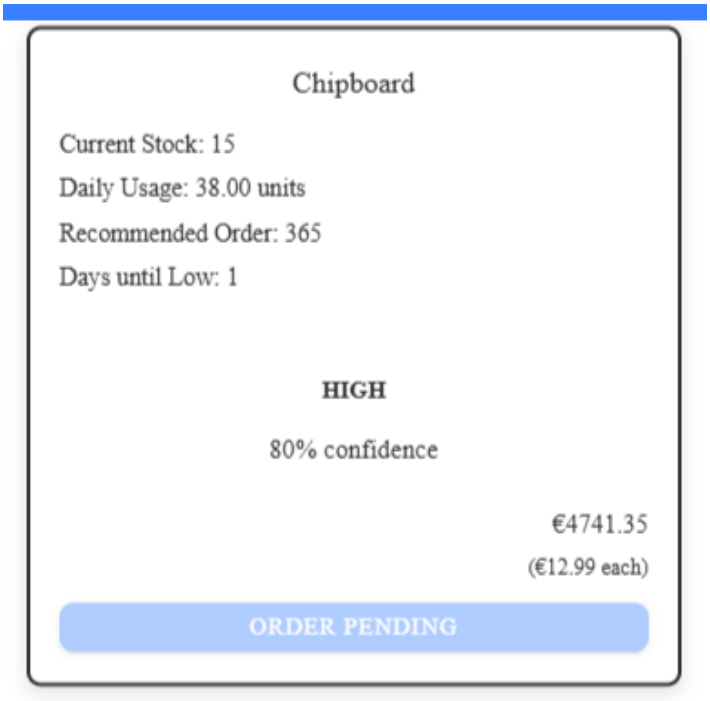


Figure 5.3: ML-powered restock suggestions based on usage and prediction confidence.

4. **Placing an Order:** When an admin agrees with the ML recommendation, they can place a restock order from the Restock page. This generates a new entry

in the *Orders* section, linked to the supplier and containing the details of the item.

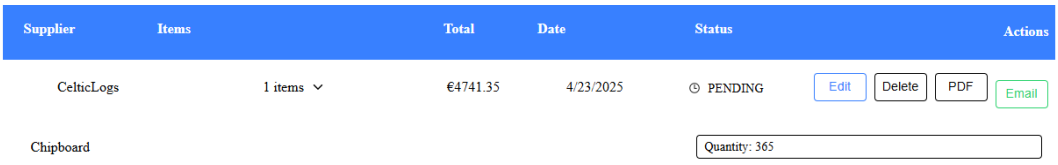


Figure 5.4: Placing an order with a supplier after the ML restock suggestion.

5. **Order Management:** Orders can be updated through multiple statuses, including Pending, Sent, Shipped, and Received. When the order is marked as received, StockWise triggers a stock update for the corresponding item.

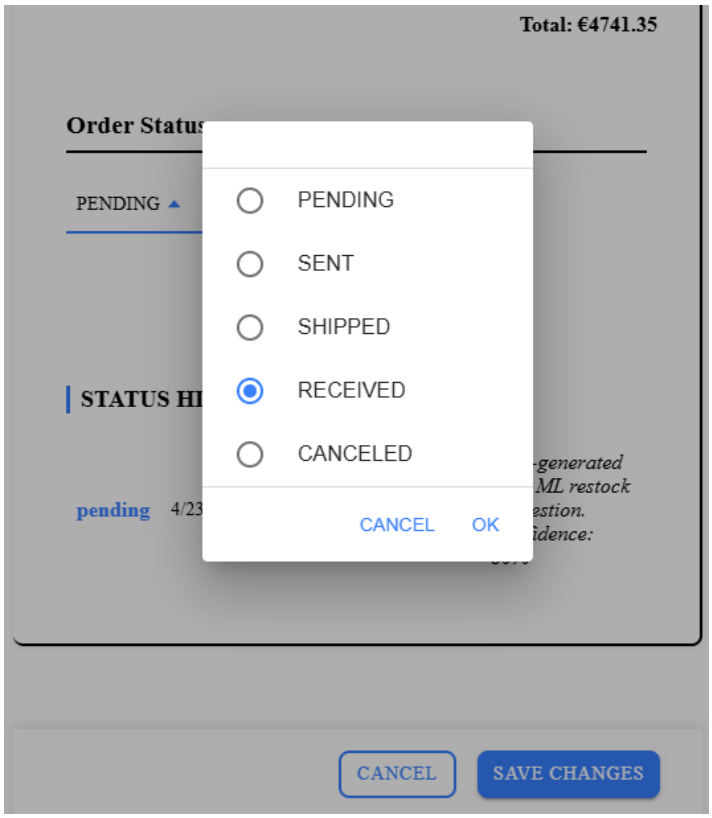


Figure 5.5: Updating an order status to “Received” to confirm delivery.

