

# Final Year Project Report

Full Unit - Final

---

## Advanced Web Development

Edward Instance

---

A report submitted in part fulfilment of the degree of

**BSc in Computer Science**

**Supervisor:** Christos Dexiades



Department of Computer Science  
Royal Holloway, University of London

April 24, 2025

# Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 14,760

Student Name: Edward Instance

Date of Submission: 24/4/2025

Signature: einstance

# Table of Contents

Abstract . . . . .	6
1 Introduction . . . . .	7
1.1 The problem . . . . .	7
1.2 Literature review . . . . .	7
1.3 Aims, objectives and user stories . . . . .	8
1.3.1 Aims . . . . .	8
1.3.2 Objectives . . . . .	8
1.3.3 User stories . . . . .	9
1.4 Professional Issues . . . . .	10
1.4.1 Shopping platforms . . . . .	10
1.4.2 Google . . . . .	11
1.4.3 Cloud Providers . . . . .	11
1.4.4 Recommendations . . . . .	12
2 State of the art web development . . . . .	13
2.1 Frontend . . . . .	13
2.1.1 Technologies . . . . .	13
2.1.2 Frameworks . . . . .	14
2.1.3 Platforms . . . . .	15
2.2 Backend . . . . .	15
2.2.1 Technologies . . . . .	15
2.2.2 Frameworks . . . . .	16
2.2.3 Platforms . . . . .	17
2.3 Database . . . . .	17
2.3.1 Technologies . . . . .	17
2.4 CI/CD . . . . .	18
2.4.1 Technologies . . . . .	18
2.4.2 Platforms . . . . .	19
2.5 Infrastructure . . . . .	19
2.5.1 Technologies . . . . .	19
2.5.2 Platforms . . . . .	19
2.5.3 Amazon Web Services . . . . .	19
2.6 Financial . . . . .	20
2.6.1 Platforms . . . . .	20

2.7	Global	20
2.7.1	Technologies	20
3	Architectures, Design Patterns and other aspects	22
3.1	Architectural paradigms	22
3.1.1	N-tier architecture	22
3.1.2	Monolithic architecture	22
3.1.3	Microservice architecture	22
3.1.4	Serverless architecture	23
3.1.5	Event driven architecture	23
3.2	Design patterns	23
3.2.1	Model view controller (MVC)	23
3.2.2	Client-Server Model	23
3.2.3	Component-based architecture	24
3.2.4	Singleton Method Design Pattern	24
3.3	Security	24
3.3.1	Authorization and Authentication	24
3.3.2	Role Based Access Control	25
3.3.3	SQL Injection	25
3.3.4	Infrastructure misconfiguration	26
3.4	Privacy	26
3.4.1	Data minimization	27
3.5	Key operational aspects	27
3.5.1	DevOps aspects	27
3.5.2	Cloud deployments	27
3.5.3	Scalability and Performance	28
3.5.4	User experience	28
4	Software Engineering	29
4.1	Version Control	29
4.1.1	Commits	29
4.2	Testing	29
4.2.1	Unit	29
4.2.2	Integration	29
4.2.3	Database	30
4.2.4	Component	30
4.2.5	End to End	30
4.2.6	Reflection	30
4.3	Implementation Practices and Code Quality	30
4.3.1	Coding Standards and Conventions	30
4.3.2	Code Reviews	31
4.4	CI/CD	31
4.5	Change Management	31
4.6	Documentation	31
5	End Product	32

5.1	System Architecture and Infrastructure . . . . .	32
5.1.1	AWS Architecture . . . . .	32
5.1.2	Docker Architecture . . . . .	35
5.2	Database Design . . . . .	36
5.2.1	Relational Database . . . . .	36
5.2.2	NoSQL Database . . . . .	36
5.3	Features . . . . .	37
5.3.1	AI Chatbot . . . . .	38
5.3.2	Searching . . . . .	40
5.3.3	User payments . . . . .	41
5.3.4	Redis Leader Election Service . . . . .	42
5.3.5	Infrastructure shared between local and remote states . . . . .	45
6	Installation and User manual . . . . .	47
6.1	Installation . . . . .	47
6.1.1	Docker . . . . .	47
6.1.2	AWS . . . . .	48
6.2	User manual . . . . .	48
6.2.1	General Use . . . . .	48
6.2.2	Admin Use . . . . .	50
7	Reflection . . . . .	52
8	Future plans . . . . .	53
	Bibliography . . . . .	56
	Appendix . . . . .	57
A	Diary . . . . .	58
A.1	Week 1 . . . . .	58
A.2	Week 2 . . . . .	58
A.3	Week 3 . . . . .	58
A.4	Week 4 . . . . .	59
A.5	Week 5 . . . . .	59
A.6	Week 6 . . . . .	59
A.7	Week 7 . . . . .	59
A.8	Week 8 . . . . .	60
A.9	Week 9 . . . . .	60
A.10	Week 10 . . . . .	60
A.11	Week 11 . . . . .	60
A.12	Week 12 . . . . .	60

A.13 Week 13 . . . . . 61

A.14 Week 14 . . . . . 61

A.15 Week 15 . . . . . 61

A.16 Week 16 . . . . . 61

A.17 Week 17 . . . . . 61

A.18 Week 18 . . . . . 62

A.19 Week 19 . . . . . 62

A.20 Week 20 . . . . . 62

A.21 Week 21 . . . . . 62

A.22 Week 22 . . . . . 63

A.23 Week 23 . . . . . 63

A.24 Week 24 . . . . . 63

A.25 Week 25 . . . . . 63

# Abstract

Online shopping has become an integral part of everyday life, with over 50 million users in the UK this year alone. This number is projected to rise by more than 10 million in the next five years [1]. Most online stores charge transaction fees for sales made on their platforms. For example, Amazon profits from transactions between third-party sellers and users [2]. This project aims to use the demand of online shopping while also prioritising users by removing transaction fees and focusing on a subscription based approach.

This will benefit both buyers and sellers across a wide range of needs and scales. Sellers will avoid high transaction fees and have more predictable costs with the subscription model, as well as being able to scale their subscription based on their usage. Buyers will also benefit from this as items will be cheaper as no one has to pay fees on each item.

This report outlines the application's aims and goals, detailing how it addresses the current challenges of online shopping platforms. There is a focus on the technologies, frameworks, and platforms used, as well as the architectural paradigms and design patterns implemented to build a robust, scalable, and secure application. Further discussion highlights the state-of-the-art approaches employed, including considerations of security, privacy, cloud deployments, and DevOps practices. As well as the focus into what was done to create the project there are also sections on the underpinning software engineering principles that shaped the development and the issues that the project faced throughout its development.

