INVENTORY MANAGEMENT SYSTEM FOR A RETAIL STORE TO OPTIMISE INVENTORY

Submitted on: Word Count:

Author: Alex Nung Student ID: 13807762 Candidate Number: A118697

BSc Data Science and Computing Final Project Report

Birkbeck College, University of London, 2025

Github repo link - https://github.com/alexnung/FinalProject

This report is the result of my own work except where explicitly stated in the text.

The report may be freely copied and distributed provided the source is explicitly acknowledged

# Abstract

Retail businesses are increasingly reliant on data to drive operational efficiency and decision-making. High-quality data is essential to ensure smooth business operations and maintain customer satisfaction. However, poor inventory management practices and data inaccuracies can lead to significant challenges, such as stockouts, overstocking, inefficient resource allocation, and missed sales opportunities. These issues can manifest as operational inefficiencies and financial losses that reduce trust in systems and processes.

When inventory management systems (IMS) cannot provide accurate, real-time data, staff must adopt workarounds, leading to inefficiencies and errors. Persistent issues can result in a loss of user confidence, resulting in reliance on external solutions or manual processes that further reduce productivity. It is insufficient to solely address data inconsistencies; businesses must understand and mitigate the broader impacts on operations and decision-making.

This project seeks to address these challenges by developing a tailored IMS that empowers retail businesses to maintain accurate stock levels, streamline operations, and enhance decision-making capabilities. By enabling real-time tracking, actionable insights, and user-friendly interfaces, the system aims to improve inventory management processes and ensure alignment with business objectives.

The solution also acknowledges the limitations of existing systems, such as high costs, complexity, and a lack of flexibility. By providing a custom-built alternative, this project offers retailers a simplified yet robust system meeting specific requirements while fostering trust, operational efficiency, and strategic agility.

Contents

[Abstract 1](#_Toc198175299)

[Introduction 4](#_Toc198175300)

[Aims and Objectives 6](#_Toc198175301)

[Project Objectives 6](#_Toc198175302)

[Literature review and context 7](#_Toc198175303)

[Case Studies 7](#_Toc198175304)

[Methodology 9](#_Toc198175305)

[Proposed Development Methodology 9](#_Toc198175306)

[Comparison of Agile with Other Methodologies 9](#_Toc198175307)

[Benefits of implementing Agile Methodology 9](#_Toc198175308)

[Challenges in Implementing Agile Methodology 10](#_Toc198175309)

[Solutions for Challenges 10](#_Toc198175310)

[Tools and software used 10](#_Toc198175311)

[Development Methodology 11](#_Toc198175312)

[Requirements, Specification and Design 12](#_Toc198175313)

[Analysis of Requirements and Design of the Proposed System 12](#_Toc198175314)

[Requirements Specification 12](#_Toc198175315)

[Functional Specification 13](#_Toc198175316)

[Comparison of Methods, Algorithms, and Data Structures 13](#_Toc198175317)

[Development 14](#_Toc198175318)

[Planning and Requirements Gathering 14](#_Toc198175319)

[Planning 14](#_Toc198175320)

[Requirement Gathering 14](#_Toc198175321)

[Deliverables 14](#_Toc198175322)

[Risk Analysis 15](#_Toc198175323)

[Database Design & System Architecture 15](#_Toc198175324)

[Database Design 15](#_Toc198175325)

[System Architecture 17](#_Toc198175326)

[Hardware and Software Requirements 17](#_Toc198175327)

[Software Requirements: 17](#_Toc198175328)

[Hardware Requirements: 17](#_Toc198175329)

[Frontend prototype 17](#_Toc198175330)

[Design the database and Backend 20](#_Toc198175331)

[Create the Frontend 22](#_Toc198175332)

[Test the system 25](#_Toc198175333)

[Implementation 26](#_Toc198175334)

[Analysis 29](#_Toc198175335)

[Test and Evaluation Plan 29](#_Toc198175336)

[Features tested 29](#_Toc198175337)

[Unit testing demo 30](#_Toc198175338)

[Conclusion 31](#_Toc198175339)

[Results 32](#_Toc198175340)

[Key Outcomes 32](#_Toc198175341)

[Data Collection 32](#_Toc198175342)

[Working System Demonstration 33](#_Toc198175343)

[Experiments and Simulations 34](#_Toc198175344)

[Validation of data integrity constraints 34](#_Toc198175345)

[Error Analysis 35](#_Toc198175346)

[Limitations and Observations 35](#_Toc198175347)

[System Limitations 35](#_Toc198175348)

[Observations 35](#_Toc198175349)

[Discussion 36](#_Toc198175350)

[Conclusion 38](#_Toc198175351)

[References 39](#_Toc198175352)

[Appendices 40](#_Toc198175353)

[Table 1 40](#_Toc198175354)

[Table 2 41](#_Toc198175355)

# Introduction

The increasing demand for IMSs emphasises its importance in modern retail operations, with an estimated 57% of retail enterprises increasing investments in software in 2024 “according to Gartner”[1] and this combined with the estimated annual growth of its market value by 9.7% of IMS’s between 2024-2031[2]. The challenges faced by retail businesses include increasing customer expectations, navigating supply chain complexities, increased procurement and infrastructure costs, and rapid technological innovations. Customers now demand a user-friendly interface, a smooth shopping experience, accurate product descriptions, and availability and timely delivery of products; this increases the importance of an efficient IMS. Also, disruptions with supply chains magnified by events out of businesses’ control, such as the COVID-19 pandemic, have stressed the importance of having resilient and adaptable systems. Addressing these concerns with an efficient IMS becomes an important factor for long-term, sustained success for retail businesses.

Additionally, the impact of social media and frequently changing consumer habits can affect how businesses are perceived, thus accentuating the importance of a smooth customer experience when shopping and an IMS is key to this. This also drives and improves operational efficiency and customer satisfaction, which is vital in the current age due to increasing competition from similar retailers and potentially negating the negative impacts of social media. Insufficient management of stock also directly impacts finances, with 62% of businesses’ finances affected by failure in inventory tracking[3]. This immediately impacts cash flow, stock capacity being one reason why businesses may fail. There is also the missed opportunity of sales and promotional optimisation, and stock shrinkage.

This project aims to address these challenges and provide a solution to mitigate them as much as possible while also being flexible enough to adapt to differing requirements. By implementing a robust IMS, the project seeks to ensure accurate, real-time tracking of stock levels, maximise efficiencies, and reduce the risk of stockouts or overstocking. The system will also include features such as insights through data analysis and user-friendly interfaces to improve accessibility. The solution aims to enhance operational efficiency, customer satisfaction, and long-term business resilience.

Due to rapid technological advancements, the IMS can be distributed across multiple platforms, enabling the creation of applications and software. This approach allows users to view, manage, and access data seamlessly while incorporating Role-Based Access Control (RBAC) to ensure secure and restricted access based on user roles.

Existing IMSs share similar features to those proposed in this project. These include:

1. **QuickBooks Commerce**: This solution offers a platform for managing inventory, orders, and sales. It includes features such as real-time stock tracking, barcode scanning, and a user-friendly interface that can integrate with multiple e-commerce platforms. Like the proposed system, QuickBooks Commerce focuses on maximising efficiency, improving stock visibility, and enabling scalability to its inventory management processes.
2. **Cin7**: This solution is a cloud-based IMS designed to handle inventory, point of sale, and order management across multiple channels. Cin7 is known for its real-time tracking features, automation tools, and advanced reporting capabilities. It also incorporates role-based access control (RBAC), which is a key feature of the proposed system, ensuring secure management of inventory data based on user roles.
3. **NetSuite ERP**: This solution is a leading enterprise resource planning system; NetSuite offers a comprehensive IMS as part of its suite of business solutions. It includes features such as real-time inventory tracking, barcode scanning, and advanced analytics. NetSuite's scalability and integration capabilities make it suitable for businesses of all sizes, aligning with the project’s goal of developing a flexible and adaptable solution for diverse retail needs.

These examples illustrate the types of IMS solutions currently available and demonstrate how the proposed project fits within this evolving landscape. By incorporating similar features, such as real-time tracking and role-based access control, the proposed system aims to enhance operational efficiency and resilience, responding to the challenges faced by modern retailers.

