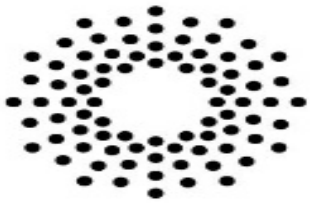

LONDON METROPOLITAN UNIVERSITY

CS6P05 Project

Decentralised Inventory Management System

Project Report

Final Submission

Name:	Kristian Spiropali
ID Number:	20027710
Date:	08/05/2023
First Supervisor:	Dr. Victor Sowinski - Mydlarz
Second Supervisor:	Dr. Sandra Fernando

Declaration

Module: CS6P05 **Deadline: 08/05/2023**

Module Leader: Dr. Ela Homayounvala **Student ID: 20027710**

PLAGIARISM

You are reminded that there exist regulations concerning plagiarism. Extracts from these regulations are printed below. Please sign below to say that you have read and understand these extracts:

Student signature: Kristian Spiropali Date: 08/05/2023

This header sheet should be attached to the work you submit. No work will be accepted without it.

Extracts from University *Regulations* on Cheating, Plagiarism and Collusion

Section 2.3: "The following broad types of offence can be identified and are provided as indicative examples…

(i) Cheating: including taking unauthorised material into an examination; consulting unauthorised material outside the examination hall during the examination; obtaining an unseen examination paper in advance of the examination; copying from another examinee; using an unauthorised calculator during the examination or storing unauthorised material in the memory of a programmable calculator which is taken into the examination; copying coursework.

(ii) Falsifying data in experimental results.

(iii) Personation, where a substitute takes an examination or test on behalf of the candidate. Both candidate and substitute may be guilty of an offence under these Regulations.

(iv) Bribery or attempted bribery of a person thought to have some influence on the candidate's assessment.

(v) Collusion to present joint work as the work solely of one individual.

(vi) Plagiarism, where the work or ideas of another are presented as the candidate's own.

(vii) Other conduct calculated to secure an advantage on assessment.

(viii) Assisting in any of the above.

Some notes on what this means for students:

1. Copying another student's work is an offence, whether from a copy on paper or from a computer file, and in whatever form the intellectual property being copied takes, including text, mathematical notation and computer programs.

2. Taking extracts from published sources *without attribution* is an offence. To quote ideas, sometimes using extracts, is generally to be encouraged. Quoting ideas is achieved by stating an author's argument and attributing it, perhaps by quoting, immediately in the text, his or her name and year of publication, e.g. "$e = mc^2$ (Einstein 1905)". A *references* section at the end of your work should then list all such references in alphabetical order of authors' surnames. (There are variations on this referencing system which your tutors may prefer you to use.) If you wish to quote a paragraph or so from published work then indent the quotation on both left and right margins, using an italic font where practicable, and introduce the quotation with an attribution.

Abstract

This paper presents a decentralised inventory management system designed to offer an optimised, easy-to-use, and performant solution for both online and offline inventory management. The system is built using Java's Spring Boot framework, which offers a robust and scalable solution for building modern, cloud-native applications. The aim is to provide an open-source solution to businesses and industries that do not have the budget or resources to create their own inventory management system.

The proposed system provides features such as user login and registration, admin functionality, security, and performance optimisation. Additionally, it includes an automated kiosk feature, which enables offline inventory management, making it convenient for customers who prefer a physical shopping experience.

The implementation of the system is thoroughly detailed, including architecture, design decisions, and deployment. I have also incorporated various testing and evaluation techniques to ensure that the system is robust, secure, and scalable. Moreover, the system is designed to be highly configurable, allowing businesses to customize it to suit their unique requirements. The system's open-source nature also provides the opportunity for the community to contribute to its development and maintenance. I envision that the system will have a significant impact on industries looking to streamline their inventory management processes while reducing costs.

I believe that the system provides a valuable solution for small and medium-sized enterprises, as it is not only cost-effective but also provides a seamless inventory management experience. The approach of implementing a decentralised inventory management system using open-source technologies provides an alternative to proprietary solutions, which can be expensive and restrictive.

In summary, this paper presents a fully optimised, easy-to-use, and performant decentralised inventory management system built using Java's Spring Boot framework. The implementation includes various features such as user login and registration, admin functionality, security, performance optimisation, and automated kiosk. I believe that such solution will offer an affordable and flexible alternative to proprietary inventory management systems, making it accessible to businesses that would otherwise not have access to such systems.

Keywords: decentralised inventory management system, Java, Spring Boot, online store, automated kiosk, user login, admin functionality, security, performance optimisation, open source, cost-effective, configurable, community, small and medium-sized enterprises.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1 Project topic and rationale

The project topic is a decentralised inventory management system designed to offer a seamless and optimised solution for businesses that require an affordable, customisable and performant inventory management system. The motivation behind this project is to provide small and medium-sized enterprises with an alternative to costly proprietary inventory management systems. The system is built using Java's Spring Boot framework and incorporates features such as user login and registration, admin functionality, security, performance optimisation and an automated kiosk.

The problem with proprietary inventory management systems is that it can be expensive, restrictive and may not cater to the unique requirements of businesses. Moreover, many of these systems are closed source, which limits the ability of businesses to customise and adapt them to their specific needs. By offering an open-source solution, I aim to provide businesses with a cost-effective and flexible inventory management system.

The proposed decentralised inventory management system also aims to address the issue of security, which is a critical concern for businesses dealing with sensitive information such as customer data and financial transactions. By incorporating robust security features such as user authentication, authorisation, and encryption, the system ensures that data is kept secure and protected from unauthorised access. The system's security features also enable businesses to comply with regulatory requirements such as GDPR and PCI-DSS. Overall, the proposed system offers a comprehensive solution that caters to the needs of businesses that require a secure, customisable, and cost-effective inventory management system.

The challenge with this project is to build a decentralised inventory management system that is easy to use, scalable and secure. Additionally, the system must cater to both online and offline inventory management, making it suitable for businesses that operate both online and in physical stores. Another challenge is to ensure that the system is highly configurable, allowing businesses to customise it to suit their specific needs.

Overall, this project is interesting and useful as it provides an affordable and flexible solution for businesses that require a customisable and performant inventory management system. Additionally, the open-source nature of the system provides the opportunity for the community to contribute to its development and maintenance, making it a collaborative effort that benefits everyone involved.

1.2 Project Aims and Objectives

Project Aims:

To develop a fully optimised, easy-to-use and performant decentralised inventory management system using Java's Spring Boot framework: The primary goal of this project is to develop a highly optimised and performant inventory management system using Java's Spring Boot framework. The system should be easy to use and user-friendly, even for those who may not have a technical background.

To provide an open-source alternative to costly proprietary inventory management systems: The aim is to provide businesses with an open-source solution that they can use to manage their inventory without having to pay for costly proprietary systems. This would reduce the cost of ownership, and businesses can allocate their resources elsewhere.

To offer a customisable and flexible inventory management solution for small and medium-sized enterprises: The proposed system is highly configurable, allowing businesses to customise it to suit their unique requirements. The system is scalable, which means it can grow with the business and adapt to changes in demand.

To ensure the system is highly configurable to cater to the unique requirements of different businesses: The system is designed to be highly configurable, enabling businesses to adjust the system to their specific needs. The system allows for customisable data fields, which means businesses can input data fields that are relevant to their business.

To incorporate robust security features to ensure data privacy and protection: Security is a critical concern for businesses when it comes to managing inventory. The proposed system incorporates robust security features such as user authentication, authorisation, and encryption to ensure data privacy and protection.

Project Objectives:

Implement user login and registration functionality to enable secure access to the system: User login and registration functionality is implemented to enable secure access to the system. This ensures that only authorised users can access the inventory management system.

Develop admin functionality to allow for efficient management of the inventory system: Admin functionality is implemented to enable efficient management of the inventory system. This allows admins to manage the inventory, add new products, and manage stock levels.

Incorporate an automated kiosk feature for offline inventory management: The proposed system incorporates an automated kiosk feature that enables offline inventory management. This is particularly useful for businesses that operate both online and in physical stores.

Ensure that the system is highly performant by implementing various optimisation techniques: The system is designed to be highly performant by implementing various optimisation techniques such as caching, database indexing, and query optimisation.

Enable the system to cater to both online and offline inventory management: The proposed system is designed to cater to both online and offline inventory management. This means businesses can manage their inventory regardless of their location.

Offer comprehensive documentation to ensure easy installation and configuration of the system: The proposed system is accompanied by comprehensive documentation that ensures easy installation and configuration of the system.

Provide ongoing support and maintenance to ensure the system remains up-to-date and secure: Ongoing support and maintenance are provided to ensure the system remains up-to-date and secure. This ensures that businesses can rely on the system to manage their inventory effectively.

Foster a community-driven development model by encouraging contributions from developers and users: The proposed system is open-source, which means that the community can contribute to its development and maintenance. This fosters a community-driven development model, where developers and users can collaborate to improve the system.

1.3 Methodology

The mixed methodology approach used in this project involved a combination of different techniques, tools and frameworks to address the unique requirements and challenges of the decentralised inventory management system. This approach allowed for a more comprehensive and holistic development process that considered the needs of different stakeholders, including users, developers and business owners. Sommerville, I., 2016.

In the frontend development, a qualitative approach was used to gather feedback from users through interviews, surveys and focus groups. This feedback was then used to refine the design and functionality of the system to ensure that it met the needs and expectations of the end-users. Additionally, a quantitative approach was used to measure the performance and efficiency of the system, using metrics such as response time, throughput and error rate.

For the backend and database development, an agile methodology was used, which involved the iterative and incremental development of the system. This approach allowed me to continuously deliver working software, gather feedback from stakeholders, and make necessary changes and improvements throughout the development process. The agile methodology also allows for close collaboration and communication between me and the my supervisors, which was essential in ensuring that the final product met the academic requirements and expectations. (Martin, R.C., 2003.)

In addition to the above methodologies, the Unified Process for system development was also used in this project. The Unified Process provided a structured and disciplined approach to the development process, which helped to ensure that the system was developed in a logical and systematic way. The Unified Process also allowed for the identification and management of risks, issues and dependencies, which helped to mitigate potential problems and ensure that the project was delivered on time and within budget.

Overall, the mixed methodology approach used in this project was effective in ensuring the successful development of the decentralised inventory management system. The combination of different techniques and frameworks allowed for a comprehensive and adaptable approach that addressed the needs of different stakeholders and ensured the delivery of a high-quality product.

1.4 The report structure

Chapter 2: Literature Review

This chapter provides an in-depth analysis of the literature on decentralised inventory management systems, including related concepts, techniques, and technologies.

Chapter 3: Requirements Analysis

This chapter outlines the requirements and objectives of the decentralised inventory management system. It includes a description of the functional and non-functional requirements, and the project scope.

Chapter 4: Design and Architecture

This chapter presents the system design and architecture based on the requirements analysis. It includes a detailed description of the software design, the database design, and the system architecture.

Chapter 5: Implementation and Testing

This chapter details the implementation of the decentralised inventory management system, including the programming languages, software tools, and techniques used. It also includes the testing plan, results, and evidence to demonstrate that the system meets the requirement specification.

Chapter 6: Evaluation and Comparison

This chapter assesses the project output and results against closely related products and/or related work on the same topic/subject area. It compares the decentralised inventory management system with other similar products and highlights its unique features and significance.

Chapter 7: Conclusion and Future Work

This chapter summarises the achievements of the project and reflects on any lessons learned. It also considers what, if anything, could have been done differently and suggests future directions for enhancing the system or extending the research. Finally, it provides concluding remarks and recommendations for further research.

Chapter 2: Background Research

2.1 Literature review of related work

Decentralised inventory management systems (DIMS) have become increasingly popular in recent years, with many companies seeking to take advantage of the benefits they offer. DIMS allows companies to manage their inventory in a distributed manner, meaning that they can track their stock levels and availability in real-time, from any location. This has the potential to significantly improve the efficiency and accuracy of inventory management, as well as reduce costs and improve customer satisfaction.

However, despite the growing popularity of DIMS, there are still some challenges and limitations associated with their use. One major challenge is the need for effective communication and collaboration between different parts of the system, particularly in cases where there are multiple nodes or locations involved. This can be particularly difficult to achieve when different parts of the system are using different technologies or protocols.

Another key limitation of DIMS is the potential for security breaches or data theft. As with any distributed system, there is always the risk of unauthorised access to sensitive data or resources. This can be particularly concerning in the case of inventory management systems, which often contain sensitive information about products, customers, and suppliers. (Li, Y., Li, D., Zhang, H., & Song, X. (2017))

Despite these challenges, the potential benefits of DIMS are significant, and many companies are investing in the development and implementation of these systems. There is a growing body of research on the topic, which has explored various aspects of DIMS, including the design and architecture of these systems, the performance and scalability of different implementations, and the security and privacy implications of their use.

Another challenge faced by traditional inventory management systems is the lack of transparency and accountability in the supply chain. With centralized systems, the responsibility of maintaining accurate records lies solely with the entity in charge, leading to potential errors, fraud, and lack of trust between different parties involved in the supply chain. (Nembhard, H. B., Hu, Q., & Zhang, Y. (2017))

Decentralized inventory management systems, on the other hand, can mitigate these issues by offering a transparent, tamper-proof, and secure way of recording transactions and tracking inventory levels. By using blockchain technology, for instance, such systems can ensure that all parties involved in the supply chain have access to the same information, without the need for intermediaries, thereby enhancing trust and accountability. (T. M. Lekshmi and M. M. Shajahan)

Moreover, traditional inventory management systems often rely on manual processes, leading to errors, delays, and inefficiencies. Decentralized inventory management systems, however, can automate many of these processes, reducing the risk of human error and improving overall performance. For instance, by using smart contracts, such systems can automatically trigger orders when inventory levels reach a certain threshold, or update records when goods are delivered or returned.

Furthermore, other key areas of DIMS include:

- Performance: As showcased by “A model-based approach for testing the performance of web applications”, the performance of inventory management systems is an important consideration, as these systems are often used in mission-critical applications. Researchers have explored various approaches to improving the performance of these systems, including the use of caching, optimization algorithms, and load balancing techniques.
- Integration with IoT devices: Some inventory management systems are integrated with Internet of Things (IoT) devices, such as sensors and RFID tags, to improve the accuracy and timeliness of inventory data. Papers such as: “Secure integration of IoT and Cloud Computing” have studied various approaches to integrating these systems, including the use of IoT protocols and standards, and the challenges and opportunities of using IoT data in inventory management.
- Scalability: As inventory management systems are used by a growing number of organizations and users, scalability becomes an increasingly important consideration. Papers, including a very informative one: “Performance, scalability and reliability issues in web applications”, have explored various approaches to ensuring that these systems can scale effectively, including the use of distributed systems and cloud-based architectures.
- Security: A great published paper, “Semantic security against web application attacks ”, displays why security is a critical consideration for any inventory management system, and researchers have studied various approaches to ensuring the security of these systems. This has included studies on authentication and access control, data encryption, and the use of security frameworks and best practices.
- Predictive analytics: Some inventory management systems include predictive analytics features that can help organizations to forecast demand and optimize their inventory levels. There are a couple of journals, one of which grasped my attention which is: “Measurement, prediction and risk analysis for Web applications”, which explain various approaches to implementing these features, including the use of machine learning algorithms and data visualization tools. I found rather interesting to implement a visual representation of the daily, weekly, monthly and yearly stock and expenses charts since it would boost the productivity of businesses.

• Cloud computing: As per “Cloud computing for Internet of Things & sensing based applications” Many inventory management systems are implemented using cloud computing platforms, and many published papers are explaining the various aspects of this approach, including the benefits and challenges of using cloud-based systems, and the trade-offs involved in different deployment models such as:

1. Public cloud
2. Private cloud or otherwise local network
3. Hybrid cloud, which is the combination of the above

In summary, a decentralized inventory management system can help alleviate many of the challenges faced by traditional systems by offering greater transparency, security, and efficiency. By leveraging technologies such as blockchain and smart contracts, such systems can provide a more reliable and trustworthy way of managing inventory, enhancing performance, and ultimately contributing to the success of businesses operating in today's highly competitive and dynamic markets.

2.2 Critical evaluation of related products/solutions

While some inventory management systems offer robust features and functionality, they can be overly complex and difficult to use. This can result in a steep learning curve for users and a lack of adoption among employees. Therefore, it is important to strike a balance between functionality and usability when evaluating inventory management solutions. (Li, L., Li, Y., & Li, B. (2019))

Many inventory management systems rely on manual data entry and batch processing, leading to delays and potential errors. However, real-time inventory tracking systems that utilize RFID or barcode scanning technology offer greater accuracy and efficiency. Therefore, businesses should carefully consider the type of tracking technology used in a potential solution.

Some inventory management systems may offer limited integration capabilities with other business systems, such as accounting or purchasing software. This can lead to data silos and redundant data entry, reducing efficiency and accuracy. Therefore, it is important to evaluate the integration capabilities of a potential solution and ensure it can seamlessly integrate with other critical business systems.

While cloud-based inventory management systems offer the benefit of anytime, anywhere access, they also come with potential security risks. Therefore, it is important to evaluate the security measures implemented by a potential solution, including data encryption, user authentication, and access controls.

Cost is always an important factor to consider when evaluating inventory management solutions. However, businesses should also consider the long-term value and return on investment (ROI) of a potential solution, considering factors such as improved efficiency, accuracy, and scalability. A cheaper solution may ultimately end up costing more in the long run if it lacks the necessary features and functionality to support business growth and success.

Example Systems:

- **QuickBooks Commerce** (formerly TradeGecko) - This cloud-based inventory management system offers strong features and integrations with other business systems, but its user interface can be complex and difficult to navigate.

- **Zoho Inventory** - This cloud-based solution offers robust features and integrations with other Zoho business apps, but some users have reported issues with bugs and customer support.

- **Fishbowl Inventory** - This on-premises inventory management solution offers strong integration capabilities with QuickBooks, but its high price point and complex setup process may be a barrier to adoption for some businesses.

- **Square for Retail** - This cloud-based solution offers strong features and affordability, but its inventory tracking capabilities may not be sufficient for larger or more complex businesses.

- **Odoo Inventory** - This open-source solution offers a range of features and customizability, but its user interface can be overwhelming and difficult to navigate for some users.

Other similar projects with the frameworks that personally helped me understand a lot of technical details are:

- “Smart Warehouse Management System Concept with Implementation”, which covers how stock planning and predictive analysis would work for a concept smart warehouse. I can apply the same concepts in my inventory management system web application, since they are very tighly related to each other.

- “Internet of things for perishable inventory management systems: an application and managerial insights for micro, small and medium enterprises”, which is not a complete application but describes how one would take shape. As described very elegantly in the below figure:

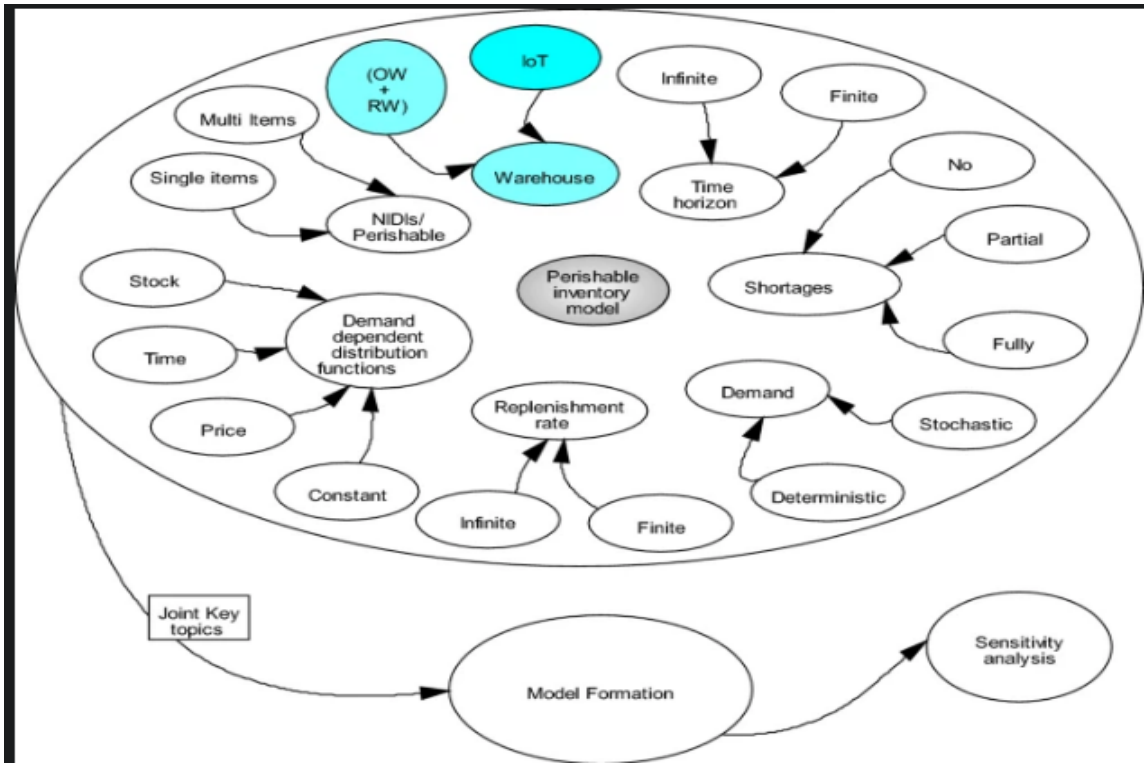

Figure 0; Internet of things for perishable inventory systems

While these are just a few examples that I have personally researched online, there are many more out there, kept hidden.

2.3 The scope of the project

The scope of the project is focused on developing a decentralized inventory management system. The system will be designed to manage the inventory of a business through an online store and an offline automated kiosk, like those found in supermarkets.

The system will have two main user types: administrators and customers. Administrators will have access to the backend of the system and be able to manage products, orders, inventory, and customer accounts. Customers will be able to browse products, add items to their cart, and make purchases through the online store or kiosk.

The system will be developed using Java's Spring Boot framework and will prioritize performance, security, and ease-of-use. The system will utilize a decentralized architecture to provide greater security and data privacy for customers and businesses.

In addition to the core functionality of the system, the project will also focus on implementing additional features such as user authentication, email notifications, and reporting capabilities. These features will be designed to provide a more complete inventory management solution for businesses of all sizes.

The project aims to provide a high-quality, customizable, and cost-effective inventory management system for businesses that do not have the budget to create their own systems. By utilizing a decentralized architecture and focusing on performance and security, the system is able to provide a reliable and scalable solution for businesses of all sizes.

In addition to the features mentioned previously, the scope of the project also includes the implementation of easy deployment options using Kubernetes and Docker's Docker Compose. The system is designed to provide a secure environment, with all aspects of security thoroughly addressed, including protection against Cross-Site Request Forgery (CSRF), Cross-Origin Resource Sharing (CORS), and the use of HTTPS encryption.

Another key aspect of the project is the collection and storage of analytics data, which includes information such as the number of products purchased, total profit percentage, number of users online, and users' geographical location. This data can be used by companies to gain a better understanding of their customers' behaviour and to make informed decisions about marketing and product development. Overall, the project's scope is to provide a robust, secure, and user-friendly decentralized inventory management system that offers a range of features and benefits to companies looking to streamline their operations and increase efficiency.