Finally the report will contain a user manual so that anyone can pickup the project and start using it either locally for development or globally as it is intended, but also so that stakeholders can gain an understanding of the platform without having required technical knowledge.

# Chapter 1: Introduction

The introduction of this report will cover both an introduction to this report but also to the project as a whole. This report is for summing up the project but to also explain decisions behind it. Firstly you will be introduced to the problem that this project is trying to solve and then how it is going to achieve that through a series of goals and achievements. Then the report describes the technical decisions that were made and the technologies used, as well as the software engineering processes that overarched these. Finally, the report will contain a user manual and a report on the professional issues faced.

There were some constraints to the project that are discussed in the relevant sections, examples of this would be one of the architectural paradigms that has to be used and one of the platforms that must be used.

## 1.1 The problem

The problem this application is trying to solve is the that selling items online is expensive and unnecessary fees are being added so that platforms can make better margins. While these fees are lucrative for the platform, they significantly reduce profit margins for sellers, particularly small-scale vendors with already tight margins, making it potentially detrimental. Large sellers are also effected by this as they could be losing up to 5% of their profits per item from fees and this adds up once at a large scale.

There are issues with a pricing model like this as lots of people do not mind paying fees because it seems like a small cost, whereas a larger monthly fee is more off putting. One of the key conditions for commercial success is the clearness and transparency of pricing [3] and there must be added value in subscription-based online services to make consumer feel that it is worth paying for [4]. Another issue with this model is that a subscription model requires constant value delivery [5], this means that a broad range of features will be required and constantly developed.

## 1.2 Literature review

The literature referenced in this report was primarily gathered during research of the project to support key arguments. Given the practical nature of this project, which is focused more on real-world implementation than theoretical exploration, the priority was on addressing the practical challenges and requirements of the application. As such, the literature review was not a primary focus. But throughout the literature review it became clear that there are less academic sources for creating a website like this project and if there are some they focus more on investigating the technologies used or the competitors strategies from a economic and business perspective. Therefore this project was based of off lots of articles written users and creators of the technologies used and then that knowledge was adapted to create the final product.



## 1.3 Aims, objectives and user stories

This section contains the aims, objectives and user stories for this project, the user stories have been put together with the aims and objectives as they are one of the biggest influencers of them. This project is a user focused project which means that the the project development is based around how users think and what they would want. In this situation the aims are focused on the overall direction of the project whereas the objectives are a means to get to the aims.

### 1.3.1 Aims

- Create a user-friendly online shopping platform: This aim focuses on ensuring that users have an easy time navigating and using the application.
- Ensure Secure and Efficient Transactions: This focuses on the security of both the transactions but also how user payment information is handled.
- Allow for multiple payment options, configurations and services: This aim is so that all of the users wishes for payments are added.
- Optimize Application Performance for All Users: This is an important aim for the project as if the application is not performant enough users could be lost.
- Implement a searching system for items that users can use to fine tune their results with.
- Prioritize user experience (UX) and interactions: This aim focuses on how the user views the application.
- Prioritize scalability: This is imperative as when users start accessing the application it will need to be ready so that it can deal with them.
- Have a broad range of user options: This aim focuses on ensuring that both buyers and sellers have multiple features to personalize and manage their interactions, profiles, and transactions according to their needs.
- Have an adaptable codebase that can be modified easily so that new features can be added easily, new developers can be on boarded quickly so that time is not wasted.
- Create a secure and reliable platform.

### 1.3.2 Objectives

- Design an easy-to-use user interface (UI) for both buyers and sellers, ensure that the UI and UX design it simple for first-time users to understand and navigate the site.
- Integrate with a trusted payment gateway to simplify and guarantee secure payments and a wide range of features.
- Develop advanced search functionality: Create a search feature that enables users to search for specific items.
- Optimize platform speed and responsiveness: Ensure that the platform runs smoothly with minimal loading times and optimized performance, even under high user traffic.

- Enable subscription management: Allow sellers to easily select and modify their subscriptions according to their needs.
- Implement customer help, such as a page for common questions or a chatbot so that users can get help if there are any issues.
- Add monitoring and insights to the application so that issues can be found but also user trends.
- Prepare for scalability so that when deployed, spikes of users will not damage the application or affect other users.
- Allow users to manage purchase history: Ensure users can easily view their purchase history and manage order details to track their shopping activities over time.
- Implement a structured testing strategy to ensure reliability and security, as well as penetration test the application.

### 1.3.3 User stories

A user story is an informal, general explanation of a software feature written from the perspective of the end user. Its purpose is to articulate how a software feature will provide value to the customer [6]. Below are some examples of user stories that are needed for this platform.

1. (High) As a buyer I want to be able to search and find specific items so that I do not waste time looking at items I do not care about.
2. (High) As a first time user I want the application to be easy to understand so that I can quickly start using it and buying or selling items.
3. (Medium) As a seller I want to have a subscription that is as close to my needs as possible so that I am not wasting money.
4. (Low) As an administrator I want to understand the activity the application is receiving as well as compare it to the past.
5. (High) As a user I want my personal and payment information to be securely stored and transmitted, so that I can feel comfortable about giving it to the platform.
6. (High) As a user I want my transactions to be secure so that I can feel confident that I will not lose any money.
7. (Low) As a buyer I want to have the option of having multiple payment options so that I can choose the best one for me.
8. (Low) As an administrator I want to have access to what subscriptions people are getting so that I can judge what is wanted.
9. (High) As an administrator I want the application to be available to all users that want to access it.
10. (High) As an administrator I want the application to be quick for all the users so that they do not get frustrated.
11. (Medium) As a super administrator I want to be able to view other administrators so that I can check who has access to what.

12. (High) As a super administrator I want to be able to change admins permissions so that we can apply the least privilege to the admins.
13. (Medium) As an admin I want fine grained access control to features so that I can control what is available to users.
14. (Low) As a buyer I want to be able to message sellers so that I can try and get a better price for an item or ask questions about it.
15. (High )As a User I want to be able to ask for help if any issues with the application occur. So that if I have any issues I can quickly resolve them.
16. (High) As a user I want to be able to view all of the past items I bought. So that I can track my purchases.
17. (High) As a seller I want the ability to easily change or cancel my subscription based on my needs. So that I can be getting the best value for my money.
18. (Medium) As a seller I want to choose how I am paid by the platform. So that I understand how I will be paid and so I can prepare for it.
19. (High) As a developer I want to be able to easily understand the codebase so that it is easy to onboard.
20. (High) As a developer I want to have our infrastructure defined as code so that we have a record of what has been done and so we can quickly created a new environment with the same configurations.