The challenges posed by global supply chain complexities and disruptions, such as those experienced during the COVID-19 pandemic, highlighted the necessity for adaptability and resilience in IMSs. The context has shaped the project to prioritise real-time tracking and accurate data management, enabling businesses to respond effectively to uncertainties. Additionally, evolving customer expectations for seamless shopping experiences have driven the inclusion of user-friendly interfaces and rapid data accessibility to enhance customer satisfaction and operational efficiency.

Financial risks linked to inventory mismanagement, such as stock shrinkage and increased carrying costs, further influenced the project's design. Features like role-based access controls will be integrated to enhance security and ensure system reliability. By addressing these contextual challenges, the project aims to provide a robust and adaptable solution tailored to the strategic needs of modern retail businesses.

The remainder of this report is structured as follows: **Chapter 2** outlines the project's aims and objectives, focusing on its specific goals and expected outcomes. **Chapter 3** provides a literature review, contextualising the relevance of IMSs in the current retail landscape and examining existing solutions and theories. **Chapter 4** details the methodology used to design and develop the system, including the tools and frameworks selected. **Chapter 5** specifies the system requirements and design principles, describing how the proposed solution addresses the identified needs. **Chapter 6** focuses on the development of the project. **Chapter 7** focuses on the implementation process, elaborating on how the system's components were developed and integrated. **Chapter 7** analyses the system's functionality, efficiency, and performance. **Chapter 8** presents the results, evaluating the system's impact and highlighting its benefits. **Chapter 9** discusses the implications of the findings, exploring challenges encountered and potential improvements. Finally, **Chapter 10** concludes the report, summarising the project's contributions and proposing directions for future work.

# Aims and Objectives

This project aims to design and implement an efficient and adaptable Inventory Management System (IMS) tailored to meet the complex needs of modern retail businesses. This system aims to improve operational efficiency, enhance customer satisfaction, and ensure long-term business resilience by providing real-time tracking of inventory, optimising stock levels, and incorporating features such as role-based access control (RBAC). By addressing the growing challenges of inventory mismanagement, supply chain disruptions, and evolving consumer demands, the system will help businesses streamline their operations, reduce costs, and improve their responsiveness to market changes.

## Project Objectives

To achieve the aims of the project, the following objectives have been defined with measurable outcomes:

1. Analyse the challenges faced by retail businesses in inventory management.
   1. **Outcome**: A detailed understanding of the key issues related to stock accuracy, supply chain, and customer expectations. This will inform the IMS design to address problems.
   2. **Measurable Outcome**: A report summarising key findings and identifying the most pressing inventory management issues.
2. Design the architecture and features of the proposed IMS
   1. **Outcome**: A detailed proposal of the IMS, including features like real-time stock tracking, RBAC, and data analysis, prioritising scalability, user-friendliness, and security.
   2. **Measurable Outcome**: A finalised design document, including system architecture diagrams and feature specifications.
3. Develop the IMS system, focusing on core functionality and integration with external platforms.
   1. **Outcome**: The development of the IMS software allows seamless inventory management across platforms, with core features such as real-time tracking and multi-platform accessibility.
   2. **Measurable Outcome**: A functioning prototype of the IMS with core features.
4. Implement user testing and feedback collection to refine the system.
   1. **Outcome**: Identify and resolve any user experience issues, bugs, or feature gaps, ensuring the system meets end-user expectations.
   2. **Measurable Outcome**: User feedback reports and a list of improvements made based on testing.
5. Assess the effectiveness of the IMS in addressing the challenges through performance metrics.
   1. **Outcome**: Evaluation of system success in solving the inventory management issues identified in objective one. Key performance indicators (KPIs) like stock accuracy levels, time saved in inventory tracking, and user satisfaction are used to measure success.
   2. **Measurable Outcome**: A report documenting the system's performance against KPIs.
6. Provide recommendations for future improvements and scalability of the IMS.
   1. **Outcome**: A plan for improving the IMS, including new features, improvements to the user interface, and scaling the system for larger businesses.
   2. **Measurable Outcome**: A report with proposed improvements, reinforced by user feedback and system performance data.

By defining and achieving the set objectives, the project aims to deliver a functional, scalable, and user-friendly IMS addressing challenges faced by retail businesses, improving operational efficiency, and providing long-term value.

# Literature review and context

Effective inventory management is essential to business operations, directly impacting profitability, efficiency, and customer satisfaction. Reports show that 20-40% of most businesses' working capital is tied up in inventory, highlighting its importance in financial management. Poor inventory practices can lead to major consequences, evidenced by the findings of a Tuck Business School study, which identified 117 errors in 25 sample spreadsheets. While 40% of these errors had minimal impact, seven resulted in severe losses ranging from $4 million to $110 million[6]. Additionally, inventory mismanagement due to out-of-stock and overstocked products was projected to cost retailers $1.77 trillion worldwide in 2023, according to IHL Group[10].

Apple CEO Tim Cook famously likened inventory to dairy products, stating, “No one wants to buy spoiled milk.” Under Cook’s leadership, Apple adopted just-in-time manufacturing, slashing its inventory turnover time from months to as little as five days, demonstrating the value of strategic inventory control[7]. Examples from major companies illustrate the risks of mismanagement. Nike faced significant challenges in the early 2000s, losing approximately $100 million in sales due to inventory problems[8]. Similarly, retail giants Kohl’s and Nordstrom recently experienced sharp declines in profit margins due to excess inventory, requiring major discounts to clear stock[9]. This shows that even industry leaders are susceptible to the dangers of poor inventory management.

The studies below explore various inventory management challenges and solutions across different sectors, drawing lessons from successful and failed implementations.

## Case Studies

1. **Goods Order Inventory System and Shopify Integration (Home Appliances Retailer)[4]**
   1. Challenges
      1. Manualworkflows lead to order delays.
      2. Fragmentedinventory tracking and lack of real-time data.
   2. Solutions
      1. Adoption of Goods Order Inventory integrated with Shopify.
      2. Real-time inventory tracking, barcode systems, and automation of processes.
   3. Results
      1. Improved efficiency, reduced manual errors, and improved customer satisfaction.
2. **Wasp Barcode Inventory Solution (Topgolf)[5]**
   1. Challenges
      1. Dependence on spreadsheets leads to inefficiencies and errors.
      2. Lack of visibility into inventory levels across multiple locations.
   2. Solutions
      1. Implementation of Wasp's inventory tracking system.
      2. Real-time updates, automated tracking, and streamlined order processing.
   3. Results
      1. Significant time savings, improved accuracy, and reduced stress on operations.
3. **Inventory Management Lessons from Industry Giants[6]**
   1. Examples: Nike, Best Buy, Target Canada, and KFC.
   2. Challenges and Failures:
      1. Excess inventory and inaccurate demand forecasting (Nike).
      2. Stock-outs during peak seasons (Best Buy).
      3. Poor implementation of new systems leading to operational breakdowns (Target).
      4. Supply chain disruptions causing widespread store closures (KFC).
   3. Lessons Learned
      1. The importance of robust forecasting systems, reliable supply chain partners, and scalable technology solutions.

The challenges and successes highlighted in the case studies and industry examples provide key insights framing the context of this project. From Apple's adoption of just-in-time manufacturing to Topgolf's implementation of automated inventory tracking, it is clear that technological integration and strategic planning are key to effective inventory management. Equally, failures like Nike's demand forecasting issues or Kohl’s struggles with excess inventory demonstrate the consequences.

This project seeks to build on these lessons by examining how businesses, especially smaller or resource-constrained ones, can adopt scalable, adaptable inventory management practices tailored to their specific operational needs. While current research and case studies emphasise the importance of advanced technology and robust forecasting systems, this project will explore how these principles can be effectively applied in smaller organisations by positioning the project within the broader context of successful and failed inventory management strategies.

# Methodology

## Proposed Development Methodology

### Comparison of Agile with Other Methodologies

The Agile methodology was chosen for this project due to its flexibility, iterative nature, and focus on delivering a Minimum Viable Product (MVP). In comparison, the Waterfall methodology, a traditional chronological approach, requires the completion of each phase before moving to the next. While this offers a structured process, it lacks adaptability against changing requirements or incorporating user feedback during development. Waterfall would be less effective for projects like this inventory management system, where requirements may evolve based on user input.