2.4 Review and justification of theories/models/development platforms/tools selected for use in the project

The choice of tools, technologies, and models used in the development of the decentralized inventory management system was based on several factors, including their suitability for the project's objectives, ease of use, scalability, and availability of support and resources.

Firstly, the system was developed using Java's Spring Boot framework, which offers several advantages such as rapid development, ease of use, and a vast community of developers providing support and resources. The use of Spring Boot also ensures that the system is scalable, maintainable, and efficient, with minimal boilerplate code required. In order to understand my choices better, lets have a look at this table below:

Architecture	**Microservices**	**Monolithic**
Scalability	Easier to scale horizontally by adding more instances of individual services	Limited scalability, typically scaled vertically
Deployment	Independent deployment of services, allowing for faster and more frequent updates	Complex deployment process for entire application, making updates slower and more difficult
Fault Tolerance	Isolating failures to individual services, allowing other services to continue functioning	A single failure can bring down the entire application
Development	Easier for multiple teams to work on different services simultaneously	Development is centralized, making it difficult for multiple teams to work on different parts of the application
Technology stack	Individual services can use different technology stacks	A single technology stack is used throughout the application
Complexity	Requires more effort to manage and orchestrate multiple services	Easier to manage and deploy a single application
Cost	Can be more expensive to implement and maintain	Generally less expensive to implement and maintain

Table 1; Comparing Microservices and Monolithic

As for the advantages of microservices, they include:

Greater scalability and flexibility, allowing for faster development and deployment

Easier fault isolation, preventing the entire application from going down in the event of a failure

Better alignment with modern DevOps practices, such as continuous deployment and containerization

Ability to use different technology stacks and programming languages, allowing for greater flexibility and innovation

Improved team agility and autonomy, as different teams can work on different services without being hindered by a centralized architecture

On the other hand, some of the consequences of using a monolithic architecture include:

Limited scalability, as the entire application must be scaled up or down at once

Greater complexity, as all components of the application are tightly coupled

Higher risk of downtime, as a single failure can bring down the entire application

More difficult deployment process, as updates must be deployed to the entire application at once

Limited technology stack and programming language options, which can limit innovation and flexibility.

To continue, the system was built using a microservices architecture, which offers several benefits, including scalability, fault tolerance, and the ability to deploy and update individual services independently. The use of Kubernetes and Docker Compose for deployment ensures that the system can be easily deployed, managed, and scaled. But some may wonder, why even bother deploying with virtualization services such as Docker or Kubernetes?

Feature	Kubernetes	Docker	Native
Orchestration	Yes, with built-in orchestration capabilities	No, requires third-party tools	No, requires manual orchestration
Scalability	Highly scalable and automated	Scalability depends on host resources	Limited scalability, manual configuration
Resource Management	Resource allocation and management	Resource management within container	No resource management
Load Balancing	Built-in load balancing features	No built-in load balancing	No load balancing

High Availability	Depends on configuration, can have availability	Can hot swap containers for high availability	No high availability
Networking	Built-in networking features	Limited networking capabilities	Basic networking capabilities
Security	Strong security features and policies	Basic security features	No built-in security features
Ease of Deployment	Requires more configuration and expertise	Easier to deploy than Kubernetes	Limited compatibility
Compatibility	Can be expensive due to resource utilization	Generally cost-effective	Cost-effective
Cost	Steep learning curve	Moderate learning curve	Easier to learn than Kubernetes
Learning Curve	- Optimized for large-scale workloads - Ability to horizontally scale resources quickly	- Fast, lightweight containerization - Better performance compared to virtual machines	- Direct access to hardware - Optimal performance and speed, but dependent on hardware
Performance	- Robust security features and policy management - Network security and encryption options	- Basic container security - Vulnerabilities may occur if containers aren't secured properly	- Security depends on the underlying operating system and hardware
Scalability	- Horizontal scaling of pods and containers - Automatic load balancing and self-healing	- Easily scalable through orchestration tools - Load balancing and resource allocation options	- Limited scalability options - Scaling requires manual configuration and resource allocation

Table 2; Comparing Kubernetes, Docker and Native

I have also seen massive development improvements using docker especially, when deploying my servers to another hardware. There is not reason to trust me, trust these folks instead: Hyungro Lee, et al. (2016), Naveed Ahmed, et al. (2019), Luiz Angelo Steffenel, et al. (2018).Martin Dyring-Andersen, et al. (2017)

Finally, in order to finish this chapter gracefully, I would like to outline that I've given some careful thought to the Java programming language facilities. Ever since I got accustomed to Rust, another statically typed language that offers ultimate performance. So why would I choose to write such critical microservice in Java, someone would ask.

Factor	Java	C	Python	Rust
Type System	Static	Static	Dynamic	Static
Memory Management	Garbage Collected	Manual	Garbage Collected	Ownership-based
Performance	High	High	Medium	High
Development Speed	Good (with libraries)	Low (without libraries)	Good (with libraries)	Excellent
Community Support	Fast	Fast	Fast	Slow (due to strictness)
Ecosystem	Very Large	Large	Large	Small but growing
Compatibility with libraries	Large, mature	Large, mature	Large, diverse	Small, growing
Security	Excellent	Excellent	Good	Moderate
Concurrency	Good	Low (due to manual memory management)	Moderate	Perfect due to Box<> type safety

Table 3; Comparing Java, C, Python and Rust

Steve Klabnik and Carol Nichols (2019), Martin Dyring-Andersen, et al. (2017), Luiz Angelo Steffenel, et al. (2018)

All in all, there are a lot of programming languages to choose from. There is no such thing as the best language of everything. Every language has its own use. Depending on the resources, manpower and purpose, one or even a combination of languages might be needed.

In this case, I have decided that Java is my pick, as mentioned before. To sum up Java in a few words: Java is a widely used, popular, and robust programming language with a large and active developer community. Its strong type system, garbage collection, and platform independence make it a suitable choice for enterprise-level projects.

Chapter 3: Requirements Analysis and Specification

3.1 Functional Requirements

The functional requirements of the system is categorised as follows:

3.1.1 User Management

The system shall allow users to register, login and update their profile information.

The system shall allow users to reset their password if forgotten.

The system shall provide different levels of access for admins and kiosk users.

3.1.2 Inventory Management

The system shall allow admins to add, update, and delete products in the inventory.

The system shall allow admins to view the inventory status in real-time.

The system shall allow kiosk users to check-out products via barcode/qrcode scanning.

The system shall automatically update the inventory status after a product has been checked-out from the kiosk.

3.1.3 Analytics and Reporting

The system shall collect and store data on products, users, and profit.

The system shall provide analytics and reports on product popularity, user demographics, and profit margins.

The system shall display the analytics and reports on the admin dashboard.

3.1.4 Live Chat Support

The system shall provide live chat support for users and admins.

The system shall allow admins to initiate live chat with users.

The system shall record the chat history for future reference.

3.2 Non-Functional Requirements

The non-functional requirements of the system is categorised as follows:

3.2.1 Security

The system shall ensure data privacy and confidentiality by using HTTPS encryption with trusted certificates.

The system shall protect against Cross-Site Request Forgery (CSRF) and Cross-Origin Resource Sharing (CORS) attacks.

The system shall allow only authorised access to data and functionalities by implementing access control measures.

The system filters access to URL endpoints based on authorization role when an entity authenticates

3.2.2 Performance

The system shall provide fast response times to ensure a seamless user experience.

The system shall have high availability to ensure that the system is always accessible.

The system shall be scalable to accommodate increasing numbers of users and transactions.

3.2.3 Deployment and Maintenance

The system shall be easy to deploy and maintain using Docker and Kubernetes.

The system shall provide server logs and health metrics for monitoring and troubleshooting purposes.

Regular maintenance is performed to ensure the system remains secure, reliable, and efficient.

In conclusion, this chapter has outlined the functional and non-functional requirements for the decentralised inventory management system. The functional requirements were categorised into user management, inventory management, analytics and reporting, and live chat support. The non-functional requirements were categorised into security, performance, and deployment and maintenance. The requirements have been carefully considered to ensure that the final product meets the needs of the users and is reliable, secure, and performant.

Use Cases

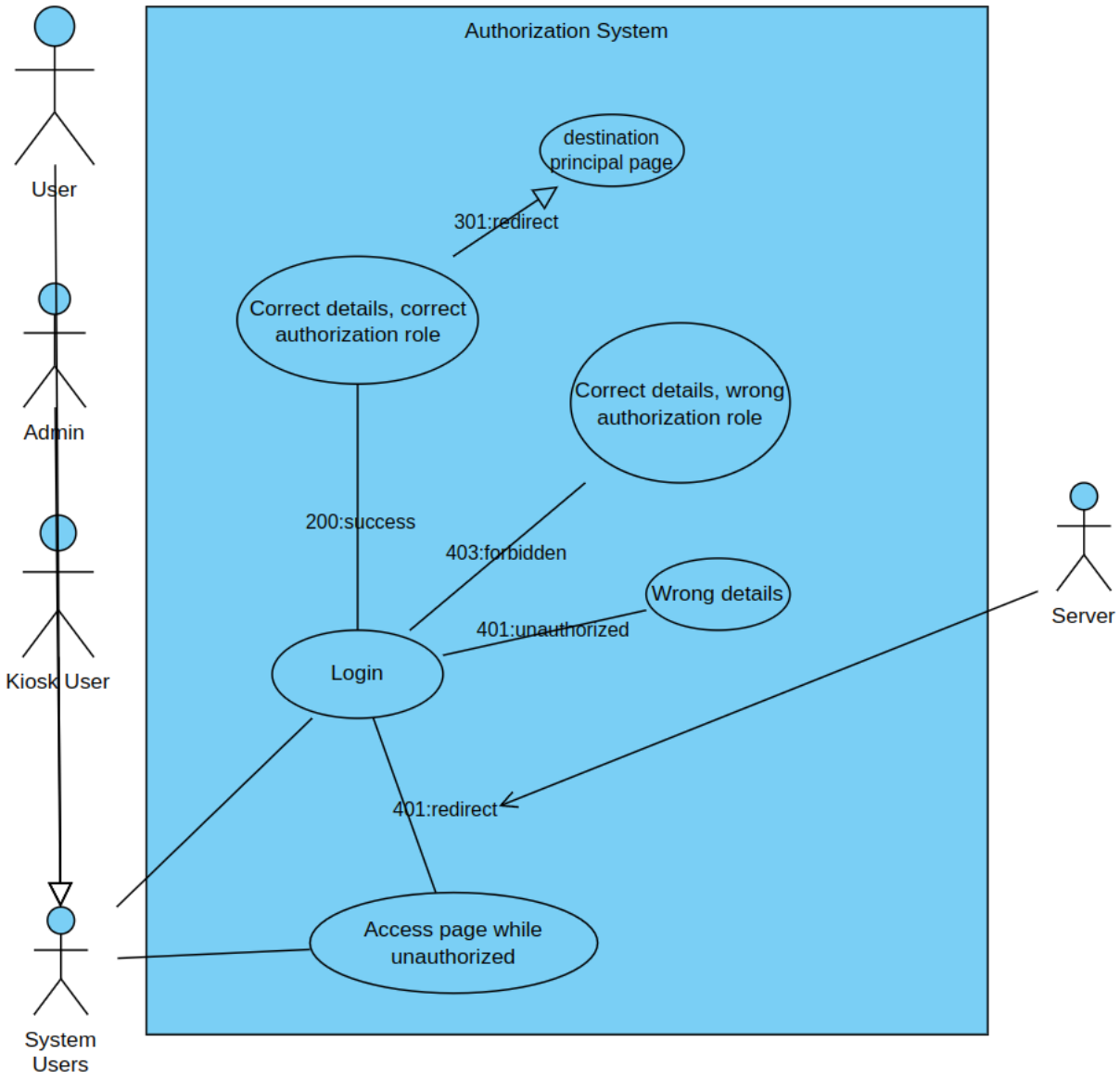

Figure 1; Authorization Use Case

I opted to go for a redirect depending on policy system. That is, if the user successfully authenticated and is authorized to access the specific resource, the server defines and sends the redirect back to the client.

If the client is not authenticated/authorized, the server redirects back to the login page.

It is a straightforward system implementation that helps us with reducing the frontend code base.

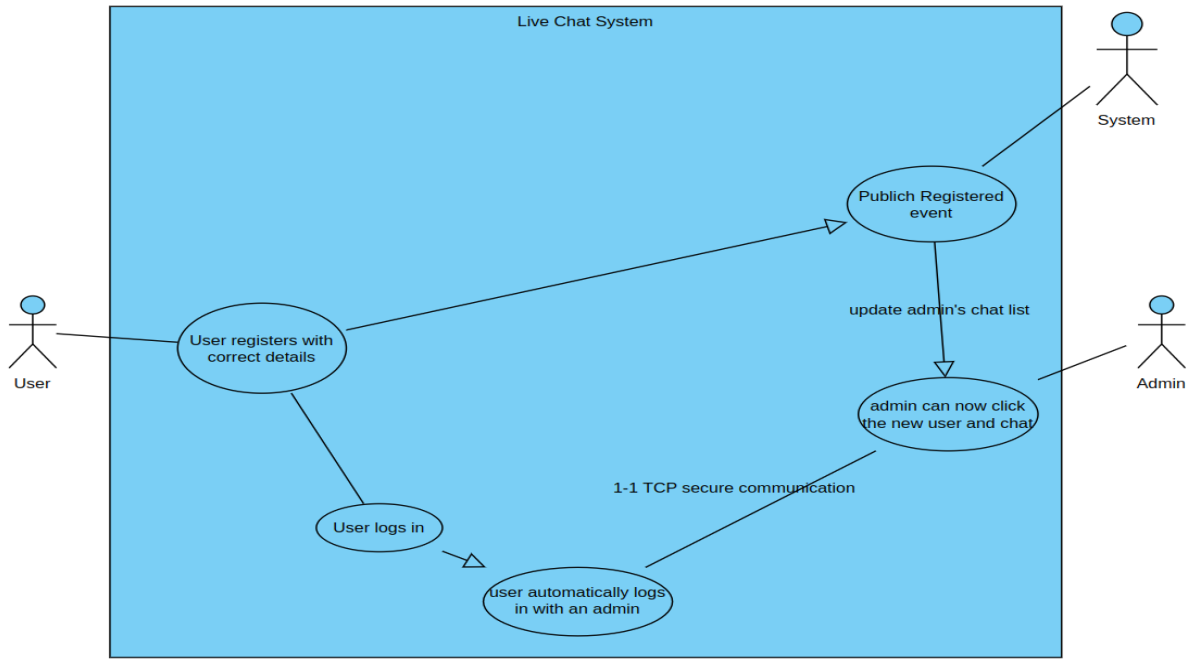

Figure 2; Live Chat system use case

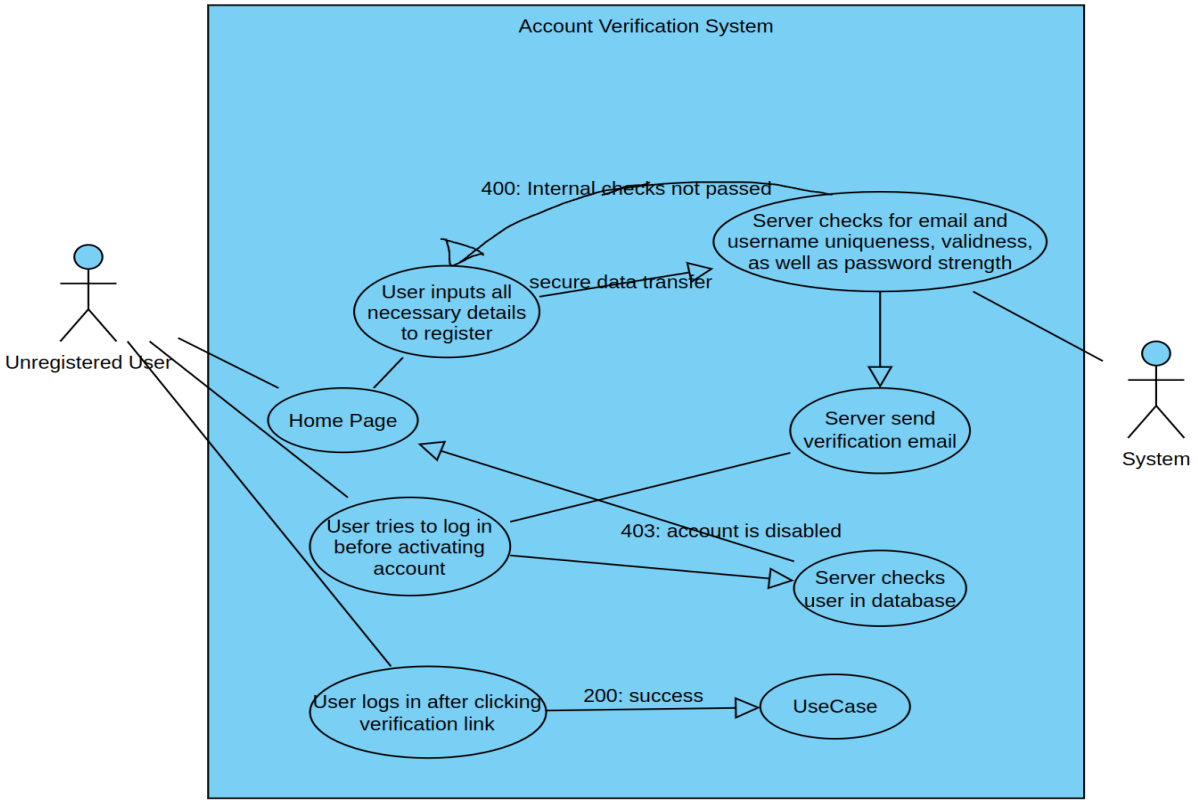

Figure 3; Account verification use case

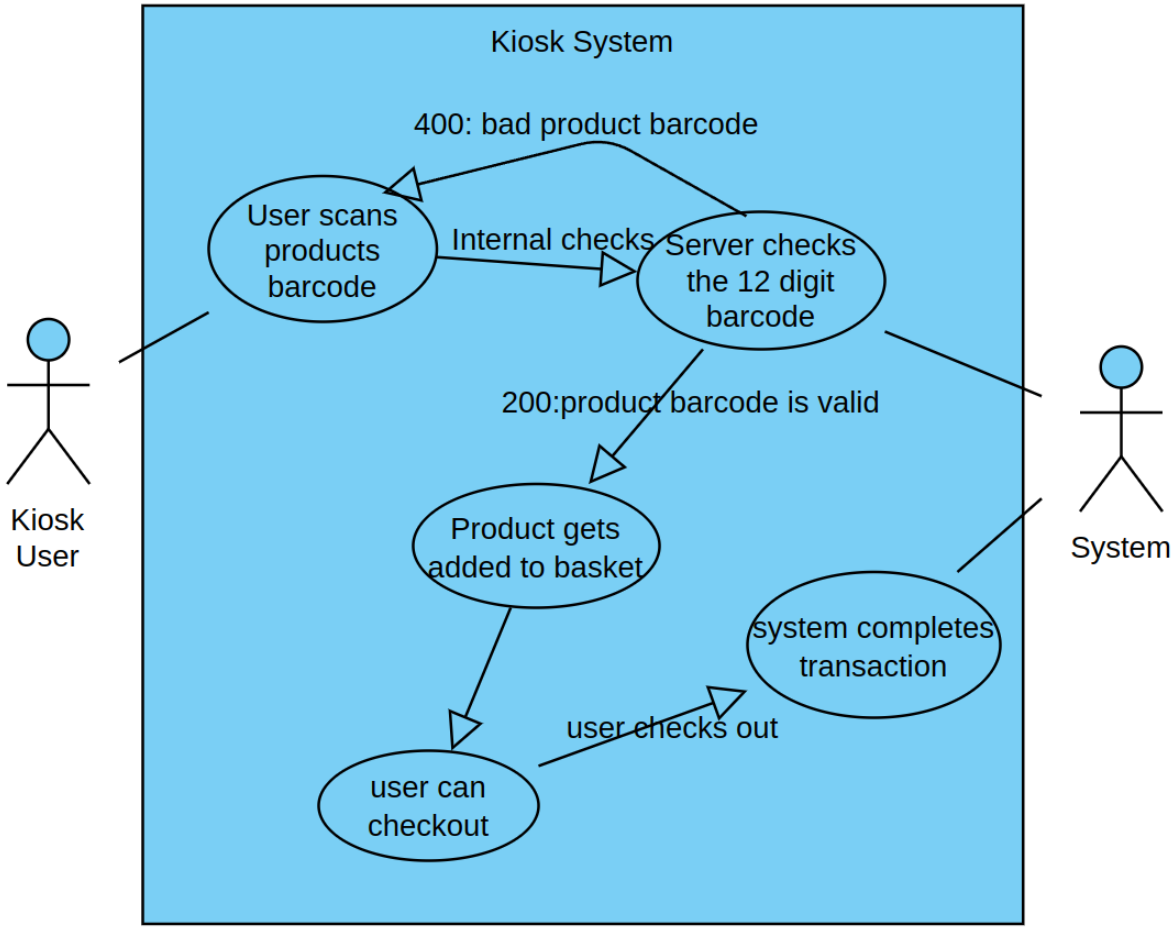

Figure 4; Kiosk system use case

There is a lot of other functionality, hidden behind the scenes. It would take a long time to explain it with use cases, so I am going to showcase such functionality from a programmer's perspective, with class, package, database and other related diagrams.

Chapter 4: Software Design

4.1 User-Interface design

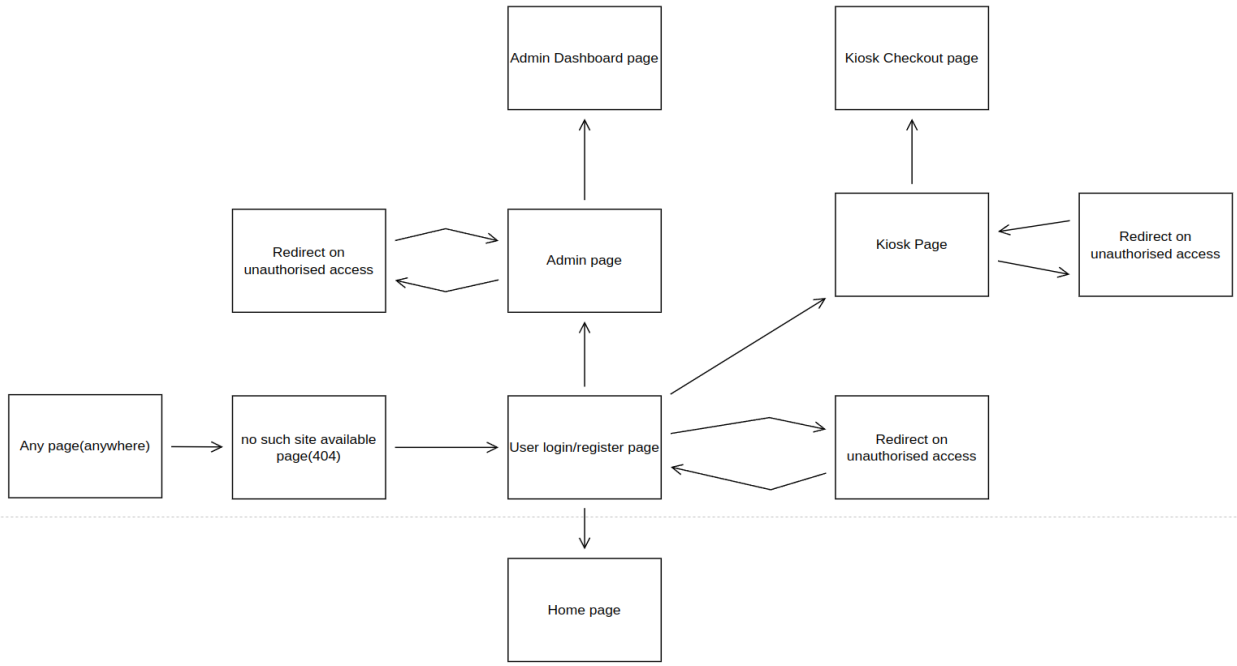

Figure 5; UI design

A couple of notes for this sitemap:

I know the Home page is also supposed to be accessed by non-logged in users. However, in order to demonstrate the many principal authentications (per role authentications) that the system supports, I implemented it with mandatory authorization.