## 1.4 Professional Issues

Professional issues arise wherever computing meets society, especially when collaborating with other professionals or when interacting with other tools and solutions. Keeping track of these issues early on is key to the trajectory of a project and trying to remedy them as soon as possible will help all parties. Professional issues have been changing as technology evolves but they also affect each project differently, and the choices made throughout the project can have consequences for not only the team working on the project but also the users. The tools and technologies mentioned earlier in this report were influenced by these decisions and later we will discuss how they were influenced.

The main issue that this project faced was to do with the disparity between large monopolies and small projects, as these large monopolies can dictate the way that a project must change. This is especially present in a Web Development project as there are stages the user faces before even interacting with a website that cannot be controlled. We will discuss the effects these monopolies have had on this project not only from the users perspective but also the developments perspective, as these issues affect the project throughout its lifecycle. Specifically, this section will examine the influence exerted by other online shopping platforms such as Amazon and Ebay, Google's search dominance and Amazon Web Services' (AWS) cloud infrastructure.. These companies create monopolies as they set standards, they have the funding to adapt to changes and they create technology that cannot be recreated.

### 1.4.1 Shopping platforms

The first monopoly that affects this project is other shopping sites such as Ebay and Amazon, these online shops are direct competition of this project and as they already have an existing

customer base and global reach. Because of this, most users will default to using these existing platforms and not come to a new one like the one made in this project, as they trust and recognise these large companies. To combat this we had to identify what we could do better such as removing platform fees and solely relying on subscription income but this will still be an ongoing issue and user acquisition will be harder due to it.

### 1.4.2 Google

Secondly Google and their search engine. Search engines have complete control over what websites show up when searched for and this directly has an impact on this project. We have had to ensure that the technologies we used would work well with search engines so that we can enhance our chances of getting users through search. This included choosing a framework with server side rendering over client side rendering as well as avoiding certain technologies known to be less search engine friendly. We have chosen Google's search engine here as it has almost 90% of the market share [7] and they have previously sued for allegedly abusing its power as the internet's main gateway to stifle competition and innovation for more than a decade [8]. One of the features that Google search has is the ability to pay for advertisements in the search results, this will move your website to the top of results and reach more users. In combination with the monopoly of existing shops this also has a negative effect on small projects such as this as there is not the funding to go up against companies worth over one billion pounds. This shows that the reliance of a single tool for search engines creates a significant professional issue that has to be considered.

### 1.4.3 Cloud Providers

This project uses AWS as a cloud provider, this was decided by looking at their pricing, services and the benefits it would bring to the project such as scalability and elasticity as discussed above. However there are numerous professional issues using a cloud provider can cause, like vendor lock in and proprietary services, data ownership, cost issues.

The most significant professional issue with cloud providers is vendor lock in and the use of proprietary services, when using services that cloud providers offer that are unique to the provider this can force a project to have to use that provider or spend time and money to migrate to a different provider. This project uses services like AWS Elastic Container Service and DynamoDB which are two services that are unique to AWS, while other cloud providers do have their own unique offering of similar services making future migration to another provider or self-hosting a complex and costly redevelopment task.

There are also issues with data ownership and residency, while all of the data is owned by the platform it is hosted on AWS' servers and underlying infrastructure, this introduces new security and compliance issues that need to be considered before thinking about deploying to the cloud. In some cases this can be a benefit as AWS may have stricter security requirements than your project so by deploying to the cloud you could increase your security.

These issues are amplified by AWS's dominant market position. The sheer breadth of services encourages deeper integration, increasing lock-in, while the complexity serves as a barrier to easy comparison or migration. But their position also allows projects to benefit from economies at scale and have lower costs overall, these costs need to be considered as using cloud providers will normally incur higher costs than self hosting and it might not be worth paying for these improved services if a project does not need it. This all needs to be thoroughly investigated and in the case of this project we decided that the benefits outweigh the issues.

### 1.4.4 Recommendations

While many projects will face issues like these, it will be hard to fix them entirely so it is recommended to understand why these issues occur and how to mitigate them. The issues can't easily be fixed as dealing with these monopolies is a greater issue than using their services, the lawsuits against Google are examples of people understanding and trying to fix these issues. Based on the experience gained from this project we recommend we recommend that projects branch out and find the best strategy for new projects, examples of this would be to compare services that cloud providers have and use a hybrid approach to benefit from all of these, or use alternative marketing like social media so a project is not relying on only search engines, and finally compare your project to your competitors and try and outperform them in cases where you can.

## Chapter 2: State of the art web development

This section contains the report describing the state-of-the-art of web development in this project. It goes over the technologies, frameworks and platforms used as well as their benefits and drawbacks. All of these were chosen for specific reasons from both a software engineering perspective and from personal consideration.

The technologies that are used refer to the programming languages, database and tools that are being used in this project. Each technology was chosen based on their features as well as a comparison to similar technologies.

The frameworks are all based of of the previous technologies and they improve the project by adding additional features capabilities. They were all chosen after reviewing competing frameworks and for their features.

### 2.1 Frontend

#### 2.1.1 Technologies

##### React

React is a JavaScript library developed by Facebook in 2013, React has a component-based architecture, it encourages the use or reusable components that can be combined together to create a user interface (UI). React components are JavaScript functions [9], this is done by using JavaScript XML (JSX), this allows developers to combine the UI and logic in one component.

Secondly, React uses a virtual document object model (virtual DOM) which means that the developers can tell React what they want the UI to look like and React will manage updating the DOM, this feature allows for faster updates and increases the performance of React applications.

Finally, React provides built-in hooks that simplify interaction with its features. These hooks help components manage state (e.g., remembering user information) [10], interact with external systems (e.g., fetching data) [10], and share data between components without passing it explicitly as props (e.g., via context) [10]. These hooks make it easier to implement advanced features as React handles much of the underlying logic.

React was chosen because of its support and performance, compared to other technologies React is likely to be the dominant framework for the foreseeable future [11] as it performed well in the benchmark section, and is also the most used and well-known framework [11]. But there is a drawback that due to the virtual DOM, React has a performance of a few milliseconds more for simple, singular DOM updates [11].

These are some of the main reasons React was chosen, but also because it is a trusted technology and is used by Facebook, Netflix and GitHub [12]. As well as being used by these companies, it is also being maintained by meta and open source contributors which means that it is still receiving fixes and updates.

## TypeScript

TypeScript is a strongly typed programming language that builds on JavaScript, giving you better tooling at any scale [13]. TypeScript is a statically typed language which means that type errors are caught at compile time and not runtime. This early detection of errors reduces the risk of bugs, making the code more reliable and easier to maintain. Overall, it provides several advantages over JavaScript [14] and helps developers write code that is easier to maintain and extend over time [14].