On the other hand, Scrum, a methodology of Agile, is also highly iterative and collaborative but differs in that it focuses on predefined roles and time-boxed sprints with strict deliverables. While this could have been a viable alternative, its rigid structure and role definitions may not align well with the project's scale and resource constraints. Agile's broader framework provides adaptability without the added complexity of defined roles.

By choosing Agile, the project benefits from a balance between structured progress and the ability to iterate and adapt, ensuring that core functionalities are prioritised through the MoSCoW framework while leaving room for enhancements based on feedback and testing during development.

The MoSCoW prioritisation is a technique which divides requirements and features into the following four groups:

* Must have - Essential features that are required for the basic functions of the system
* Should have - Useful features that should be delivered if there is time
* Could have - Useful features that could be delivered but have less priority than “Should have”
* Won’t have - Features that won’t be present in the current model

### Benefits of implementing Agile Methodology

This approach allows for the development of core functionalities as a priority to deliver an MVP and follow the MoSCoW prioritisation to ensure that key functions are offered, and extra features can be added later. This also provides the opportunity to review sprints to identify potential improvements that can be implemented, and also how the system functions as a whole with the new additions. This approach also emphasises the importance of user feedback. Due to the product being used entirely by the client, getting their input during the development process through building usable prototypes is vital to ensure the successful delivery of the product. Another benefit is that it can account for all ranges of finances, and breaking the project into smaller increments can help identify and mitigate risks, as functional prototypes at each sprint provide the opportunity to identify risks and address them either during the sprint or at the next sprint. As IMS are crucial to business operations, being able to address integration issues, user adoption, and adjusting requirements is vital to ensure the success of this project.

It is also a common practice for agile development to utilise entity relationship diagrams (ERD). It is especially beneficial in this case, given that this project involves databases, as it provides a clear and adaptable representation of data flow and structure. This provides a high-level plan that can include details of relationships, entity attributes and promotes collaboration because this can be shared with those involved to get feedback to make improvements. It can also help with prototyping as it provides a design basis in the early stages of development and ensures it aligns with user requirements. It can also display complexities and dependencies to help prioritise essential tasks and potential prerequisites for other functions.

### Challenges in Implementing Agile Methodology

Implementing Agile for this project presents unique challenges, one significant hurdle is the resource constraint due to constant iterations, constant testing and user feedback, which all require consistent engagement and dedicated developers. Agile thrives on collaboration and iterative feedback, which can be difficult without a team, leading to time management issues and potential bottlenecks in the development process.

Another challenge is managing stakeholder expectations effectively. Engaging with stakeholders or potential users to gather regular feedback can become cumbersome without a dedicated team. Ensuring clear communication and maintaining a realistic timeline while incorporating iterative feedback cycles can demand significant effort. The lack of peer reviews and collaboration can result in limited perspectives on problem-solving and design decisions.

Maintaining the adaptability of Agile while adhering to its core principles can also be demanding for this project. Managing sprints, defining priorities using frameworks like MoSCoW, and continuously assessing and revising the development strategy require discipline and meticulous planning. Furthermore, addressing integration issues, ensuring usability, and building a robust, user-friendly system without support from a team might lead to increased workload and stress.

### Solutions for Challenges

To address these challenges, Agile practices and tools will be utilised specifically for this project:

1. Simulation of collaboration with tools: As the agile methodology is team-focused, I will simulate team collaboration using JIRA to plan and track tasks effectively. This will allow requirements to be broken down into manageable tasks in sprints, visually display project progress and prioritise tasks/features using the MoSCoW framework. For this project, I will use JIRA to help track feedback to ensure “Must Have” functionalities are prioritised.
2. Continuous Integration (CI) and Continuous Delivery (CD): One of the key practices in agile methodology is continuous integration (CI) and continuous delivery (CD), which focuses on developing and deploying code but also running automated testing. This will provide the scope for quality assurance. For this project, I will use testing frameworks to validate data processing functionalities at each sprint; automated testing can proactively identify issues and resolve them quickly to prevent delays during testing.
3. Self-Reflection: The emphasis on CI during sprint reviews can help reflect on progress and compare it with expected progress. For this project, I will use each sprint as an opportunity to reflect on progress by comparing outcomes against objectives and feedback to assess the success of implementing features.
4. Documentation and Version Control: Due to the collaborative nature, having version control and documentation of changes is essential to Agile. For this, I will use GitHub for version control, enabling me to track changes and revert to previous versions if needed. This will be utilised with code comments to explain what each code segment does, and commit comments to also understand the changes for each new version.

## Tools and software used

MySQL Workbench: This will be used to design, manage and implement the database for the application. The ERD will be used as the structural basis for entity creation, entity relations and queries for the system functionalities. Workbench also can produce ERDs which represent the databases’ structure based on the databases created within its schemas, ensuring adherence to the project requirements.

Visual Studio Code (VS Code): This is the IDE that will be used to manage and write the code for the frontend and backend. Python will be written for the backend, while HTML, CSS and JS will be written for the frontend to provide a user interface that is integrated with the database, with testing conducted throughout. VS Code supports multiple programming languages and frameworks while having plugins that support integration and testing.

GitHub: This will be used for version control, documentation, and to simulate collaboration. By regularly committing with detailed descriptions, ensuring that changes are tracked. Along with branching for the development of features, with testing to ensure complete integration before merging with the main branch to minimise the risk of errors. Version control is an essential part of agile development, and GitHub’s features allow tracking and reverting of changes and documenting changes.

Postman: This will be used to test API endpoints and verify database interactions. The verification of endpoints for creating, reading, updating and deleting (CRUD) data ensures that the API meet project requirements with smooth database integration. This is crucial because this confirms that data is accurately communicated between the database and the backend, and also, Postman allows this to be tested in a user-friendly way.

Python: This will be used as the main programming language for backend development, using frameworks for the integration. This will set up help functions connect with the database to handle requests via a MySQL connector, and also implement logic with efficient and manageable code. Python, combined with the available frameworks, allows for flexibility and scalability when creating APIs and handling database interactions. It also has a vast library that supports testing.

## Development Methodology

1. Planning and Requirements Gathering
   1. Outline the system's goals, such as tracking inventory, managing orders, and generating reports.
   2. Decide on core functionalities like user management, stock management, order tracking, and transaction history.
   3. Understand what data needs to be stored and processed (e.g., products, warehouses, stock levels).
2. Database Design & System Architecture
   1. Finalise the ERD and SQL scheme, including relationships between entities
   2. Plan API endpoints for database interactions
   3. Create mock-ups for the website interface
3. Set Up the Environment
   1. Install Python
      1. Install Flask: pip install flask
      2. Install MySql Connector: pip install mysql-connector-python
   2. Set up MySql
      1. Install MySql Server and a database management tool like phpMyAdmin or MySql Workbench.
   3. Prepare frontend tools (e.g. VS Code and Chrome)
4. Design the database and Backend
   1. Create the required SQL tables
   2. Set up a web framework
      1. Set Up Flask (Create a Python file (app.py) to handle server-side logic.)
   3. Connect the backend to MySql
      1. Use mysql.connector to connect to the database
      2. Install pip install flask-cors
   4. Create API endpoints
      1. For Example: /add-product: adds a new product, get, update and delete
5. Create the Frontend
   1. Structure the website using HTML, JS and CSS
6. Test the system
   1. Run the Flask server: python app.py
   2. Open HTML to test the form
   3. Verify that the data is inserted correctly in MySql
7. Deploy the system
   1. Connect the front and backend
8. Maintain and enhance the system

# Requirements, Specification and Design

## Analysis of Requirements and Design of the Proposed System

System: The project involves developing an inventory management system (IMS) to address challenges in stock tracking, order processing, and report generation for a small-to-medium enterprise. Key issues include real-time inventory updates, user-friendly interfaces, and scalable database architecture. Stakeholder feedback revealed the necessity for intuitive user management and customizable reports.

Software: The software should support CRUD operations for inventory, provide role-based access for users, and enable seamless data visualisations through dashboards. The software must be scalable to handle increasing data loads as the business grows.