Any page is going to go to the No such Site available (404 page). That may happen if the server is still in startup phase or when the server is being shut down. It may also happen with a wrong redirect, should the code be changed in the future, so it is a good sign of something internally going wrong.

4.2 Database tables' structure design

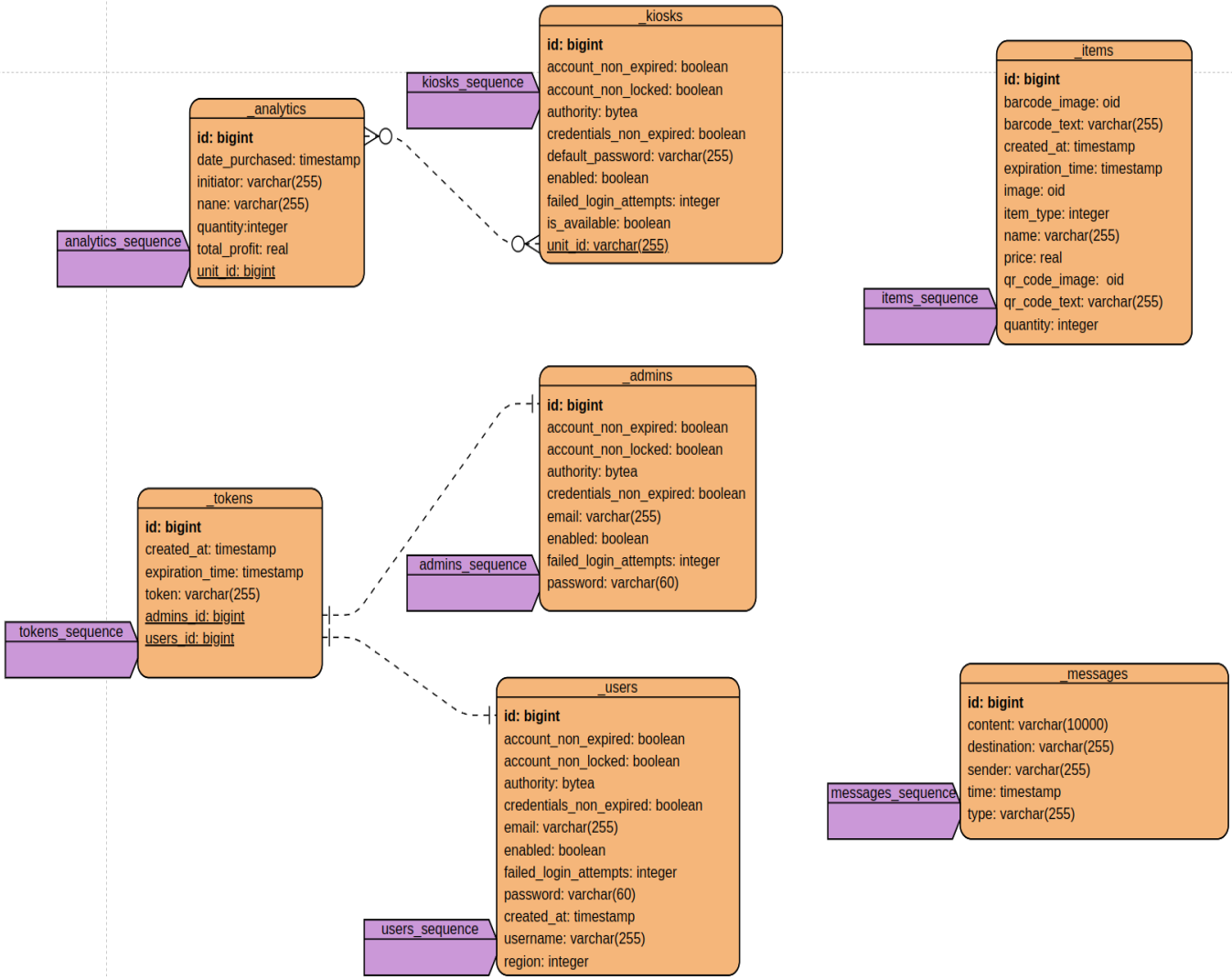

Figure 6; Database entity diagram

A couple of notes to keep in mind:

I went for a very straight-forward database design. Based on my Gantt chart diagram and many other factors, I did not want to make any mistakes halfway through the project.

Every database entity has their own generators. Although I am going to demonstrate it later, I have added restrictions on most database entity elements, such as enforce uniqueness in email and username of admins and users, passwords should be "strong", some elements cannot be null etc.

4.3 Main components of the software architecture

I cannot really pinpoint the exact class of the system that holds the entire system up and running. Instead, there are multiple asynchronous classes working together as part of a microservice. If one service is not working, the server will not stop functioning. If it can proceed with the data processing, it is going to. For example, even if the email validation microservice stops working, the user is going to be registered to the system, but be forever disabled. It all depends on how everyone will look at what functions are “necessary” and what not.

User Management Package Diagram:

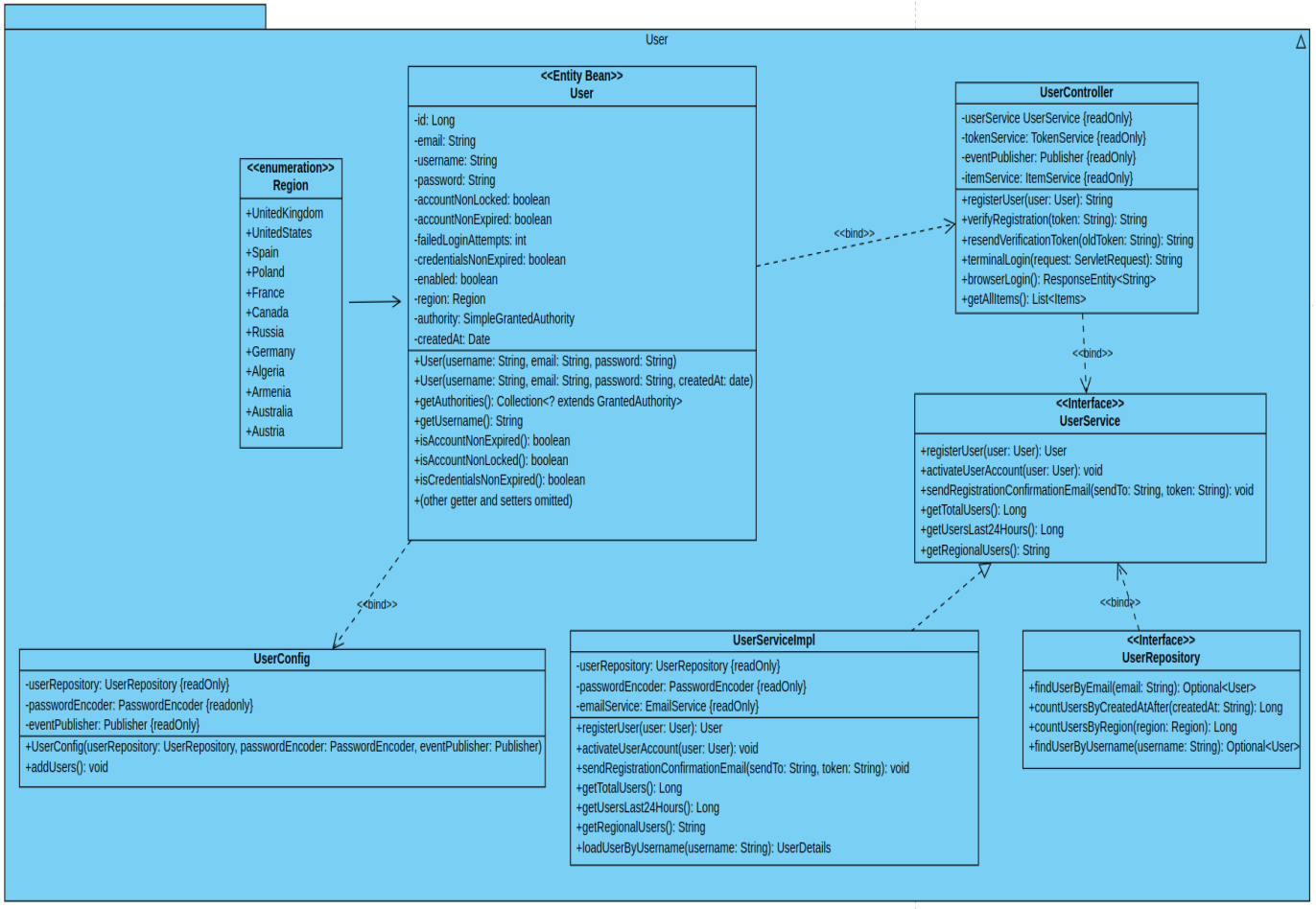

Figure 7; User package diagram

Pretty much a self-explanatory image, saving some explanation. Every class, apart from the UserConfig is needed. The UserConfig is not going to be present at production either way.

By declaring interfaces, I can implement them however I want. I am only defining the necessary functions that the API should have and implementing them, giving freedom to the developers.

Token Package Diagram:

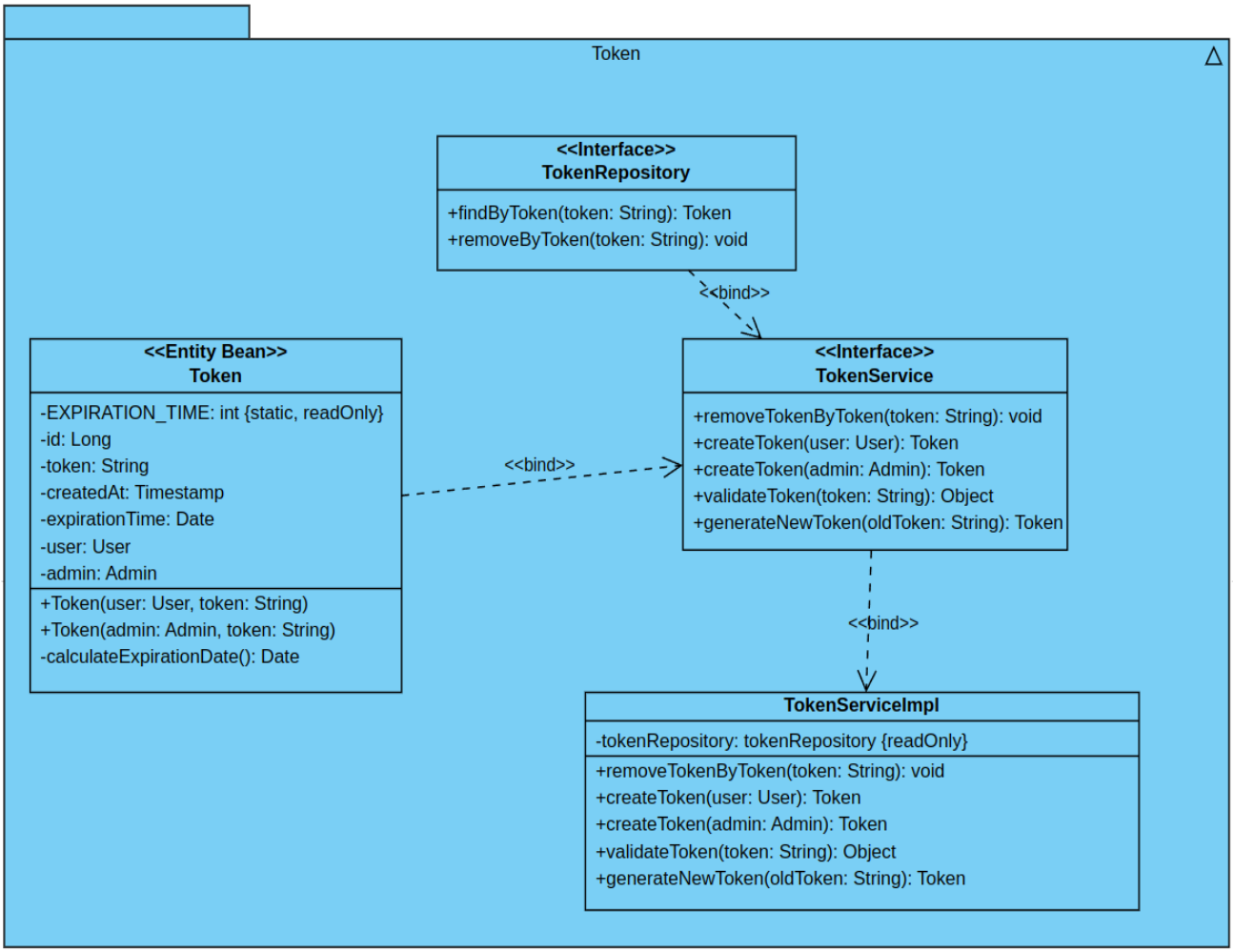

Figure 8; Token package diagram

Some notes for this Class:

You can notice that the Token Entity has foreign keys of both user and admin id. Also, the TokenService has 2 identical functions:
- createToken(user: User): Token

- createToken(admin: Admin): Token

The purpose of this is to “recycle” the TokenService, instead of creating some other, independent services just for the token creation of user and admins.

The other logic is the same as before.

Kiosk Management Package Diagram:

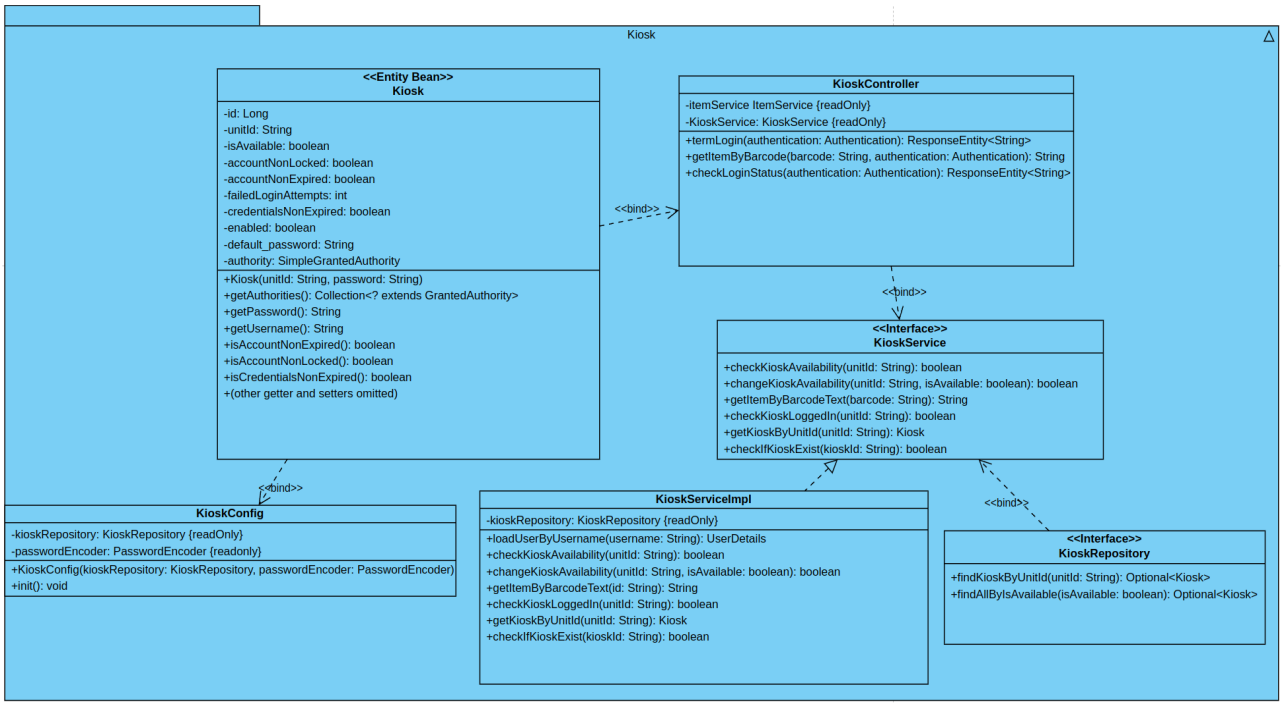

Figure 9; Kiosk management package diagram

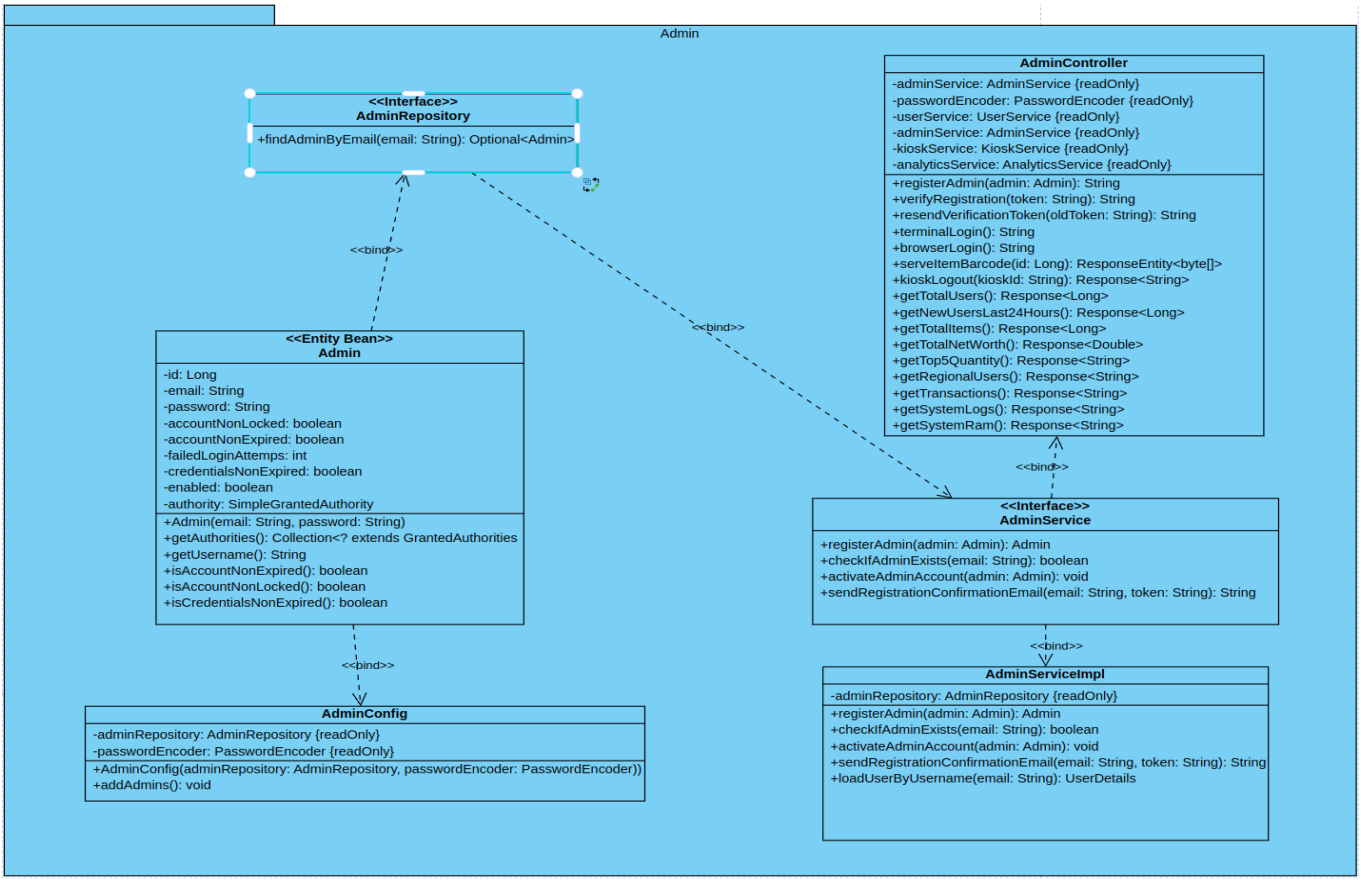

Figure 10; Admin management package diagram

Chat Management Package Diagram:

So as not to repeat the images before, the Analytics Package is more or less the same as the Admin and Users package diagram, although it has different fields used for collecting Analytics on users, admins, products and kiosks, after which it displays them in the Admin page.