## 2.1.2 Frameworks

### Next.js

Next.js is a framework built on top of React, Next.js provides both client and server components, this allows you to create a UI using react and then handle logic with Next.js features. Some of these features include advanced routing and nested layouts, middleware, API routes and client and server side rendering.

As Next.js is a full stack framework you could build a full web application using only it, but you do lose benefits of other frameworks which is why this project uses a separate main backend. Having access to both client and server components means that some queries and logic can happen securely on the server while not having to make calls to the main backend.

After doing a comparison to other frameworks Next.js stands out based on it's amount of features and ability to scale and handle high workloads. As this project will be dealing with a large amounts of items being sold on it, it is imperative that they are loaded correctly. Compared to a framework like Vite which is built for speed and flexibility, Next.js excels in handling complex information which an e-commerce application has. Most other frameworks focus on either client or server side speed whereas Next.js aims to offer benefits from both. For example Astro is a framework that is server first, Astro excels in speed with static content, while Next.js offers dynamic server-side rendered pages, ensuring a fast, SEO-friendly user experience [15]. For this application Next.js is the better option as the content of this application will change a lot and the search engine optimization (SEO) will help users find the application.

Next.js is also used by large companies for their enterprise level applications, companies such as Nike, Stripe and Spotify all use Next.js [16]. This shows that it is trusted and can be used to create applications with high performance and high loads.

### Tailwind CSS

Tailwind is a utility-first CSS framework [17], Tailwind provides utility classes that can be applied directly to HTML, these classes provide a consistent development experience unlike traditional CSS where each CSS file could be formatted differently or use different naming cases. Compared to traditional CSS where the styling for elements is in different folders having the styling inline with the HTML makes it easier to understand and change. Tailwind automatically removes all unused CSS when building for production [17], this means that the build will be optimized which will increase the overall performance of the application. Tailwind integrates with both Next.js and React which means that it is easy to add to the project.

## Apollo Client

Apollo Client is a comprehensive state management library for JavaScript. It enables you to manage both local and remote data with GraphQL. Use it to fetch, cache, and modify application data, all while automatically updating your UI [18].

Apollo Client was chosen for this project because of all the features it provides, having efficient datafetching and caching is crucial to any application and having it built in using a library improves development and also performance. Development is improved as all of these features do not need to be manually created and performance is ensured as there is a large community maintaining and using this, so lots of existing errors have been fixed since being found.

Apollo Client easily interacts with React and Next.js by using React hooks, such as UseQuery and UseMutation. This integration allows for both a better development experience as well as ensuring less errors when the two frameworks interact in production.

### 2.1.3 Platforms

#### Cypress

Cypress is a testing platform that allows for frontend component and end to end testing, it is easy to add provides frontend testing, it was chosen for its range of features, industry trust and ease of addition.

## 2.2 Backend

### 2.2.1 Technologies

#### Java

Java was created in 1995 and has been in use since then, which makes it a reliable and well supported programming language. Java has a large ecosystem with lots of community support, because of this there are tools for most

Java's "write once, run anywhere" (WORA) strategy is one of its biggest benefits and it ensures that code written in Java can run on any platform with a Java Virtual Machine (JVM). This will be beneficial to this project as multiple people will need to run the application and users will be able to easily deploy and maintain web applications across several operating systems and server configuration [19].

Java's architecture encourages scalability thanks to built-in support for multi-threading and concurrency [19]. As the number of potential users for this application is unknown it is worth preparing for a both a large and unpredictable amount of users. Java's scalability is perfect for this and paired with the correct frameworks it will provide a robust backend for the project.



## 2.2.2 Frameworks

### Apollo Gateway

An Apollo Gateway sits in front of GraphQL subgraphs and allows for a federated architecture. This means that the backend services can be completely independent but under one endpoint. Currently this project has it setup but it is not being used as the backend is currently one service and not split up.

### Spring Boot

Java Spring Boot is an open-source tool that makes it easier to use Java-based frameworks to create microservices and web apps [20]. It enables this project to quickly build a Spring based application, Java Spring Boot is one specific module that is built as an extension of the Spring framework [20].

Spring Boot provides Convention over configuration [20] this means that developers only write code for the unconventional aspects of the app they're creating [20]. This means that the application will automatically run without the developer having to create configuration.

The primary benefit of using Spring Boot in this project would be not having to create configurations for Spring and for minimizing the amount of boilerplate code needed [20]. This means that development of the application can be prioritized. Secondly, Spring Boot provides a suite of development and testing tools designed to streamline the process of building and validating applications. Features like embedded http servers [20], Actuator for monitoring and management, and starter dependencies make it easier to set up, test, and deploy applications efficiently.

### Netflix DGS

The Netflix DGS (Domain Graph Service) framework simplifies the process of creating GraphQL services with Spring Boot. The framework provides an easy-to-use annotation based programming model, and all the advanced features needed to build and run GraphQL services at scale [21].

The DGS framework is actively maintained by Netflix and supported by an active community. Netflix itself relies on the DGS framework to power its GraphQL architecture, demonstrating its ability to handle high-load, critical applications. Given Netflix's scale, with 238.3 million subscribers as of 2023 [22], the framework's proven reliability in such a demanding environment makes it a compelling choice for this project.

DGS was chosen over GraphQL Java because of the reasons above but also because of its development support. DGS has more support for annotations, whereas GraphQL Java requires more boilerplate code for the same amount of functionality. This drastically improves development speed. DGS is also schema first and allows for automatic wiring of datafetching and types based on naming conventions, whereas for GraphQL Java you need to manually wire it together using RuntimeWiring.

## 2.2.3 Platforms

### OpenAI

OpenAI is an artificial intelligence(AI) company that creates AI models as well as hosts them for people to use. This project is using their hosted models as a chatbot, OpenAI provides an API interface to query the models and that is what is being used. OpenAI was chosen as they are one of the leading platforms for AI and they offer reasonable pricing, the choice to use a platform instead of self hosting a model was due to the ease of integration and training as OpenAI handles all of this for the platform.

### Apache Kafka

Apache Kafka is an open-source distributed event streaming platform [23] used for for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications [23].

Within this project it is used to handle real time streaming of events such as the creation of bids and the closing of auctions. The importance of Kafka will be discussed later but the choice of it over other message queues and streaming services like RabbitMQ and AWS Kinesis was because of its core capabilities, ecosystem and trust.

Kafka also allows for decoupling of the producers and consumers of events which means they can be independently scaled, and also created by separate teams. This enhances the project real-time data handling but also the development experience.