6. **Stock Update:** When the order status is set to Received, the stock level in the inventory is automatically updated with the new quantity from the order. This closes the loop and ensures accurate inventory levels.

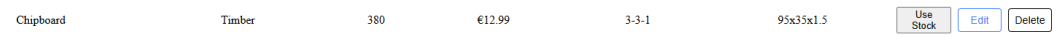


Figure 5.6: Inventory was automatically updated after receiving the order.

5.6 System Accuracy Comparison

Feature Tested	Manual System	StockWise
Stock Check Frequency	Weekly	Real-time
Time to Restock	Avg. 2 days delay	Instant predictions
User Roles	None	Admin, Manager, Employee
ML Predictions Accuracy	Not available	85–95% (estimated)

Table 5.2: Comparison of Manual Inventory vs. StockWise

5.7 StockWise Evaluation: Conclusion

After evaluating StockWise, there is a clear picture painted where all objectives were reached, and some exceeded the initial expectations. Although machine learning was never promised in the proposal, there was a significant change in how StockWise was going to perform. Moving more towards the user's needs as the development process progressed.

Integrating real-time updates, predictive insights and advanced user authentication has really elevated StockWise, where small businesses would not have had access to these features before when starting off.

StockWise has demonstrated how technical and relevant an inventory management system can become to help small businesses that do not have the funding to afford ERP systems such as SAP or Oracle. Its cloud-native technologies, along with machine learning capabilities and user-centric design, prove to be a professional solution for companies needing a starter inventory management system.

Chapter 6

StockWise Conclusion

In this dissertation there has been an in-depth examination into the planning, design, development, and evaluation of StockWise. From the outset, there was always a focus on using the knowledge gained from SAP, which deals with ERP systems for medium to large businesses. Focusing on user-friendly design which is tailored towards small businesses that often lack the tools StockWise can offer. This dissertation outlines how cloud-native tools along with machine learning are the future for all inventory management systems.

6.1 Revisiting Objectives

In the introduction there were project objectives that guided the development process:

- **Full CRUD Functionality:** CRUD is supported on orders, supplier, and inventory pages. These CRUD operations are done through Firestore using its NoSQL collections where the data can be stored securely.

- **Smart ML-Powered Restocking:** A customised API which handles machine learning to help users with fast and intelligent predictions. StockWise is committed to delivering an advanced product for small business users who do not have access to machine learning.

- **User-Friendly UI for All Ages:** Utilises an impressive UI comfortable for users of all ages and technical skill levels.

- **Role-Based Admin Access:** Differentiates between admins and users. Admins hold data integrity to a high standard so no outside user can interact without approval.