Data Processing/AI/Machine-Learning Pipeline: Initial pipeline requirements consist of data collection from user inputs and transactional logs, data preprocessing to identify trends in stock movement and implementation of predictive analytics to forecast inventory needs using machine learning in future iterations. The initial design will be a modular pipeline with clearly defined preprocessing, processing, and visualisation stages, ensuring easy debugging and updates.

## Requirements Specification

Some example functional requirements for the system will be:

* Inventory Management:
  + Add, update, delete, and view product details.
  + Track stock levels in real-time.
* Order Processing:
  + Process purchase orders and generate invoices.
  + Notify users about low stock levels.
* User Management:
  + Role-based access (Admin, Manager, Staff).
* Report Generation:
  + Generate reports on stock levels, sales, and order history.

Non-Functional Requirements for the system will be:

* Budget to be under £500.00.
* Project completion within six months.
* Compatible with Windows and web browsers.
* Backend hosted using Python
* Database hosted on MySql.

Performance Requirements:

* System latency under 500ms for all operations.
* Support for concurrent users (minimum 8).

Verification Requirements:

* API tests via Postman to ensure functional integrity.

Validation Criteria:

* User acceptance testing with real-world data from stakeholders.

## Functional Specification

System Expectations:

* Relational database with entities for Products, Users, Orders, and Reports.
* Intuitive UI for data entry and dashboards for visual analytics.

## Comparison of Methods, Algorithms, and Data Structures

The system structure will follow a modular design, allowing for isolated updates to specific components (e.g., frontend vs backend) along with dataflow, providing real-time updates between the frontend UI and MySql database using RESTful APIs providing seamless integration between the user interactions and backend processes. This provides ease of maintenance, clear separation of concerns and simplified debugging.

I will be using CRUD (Create, Read, Update and Delete) operations as part of MySql queries as a priority, which will utilise direct SQL queries. I have chosen this method because this is a more direct and efficient method for this particular use case compared to methods such as Object-Relational Mapping (ORM), as while ORM is easier for developers, it has the downsides of performance overheads for complex queries and less control over the system [11]. Given extra time, I will try to implement predictive analytics using machine learning algorithms such as ARIMA. The reason ARIMA has been chosen in this scenario is that it is highly effective with time series analysis to detect trends and seasonality to help identify optimal stock and sales operations for the business.

I will be using a relational database normalised to Third Normal Form (3NF) to reduce redundancy, and also JSON for the API data format for efficient data transfer. A relational database design has been chosen because it ensures data accuracy and consistency, handling complex data relationships and ACID (Atomicity, Consistency, Isolation, Durability) compliance. Compared to other databases like NoSql databases, which can have data discrepancies and more complex queries when handling relationships between different entities, which can have serious consequences, as mismanagement of stock can lead to major losses. JSON has been chosen because of its readability, ubiquitous support, flexibility and its effectiveness with web interactions; meanwhile, other formats such as XML are more resource-intensive and more complex, making them less user readable.

The system is planned to have simple role-based access control (RBAC), with reports generated depending on user permissions. The system will also use real-time inventory management to mitigate stock issues, along with a modular software architecture to ensure scalability and maintainability. This design also provides opportunities for scalability if required and potential AI integration for long-term goals.

# Development

## Planning and Requirements Gathering

### Planning

The objective of the project is to develop a robust inventory management system (IMS) to streamline stock tracking, processing orders and generating reports for small enterprises. Stakeholders (e.g. managers, staff and IT personnel) will need to be identified and consulted to gather requirements, by using methods such as surveys and interviews to understand required and desired functionalities, interface design and bottlenecks when using the current system and also the development timeline. The proposed timeline will be:

* Gathering requirements: 2 weeks
* Designing: 3 weeks
* Developing: 10 weeks
* Testing and deployment: 3 weeks

### Requirement Gathering

* Functional Requirements:
  + Inventory Management
    - Create, Read, Update and Delete product details
    - Real-time stock updates
    - Tracking the movement of stock
  + Order Processing:
    - Process orders
    - Generate invoices
    - Notify users with relevant permissions about low stock levels
  + User Management
    - Role-based access controls (Admins, Managers and staff)
    - Maintaining user records with secure login credentials
  + Report Generation
    - Generate detailed reports on stock levels, sales and order history
    - Export generated reports as PDFs or Excel documents
* Non-Functional Requirements
  + System latency under 500ms for common operations
  + Support at least 8 concurrent users
  + Compatibility with Windows and web browsers
  + Secure hosting environment
  + Scalable design to accommodate future growth
* Data Requirements
  + Relational database design using normalised schema (3NF)
  + Key entities within the database are Users, Orders, Reports, Products and Invoices
* Validation requirements
  + Postman to test API requests
  + User acceptance testing (UAT)

### Deliverables

* A fully functional IMS with order processing, user management and reporting
* User documentation and training resources
* Deployment and product maintenance post-launch

### Risk Analysis

* Risks
  + Delays in stakeholder feedback - Unavailability of stakeholders during review phases or delayed approvals/modifications can delay progress potentially causing missed deadlines.
  + Budget constraints influencing tool selection - The budget restricts the choice of software or tools used for services, leading to potential compromises.
  + Scalability issues with initial design - If the initial design isn’t optimised for growth, the addition of new features or increased demands becomes more challenging.
  + System downtime during deployment - A poorly planned migration process from development to production can result in prolonged downtime.
  + Data loss during migration - Errors or omissions during data transfer to the new database can lead to missing or corrupt data, affecting operations.
  + Users being hesitant to use the new system - Staff may be accustomed to the existing system and workflows.
  + Hardware limitations - Local machines within the intended environment may not be capable of handling large datasets
  + Cybersecurity threats - Targeted attacks such as SQL injection or phishing can compromise data and result in a violation of compliance.
* Mitigation
  + Frequent progress reviews - Scheduling of regular check-ins with stakeholders ensures consistent feedback, minimising bottlenecks during the decision-making process.
  + Prioritise critical features using minimum viable product (MVP) - Focus on delivering essential features while deferring advanced ones, which allows early deployment.
  + Design using modular architecture for future scalability - This breaks down the system into separate components to facilitate easier updates or enhancements without interrupting the whole system.
  + Develop and test in phases - By deploying the system in stages, this helps to minimise disruptions and resolve issues incrementally.
  + Data backup and validation during migration - Implementation of automated tools to validate data accuracy and maintain backups of original data.
  + User training and support - Provision of workshops, tutorials and a helpdesk can reduce the learning curve and address concerns.
  + Monitor and optimise system performance - Regularly evaluating the host machine's resources and adjusting configurations or upgrading hardware where necessary.
  + Implementation of security measures - Use of input validations and enforcing strong password policies.

## Database Design & System Architecture

### Database Design

The database design for the IMS will utilise a normalised relational database structure adhering to 3NF that will allow maintaining data integrity and reducing redundancy, and also ensuring that the database is efficient and scalable. It will consist of the following entities (see below an ERD[Figure 1] displaying the entities and their relationships):

A screenshot of a computer screen

Description automatically generated

(Figure 1)

* Users
* Product\_Categories
* Products
* Orders
* Order\_Items
* Reports
* Invoices

It will also consist of the following relationships:

1. **Users**:

* Linked to **Orders** through customer\_id (a user can place multiple orders).
* Linked to **Reports** through user\_id (a user can generate multiple reports).
* Linked to **Invoices** through customer\_id (a user can have multiple invoices).

1. **Product\_Categories**:

* Linked to **Products** through category\_id (a category can have multiple products).

1. **Products**:

* Linked to **Orders** via **Order\_Items** (a product can be part of multiple orders).

1. **Orders**:

* Linked to **Order\_Items** through order\_id (an order contains multiple items).
* Linked to **Invoices** through order\_id (an order generates one invoice).

1. **Order\_Items**:

* Links **Orders** and **Products** (many-to-many relationship).

1. **Reports**:

* Linked to **Users** through user\_id (reports are generated by users).

1. **Invoices**:

* Linked to **Orders** through order\_id (one invoice per order).

Based on the decided database design, it adheres to the 3NF relational database structure as:

* Each table represents a single entity or relationship with no repeating groups.
* Functional dependencies ensure that no partial dependency exists in any table.
* Transitive dependencies are eliminated, ensuring all non-key attributes are dependent only on the primary key.