## 2.3 Database

### 2.3.1 Technologies

#### PostgreSQL

PostgreSQL is a powerful, open source object-relational database system that uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads [24]. The relational databases considered for this project were PostgreSQL, MySQL and SQLite.

PostgreSQL was chosen as it offers more features than MySQL. It gives you more flexibility in data types, scalability, concurrency, and data integrity [25]. As well as that PostgreSQL is always ACID compliant [25] whereas MySQL is not always ACID compliant. Secondly, PostgreSQL has more features, better performance and scalability when compared to SQLite [26]. Overall PostgreSQL was the best choice for database as it has a rich set of features and outperforms its competitors.

#### DynamoDB

DynamoDB is a serverless NoSQL database offered by AWS, DynamoDB was chosen over other NoSQL databases such as MongoDB as the project was already integrated with AWS at the time of needing a database and the serverless benefits were the deciding factor. A

NoSQL database was added to the project alongside the Relational Database as it provides more scalability, flexibility, and higher performance. It is being used in situations where the data is constantly changing and not all fields are required.

## **Redis**

Redis is an in-memory key-value database that is being used as a cache in this project. The reason Redis was added as a cache was for reducing latency on frequent database queries while also decreasing the database load, and for improving scalability by spreading the load and having both the database and the cache scale independently. Another reason it was chosen was because the cache can be ran both containerised as well as in AWS ElastiCache which means that it is ideal for both deployments on AWS or using docker and the integrations with the backend are also a benefit.

## **Liquibase**

Liquibase is a database change management solution, it adds version control, compliance and continuous integration and continuous delivery (CI/CD) to databases. Liquibase allows developers to write changelogs which define the database changes, and these changelogs can be tracked by version control and Liquibase creates a checksum for each changeset and if a mismatch is detected on deployment an error is thrown which helps prevent unwanted changes.

# **2.4 CI/CD**

## **2.4.1 Technologies**

### **GitLab CI/CD**

GitLab CI/CD is a continuous method of software development, where you continuously build, test, deploy, and monitor iterative code changes [27]. CI/CD aims to streamline and improve the software development life cycle and it can be achieved by using different tools. There are many other tools for CI/CD such as Jenkins, CircleCI and GitHub actions but Gitlab CI/CD was chosen as the project is using GitLab as a version control system and it makes development easier by using the same system.

### **GitLab-managed Terraform**

GitLab managed terraform is a service offered by GitLab that allows this project to securely store state files as well as having other features such as locking. It also allows the same states to be accessed across CI pipelines and on developers machines.

## 2.4.2 Platforms

### GitLab

GitLab is a DevSecOps platform, it is required for this project as a version control system, but this project also uses it for its pipelines which were discussed above. Having both automated pipelines and version control are crucial to this project as they aid development by preventing mistakes, accidental code loss and by making CI/CD easier to use.

## 2.5 Infrastructure

### 2.5.1 Technologies

#### Terraform

Terraform is an infrastructure as code (IaC) tool that lets you define both cloud and on-prem resources [28]. Terraform was chosen over other IaC tools such as cloudformation as it is framework agnostic so it can be adapted in the future if the infrastructure changes.

Terraform uses a declarative syntax which means that developers only need to define the desired state of their infrastructure and Terraform makes the steps to get to that state. This simplifies the development and management processes which makes it a good tool to use for this project.

Finally, Terraform uses a modular structure, which means pieces of infrastructure that will be reused lots can be defined as modules and easily reused. This will benefit the project by helping with development but also by keeping a cleaner code base.

### 2.5.2 Platforms

### 2.5.3 Amazon Web Services

Amazon Web Services (AWS) is the world's most comprehensive and broadly adopted cloud [29], AWS is the leading cloud and they have significantly more services, and more features within those services, than any other cloud provider [29]. As well as that AWS has a large community of customers and partners which means that there is lots of support for developers using it.

While cloud computing has seen significant growth, AWS faces strong competition, particularly from Microsoft Azure and Google Cloud Platform (GCP). AWS holds a 31% market share, compared to Azure's 20% [30]. The choice of a cloud provider often depends on existing infrastructure and specific service needs. However, as this project is being developed from scratch, those considerations are less critical. AWS also provides a generous free tier that means that you get access to some services for a limited time for free, this influenced the choice as it means there will be little to no development cost.

AWS was selected for this project due to its extensive range of services, its ability to scale applications effectively, and its proven reliability in production environments. By leveraging AWS, this project benefits from a robust, scalable, and globally available deployment

environment.

## 2.6 Financial

### 2.6.1 Platforms

#### Stripe

Stripe is a fully integrated suite of financial and payment products [31]. This platform is essential for the application as it handles user subscriptions and user-to-user payments. By using Stripe the payment systems do not need to be manually set up, which saves in development time but also ensures that the systems used are globally compliant.

Stripe handles secure payment processes, such as encryption, tokenization, and fraud prevention, as they are built into Stripe's infrastructure, all of this can be accessed by using the Stripe API's, libraries and SDK's. Stripe has always been focused on developers since it was created which means that the development experience and support is very good which is another reason it was added to the project.

## 2.7 Global

### 2.7.1 Technologies

#### GraphQL

GraphQL is a query language for APIs and a runtime for fulfilling those queries with your existing data [32]. GraphQL is an alternative to REST, REST is the default for application program interfaces (API) but it has some downsides. Some of these downsides include over-fetching and each client need to know the location of each service [33]. GraphQL solves these issues by letting the client choose which data it requests and all the data is behind one endpoint.

In this project GraphQL has been chosen to be the primary API but there will also be some RESTful API's for simpler tasks such as health checks.

#### Docker

Docker is a containerization tool, a Docker container allows a user to run code no matter what they are running on, a container is based on an image which is a script that installs all of the required dependencies and sets up the application.

Containers make it easy to share, build and deploy applications anywhere. As the container only has the required dependencies it means that the final product is lightweight which makes them more efficient when running. Finally, by using Docker containers the application is more scalable if built correctly, as the amount of containers can easily be scaled up.

Overall, Docker was added to this project to allow for easy sharing of the application when

submitting code but also as a way to potentially deploy the application.

## Chapter 3: Architectures, Design Patterns and other aspects

### 3.1 Architectural paradigms

#### 3.1.1 N-tier architecture

N-tier architecture divides an application into logical layers and physical tiers [34]. Layers are a way to separate responsibilities and manage dependencies. Each layer has a specific responsibility [34]. The reason for choosing this pattern is because it is required by the project requirements. This project has a web tier (Next.js application), application layers (Spring Boot application and Cognito) and the database layer. These layers all interact with each other, for example the web layer will request data from the application layers which then fetches the data from the database layer.