However, I am going to show the Chat package diagram, since it is fundamentally different:

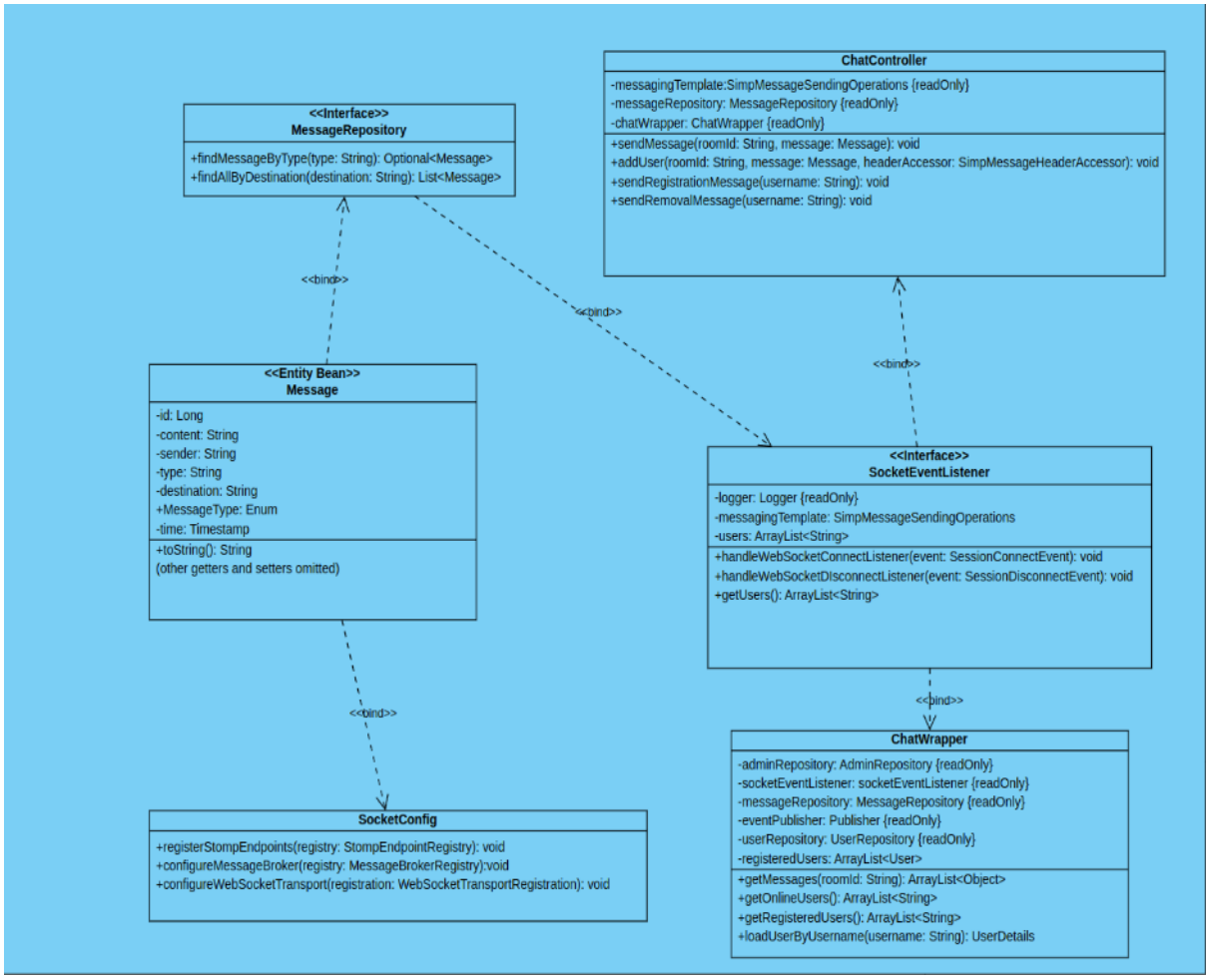

Figure 11; Chat package diagram

A couple of notes here:
SocketConfig class is the actual configuration about buffers, maximum size limits, one-offs and etc. It is useful if one wants to use RabbitMQ, a technology which I did not utilize.

One may also fine tune the server's capabilities with the SocketConfig file template.

The rest of the classes are pretty much straight forward.

There is one more minor class, the WebController class which just maps the actual files to the urls.

So for the admin page, http://localhost/admin → I have set the viewname of admin.

Note, I have not actually defined a view name for the root path, http://localhost/

That is because springboot automatically picks that up, as long as I have an index.html in the root path of the resources folder. The rest of the html pages go to templates. All of the other static content go in the static folder, each one to their own respectively.

Security Configuration Package:

This one was the hardest to do. Since I had so many principal users(or otherwise authorities), I had to define multiple configurations. But Spring Boot does not really allow you to load a lot of authentication entry points for the same path.

What I did instead is specify a lot of static classes with an order, starting from 1. So, when a user/admin/kiosk user goes to authenticate, for example, in the http://localhost/admin page, at most one is going to be be picked in order.

Check on the @Order() annotation on every static class.

So in the above example a user could login as an admin?

Wrong. Since I have also specified antMatchers() which is regex paths for every static paths. So admins take the /admin/** path, users take the /users/** path and so on.

I also have included a caching mechanism for lazy loading some files in logs.

Now let's discuss the importance of the CustomEntryPoint.java file. Without that file, anytime a user is not authenticated and goes in a privileged webpage OR enters a password

incorrectly OR does not have necessary principal roles to access a webpage, a popup would drop down to authenticate. I do not want that. I want my nice and customized login pages to handle that job.

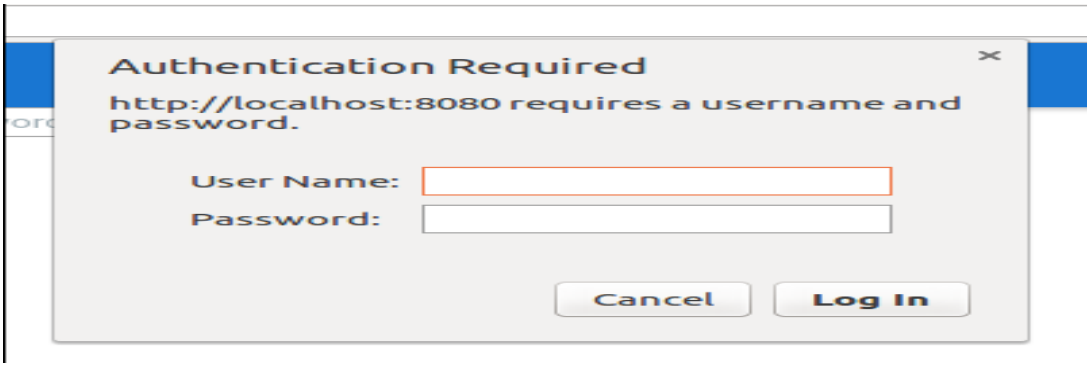

Figure 12; Browser's innate authorization mechanism

4.4 Use case realisation

User Registers/Logs in:

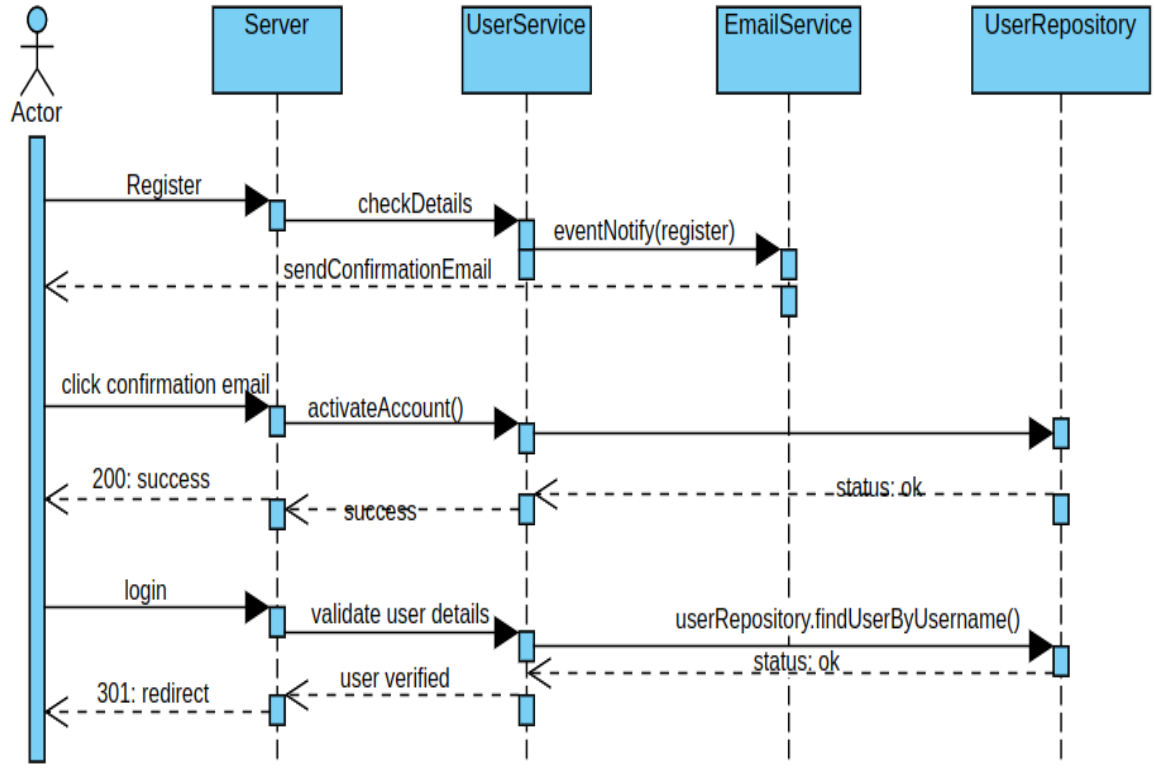

Figure 13; Main sequence diagram

As seen, the Server here is the UserController, but I left it as Server for clarity's sake. The middleman is UserService as always, with the database management being the User Repository.

Note, UserService does not directly communicate with EmailService, that would be a violation of the contract. Instead, what happens is that it sends an asynchronous event, and another async listener picks it up and communicates with EmailService.

Controller can communicate directly with the Services, but not vice versa. A Service may not be the one initiating and hereby calling a controller. A Service should only reply to the controller.

Another note is that a Controller should never directly communicate to a repository without a Service. Otherwise, the server might end up having Circular dependencies issues which is another world of hell.

Chapter 5: Implementation and Testing

5.1 Software Implementation

In this section, I am going to provide details on how the design specifications outlined in the previous chapter were implemented. I carefully considered various alternative implementation methods, ultimately selecting the one that would best meet the requirements of the decentralised inventory management system (DIMS).

To implement the DIMS, I utilised a range of software development techniques, programming languages, and software tools. For example, I utilised agile project management methodologies to ensure continuous feature integration for timely delivery of project milestones, since it also suit my needs. I have also adopted a microservices architecture to enable the creation of a scalable and resilient system, which can handle high levels of user traffic and demand.

In addition, I employed containerisation techniques with Docker and Kubernetes, which as discussed allows for the easy deployment and maintenance of the system. To ensure security and protection against potential threats, I implemented https encryption with trusted certificates, CSRF, and cross-origin protection.

Furthermore, I utilised various programming languages including Javascript, and HTML to develop the front-end and back-end components of the DIMS. I used Springboot, a Java-based web framework, to build the server-side component of the system, and used mostly Jquery a Javascript-based library, to build the client-side component. Additionally, I utilised Redis, a very fast in-memory database, to store and manage data generated by the system.

The technical description of the implemented software includes various features such as the ability for kiosks to be able to scan barcodes and QR codes, allowing users to checkout items easily. Admin users can manage the system and access various analytics, including information on product sales, user behaviour, and profitability. The system also includes a live chat functionality which enables users to communicate with the administrators in real-time, as well as a server log and server health metrics for system monitoring and troubleshooting. Overall, the technical implementation of the DIMS has successfully realised the design specifications outlined in the previous chapter.

5.2 Software Testing

In this section, I am going to talk in detail about the testing process used to demonstrate that the system I have developed works as intended and meets the requirements specified in Chapter 3 of this report. A comprehensive test plan was created to ensure that all aspects of the system were thoroughly tested, including functional and non-functional requirements.

The test plan includes manual testing methods. Manual testing was carried out to ensure that the system was easy to use and that all features were working as expected as well as to test the system's performance, security, and scalability.

I will start with user management testing first:

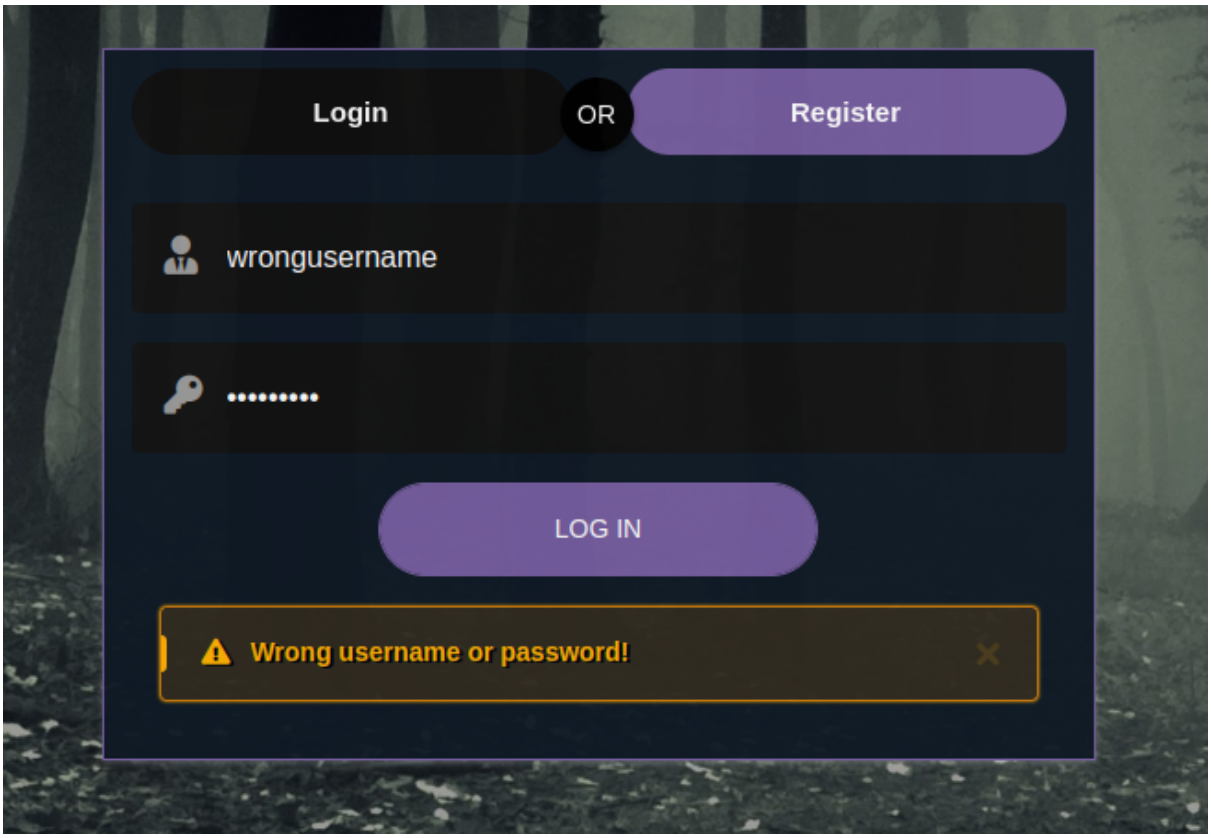

Figure 14; Wrong username log in

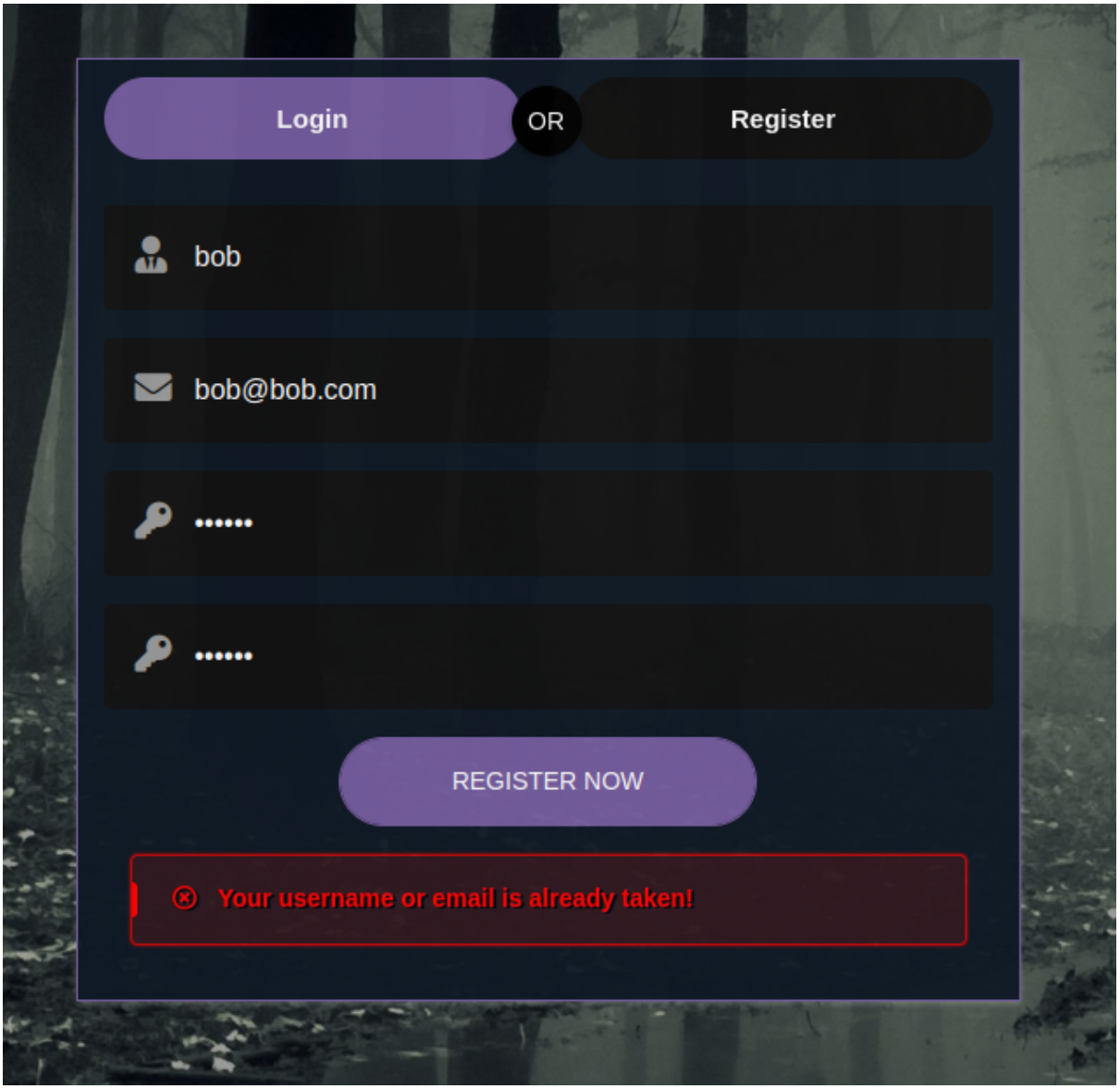

Figure 15; Taken username/email register

Figure 16; POST failed devtool

As seen, the server successfully replies with 401; unauthorised.

Let us try to login now:

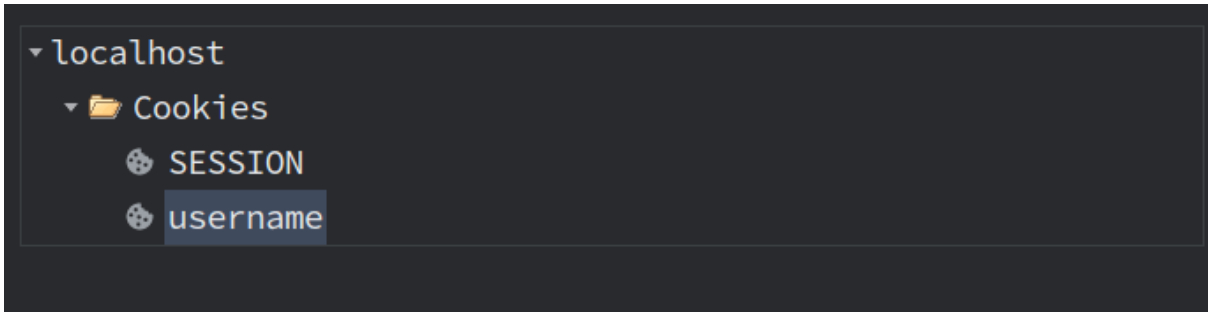

Figure 17; Browser cookie set

First thing I noticed immediately is that the cookies were set properly. Both the username and the SESSION. If I were to use a production mode of this server, as in use official signed certificates, there would be one more, namely XSRF-Token.

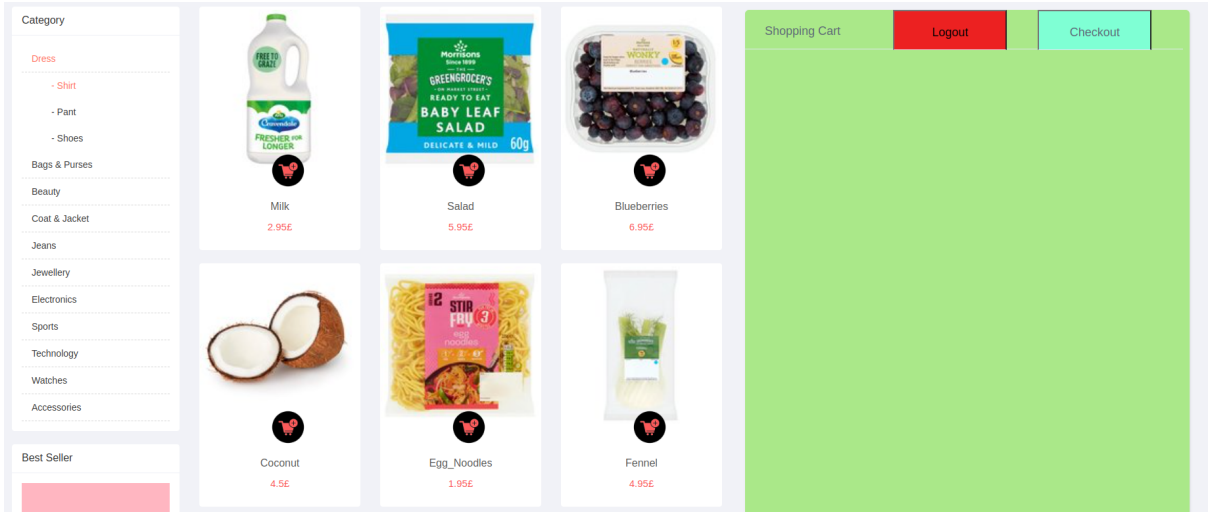

Figure 18; User login main page

Add item to cart:

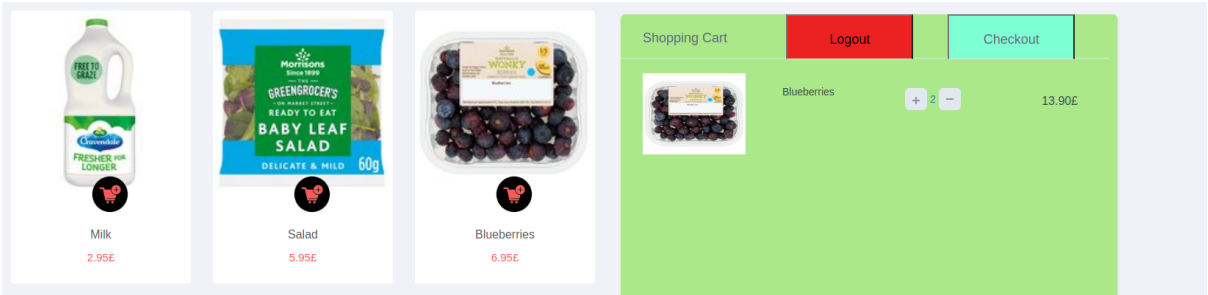

Figure 19; Adding an item to basket

Remove 1 quantity of blueberry from cart and add another

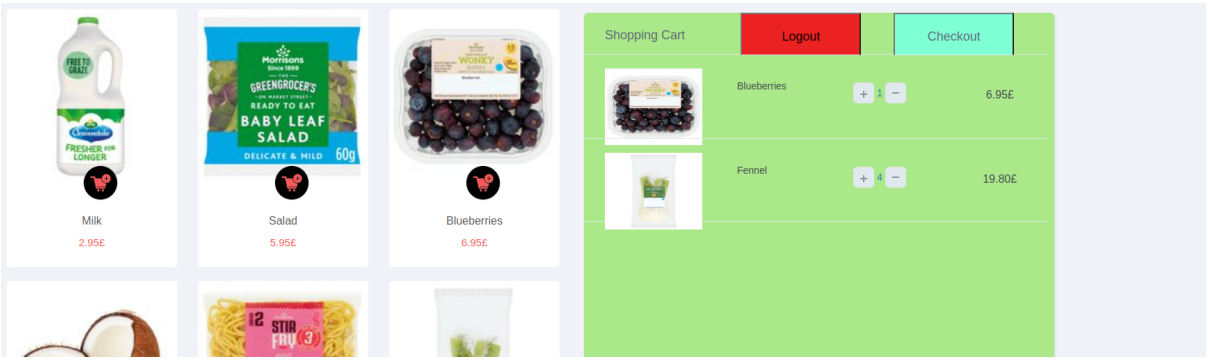

Figure 20; Remove 1 quantity from item

Completely remove an item with the minus sign:

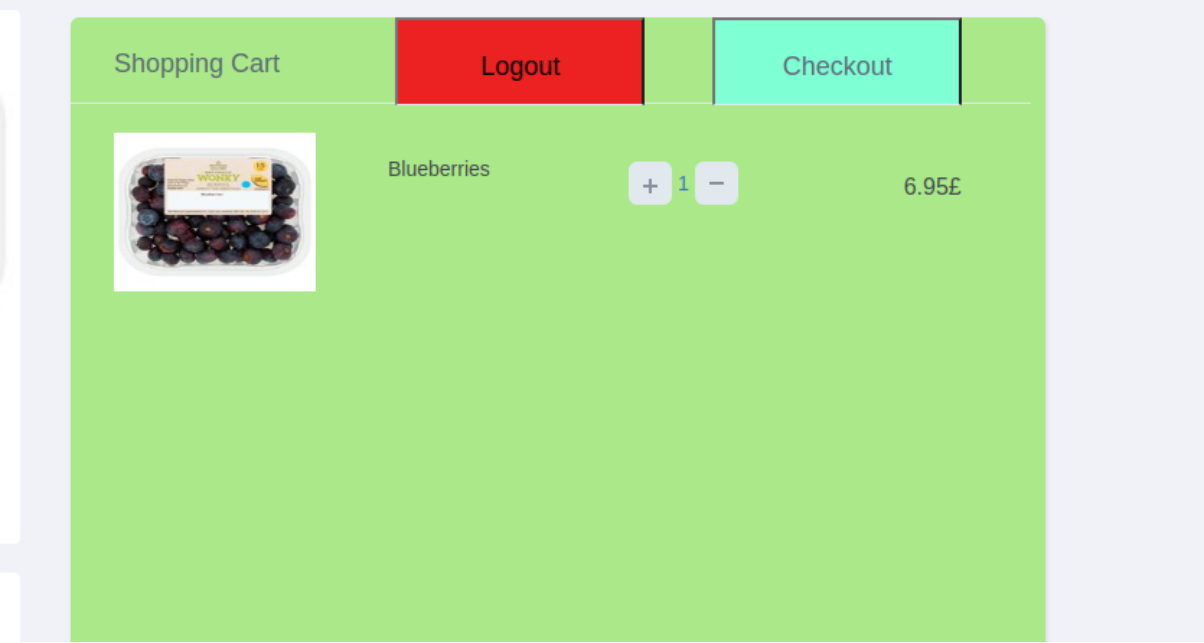

Figure 21; Completely remove an item from basket

Checkout 1(I have not really added a success message, it is only going to throw errors if need be)

```
backend-server  | Hibernate:
backend-server  |     select
backend-server  |         nextval ('analytics_sequence')
backend-server  | Hibernate:
backend-server  |     insert
backend-server  |     into
backend-server  |         _analytics
backend-server  |         (date_purchased, initiator, name, quantity, total_profit, unit_id, id)
backend-server  |     values
backend-server  |         (?, ?, ?, ?, ?, ?, ?)
backend-server  | Hibernate:
backend-server  |     select
backend-server  |         nextval ('analytics_sequence')
backend-server  | Hibernate:
backend-server  |     insert
backend-server  |     into
backend-server  |         _analytics
backend-server  |         (date_purchased, initiator, name, quantity, total_profit, unit_id, id)
backend-server  |     values
```

Figure 22; Checkout completed server logs

And if I press logout button:

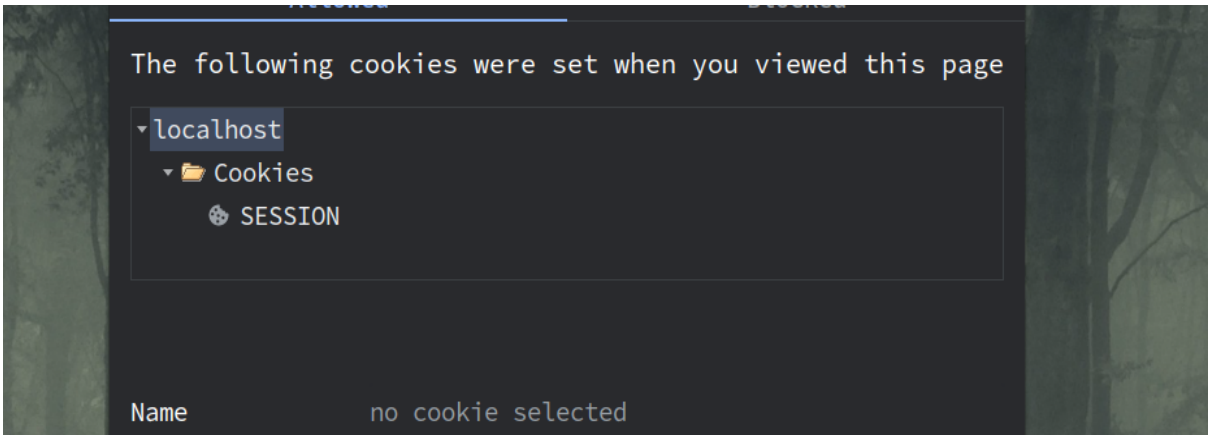

Figure 23; user logout clicked result

Note that the Session is not a stale one. The moment that I sent a request to logout(GET request), that get request replaced the Session cookie with another one, invalid in addition to erasing the username cookie.

On the bottom of the page I have added some useful links:

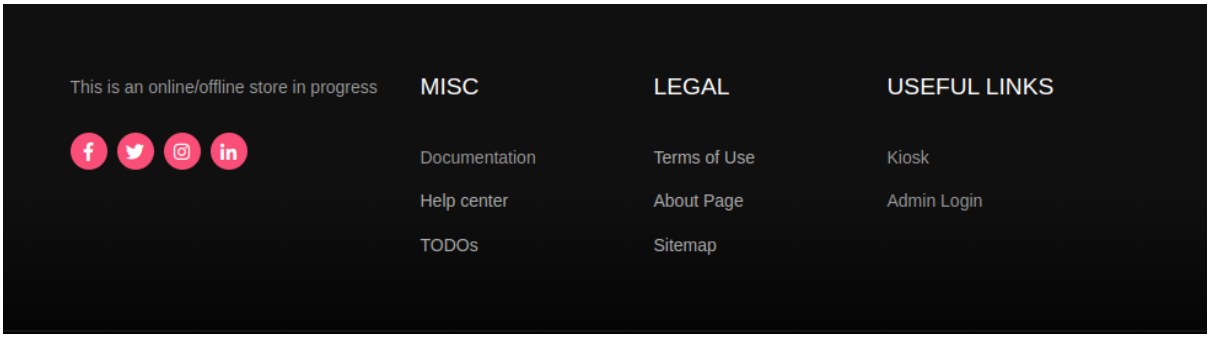

Figure 24; User login footer links

Let go ahead and enter the kiosk link:

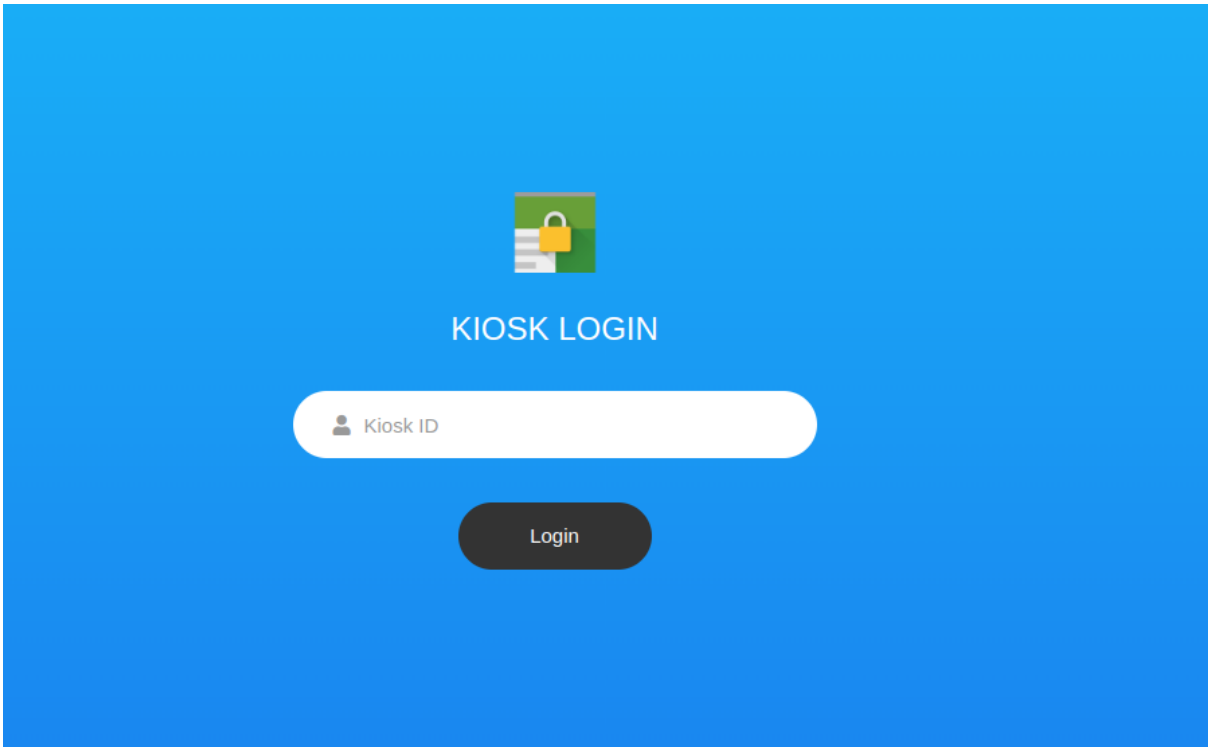

Figure 25; kiosk login page

Notice that in this case, I only need a Kiosk ID, not a username. This is intentional, as in Morrison's I have seen that they only need a passcode of some sort and the managers/shift helpers can login.

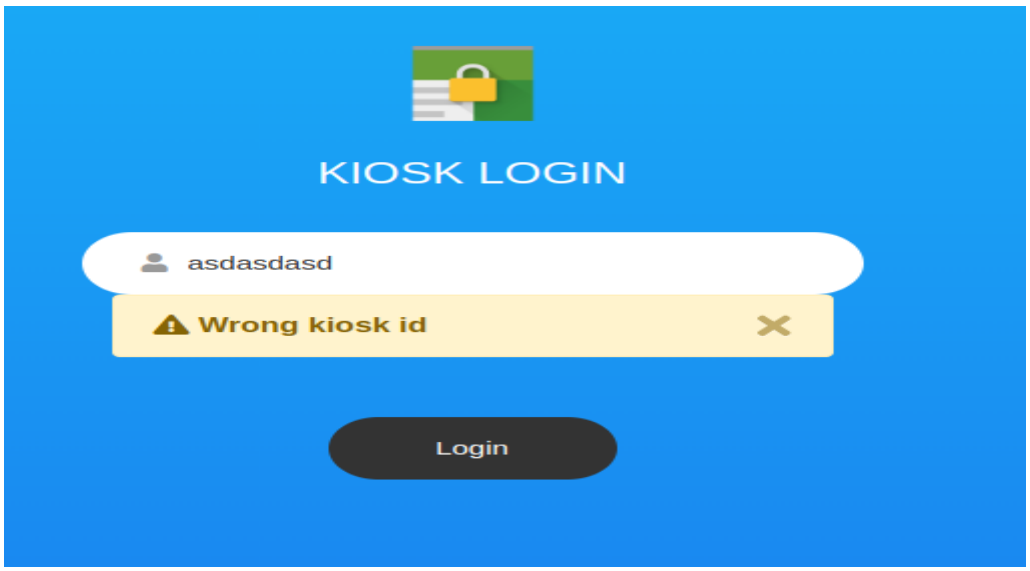

Figure 26; Wrong kiosk id warning

Upon entering correct login details:

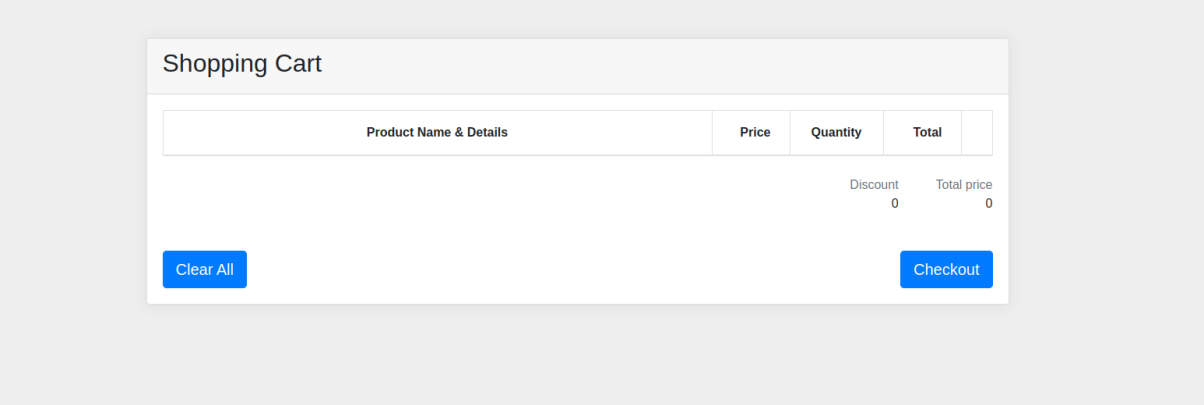

Figure 27; kiosk login success

While I can not really demonstrate it with a picture, this page has a hidden event listener that is only going to focus in a textbox continuously. Also, input from a keyboard is not going to work. In theory, It will, but one has to enter a valid 12 digit correct RFC barcode number within 50ms.

Yes that is intentional, it is meant to be used with a barcode scanner. So the best I can do to show that this test is working is like this:

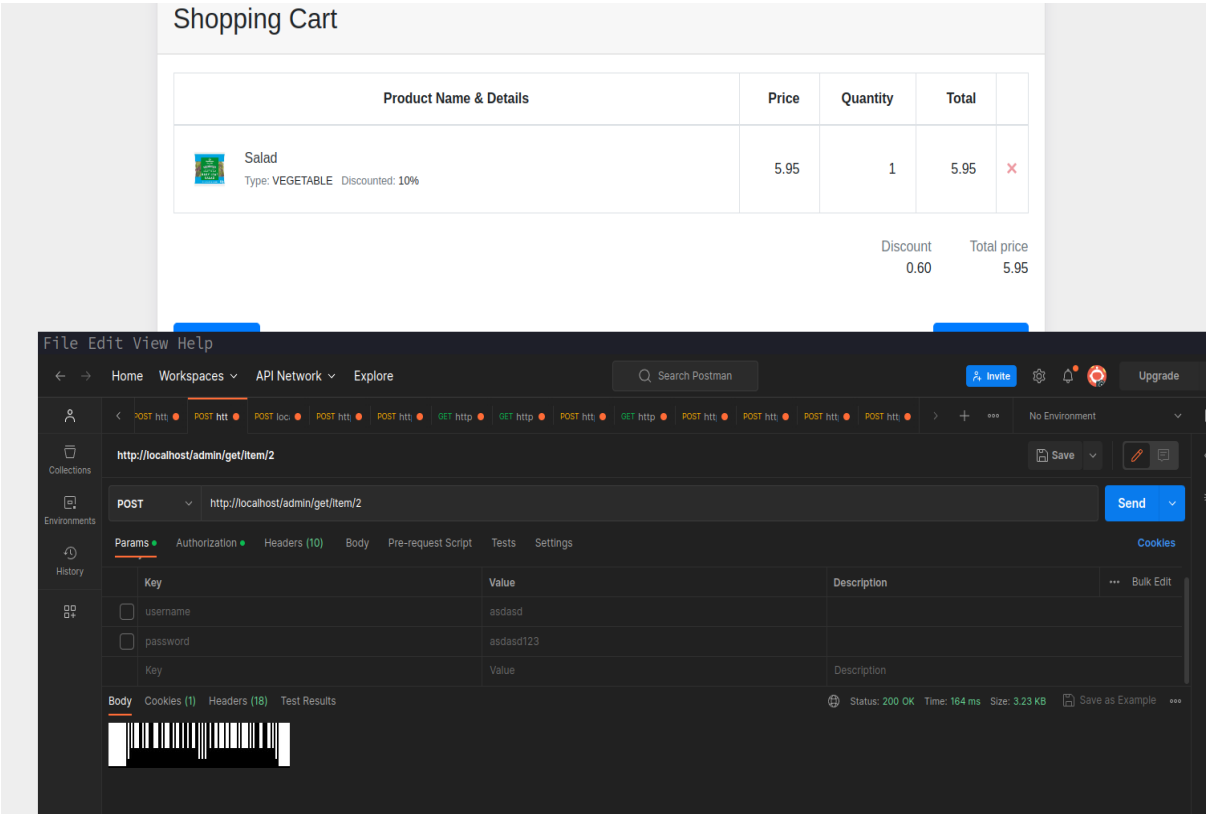

Figure 28; Scanning item with barcode scanner demonstration

Without a barcode scanner it is impossible for the frontend to display a product. So I am going to leave it at that.

Checkout works as well, a popup is displayed:

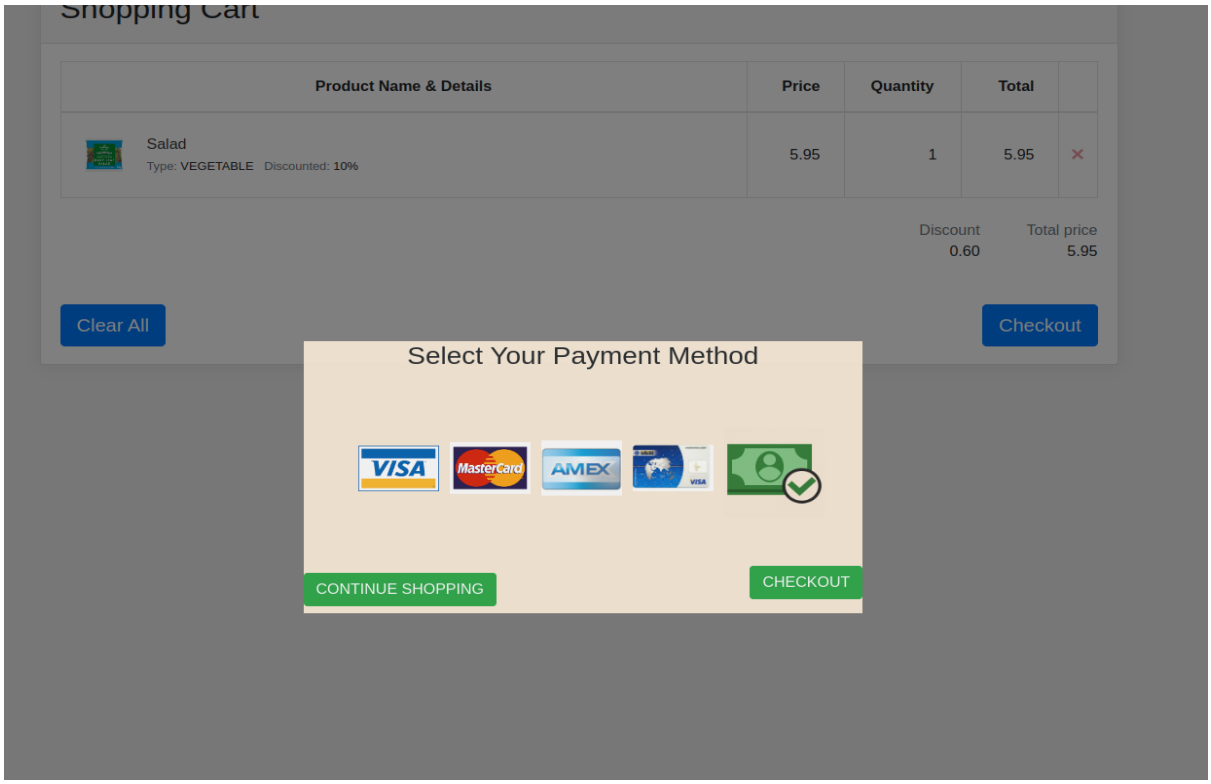

Figure 29; kiosk payment options

upon selecting a payment method:

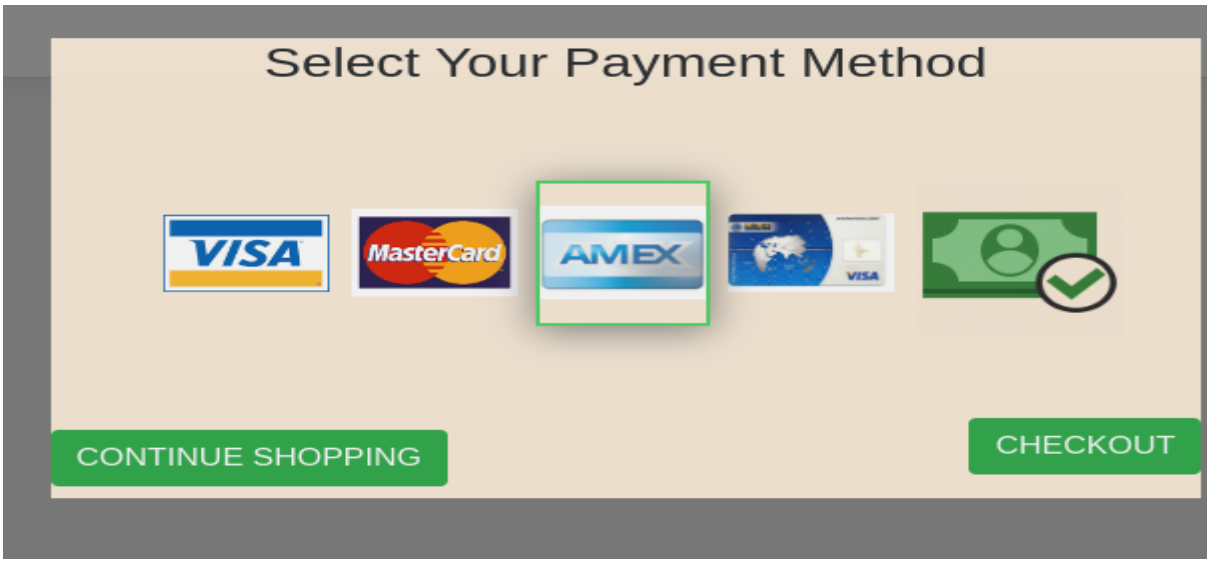

Figure 30; payment options selected

and when Checkout is pressed on that popup, transaction finishes and the kiosk returns to its initial state:

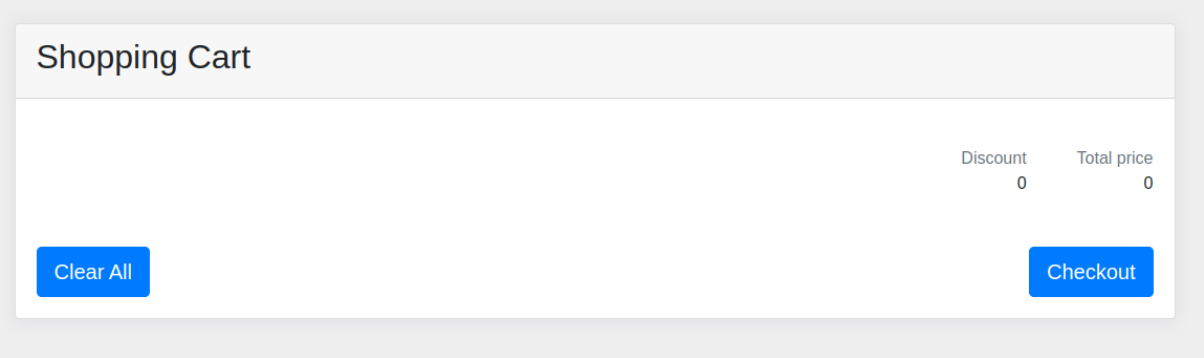

Figure 31; payment success, kiosk reset

Lets go and see what the admin is all about:

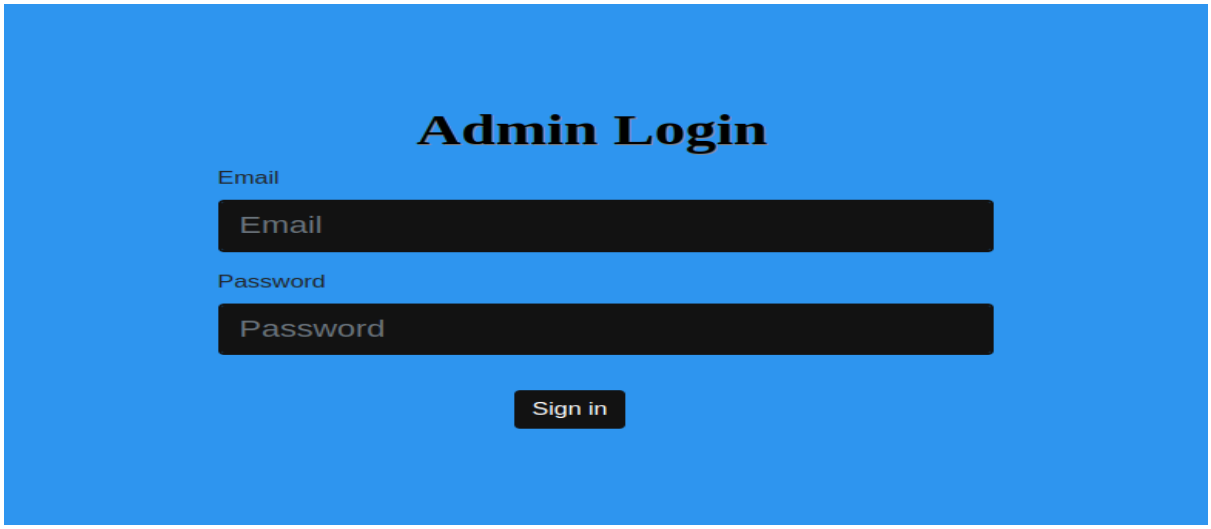

Figure 32; Admin login page

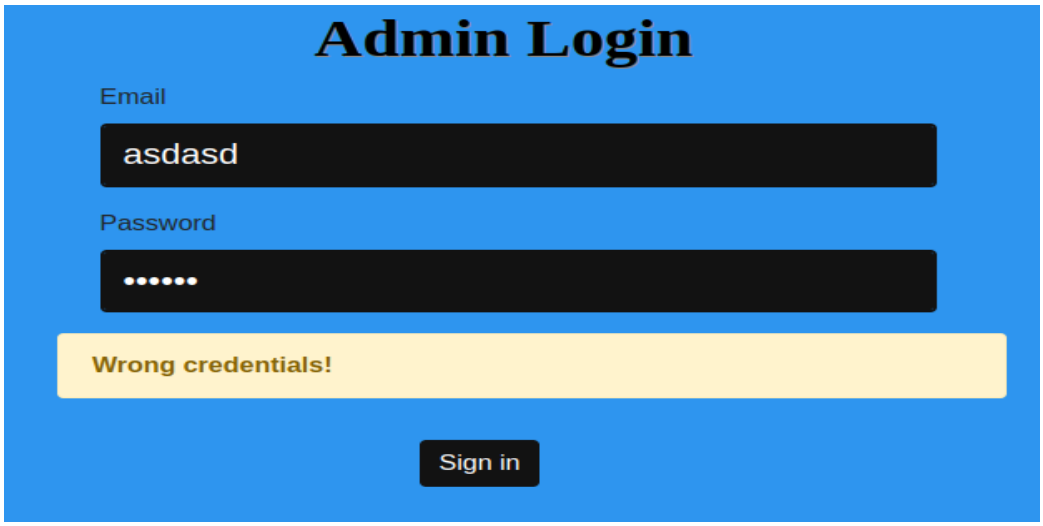

Figure 33; Admin wrong details popup

And when I correctly enter the details:

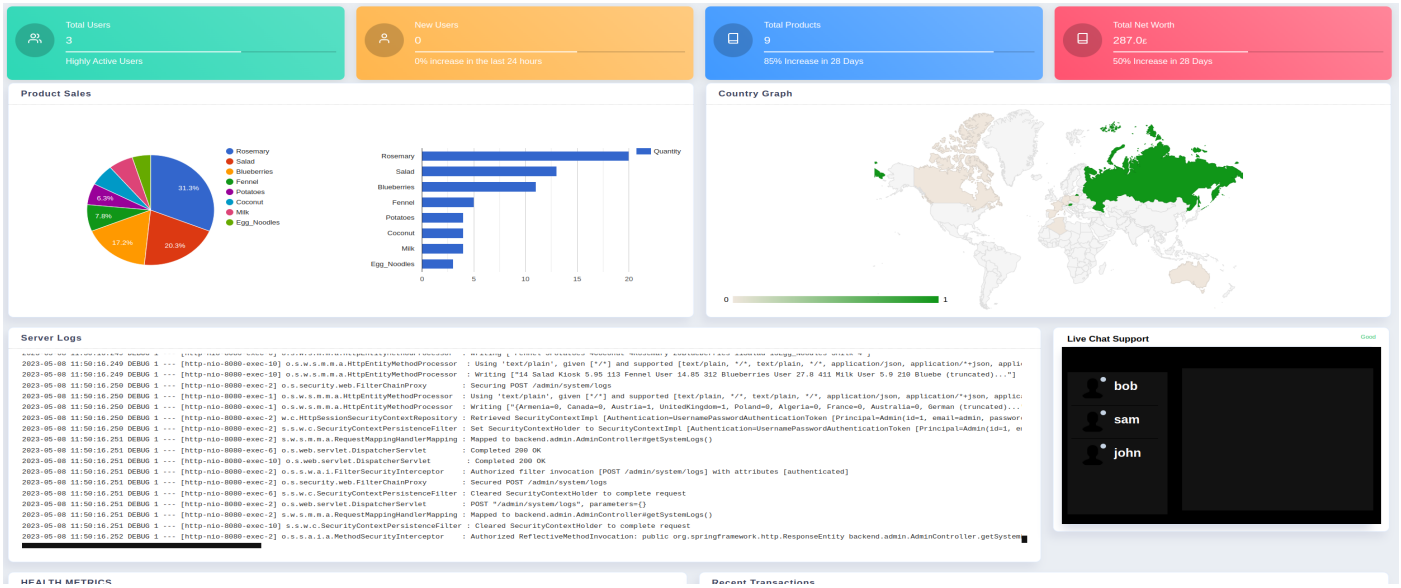

Figure 34; Admin dashboard top part

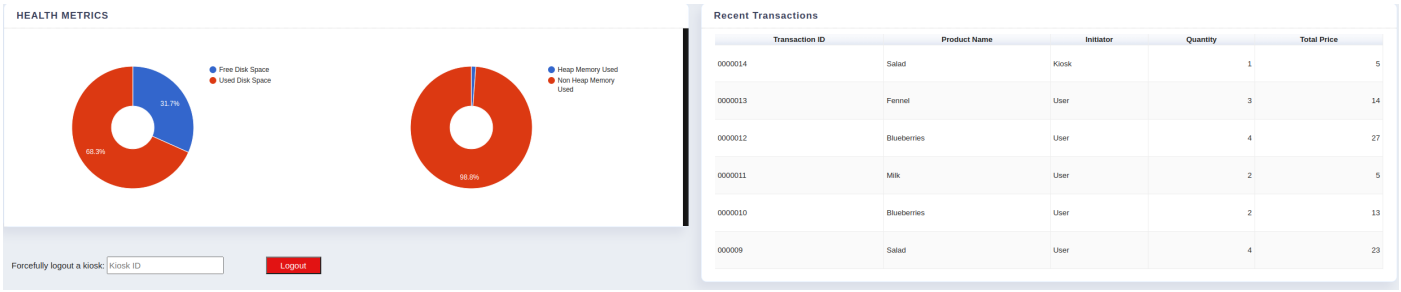

Figure 35; Admin dashboard bottom part

Note the Live Chat support on the right? It works!

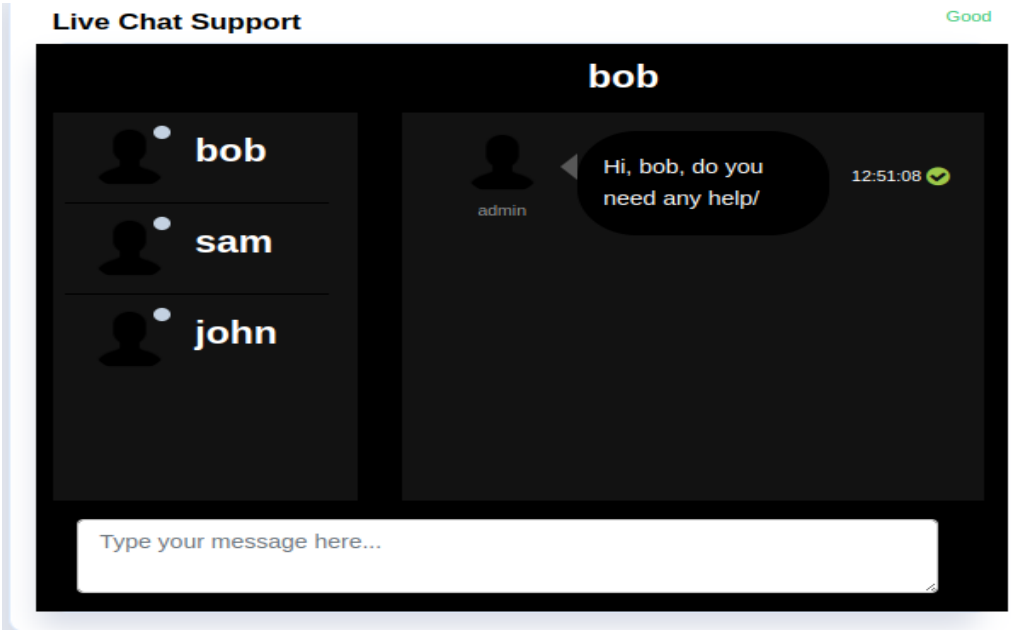

Figure 36; Admin live chat functionality

And from bob's user perspective:

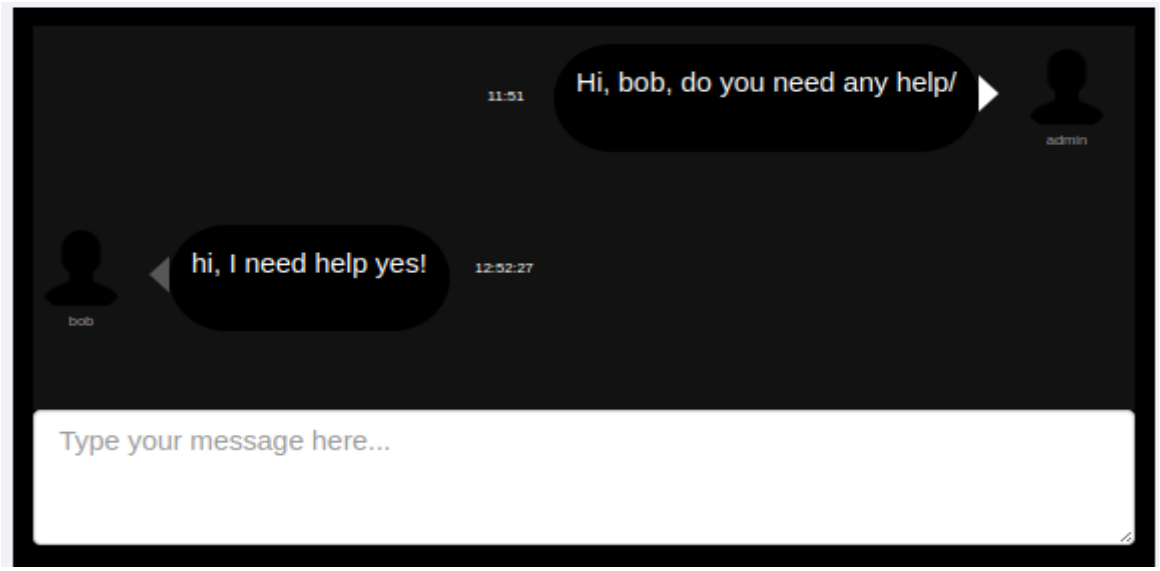

Figure 37; User live chat functionality

There is also a force logout button for the kiosks, since the kiosks are not meant to be logged out, but should anything happen:

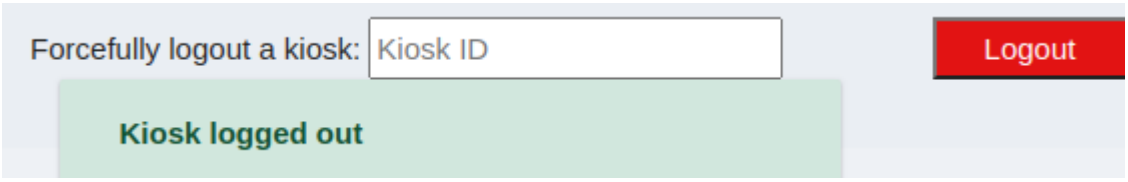

Figure 38; Admin forcefull logout kiosk

Wrong or nonexistent kiosk id:

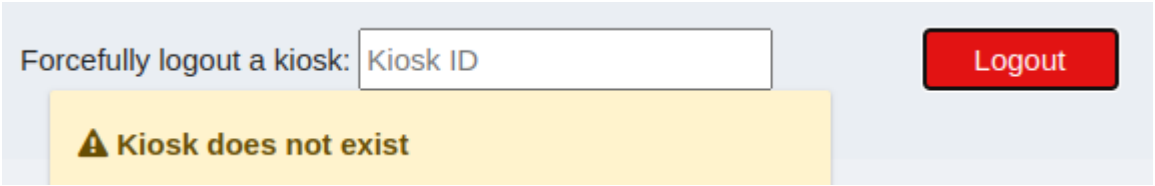

Figure 39; Kiosk does not exist popup

Or just a kiosk that is already logged out:

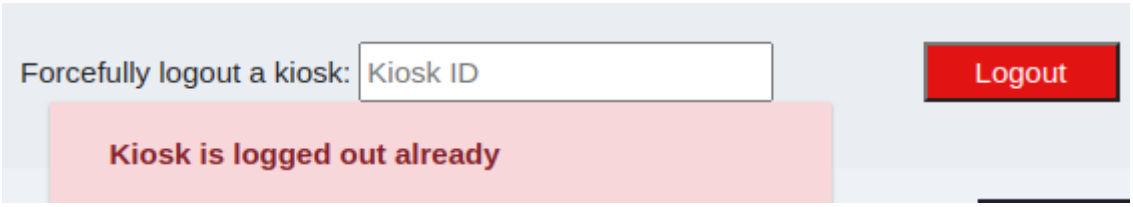

Figure 40; Kiosk logged out already popup

And last, but not least, the Admin's very own logout button:

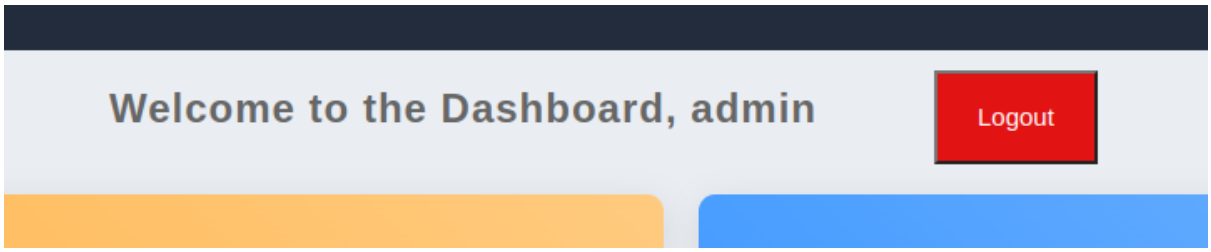

Figure 41; Admin greeter and logout button

on clicking it this is what happens:

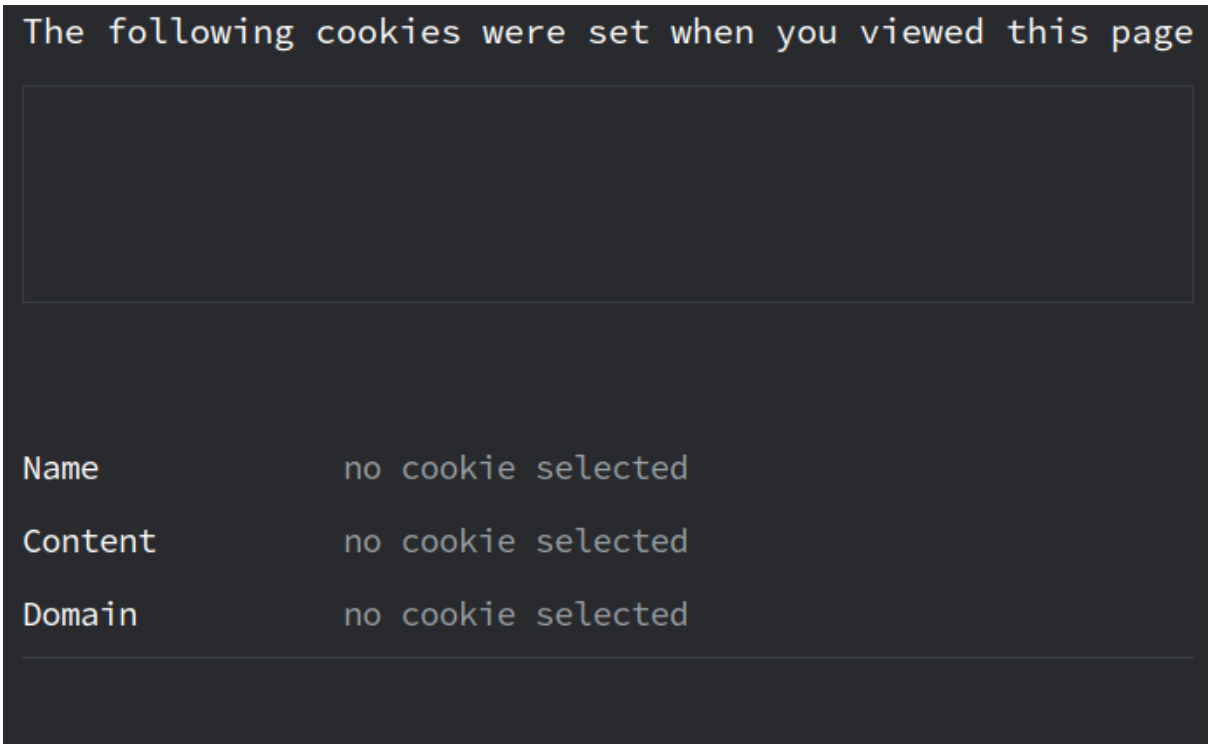

Figure 42; Admin logout button clicked

The cookies and cache are cleared, if any, and then sent back to the login page. If for any reason a page that is not existent is accessed:

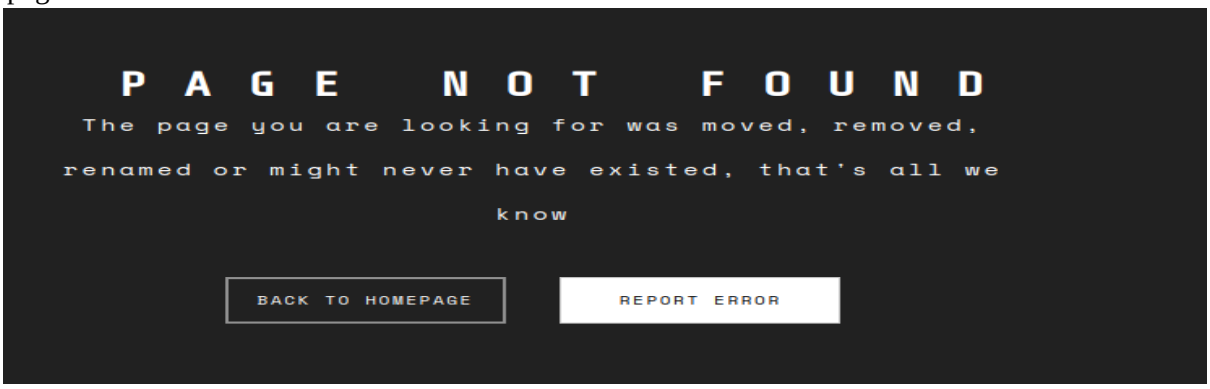

Figure 43; Page not found

I left the best for last. Users normally need to verify their email addresses via a token. I have implemented that very logic in the system.