### System Architecture

The system will be deployed locally initially to ensure that it is compatible with the planned hardware, and at a later date, it will have a client-server architecture for seamless communication and will use the following software:

* Frontend: An HTML interface utilising JS and CSS for a user-friendly interface
* Backend: A Python Flask application to handle business logic and API endpoints
* Database: MySql installed locally to interact with the backend

The system will have the following features:

* Real-time operations: Local hosting ensures minimal latency when users interact and use the system
* Scalability considerations: The system architecture will allow transition to cloud hosting when required
* Security measures: The use of RBAC and local firewall configurations will protect the system during deployment

While initially the system will be deployed locally, the design will consider scalability in the case of expanding requirements, future growth and transitioning to a cloud-based setup (e.g. AWS and Google Cloud). This allows for increased user capacity, remote access for multiple locations, improved data redundancy and data recovery measures.

## Hardware and Software Requirements

### Software Requirements:

* Operating systems (OS) - Windows 10/11
* Database - MySql community server
* Backend - Python with Flask
* Frontend - HTML5, CSS3 and JS
* Dependencies/Packages:
  + Flask (As part of Python)
  + MySql connector (Executing SQL queries from Python)
  + Pandas (Data processing and report generation)
  + CSS (for frontend styling)

### Hardware Requirements:

* Minimum hardware specifications:
  + Processor - Dual-core 2.0 GHz or higher
  + RAM - 4GB
  + Storage - 20 GB free space

## Frontend prototype

Proposed design for the frontend:

A screenshot of a graph

Description automatically generated

(Figure 2)

A screenshot of a website

Description automatically generated

(Figure 3)

A screenshot of a computer

Description automatically generated

(Figure 4)

## Design the database and Backend

The database has been designed with the entities and their relationships as described using the following code snippet within MySql:

Users table

A screenshot of a computer code

Description automatically generated

(Figure 5)

Product\_Categories table

A close-up of a computer code

Description automatically generated

(Figure 6)

Products table

A screen shot of a computer

Description automatically generated

(Figure 7)

Orders table

A screen shot of a computer code

Description automatically generated

(Figure 8)

Order\_Items table

A screen shot of a computer code

Description automatically generated

(Figure 9)

Reports table

A screenshot of a computer code

Description automatically generated

(Figure 10)

Invoices table

A computer code with text

Description automatically generated with medium confidence

(Figure 11)

Backend code:

db\_util.py

A screen shot of a computer program

Description automatically generated

(Figure 12)

mysql\_connect.py

A screen shot of a computer

Description automatically generated

(Figure 13)

app.py (see the app.py file in the repository)

## Create the Frontend

The pages for the frontend have been created using HTML and CSS. The initial designs have sample data but I have tried to replicate the design, so it is uniform for all pages I will aim to integrate login credentials and the API requests for the relevant pages.

Dashboard

A screenshot of a computer

Description automatically generated

(Figure 14)

Orders

A screenshot of a computer

Description automatically generated

(Figure 15)

Invoices

A screenshot of a computer

Description automatically generated

(Figure 16)

Products

A screenshot of a computer

Description automatically generated

(Figure 17)

Inventory

A screenshot of a computer

Description automatically generated

(Figure 18)

Reports

A screenshot of a computer

Description automatically generated

(Figure 19)

## Test the system

To test the functionality of the proposed system and entity relationships, I created sample data to ensure that it acts as planned and verified the foreign keys act as intended via invalid SQL inputs:

An invalid SQL input for the Users table that misses out the username, as that is a required input

A close-up of a computer code

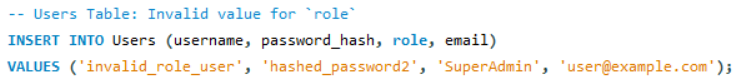
Description automatically generated

(Figure 20)



(Figure 21)

An invalid SQL input for the Users table that inputs a non-approved value

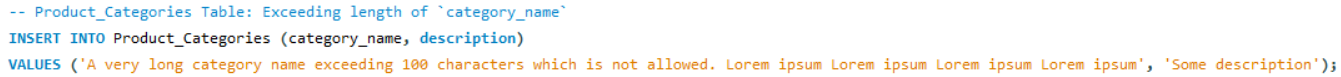


(Figure 22)



(Figure 23)

An invalid SQL input for the Product\_Categories table that inputs a value into the category\_name that exceeds the allowed character limit

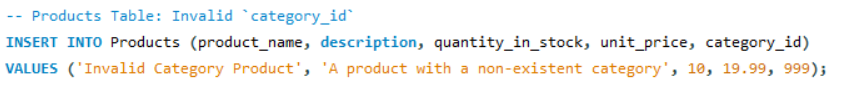


(Figure 24)

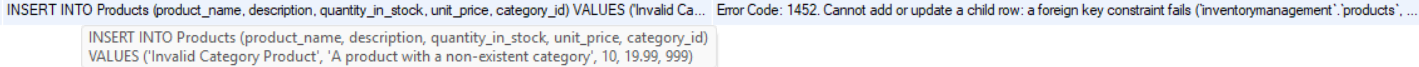


(Figure 25)

An invalid SQL input for the Products table that tries to reference a non-existent category\_id

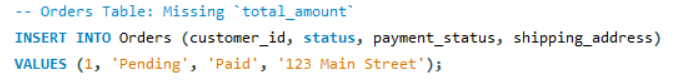


(Figure 26)



(Figure 27)

An invalid SQL input for the Orders table that is missing the total\_amount field

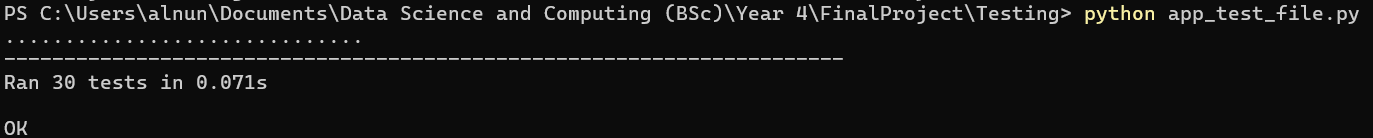


(Figure 28)



(Figure 29)

I have also conducted unit testing for the Python code connecting the frontend to the backend for the following files: app.py, db\_util.py and mysql\_connect.py. Tests have been written and performed for all scenarios, with every test passing.



(Figure 30)

A screen shot of a computer

Description automatically generated

(Figure 31)

A black screen with white text

Description automatically generated

(Figure 32)

# Implementation

For the implementation of this project, and based on the requirements initially set out, core system functions have been satisfied as the system has CRUD functionality for all major entities. While also handling data manipulation for the database via the frontend to minimise errors, and data validation only to allow permitted values and enforcing required values based on entity setup. RESTful APIs were developed using Flask, which handles data manipulation and retrieval for the frontend interface. The use of the APIs also provides efficiency for users as any operations they need (e.g. Create, Read, Update or Delete) are handled using forms or buttons.

Below is an example where I am adding a new product where all relevant fields are filled out[Figure 33], and the “Create” request is sent to the database, which appears in the table that is auto-populated for all existing products on the product page[Figure 34]. I verified that it has appeared in the database by running an API request on Postman, searching specifically for the newly generated product, which appears[Figure 35]. The user is then able to delete individual products via a delete button at the end of every product in the list[Figure 34] when they click “Delete”, they are prompted with a confirmation message to check they have not clicked the link accidentally[Figure 36]. Once the user confirms they want to delete that record, it is then removed from the database[Figure 37]. This is one of the key functionalities of the system, with the same implementation being applied to all relevant entities. There is also data validation in place to only accept the correct data type in each field, and also provision of a dropdown list for product category to only allow existing fields.

A screenshot of a computer

Description automatically generated

(Figure 33)



(Figure 34)

A screenshot of a computer

Description automatically generated

(Figure 35)

A screenshot of a computer

Description automatically generated

(Figure 36)

A screenshot of a computer

Description automatically generated

(Figure 37)

Key features have also been included in the system where the frontend is dynamically built using HTML, CSS and JavaScript to allow the tables and forms to be updated so that they appear in the relevant areas. For example, when a user wants to delete a record on the system, they do not need to refresh the page to see that it has been removed; they will see an alert message, and once they clear the alert message, the record is subsequently removed from the database and the frontend. Also

Also, triggers have been set up for the system so that an invoice is automatically created once an order is created. Also, when an order is updated, it is updated for the invoice where relevant, such as updating the status, total amount, or even deleting an order from the system[Figures 15 & 16]. Also related to orders, when creating or updating an order, when users select a product and the quantity for an order, it dynamically calculates the price without the user having to reduce the risk of human error.