There are many benefits and concerns to using N-tier architecture, firstly each layer can be scaled independently, this is extremely useful when certain layers are experiencing more stress than others as that layer can be scaled up without needing to scale up the whole application. As well as that there is also an inherent security benefit, this is because there is the option to add security checks in the gaps between the layers for example, it is easy to apply network security group rules and route tables to individual tiers [34]. This can be easily added by restricting the origins of requests that are allowed. Although there are many benefits there are issues like the fact that when requests start passing through many layers it decreases observability and makes it harder to test or change these layers.

#### 3.1.2 Monolithic architecture

Monolithic architecture is a architectural paradigm where all of the application is packaged and deployed all together, and monolithic applications are usually deployed in virtual machines and if it needs to scale then more virtual machines are added with only the application running in them. There are many other issues with this architecture apart from its scalability, such as how tightly coupled the application is together, this is bad because it means that one change, such as updating a database schema, would mean also updating other systems that use the database. Overtime monolithic codebases can get hard to understand as so much functionality is contained in the codebase. While monolithic architecture is quick to start developing with, it causes many issues which is why it was not picked for this project.

#### 3.1.3 Microservice architecture

Microservice architecture is the opposite of monolithic architecture, they are an architectural and organizational approach to software development where software is composed of small independent services that communicate over well-defined API [35]. Each service in a microservice architecture is designed to be deployed and run independently of the other services while also being designed for a set of capabilities and focuses on solving a specific problem [35]. This drastically improves scalability and fault tolerance because each service can be individually scaled and if one goes down it work affect the others. While the services are easy

to update and change, the interactions and API's need to be carefully updated as a loss of communication between services is detrimental.

### 3.1.4 Serverless architecture

Serverless computing is an application development and execution model that enables developers to build and run application code without provisioning or managing servers or back-end infrastructure [36]. Serverless architecture heavily depends on what services you have access to which is why it is not always the best choice for an application. Although with serverless you do not have to worry about provisioning servers or paying for downtime which saves on development time and money.

### 3.1.5 Event driven architecture

Event-Driven Architecture (EDA) is a software architectural paradigm that centres around the generation, transmission, processing, and storage of events [37]. In EDA, producers publish events (such as a user signing up or completing a checkout), which are then consumed by event consumers. These consumers trigger specific processing routines or business logic in response to the event [37].

This project implements an event driven approach in some sections, such as the bidding service. Apache Kafka is used to stream the events and for example when an auction finishes the auction or event is sent for processing or when a bid is placed it is streamed and notifies the interested parties. Using event driven architecture here means that the producing and consuming of bids can be done separately and scaled independently which improves the tolerance of the system.

## 3.2 Design patterns

### 3.2.1 Model view controller (MVC)

The MVC design pattern breaks an application into three parts: the Model (which handles data), the View (which is what users see), and the Controller (which connects the two) [38]. The flow of MVC is that the user interacts with the view (frontend), then the controller (back-end) requests or updates the model (database) based on the request from the view, the view sends the controller the response and the controller updates the view. MVC helps organize code and manage complexity [38] in complex applications, MVC makes applications easier to test as you can test each component individually. However, it can also add complexity if it is not required, particularly in small-scale applications, or if it is not implemented properly, so it is worth checking if it is needed in a project.

### 3.2.2 Client-Server Model

The Client-Server splits systems into two components, the client and the server. The clients request information from a server, for example the frontend requesting data from the backend. The server will receive these requests from clients and it will fetch the data requested and return it. There are concerns with this model as if the server fails or gets overwhelmed by



the clients sending it requests, it can shutdown.

This is being used inside the frontend of the application, specifically it is being used for controlling authentication and financial transactions. It is being used there as it is secure from users and bad actors and none of the information in these flows are exposed to the browser.

### 3.2.3 Component-based architecture

Component-based architecture is a framework for building software based on reusable parts [39], these components can be reused throughout an application. An example of this is React components which were discussed above, components are encouraged in React as lots of UI is reused, such as buttons or input fields. The main benefit of this architecture is the reusability of these components and because it adds consistency to the UI.

### 3.2.4 Singleton Method Design Pattern

The Singleton Method Design Pattern ensures a class has only one instance and provides a global access point to it. It's ideal for scenarios requiring centralized control, like managing database connections or configuration settings [40]. An example of a singleton in this project is the singleton instance of Stripe used in the frontend. This benefits the project as there is only one instance interacting with Stripe so it is less likely for duplicate actions to occur when interacting with Stripe.

## 3.3 Security

This section contains some of the security features that were considered for this project. During the project the OWASP top 10 Web Application Security Risks [41] were considered as they are all likely risks that could be exploited.

### 3.3.1 Authorization and Authentication

Authorisation is the process of granting or denying access to resources based on a user's identity and privileges. Authentication is the process of confirming the user is who they say they are. They can be applied in many ways and below are three of the main ways they were implemented in this project.

#### API key

API key authorization is the process of using a secret key that a client provides when making an API call. This has been implemented into the backend so that only authorized users can make API calls. The way this works is that the security config checks the X-API-KEY header in the request and compares it to a saved key, this is explained more in the features section below.

## **JSON web token (JWT)**

A JSON web token(JWT) is JSON Object which is used to securely transfer information over the web(between two parties). It can be used for an authentication system and can also be used for information exchange. The token is mainly composed of header, payload, signature [42]. They are being used in this project for the backend to verify if a client is allowed make a request to the backend. It does this by checking the claims of a client and confirming them with the JWT issuer. JWT's are only available for a limited time before they need to be refreshed which helps increase security as it means old tokens cannot be reused.

### **3.3.2 Role Based Access Control**

Role-based access control (RBAC) is a model for authorizing end-user access to systems, applications and data based on a user's predefined role [43]. It is used in this project primarily for managing site admins, who have more access than users, as well as controlling the fine grained access an individual admin will have, for example some admins can alter permissions but others cannot. This provides the platform with better security as each user only has access to what they need and not anything else.

### **3.3.3 SQL Injection**

SQL injection was the top of the OWASP top 10 for years and has recently been to third [41], these attacks can lead to large data breaches and they are easy to do. SQL injection is a type of an injection attack that makes it possible to execute malicious SQL statements against a database, it usually occurs when there is no input sanitation or when a web application directly interacts with a database.

```
1  RETURN QUERY EXECUTE format(  
2      'SELECT  
3          i.item_id,  
4          i.name,  
5          i.description,  
6          i.is_active,  
7          i.ending_time,  
8          i.price,  
9          i.stock,  
10         i.category,  
11         i.images,  
12         i.seller_id,  
13         i.final_price  
14     FROM items i  
15     WHERE similarity(i.name, $1) > 0.3  
16     AND is_active = true  
17     ORDER BY %I %s  
18     LIMIT $2 OFFSET $3',  
19     order_by,  
20     order_direction  
21 )  
22 USING search_text, page_size, page * page_size;
```

CodeBlock 3.1: SQL for searching for items.