So when the user registers:

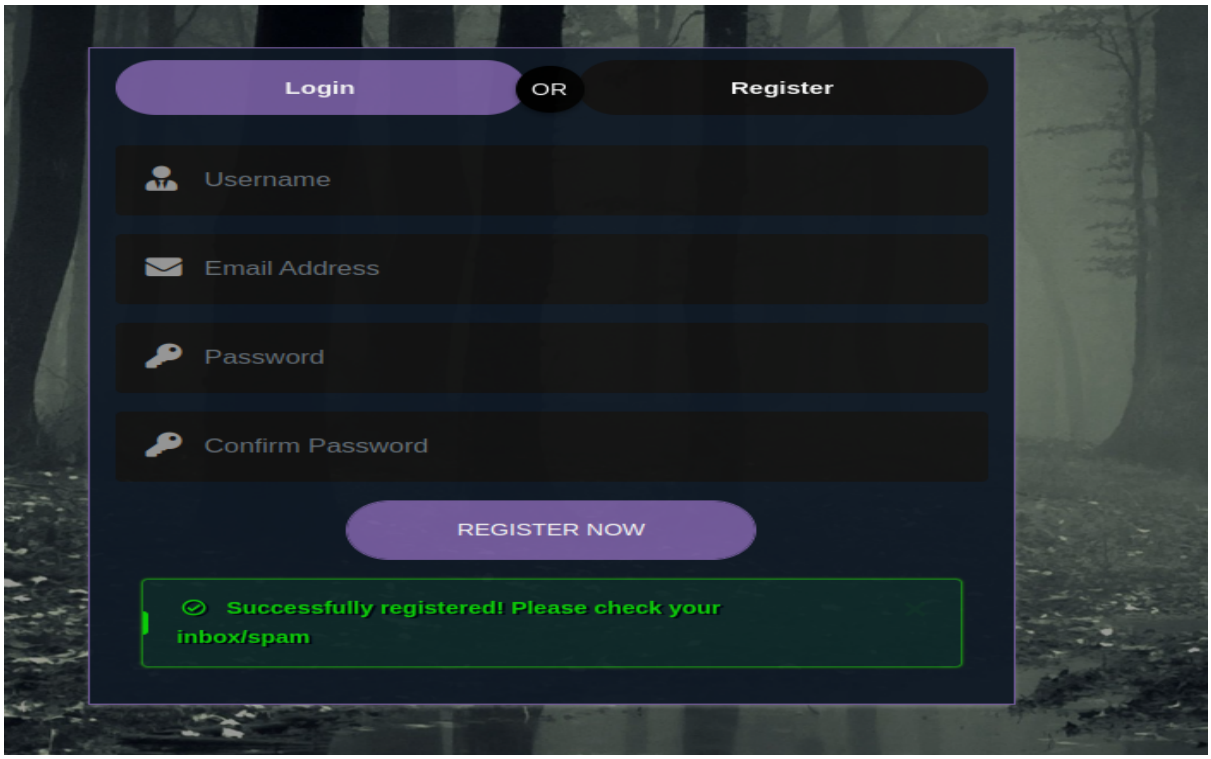

Figure 44; user registered

If the user tries to login before verifying:

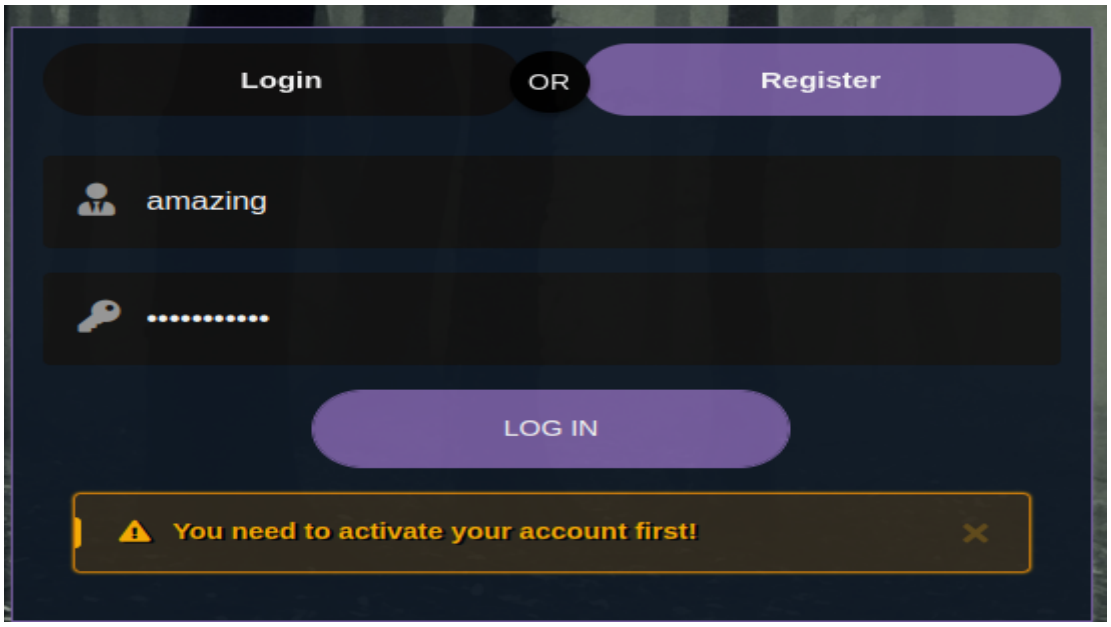

Figure 45; User login without activating their account

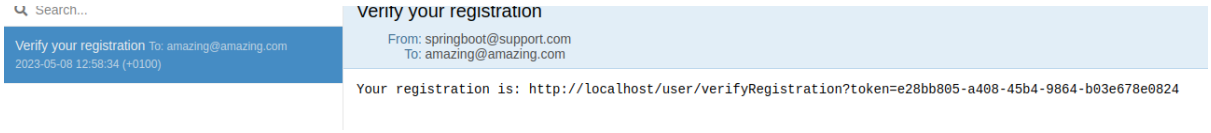

Figure 46; User verification email

After verification is done:

Figure 47; User verified successfully message

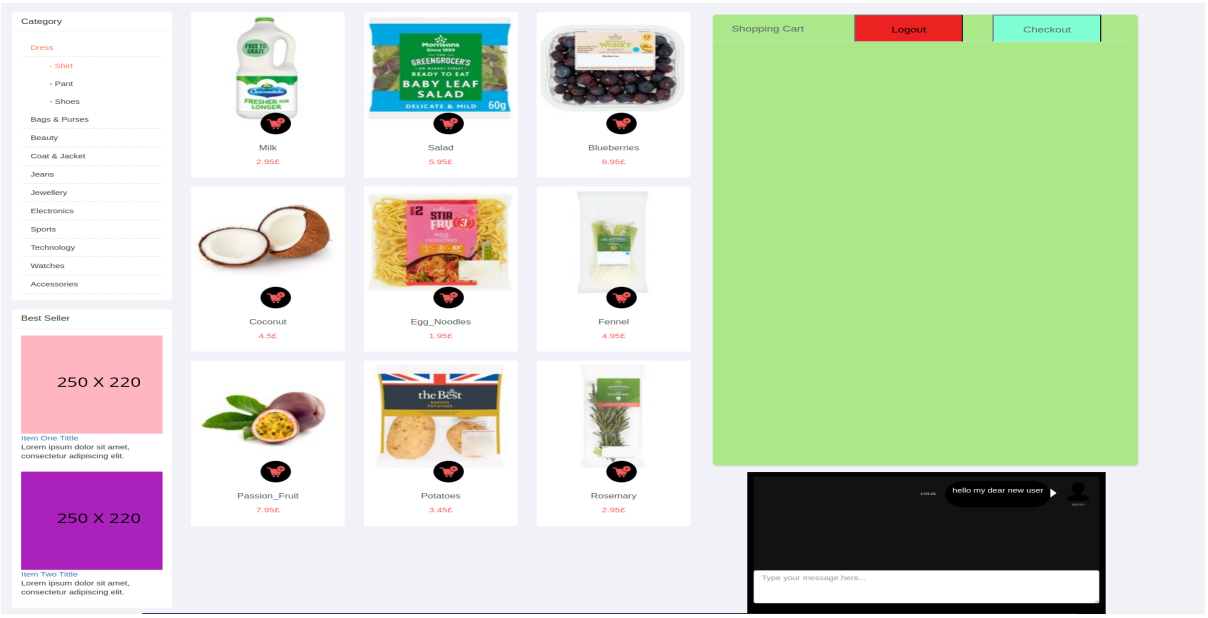

Figure 48; user greeted by admin on register

As for the admins side, a new user entered their user list:

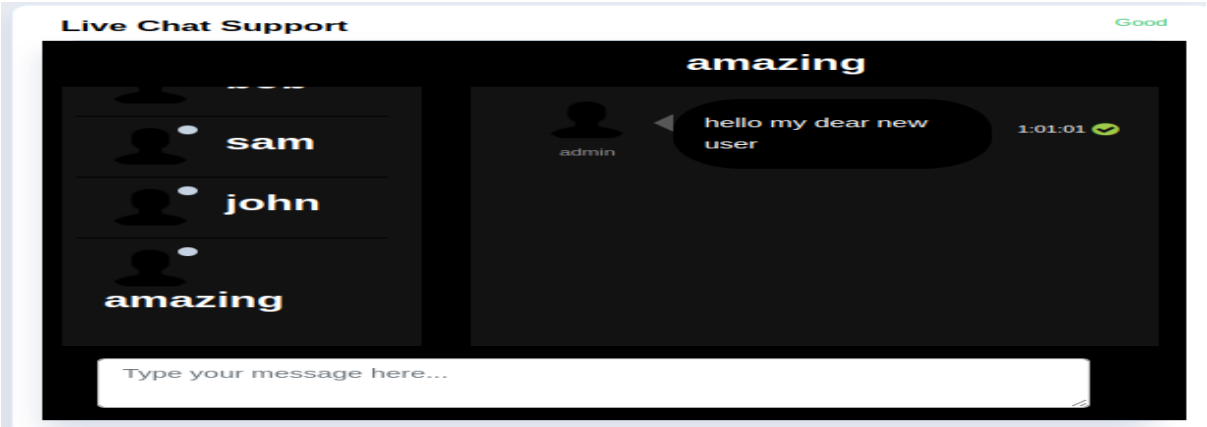

Figure 49; user replying to admin

Testing Plan Table

Feature	USER	ADMIN	KIOSK	ANALYTICS	TOKEN
Login	Works	Works	Works	Works	NA
Register	works	works/ commented	works	NA	NA
Verify	Works	Works	NA	NA	NA
CHAT	Works	Works with support for dynamically adding users to list on register	NA	NA	NA
Error Reporting	All cases covered	All cases covered	All cases covered	All cases covered	All cases covered
Checkout	Works, no success reporting	Works, not in the admin page	Works, success notification included	Works for fetching checkout details	NA
Authentication	Works for all /user/**	Works for all /admin/**	Works for all /kiosk/**	Works for all /analytics/**	NA
Authorization	ROLE_USER principal role	ROLE_ADMIN principal role	ROLE_KIOSK principal role	ROLE_* principal role	NA
Server logs	Unauthorised	Works for admin panel	Unauthorised	Unauthorised	NA
Server metrics	Unauthorised	Works for admin panel	Unauthorised	Unauthorised	NA
Last transactions	Unauthorised	Works for admin panel	Unauthorised	Unauthorised	NA
Server Net Worth	Unauthorised	Works for admin panel	Unauthorised	Unauthorised	NA
Server products total	Unauthorised	Works for admin panel	Unauthorised	Unauthorised	NA
Removed browser auth popup	Yes for all endpoints	Yes for all endpoints	Yes for all endpoints	Yes for all endpoints	NA

Table 4; Testing plan table

This table sums up all direct and indirect testing that could be done. For the indirect ones, while I did not put explicit unauthorized calls to user homepage, for example, it automatically means that If a user tried to enter an admin's endpoint, they would be greeted with a nice 401 or 403.

The test results were documented in a comprehensive report, which included evidence of the testing sessions, computer screen shots, and detailed explanations of any issues that were found. The report also provided evidence of how much of the solution was implemented and working correctly, as well as evidence that the final implementation correlated with the initial specification.

Overall, the testing process was successful, and I were able to demonstrate that the system I have developed works as intended and meets the requirements specified in Chapter 3 of this report. The use of both manual and automated testing methods ensured that all aspects of the system were thoroughly tested, and any issues that were found were addressed promptly. The documentation of the test results provided evidence that the final implementation correlated with the initial specification, and the system met all of the requirements

.

Chapter 6: Evaluation of Results

This chapter evaluates the extent to which the developed decentralized inventory management system has met the project's aim and objectives. The primary aim of this project was to develop a system that focuses on security, performance, easy deployment, and online user support, as well as having administrators and kiosks.

The developed system has achieved this aim by providing robust security features such as HTTPS encryption with trusted certificates, cross-site request forgery (CSRF) protection, and cross-origin resource sharing (CORS) protection. Performance has been a key consideration in the development process, and the system has been designed to be deployed with Docker and Kubernetes, providing efficient resource utilization and scalability.

The system includes user management functionality, with administrators and kiosks as key user roles. Kiosks are offline automated machines that allow in-person users to check out products using barcode or QR code scanning. Online users have the ability to register, login, and communicate with admins through live chat.

The system also provides analytics functionality for products, users, profit, and more, with these metrics displayed on the admin page. Additionally, a server log and server health metrics are available on the admin page for maintenance purposes.

The functional requirements of the system were achieved through the use of the Use Case technique, which facilitated the elicitation and representation of functional requirements. The non-functional requirements were achieved through the use of appropriate software development methodologies, such as agile development and continuous integration and delivery.

To assess the project's success in solving the stated research problem, the developed system was compared to closely related products and related work in the chosen topical area of the project. This comparison revealed that the developed system is unique in its focus on security, performance, and online user support, while also providing robust user management functionality. (A. H. A. Abdullah, A. Ismail, and H. Zainuddin)

Due to the proprietary nature of many existing DIMS, direct comparison with more advanced systems was not possible. However, the development of this system offers a number of significant advantages over traditional inventory management approaches. (T. M. Lekshmi and M. M. Shajahan) These include improved security measures such as https encryption, csrf and cross origin protection, as well as the use of Docker and Kubernetes for easy deployment and maintenance. The inclusion of kiosks and live chat support for users also sets this system apart, as does its comprehensive analytics capabilities for tracking product data and profitability. Overall, these features make the system more secure, efficient, and user-friendly than many existing DIMS solutions.

In terms of performance, the developed system has been optimized for high scalability, enabling it to handle large amounts of data and users without any degradation in performance. This is achieved through the use of containerization technologies such as Docker and Kubernetes, which enable the system to scale up or down depending on the workload.

In conclusion, the developed decentralized inventory management system has achieved the project's aim and objectives by providing robust security, efficient performance, and user-friendly functionality. The use of appropriate software development methodologies and techniques, such as the Use Case technique, enabled the achievement of both functional and non-functional requirements. The system has also solved the stated research problem by providing a unique solution to inventory management that is both secure and user-friendly.

Chapter 7: Conclusions

7.1 A summary of what has been achieved in the project

In this project, a decentralized inventory management system (DIMS) was developed with a focus on security, performance, and ease of deployment. The system includes features such as support for admins, kiosks, and online users, as well as analytics and monitoring capabilities. The system also utilizes Docker/Kubernetes for easy deployment and HTTPS encryption with trusted certificates, CSRF, and cross-origin protection for enhanced security.

One of the key achievements of this project is the development of a highly secure and performant DIMS that is easy to deploy and use. The use of modern technologies such as Docker and Kubernetes ensures that the system is highly scalable and reliable, while the inclusion of HTTPS encryption and other security features ensures that user data is protected at all times.

Another unique feature of this DIMS is its support for kiosks, which allows in-person users to easily checkout products using barcode or QR code scanning. This feature can greatly improve the user experience and make the system more accessible to a wider range of users. Additionally, the inclusion of analytics and monitoring capabilities allows admins to gain valuable insights into user behavior and product performance, which can inform future business decisions.

Overall, the development of this DIMS represents a significant achievement in the field of inventory management systems, and has the potential to greatly improve the efficiency and security of businesses in various industries.

7.2 Reflections and lessons learned

In the course of the project, it became evident that there are certain legal considerations that should be taken into account when developing a decentralised inventory management system. One of the key legal issues that emerged was data privacy and protection, particularly with respect to the collection, storage, and use of user data. As the system collects and stores user data, it is important to comply with relevant data protection regulations, such as the General Data Protection Regulation (GDPR).

In addition to data protection, there are also legal issues related to the use of third-party software and components, particularly those that are licensed under open source licenses. I had to carefully review the terms and conditions of these licenses and ensure that they were in compliance with the relevant license agreements.

Moreover, it is important to consider the intellectual property rights associated with the project, including any software, hardware, or other products that were developed. I had to ensure that all

intellectual property rights were properly assigned and that any third-party intellectual property rights were respected.

Finally, I had to consider any potential liability issues that may arise from the use of the system, particularly in cases where the system may be used in a commercial setting. This required a careful review of any potential risks associated with the system and the implementation of appropriate risk management strategies.

7.3 Future work

I researched a lot on where I could go with this project. The most interesting things that I found so far can be summed up likewise:

- Integration with blockchain technology: The use of blockchain technology can further enhance the security and transparency of the inventory management system. It can ensure that all transactions are secure and immutable, which adds an additional layer of trust and authenticity to the system.

- Integration with machine learning algorithms: The use of machine learning algorithms can help in predicting inventory levels, identifying patterns, and forecasting demand. This help the backend in streamlining the inventory management process and reducing inventory holding costs.

- Integration with IoT devices: The integration of IoT devices can help in tracking inventory in real-time, reducing errors in the inventory management process, and automating manual processes. It can also help in improving the accuracy of demand forecasting.

- Expansion to other industries: The inventory management system can be expanded to other industries such as retail, manufacturing, and logistics. This requires the development of industry-specific features and customisations.

- Integration with other systems: The system can be integrated with other systems such as accounting and supply chain management software to provide a comprehensive solution to businesses.

- Developing a mobile application: Developing a mobile application can provide users with easy access to the inventory management system, enabling them to monitor inventory levels and perform inventory transactions on the go.

- Integration with e-commerce platforms: The integration of the inventory management system with e-commerce platforms can help in automating the order fulfilment process, reducing lead times, and improving customer satisfaction.

Appendices

Appendix A: Project Management

1.1 The original project plan from the Proposal

No.	Activities	Estimated Duration	Activity Description
1	Implement Questionnaires and gather Data	2 weeks	Hand out questionnaires on local residents in order to gather analytic data for the User Interface component
2	Review Data and gather overall statistics	2 week	Review all data collected, categorise it and cross examine it with real life world stock market platforms
3	Implement Backend Services	5 weeks	Start developing services which will be used to interact with Database and User Interface for data transfer
4	Construct User Interface System	4 weeks	Create User Interface for the user interaction
5	Create Database Interaction Services and other components	3 weeks	Create and revisit the data handling services as well as other components tighly related to it
6	Design secure data transfer protocols	2 weeks	Thoroughly harden all aspects of the web application, especially the user data transfer services
7	Construct User Interface System	4 weeks	With the data collected from analytics, construct a user-friendly UI
8	Systems' Testing	3 weeks	Conduct tests on both Systems
9	Evaluation of Developed Systems'	2 weeks	Evaluate and check if the system is up to standards before deploying
10	Finalise Documentation report	3 weeks	Complete project documentation with spell-checks, format fixing and technical system resources required
	Total Duration	26 weeks	

Table 5; Original project plan

1.2 Review of the project process

Ahead of schedule

No.	Activities	Estimated Duration	Activity Description

1	Implement Questionnaires and gather Data	2 weeks	Hand out questionnaires on local residents in order to gather analytic data for the User Interface component
2	Review Data and gather overall statistics	2 week	Review all data collected, categorise it and cross examine it with real life world stock market platforms
3	Implement Backend Services	10 weeks	Start developing services which will be used to interact with Database and User Interface for data transfer
4	Construct User Interface System	4 weeks	Create User Interface for the user interaction
5	Create Database Interaction Services and other components	3 weeks	Create and revisit the data handling services as well as other components tighly related to it
6	Design secure data transfer protocols	2 weeks	Thoroughly harden all aspects of the web application, especially the user data transfer services
7	Construct User Interface System	4 weeks	With the data collected from analytics, construct a user-friendly UI
8	Systems' Testing	3 weeks	Conduct tests on both Systems
9	Evaluation of Developed Systems'	1.5 weeks	Evaluate and check if the system is up to standards before deploying
10	Finalise Documentation report	3.5 weeks	Complete project documentation with spell-checks, format fixing and technical system resources required
	Total Duration	33 weeks	

Table 6; Original Gantt chart

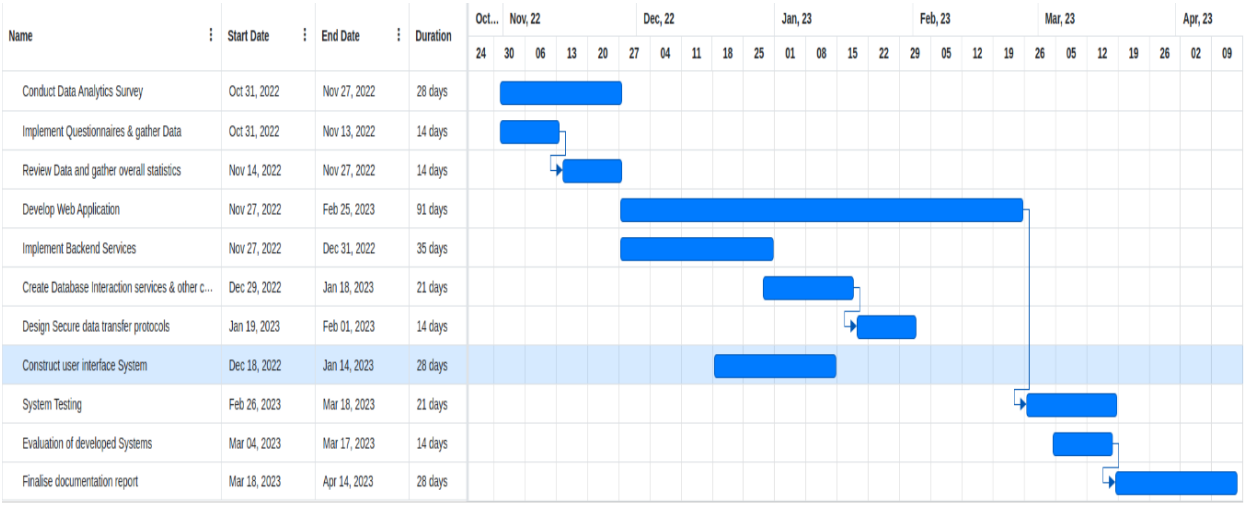

Table 7; Original Graphical Gantt Chart

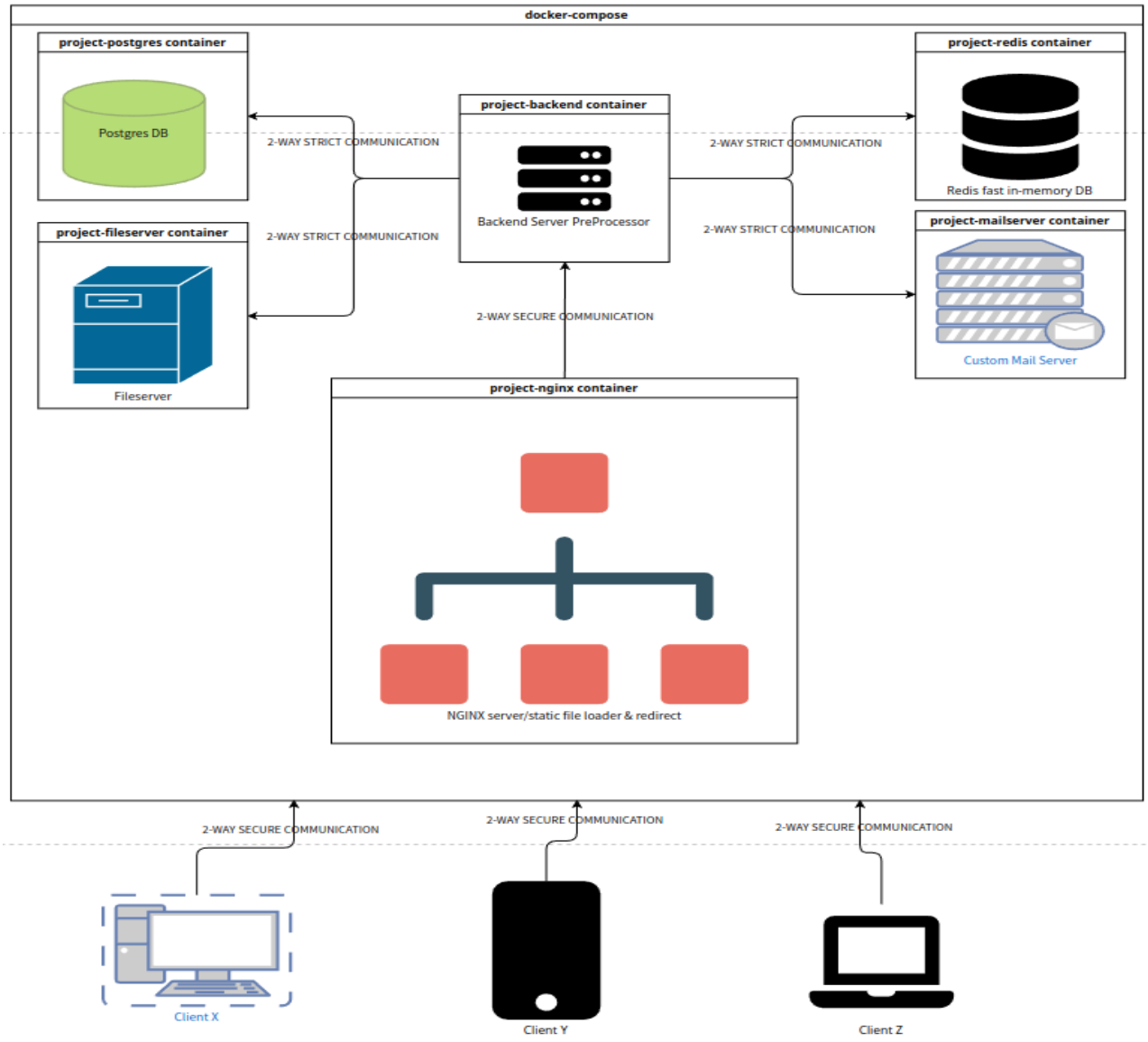

Figure 50; High level project logic

1.3 Amendments to the original plan

1.3.1 Remedial actions should be included if you're behind the schedule

I have gotten very far, no feature that was mentioned above is not implemented. The project is way ahead of schedule considering the implementation as well as the Interrim submission report.