Also, as the user will only be interacting with the database via the frontend, any unnecessary details are avoided, introducing simplicity for users and ease of training when required. The system also utilises Flask-CORS, which enables cross-origin requests to allow users to access data from multiple entities within the same page without having to change pages. The schema has been represented in the ERD[Figure 1].

Testing has also been conducted to ensure that all API requests perform as expected, database validation, unit testing for the development code and errors are handled appropriately ([here](#_Test_the_system)). Due to time constraints, some features are currently still in development, such as integration of user authorisation, support for concurrent users and online deployment to allow use across multiple systems.

# Analysis

## Test and Evaluation Plan

The test and evaluation plan for this project was meticulously designed to ensure that all system functionalities and performance requirements were met. The primary focus was on validating system behaviour across different scenarios to ensure reliability, usability, and accuracy. A combination of manual testing, automated unit testing, data validation testing, and user testing was employed. The methods used to verify the system against its requirements included the following:

* Software Testing

Core system functionalities, particularly CRUD operations (Create, Read, Update, Delete), were tested via unit testing. Key aspects included:

* Testing for expected behaviour under normal conditions, such as successful record creation or retrieval.
* Identifying system robustness by simulating unexpected or erroneous inputs to ensure that proper error-handling mechanisms were triggered.
* Ensuring comprehensive test coverage by creating test cases for all essential entities (e.g., Products, Users, Orders).
* Unit tests simulated scenarios like missing mandatory fields or exceeding field constraints to confirm that appropriate error messages were displayed, and invalid operations were blocked.
* Simulation

Simulations were conducted to evaluate CRUD operations within the database. The goals were:

* To verify that records were accurately created, updated, and deleted.
* To ensure changes persisted in the database as expected after each operation.
* To identify potential system failures in data processing or transaction handling.
* Simulations involved scripted sequences of database interactions that mimicked real-world usage patterns (e.g. creating an order automatically creates an invoice that mirrors the details)
* Data Validation Testing

Data validation tests ensured that the system only accepted and processed accurate and acceptable inputs. This involved:

* Testing input forms to confirm that constraints (e.g., maximum lengths, allowed formats, required fields) were enforced.
* Validating database rules, such as foreign key constraints and unique identifiers, to maintain data integrity.
* Verifying invalid inputs, such as incorrect data types or out-of-range values, was rejected with clear error messages displayed to the user.
* Fields like "Status" in “Orders” were tested to ensure they only accepted the provided Enum values, and invalid attempts triggered appropriate errors.
* User Acceptance Testing (UAT)

UAT was conducted to evaluate the system's usability and confirm that it met end-user requirements. This involved:

* Preparing user test cases for the frontend, focusing on workflows users were likely to follow.
* Testing the user interface by observing system behaviours during user interactions.
* UAT included verifying that users could easily create and manage orders through the interface and that all operations provided clear success or error feedback.

### Features tested

Testing was carried out using manual and automatic methodologies, focusing on edge cases and critical paths. The features to be tested for the project were:

|  |  |  |  |
| --- | --- | --- | --- |
| Feature | Test Description | Status | Test Method |
| Users Management | Creating, reading, updating, and deleting | Passed | Unit testing, simulations, data validation, and UAT |
| Product\_categories Management | Creating, reading, updating, and deleting | Passed | Unit testing, simulations, data validation, and UAT |
| Products Management | Creating, reading, updating, and deleting | Passed | Unit testing, simulations, data validation, and UAT |
| Orders Management | Creating, reading, updating, and deleting | Passed | Unit testing, simulations, data validation, and UAT |
| Order\_Items Management | Creating, reading, updating, and deleting | Passed | Unit testing, simulations, data validation, and UAT |
| Reports Management | Creating, reading, updating, and deleting | Passed | Unit testing, simulations, data validation, and UAT |
| Creating invoices when an order is created | Creating | Passed | Simulations |
| Updating and deleting invoices upon changes in an order | Updating and deleting | Passed | Simulations |
| Provision of prefilled product\_category fields on the frontend | Reading | Passed | Simulations and UAT |
| Automatically displaying table records on the frontend | Reading | Passed | Simulations, data validation, and UAT |
| Mysql connection with the frontend | Connecting | Passed | Unit testing and simulations |
| Searching for individual value on frontend | Filtering | Passed | Simulations and UAT |

### Unit testing demo

A key component of testing is unit testing. Below is an example unit test to validate the create product function for the system.

A screen shot of a computer program

Description automatically generated

(Figure 38)

This test checks whether the product creation functions correctly by inserting a mock record into the database. It is located in the following directory of the project /Testing/app\_test\_file.py.

### Conclusion

The system underwent comprehensive testing through a multifaceted approach that included unit testing, simulation, data validation, and user acceptance testing (UAT).

Unit Testing ensured the core functionalities, such as CRUD operations for all entities, were robust and performed accurately under various conditions. These tests validated the system’s behaviour with expected and unexpected inputs, confirming proper error handling and resilience to edge cases.

Simulations provided a controlled environment to assess how CRUD operations interacted with the database, verifying that changes to records were correctly executed and persisted as intended. These tests also highlighted potential bottlenecks or errors in the data processing pipeline, allowing for timely corrections.

Data ValidationTesting focused on safeguarding the accuracy and integrity of the data. This step ensured that only valid and acceptable inputs were processed and stored, while invalid entries were appropriately flagged and rejected. The system’s ability to enforce these constraints guaranteed reliable and consistent data throughout its lifecycle.

Finally, User Acceptance Testing (UAT) evaluated the system’s frontend usability and adherence to user requirements. By following predefined test cases, this phase confirmed that the user interface was intuitive, accessible, and met the needs of its intended users. The feedback from UAT was instrumental in refining user workflows and ensuring the overall solution aligned with user expectations.

In conclusion, the combination of these testing methods not only validated that key system functions were operating correctly but also demonstrated the reliability, stability, and readiness of the system for deployment in a real-world environment. This robust testing process provided confidence in the system's ability to handle operational demands while maintaining data integrity and delivering a seamless user experience.

# Results

The project yielded a functional system that met the majority of requirements and achieved the key system objectives. Below is a breakdown of the results, supported by relevant screenshots, tables, and figures to demonstrate the system's performance and behaviour across various metrics.

## Key Outcomes

### Data Collection

* + - Comprehensive datasets for CRUD operations on entities like Products, Users, Orders, and Inventory were successfully managed, validated, and processed.
    - Test data used for simulations and validations confirmed data integrity under multiple scenarios.

A screenshot of a computer

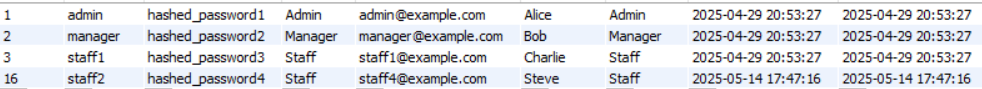
Description automatically generated

(Figure 39)

A close-up of a computer screen

Description automatically generated

(Figure 40)

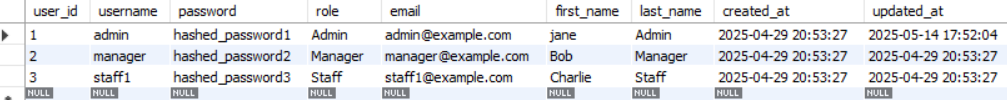


(Figure 41)

A screenshot of a computer

Description automatically generated

(Figure 42)



(Figure 43)

The above screenshots demonstrate system functionality for CRUD operations. I have used the users table for my example, where I have displayed adding[Figure 40], reading[Figure 41], updating[Figure 42], and deleting[Figure 43].

### Working System Demonstration

* + - The system was thoroughly tested, and its functionalities were verified through unit tests, user acceptance testing, and data simulations.