There are protections against this inside this project, the primary protection is the library that is being used to interact with the database has built in protection against this, as well as that in the cases where user input is directly affecting queries such as in the search queries the data is being formatted so that PostgreSQL ensures that the inputs are treated as data not as commands. Above is an example of this happening inside the application.

### 3.3.4 Infrastructure misconfiguration

Infrastructure misconfiguration can cause serious vulnerabilities and insecure design and security misconfiguration are on the OWASP 10 [41]. To try and mitigate these issues, security scanning on the IaC was added using Trivy.

## 3.4 Privacy

Privacy is a key feature for most users as they want to know how their information is being handled and protected, most of the features to protect user privacy come under the security aspects but these sections here will help minimize the impact on a user if their information was leaked. A user will also have access to delete all of their personal information that the application has collected. There is some data that cannot be deleted such as transactions with others so platform data will remain but personal information once.

### 3.4.1 Data minimization

Data minimisation means collecting the minimum amount of personal data that you need to deliver an individual element of your service [44]. The way this is being implemented in this application is that every piece of user data is only saved if necessary and if it is then it is saved in the correct place, for example payment details never leave Stripe.

## 3.5 Key operational aspects

### 3.5.1 DevOps aspects

The DevOps aspects that have been introduced to this project so far, have been CI/CD with steps for testing, building and deploying. DevOps aspects have been heavily considered for this application as each part of it has tests, will need to be deployed and once deployed they will need to be monitored. It is not a requirement for any of this but it is a good challenge and every production web development project has needed to be deployed, so it is also a realistic and helpful challenge.

The DevOps aspects that have been implemented include: running tests, linting code, building infrastructure and services and finally deploying code, this will be fully explained in the End Product chapter and the Software Engineering chapter.

#### Infrastructure as Code (IaC)

IaC was used in this project as it means that all of the infrastructure that will be created has been tracked with version control, it can easily be replicated using Terraform. As well as that manual infrastructure management is time-consuming and prone to error and IaC can be used to control costs, reduce risks, and respond with speed to new business opportunities [45]. Whereas some of the benefits are that you can easily duplicate an environment, reduce configuration errors and scalability. More of the information about IaC in this project can be found in the terraform section above or the IaC feature section below.

### 3.5.2 Cloud deployments

As discussed above cloud deployments are apart of this project, this was done using AWS. The reason cloud deployments were considered over on-premises deployments is because they are infinitely scaleable, servers do not need to be managed, expensive capital expenditure costs can be avoided and more. As well as the benefits this project would get from being deployed. There is also the benefit of having multiple deployment options and the ability to swap if there is a surge of users. Cloud deployments also offer the ability to go global in minutes [46] and disaster recovery, trying to implement these would take a lot longer and it would be more expensive as we would not benefit from economies at scale. Below in the End Product chapter there are more details about how the deployments were implemented.

### 3.5.3 Scalability and Performance

The project is being built with scalability and performance as one of its main priorities, this all started with the technology and framework choices and so that there are no bottlenecks later in the project that need to be fixed. This was also considered when choosing cloud deployments as the benefits from the cloud can be combined with the applications features to reach a better potential. Another way to ensure these is testing, testing will be able to help determine if the application can scale and performs well. Both scalability and performance are important for when users start using the application or if there is an unexpected increase of users, these situations are both beneficial to the application so it is worth being prepared for them.

### 3.5.4 User experience

User Experience is at the core of this project, as this is a project designed to be used the experience of those users needs to be prioritised. To do this UI libraries have been researched and used, as they have professionals working and testing these components as well as having a large community backing them and giving advice, so it benefits the project by using their knowledge to help us. Secondly, over the next couple of months we are planning to get a small group of potential users to have a look at the application and give feedback. This will help find errors that have not been found as well as advice on what we could be improved.

## Chapter 4: Software Engineering

This chapter contains the information about some of the software engineering principles that were applied throughout the development process. It highlights the benefits but also some of the drawbacks of what was done.

### 4.1 Version Control

Version control, specifically Git, was used to manage the project's codebase hosted on GitLab. It allowed for tracking changes, managing different lines of development via branching. The branch strategy that was used would involve having three main branches main, development and test, these branches would track the code changes for each environment and then if a developer wanted to add a feature they would create a branch off of development named feature, create what they wanted to, create a pull request back into development, then it could go to test and finally main. This strategy ensures that the code is reviewed and tested at many stages of the development. As well as the development benefits it also allows for viewing of the history and rollback capabilities in case there are any issues.

#### 4.1.1 Commits

Commit structure for this project followed the conventional commits guide, which means that all commits are standardised and can be understood quickly, for example if a developer was looking for a specific bug fix they only have to search the commits with messages starting with fix:.

### 4.2 Testing

This section contains the information about the testing strategy that was implemented and the testing suite that was created.

#### 4.2.1 Unit

The majority of tests that were created were Unit tests, they serve the purpose of verifying that code blocks would work correctly, it tests all of the functions in the backend but when they are isolated, they mock interactions with other services.

#### 4.2.2 Integration

Integration tests cover test the interactions between different services, they are one level higher than Unit tests, these were added to test API's as a whole and how they would interact with the database, cache or external services.

### 4.2.3 Database

There was a suite of database tests which tested the SQL functions, constraints and procedures. These are essential to confirming that the implementation of the data model that was created was correct. These tests run on a separate database schema but are meant to be ran on an empty database as they create and destroy data in the tables and it should not be ran against production data.

### 4.2.4 Component

Component tests were added to test the user interface and loading speeds of the individual frontend components, they can help identify bottlenecks, UI degradation and more.

### 4.2.5 End to End

Finally, End to End tests were added to the frontend, these tests test the user flow and interactions with the project, they can also be used as smoke tests to check that all functionality is working.

### 4.2.6 Reflection

Overall the test suite did slow development slightly as there were a lot of levels of tests to create but it does mean that the end product is more reliable as it has been tested. As well as that there could have been improvements to the frontend testing as there were not as many tests made compared to the backend.