1.4 Lessons learned in project management

After completing this project, I have learned several valuable lessons in project management. One of the most important lessons I learned is the importance of setting realistic timelines and milestones. During the course of the project, I found that some of the tasks took longer than anticipated, and this caused delays in other aspects of the project. In the future, I am going to ensure that I have more accurate estimates of the time required for each task and build in extra time as a buffer to account for any unforeseen issues that may arise.

Another lesson I learned is the importance of effective communication throughout the project. Clear and consistent communication with all stakeholders is essential to ensure that everyone is aware of the project's progress and any issues that arise. In the future, I am going to establish more effective channels of communication and schedule regular progress updates to keep everyone informed.

Finally, I learned that project management requires a great deal of flexibility and adaptability. Despite careful planning, unforeseen issues can arise, and it's important to be able to adjust plans and strategies accordingly. In the future, I will make sure that I am more flexible and prepared to make changes to the project as needed to ensure its success.

Appendix B: Survey responses

Some snapshots from the forms that I used to collect useful data for the frontend. Although the results are more or less equal, it helped me, nonetheless, to keep in mind certain things while designing the frontend.

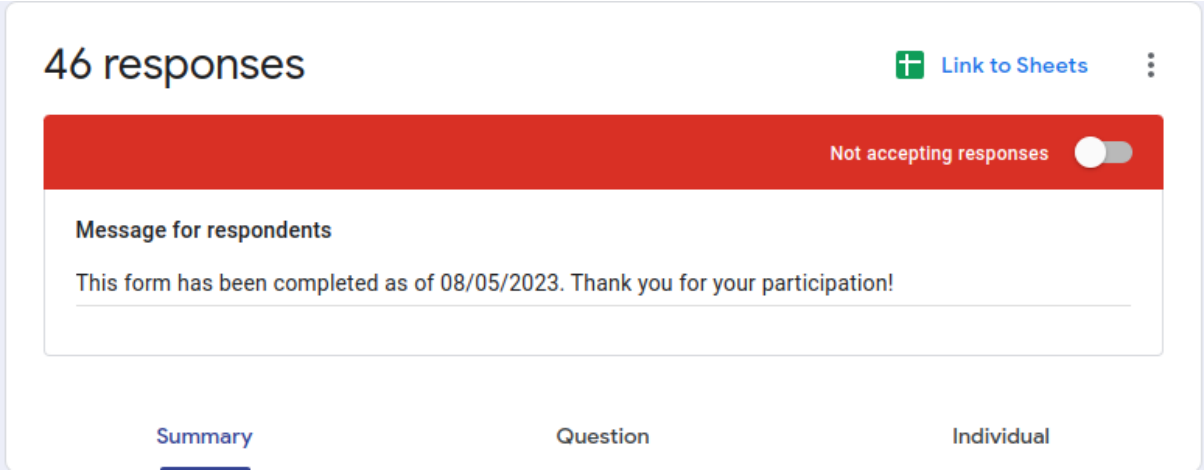

Figure 51; Survey total respondents

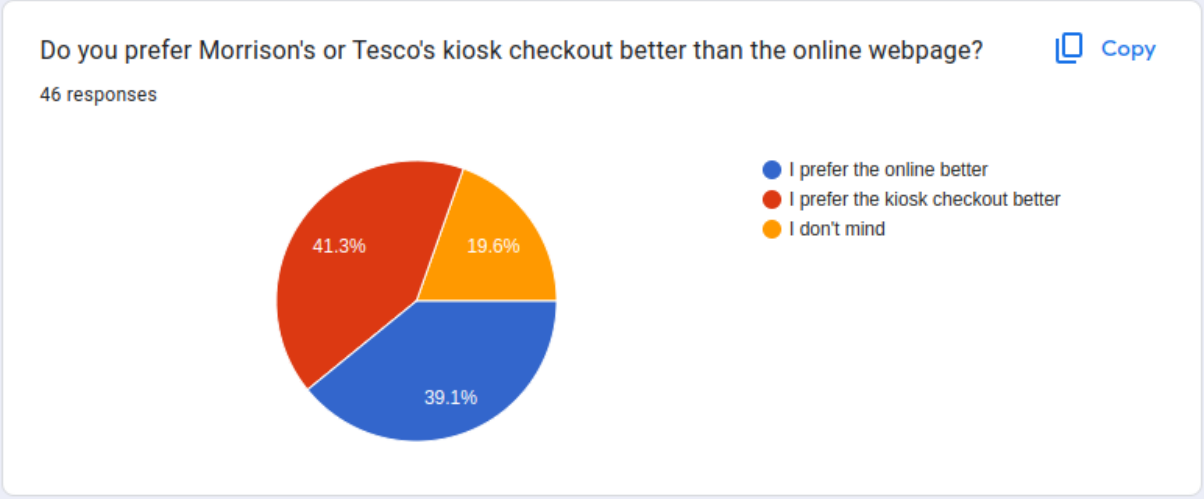

Figure 52; Survey part 1/6

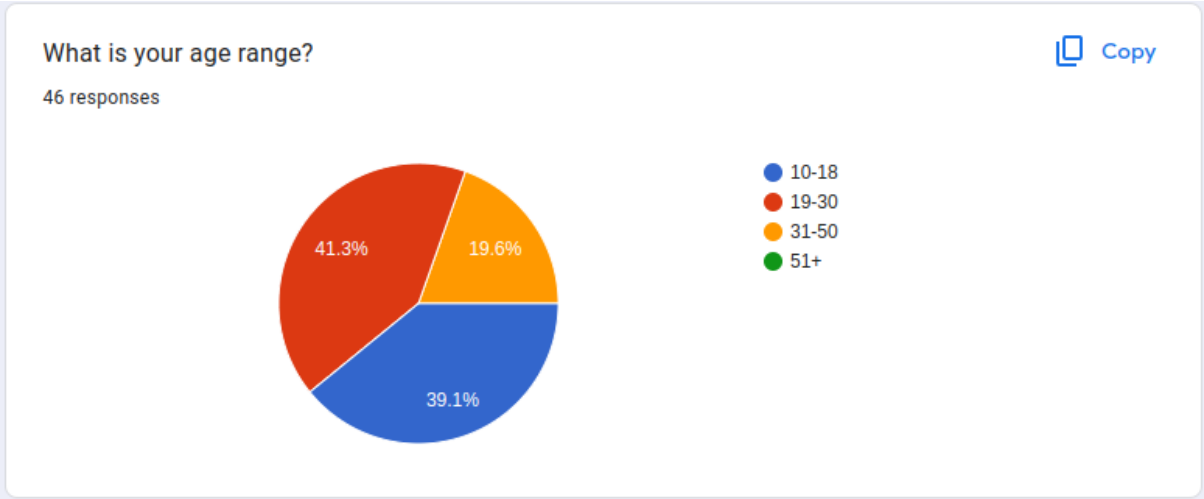

Figure 52; Survey part 2/6

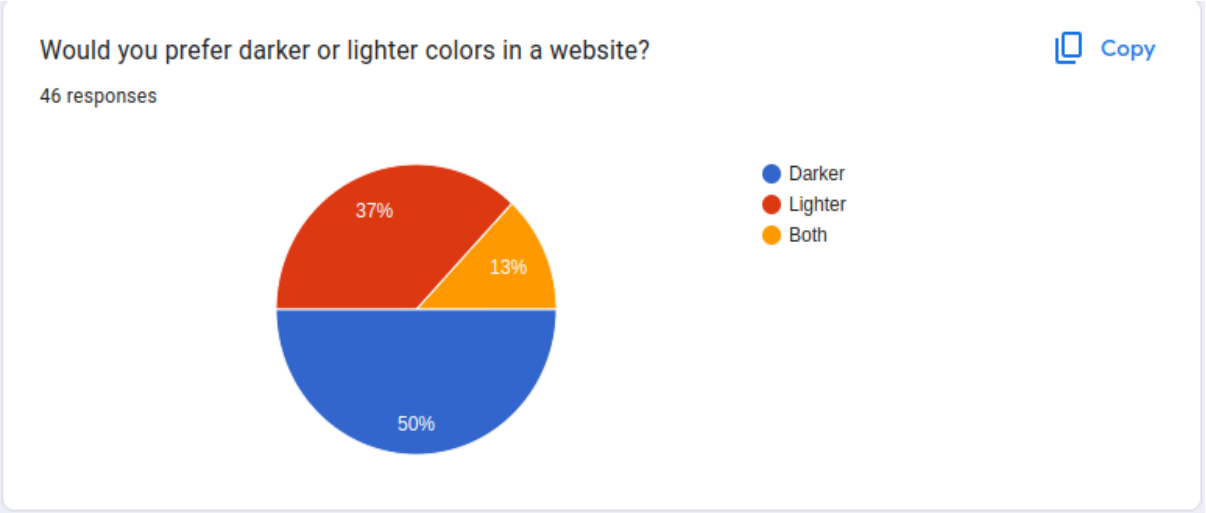

Figure 53; Survey part 3/6

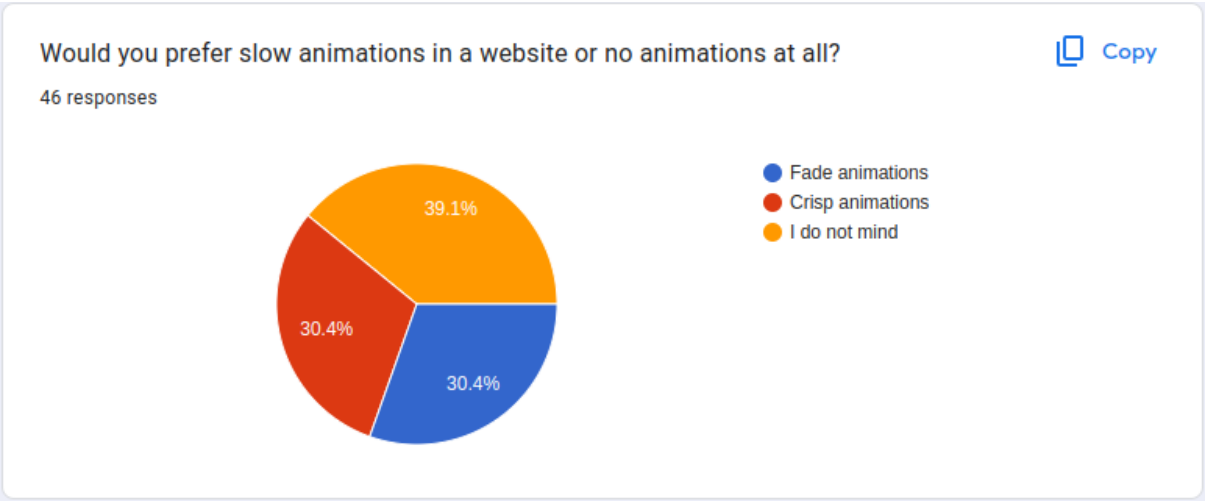

Figure 54; Survey part 4/6

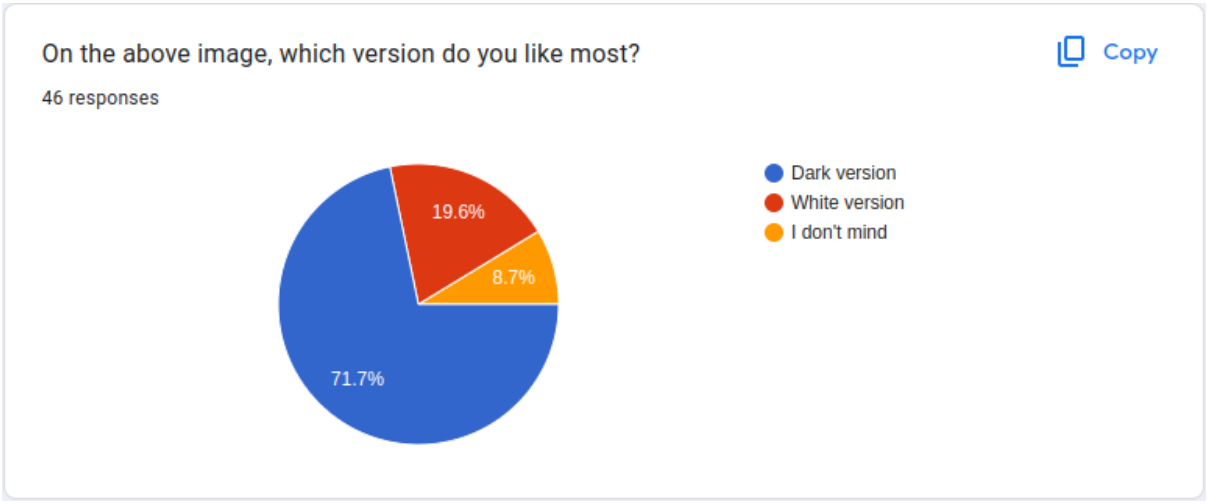

Figure 55; Survey part 5/6

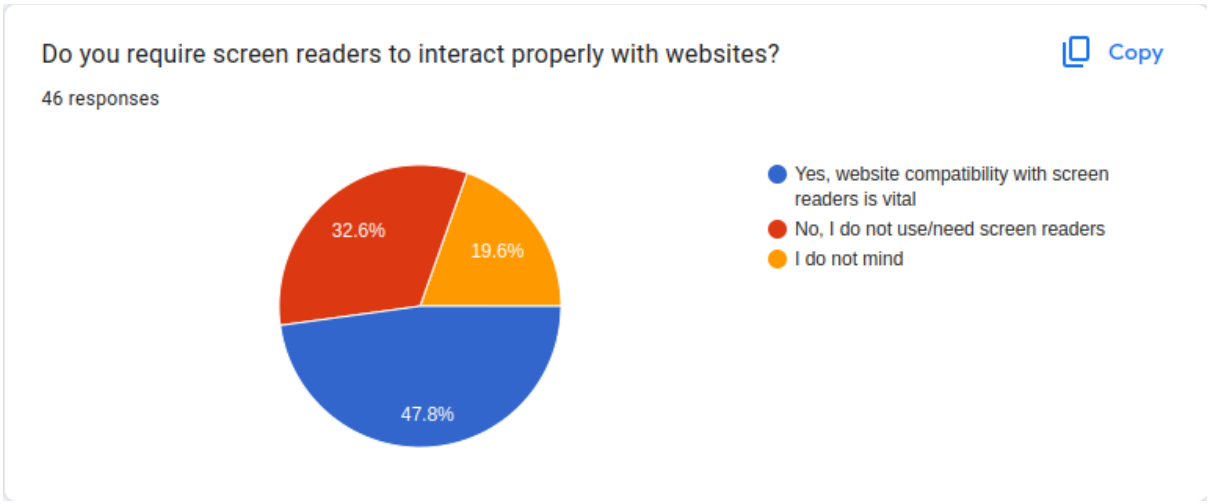

Figure 56; Survey 6/6

Acknowledgements

I would like to express my gratitude to Dr. Victor Sowinski - Mydlarz and Dr. Sandra Fernando for their invaluable support and guidance throughout the project. Their expertise, knowledge and encouragement have been instrumental in shaping my research and improving the quality of this paper.

Dr. Victor’s unwavering dedication to this project has been truly inspiring, and his insightful feedback and suggestions have helped me to push the boundaries of my research. His commitment to excellence and his passion for the subject matter have motivated me to strive for the highest standards.

I would also like to thank Dr. Sandra for her tireless efforts in providing me with constructive criticism and feedback, which has been critical to my success in this project. Her expertise in the field and her excellent communication skills have been an asset to my research, and I am grateful for her contributions.

I am deeply indebted to both Dr. Victor and Dr. Sandra for their invaluable contributions, support, and encouragement throughout this project. Their guidance has been instrumental in shaping my research, and I am honored to have had the opportunity to work with such talented and dedicated individuals.

References

- Li, Y., Li, D., Zhang, H., & Song, X. (2017). Decentralized inventory management in a two-stage supply chain with competing retailers. International Journal of Production Economics, 185, 69-78.
- Li, L., Li, Y., & Li, B. (2019). Blockchain-enabled decentralized inventory management in supply chains. International Journal of Production Research, 57(4), 981-1000.
- Nembhard, H. B., Hu, Q., & Zhang, Y. (2017). Decentralized Inventory Management in Multi-Echelon Supply Chains with Demand Uncertainty. In Proceedings of the 2017 Winter Simulation Conference (pp. 1862-1873).
- "Performance Comparison of Java, C++, and Python for Scientific Application" by Hyungro Lee, et al. (2016): https://ieeexplore.ieee.org/document/7472727
- "A Comparative Study of Containerization Tools: Docker, LXC, LXD, OpenVZ, and Linux-VServer" by Naveed Ahmed, et al. (2019): https://ieeexplore.ieee.org/document/8790696
- "A Comparative Study of Java, Ruby, and Python for High-Performance Computing Applications" by Luiz Angelo Steffenel, et al. (2018): https://ieeexplore.ieee.org/document/8414519
- "Rust: A Secure and Safe Systems Programming Language" by Steve Klabnik and Carol Nichols (2019): https://ieeexplore.ieee.org/document/8725716
- "Performance Comparison of Rust and C++ for Scientific Computing" by Martin Dyring-Andersen, et al. (2017): https://ieeexplore.ieee.org/document/8107372
- Martin, R.C., 2003. Agile software development: principles, patterns, and practices. Prentice Hall.
- Sommerville, I., 2016. Software engineering. Pearson Education Limited.
- A. H. A. Abdullah, A. Ismail, and H. Zainuddin, "Decentralized inventory management system in supply chain network," in 2016 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM), 2016, pp. 251-255.
- H. Adachi and K. Nakamura, "Design of a decentralized inventory management system for a supply chain network," Journal of Japan Industrial Management Association, vol. 66, no. 4, pp. 199-209, 2015.
- Y. Ma, X. Li, and G. Lu, "Research on a decentralized inventory management system in the supply chain based on game theory," Journal of Applied Mathematics, vol. 2014, Article ID 628536, 10 pages, 2014.
- T. M. Lekshmi and M. M. Shajahan, "Decentralized inventory management system for a multi-product supply chain under demand and lead time uncertainty," Journal of Industrial and Management Optimization, vol. 16, no. 1, pp. 301-319, 2020.

Bibliography

- R. S. Pressman, "Software Engineering: A Practitioner's Approach," McGraw-Hill, 2010.
- S. McConnell, "Code Complete: A Practical Handbook of Software Construction," Microsoft Press, 2004.
- M. Fowler, "Refactoring: Improving the Design of Existing Code," Addison-Wesley Professional, 2018.
- M. Kuhn, "Building Microservices: Designing Fine-Grained Systems," O'Reilly Media, 2015.
- J. E. Sommerville, "Software Engineering," Pearson Education, 2016.
- B. W. Boehm, "Software Engineering Economics," Prentice-Hall, 1981.
- J. Ousterhout, "Tcl and the Tk Toolkit," Addison-Wesley Professional, 1994.
- R. Pike, D. Presotto, S. Dorward, B. Flandrena, K. Thompson, H. Trickey, and P. Winterbottom, "Plan 9 from Bell Labs," ACM Transactions on Computer Systems, vol. 8, no. 2, pp. 191-223, 1990.
- N. Jones, B. Noble, and R. Biddle, "Empirical software engineering: An emerging area," IEEE Computer Society Press, 1991.
- J. F. Ramil and M. L. Peters, "Software Metrics: A Rigorous and Practical Approach," CRC Press, 2002.
- Booch, G., Rumbaugh, J. and Jacobson, I., 2005. The unified modeling language user guide. Addison-Wesley Professional.
- Gamma, E., Helm, R., Johnson, R. and Vlissides, J., 1994. Design patterns: elements of reusable object-oriented software. Addison-Wesley Professional.
- Fowler, M., 2002. Patterns of enterprise application architecture. Addison-Wesley Professional.
- Beck, K., 2000. Extreme programming explained: embrace change. Addison-Wesley Professional.
- McConnell, S., 2004. Code complete. Microsoft Press.
- Ambler, S.W., 2002. Agile database techniques: effective strategies for the agile software developer. Wiley.
- Hunt, A. and Thomas, D., 1999. The pragmatic programmer: from journeyman to master. Addison-Wesley Professional.