A screenshot of a computer

Description automatically generated

(Figure 44)

A screenshot of a computer

Description automatically generated

(Figure 45)

A screenshot of a computer

Description automatically generated

(Figure 46)

A screenshot of a computer

Description automatically generated

(Figure 47)

A screenshot of a computer

Description automatically generated

(Figure 48)

A screenshot of a computer

Description automatically generated

(Figure 49)

The above screenshots show the frontend page for products and the form displayed[Figure 44,45&46] when trying to create a new product, along with the data validation[Figure 44,45] and alerts when an invalid or blank field is present. It also shows the provision of all existing product\_categories as a pre-filled dropdown[Figure 46] that users would select from which is used to populate the product\_category[Figure 48&49] field based on their selection and this data is then displayed onto the Inventory page[Figure 49].

The below table is a summary of the unit tests created and ran with the metrics and whether they passed[See [table 1](#_Table_1) in the Appendices].

## Experiments and Simulations

### Validation of data integrity constraints

The objective of this testing was to ensure that the system accurately processes, validates, and stores data inputs in the local environment without introducing errors or inconsistencies. Proper data integrity is essential for the functionality and reliability of the system, especially given its dependency on database operations. As such the following tests were conducted:

[See [table 2](#_Table_2) in the appendices]

## Error Analysis

**Observed Errors**

1. **Data Entry Errors**

* Some fields allowed invalid values (e.g. quantity in stock allows negative values)[Figure 62]
* Adding extra validation measures to only allow values equal to or greater than 0

## Limitations and Observations

### System Limitations

Currently the system is setup to be deployed on a singular machine locally without any testing with online infrastructures for deployment, as such it is unknown how the system will operate with concurrent users and if there are any performance or system issues with virtual deployment. Issues that may arise are:

* Installation on multiple devices concurrently
  + - Simultaneous setup on multiple devices may overload the database, leading to slow responses or connection timeouts
    - Concurrent installations may lead to incomplete of conflicting configurations if not synchronised properly
    - During the setup phase, unauthorised devices may attempt to connect, presenting security risks
* Setup of the system over a wireless connection
  + - Unstable/slow network connections can lead to incomplete setup or data loss
    - Some required ports or protocols may be blocked hindering setup if firewalls or router have network configurations
    - If an unsecured connection is used at any point for the system, it can expose the system to potential attacks or breaches
* Scaling of the system when required
  + - The addition of new modules or modifying existing ones may introduce bugs or compatibility issues
    - Increased operational and/or cloud costs if scaling to handle increasing load or users
    - Increased complexity as larger systems require more complex tools for basic operations
* Concurrent system users
  + - A high number of concurrent users can lead to database locking, slow performance or blocking of operations
    - Multiple users accessing or modifying the same data simultaneously can cause data inconsistencies
    - Mismanaged session management can lead to unauthorised access or user conflicts
* Compatibility with screen readers
  + - Some UI elements may not be fully compatible with screen readers creating usability issues
    - Insufficient testing across different screen readers can miss compatibility issues, especially less commonly used devices
    - Implementation of compatibility features such as ARIA can increase code complexity, potentially impacting performance

### Observations

Positive observations

* + **System Functionality**
    - The system successfully handles CRUD operations for all core entities, ensuring data integrity and consistent performance during local testing.
    - Validation mechanisms effectively reject invalid inputs, maintaining database accuracy.
* **User Interface**
  + - The frontend design was intuitive and user-friendly during user acceptance testing (UAT)
    - Responsive elements ensure smooth navigation and interaction across different screen sizes.
  + **Data Management**:
    - Foreign key constraints and database rules ensure seamless linking between related data tables.
    - Simulations confirmed accurate processing of edge cases, highlighting the system's robustness under varied inputs.
* **Error Handling**
  + - Clear and descriptive error messages guide users when invalid operations are performed.
    - Graceful handled expected errors without crashing or affecting other functionalities.
  + **Scalability Potential**
    - The modular structure of the codebase allows for potential enhancements, such as integrating new features or transitioning to an online environment.

Negative Observations

* + **Limited Deployment Environment**
    - The system is currently restricted to local operation, with no testing conducted for deployment in distributed or cloud environments.
    - Lack of performance data for concurrent multi-user scenarios leaves scalability untested.
* **Network Dependency Risks**
  + - The system’s performance over wireless networks has not been evaluated, potentially causing delays or failures in future online deployments.
    - Firewall configurations and port accessibility for future deployments have not been validated.
  + **Accessibility Gaps**
    - Limited testing for accessibility features may result in usability challenges for users relying on assistive technologies, such as screen readers.
    - Absence of ARIA attributes or semantic HTML in some areas may hinder accessibility compliance.
* **Security Concerns**
  + - Operating locally without encryption or authentication mechanisms makes the system vulnerable to unauthorized access.
  + **Unexplored Extensibility**
    - Adding new features or modifying existing functionalities may introduce unforeseen bugs due to insufficient integration testing.

**Recommendations**

Addressing the negative observations through additional testing, security enhancements, accessibility upgrades, and infrastructure planning will significantly improve the system's readiness for broader deployment and scalability.

# Discussion

Critical evaluation of the results

● Reflection on the approach followed and the project in general

➢ Include strong and weak points

➢ Lessons learn

➢ Design decisions that could have been made differently, given the experience of the project

➢ Ways in which the project could be improved, or extended

# Conclusion

Were the original aims and objectives, achieved:

➢ If a hypothesis was to be proved, was it proven?

● The project does not need to be a success

✔ Good understanding of why it succeeded, or failed, must have been demonstrated

● What challenges were faced?

● What should the future direction of the project, be?

● What experience was gained?

● What things did you learn to do better?

# References

[1] - Rinf.tech, 2024. *Demand forecasting and inventory management in retail: Trends and challenges*. [online] Available at: <https://www.rinf.tech/demand-forecasting-and-inventory-management-in-retail-trends-and-challenges/> [Accessed 20 November 2024].

[2] - Fortune Business Insights, 2024. *Inventory management software market*. [online] Available at: <https://www.fortunebusinessinsights.com/inventory-management-software-market-108589> [Accessed 20 November 2024].

[3] - Procurement Tactics, 2024. *Inventory management statistics*. [online] Available at: <https://procurementtactics.com/inventory-management-statistics/> [Accessed 22 November 2024].

[4] - Goods Order Inventory, 2021. *Home appliances retailer*. [pdf] Available at: <https://www.goodsorderinventory.com/wp-content/uploads/2021/08/Home-Appliances-Retailer.pdf> [Accessed 28 November 2024].

[5] - Wasp Barcode, 2024. *Topgolf case study: Inventory control solutions*. [online] Available at: <https://www.waspbarcode.com/inventory-control/topgolf> [Accessed 28 November 2024].

[6] - Asset Panda, 2024. *Catastrophic inventory mistakes by huge brands and how to avoid them*. [online] Available at: <https://www.assetpanda.com/resource-center/blog/catastrophic-inventory-mistakes-by-huge-brands-and-how-to-avoid-them/> [Accessed 2 December 2024].

[7] - Investopedia, 2024. *Inventory management*. [online] Available at: <https://www.investopedia.com/terms/i/inventory-management.asp> [Accessed 2 December 2024].

[8] - Ordoro, 2012. *Nike's inventory management solution*. [online] Available at: <https://blog.ordoro.com/2012/01/24/nikes-inventory-management-solution/> [Accessed 5 December 2024].

[9] - Retail Dive, 2023. *Macy’s inventory: 2022-2023*. [online] Available at: <https://www.retaildive.com/news/macys-inventory-2022-2023/644431/> [Accessed 5 December 2024].

[10] - Food Institute, 2024. *Why inventory distortion costs retailers trillions*. [online] Available at: <https://foodinstitute.com/focus/why-inventory-distortion-costs-retailers-trillions/> [Accessed 22 December 2024].

[11] - Apicraft, 2024. *SQL vs ORM: Choosing the right tool for the job*. [online] Medium. Available at: <https://medium.com/@apicraft/sql-vs-orm-choosing-the-right-tool-for-the-job-e0bc8c6fbe62> [Accessed 10 Jan 2025].