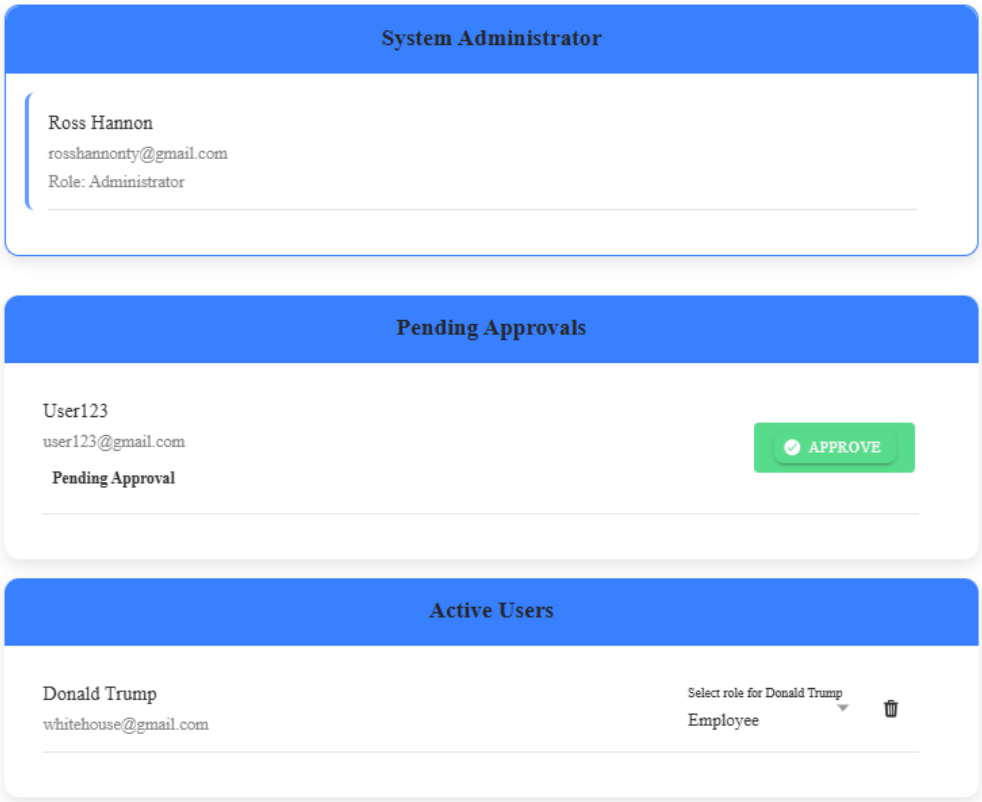


Figure 6.1: Admin Access Panel: Pending users require admin approval before gaining access to StockWise. Active users are managed in real-time, and roles can be updated dynamically.

6.2 Reflections on Learning and Development

The development of StockWise provided a substantial amount of learning opportunities, such as cloud deployment, full-stack web development, machine learning and API integration, which have all given back vast knowledge. There was always a huge significance around user-centric design, which is needed when dealing with users who may not be technical. Machine learning predictions, along with API services and cloud architecture, brought StockWise to the next level through the integration of advanced and effective technologies, which provided valuable insights for future projects.

6.3 Beyond the Objectives

With machine learning not being part of the original plan, it very quickly became the main focus of StockWise, as it provided its most powerful features. Using its predictive knowledge bridged a gap between normal inventory tracking abilities and a very advanced intelligent automation system. Going forward, there is a lot of room to advance this machine learning model and deeper integration to unlock greater automation potential.

6.4 Concluding Remarks

StockWise was a major project to work on, having gained a vast amount of knowledge from its development. It has delivered a small enterprise-grade functionality available for small businesses which provide a steady flow between complexity and usability.

As this dissertation concludes, it's important to reflect on the technical achievements delivered, which are now a solution for real-world problems. Small startups growing rapidly and having to rely on manual inventory management can now adopt StockWise, as it can offer efficiency, intelligence, and empowerment.

Appendix

GitHub Repository Link

The full StockWise project source code can be accessed at the following GitHub repository:
`https://github.com/rosshannon7677/StockWise`

Project Demonstration Screencast

A short screencast demonstrating the main features and functionality of StockWise is available at:
`https://www.youtube.com/watch?v=g-QFwO2XuKQ`

Bibliography

[1] Inside Logistics. Small businesses struggle with inventory, survey finds. https://www.insidelogistics.ca/e-commerce/small-businesses-struggle-with-inventory-survey-finds-187881, 2023. Accessed: 2025-04-24.

[2] MaybeWorks Blog. Pros and cons of the ionic framework. https://maybe.works/blogs/pros-and-cons-of-ionic-framework, 2023. Accessed: 2025-04-24.

[3] Small Business Inventory Management Blog. Ai for inventory management. https://small-business-inventory-management.com/blog/inventory-management/ai-for-inventory-management.html, 2024. Accessed: 2025-04-24.

[4] Ken Schwaber and Jeff Sutherland. The scrum guide. https://scrumguides.org/scrum-guide.html, 2020. Accessed: 2025-04-20.

[5] Atlassian. Jira software guide: Agile development. https://www.atlassian.com/software/jira/guides/agile, 2024. Accessed: 2025-04-20.

[6] Teamhood. Agile project management: Sprint planning guide. https://teamhood.com/project-management/agile-project-management-guide/, 2025. Accessed: 2025-04-20.

[7] Atlassian Community. How kanban boards improve workflow in jira. https://community.atlassian.com/t5/Jira-articles/How-Kanban-Boards-Improve-Workflow-in-Jira/ba-p/2638129, 2025. Accessed: 2025-04-20.

[8] Alex Banks and Eve Porcello. *Learning React: Functional Web Development with React and Redux*. O'Reilly Media, Inc., 2nd edition, 2020.

[9] Ionic Framework Team. Ionic framework documentation. https://ionicframework.com/docs, 2024. Accessed: 2025-04-20.

[10] Firebase Documentation. Cloud firestore offline data. https://firebase.google.com/docs/firestore/manage-data/enable-offline, 2024. Accessed: 2025-04-20.

[11] FastAPI Documentation. Fastapi: The high-performance python web framework. https://fastapi.tiangolo.com, 2024. Accessed: 2025-04-20.

[12] Matplotlib Contributors. Matplotlib quick start guide. https://matplotlib.org/stable/users/explain/quick_start.html, 2024. Accessed: 2025-04-20.

[13] Chart.js Contributors. Chart.js official documentation. https://www.chartjs.org/docs/latest/, 2024. Accessed: 2025-04-20.

[14] Firebase Documentation. Firebase firestore overview. https://firebase.google.com/docs/firestore, 2024. Accessed: 2025-04-20.