# Appendices

## Table 1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Entity** | **Operation** | **Metrics** | | | **Pass/**  **Fail** |
| Response Time | DB Operations | Status Code |
| Users | Create | Less than 200ms | 1 Insert query | 201 | Pass |
| Read all | Less than 200ms | 1 Select query | 200 | Pass |
| Read one | Less than 200ms | 1 Select query | 200 | Pass |
| Update | Less than 200ms | 1 Update query | 200 | Pass |
| Delete | Less than 200ms | 1 Delete query | 200 | Pass |
| Invalid create | Less than 200ms | None | 400 | Pass |
| Invalid read | Less than 200ms | 1 Select query | 404 | Pass |
| Invalid update | Less than 200ms | 1 Update query | 404 | Pass |
| Invalid delete | Less than 200ms | 1 Delete query | 404 | Pass |
| Product\_categories | Create | Less than 200ms | 1 Insert query |  | Pass |
| Read all | Less than 200ms | 1 Select query | 201 | Pass |
| Read one | Less than 200ms | 1 Select query | 200 | Pass |
| Update | Less than 200ms | 1 Update query | 200 | Pass |
| Delete | Less than 200ms | 1 Delete query | 200 | Pass |
| Invalid create | Less than 200ms | None | 200 | Pass |
| Invalid read | Less than 200ms | 1 Select query | 400 | Pass |
| Invalid update | Less than 200ms | 1 Update query | 404 | Pass |
| Invalid delete | Less than 200ms | 1 Delete query | 404 | Pass |
| Products | Create | Less than 200ms | 1 Insert query | 404 | Pass |
| Read all | Less than 200ms | 1 Select query | 201 | Pass |
| Read one | Less than 200ms | 1 Select query | 200 | Pass |
| Update | Less than 200ms | 1 Update query | 200 | Pass |
| Delete | Less than 200ms | 1 Delete query | 200 | Pass |
| Invalid create | Less than 200ms | None | 200 | Pass |
| Invalid read | Less than 200ms | 1 Select query | 400 | Pass |
| Invalid update | Less than 200ms | 1 Update query | 404 | Pass |
| Invalid delete | Less than 200ms | 1 Delete query | 404 | Pass |
| Orders | Create | Less than 200ms | 1 Insert query | 404 | Pass |
| Read all | Less than 200ms | 1 Select query | 201 | Pass |
| Read one | Less than 200ms | 1 Select query | 200 | Pass |
| Update | Less than 200ms | 1 Update query | 200 | Pass |
| Delete | Less than 200ms | 1 Delete query | 200 | Pass |
| Invalid create | Less than 200ms | None | 200 | Pass |
| Invalid read | Less than 200ms | 1 Select query | 400 | Pass |
| Invalid update | Less than 200ms | 1 Update query | 404 | Pass |
| Invalid delete | Less than 200ms | 1 Delete query | 404 | Pass |
| Order\_items | Create | Less than 200ms | 1 Insert query | 404 | Pass |
| Read all | Less than 200ms | 1 Select query | 201 | Pass |
| Read one | Less than 200ms | 1 Select query | 200 | Pass |
| Update | Less than 200ms | 1 Update query | 200 | Pass |
| Delete | Less than 200ms | 1 Delete query | 200 | Pass |
| Invalid create | Less than 200ms | None | 200 | Pass |
| Invalid read | Less than 200ms | 1 Select query | 400 | Pass |
| Invalid update | Less than 200ms | 1 Update query | 404 | Pass |
| Invalid delete | Less than 200ms | 1 Delete query | 404 | Pass |
| Reports | Create | Less than 200ms | 1 Insert query | 404 | Pass |
| Read all | Less than 200ms | 1 Select query | 201 | Pass |
| Read one | Less than 200ms | 1 Select query | 200 | Pass |
| Update | Less than 200ms | 1 Update query | 200 | Pass |
| Delete | Less than 200ms | 1 Delete query | 200 | Pass |
| Invalid create | Less than 200ms | None | 200 | Pass |
| Invalid read | Less than 200ms | 1 Select query | 400 | Pass |
| Invalid update | Less than 200ms | 1 Update query | 404 | Pass |
| Invalid delete | Less than 200ms | 1 Delete query | 404 | Pass |

## Table 2

|  |  |  |  |
| --- | --- | --- | --- |
| Entity | Input | Expected outcome | Result |
| Users | Valid creation | User successfully created | Pass |
| Missing email field. | Unsuccessful query | Pass |
| Invalid role. | Unsuccessful query | Pass |
| Product\_categories | Valid creation | Category successfully created[Figure 51] | Pass |
| Missing category\_name | Field required[Figure 50] | Pass |
| Products | Valid creation | Product successfully created[Figure 48] | Pass |
| Text instead of number | Enter int instead[Figure 45] | Pass |
| Missing stock quantity. | Field required[Figure 44] | Pass |
| Orders | Valid creation | Order successfully created[Figure 52&53] | Pass |
| Auto calculation of total amount | Total displayed[Figure 54] | Pass |
| Reports | Valid creation | Report successfully created | Pass |
| Non-JSON entry for report data | Invalid datatype[Figure 55] | Pass |
| Invoices | Creation of Invoice upon creation of order | Invoice created with same order details[Figure 15&16] | Pass |
| Updating Invoice upon updating order | Invoice updated with same order details[Figure 56,57&58] | Pass |
| Database | Valid connection settings. | Connected[Figure 59] | Pass |
| Invalid connection | Couldn’t connect[Figure 60] | Pass |
| Disconnected database. | Disconnected[Figure 61] | Pass |
| Frontend | Form inputs with valid data. | Successful update[Figure 56] | Pass |
| Empty required fields. | Fields required[Figure 50] | Pass |
| Incorrect data type inputted | Enter int instead[Figure 45] | Pass |

A screenshot of a computer

Description automatically generated

(Figure 50)

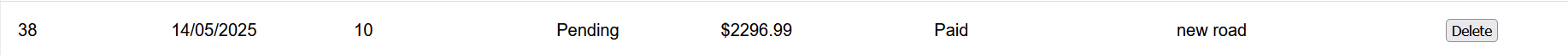
A screenshot of a computer

Description automatically generated  
(Figure 51)

A screenshot of a computer

Description automatically generated

(Figure 52)



(Figure 53)

A screenshot of a computer

Description automatically generated

(Figure 54)

A screenshot of a computer

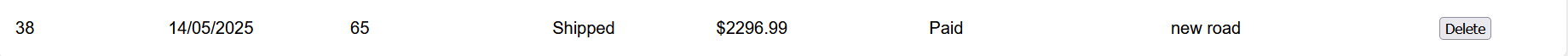
Description automatically generated

(Figure 55)

A screenshot of a computer screen

Description automatically generated

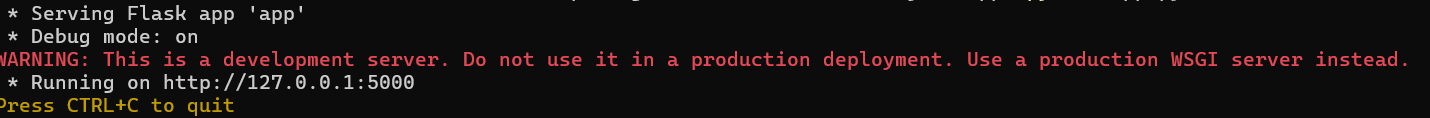
(Figure 56)



(Figure 57)



(Figure 58)

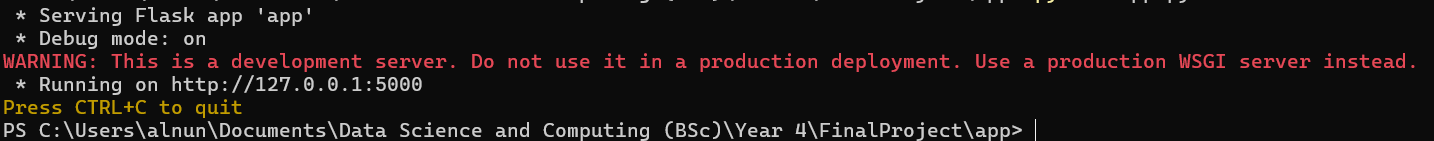


(Figure 59)

A screenshot of a computer

Description automatically generated

(Figure 60)



(Figure 61)



(Figure 62)

User Manual

● Relevant code

● Technical information

● Transcripts of interviews that might have taken place

● Detailed data

● Extensive presentation of test, or evaluation, results

✔ Link to this section from from within the main text in the results, or discussion