## 4.3 Implementation Practices and Code Quality

### 4.3.1 Coding Standards and Conventions

Established coding standards and conventions were enforced across the codebase to ensure consistency, readability, and maintainability. This was primarily achieved through automated tooling:

- **Linting:** ESLint and others linting tools depending on the language stack were configured with predefined rule sets to automatically detect and often fix stylistic issues, potential bugs, and deviations from agreed-upon conventions during development.
- **Checkstyle:** For Java components, Checkstyle was used to verify adherence to specific coding standards regarding formatting, naming conventions, and code structure.
- **Prettier:** An opinionated code formatter like Prettier was likely used to automatically enforce consistent code formatting, minimizing debates about style during code reviews.

Adherence to these standards makes the code easier for all developers to understand and helps prevent common errors.

### 4.3.2 Code Reviews

As discussed above, code reviews would occur at every stage of the codes lifecycle, a pull request would be created for any new code entering the codebase and as it makes its way through the environments it gets reviewed again.

## 4.4 CI/CD

The CI/CD pipelines will be discussed throughout this project but as a high level overview they were used to automate testing, building and deployment of code. The benefits of this were that code would automatically be verified to be working, to some degree, but also if broken code was pushed it would be caught before deploying, and it would save developer time as they did not have to manually deploy each stage of code.

## 4.5 Change Management

The database uses a change management tool called liquibase which was discussed above. The benefits of this are that the changes to the database are tracked and monitored, which means that there is a changelog to the database and any hidden or unwanted changes can be caught by the comparison of what has already been applied.

## 4.6 Documentation

One of the benefits of using GraphQL is that its schema works as API documentation as well, which meant the it was easy to keep this updated. For code documentation, two tools were used, on the backend Javadoc was added and the frontend used TypeDoc.



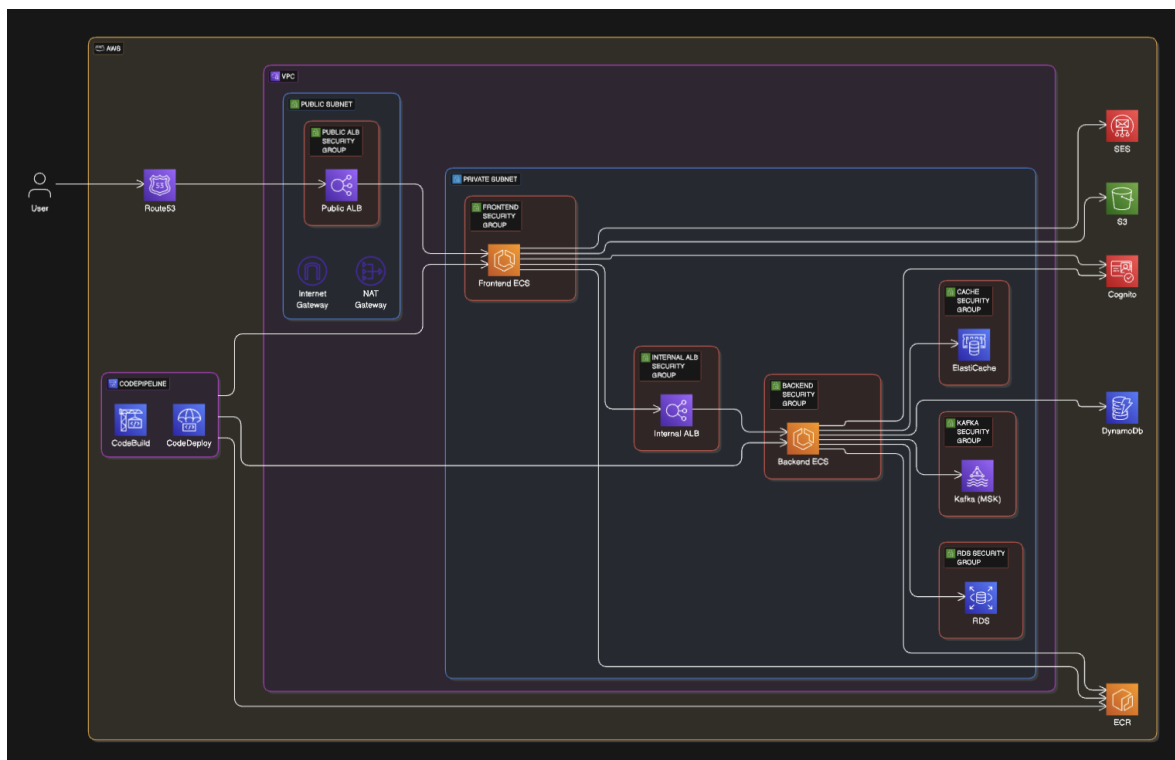
## Chapter 5: End Product

This chapter contains all the information about what has been completed, it has been split up into the architecture and infrastructure, the database design and the key features. Each of those sections will go into depth about what is being used and why.

### 5.1 System Architecture and Infrastructure

There are two main ways to deploy this project and later in the Installation and User manual chapter, you will be shown how to deploy to both of them, but this section goes into depth about how it is laid out and what services are being used.

#### 5.1.1 AWS Architecture



The diagram above shows the AWS architecture, it includes three main sections, the services inside the Virtual Private Cloud (VPC), the Global AWS Services and the services in Code-pipeline. While all of these services interact, there separation is important, and it is based on how AWS setup their services. The services that were used were primarily chosen based on the initial technology stack but they also have their own benefits which will be discussed.

#### VPC Services

The services inside the VPC are deployed across multiple availabilty zones and if needed they can be deployed across different AWS regions for enhanced global coverage. The diagram shows the interactions between each services.

**Subnets** Subnets are range of IP addresses, the difference between the public and private subnets is that the public ones have direct access to an internet gateway and public traffic can be made accessible to them, whereas the private subnets do not allow any public ingress and they can egress using the NAT Gateway.

**Security Groups** A security group controls the traffic that is allowed to reach and leave the resources that it is associated with [47]. They surround all of the services and only allow the correct traffic, for example the backend ALB will only accept traffic from the frontend security group. This adds an extra layer of security to the deployment.

**Load Balancers** Application Load Balancers (ALB's) are used to automatically distribute incoming application traffic across multiple targets, such as the ECS services above. This project utilizes two different Load balancers, there is a public ALB which is accessible from the internet and it routes user traffic to the frontend containers. Then there is an internal ALB which routes the frontend's requests to the backend containers, this architecture was created so that the two services can scale independently. The ALB's have health checks for their downstream services so they will only send traffic to healthy tasks and will force redeployment of unhealthy tasks.

**Elastic Container Service (ECS)** The ECS services contain docker containers for a specific task, such as the frontend and backend of this project. They are serverless as they are being run with AWS Fargate, which means that the underlying servers are managed by AWS.

**ElastiCache** Amazon ElastiCache is a serverless, fully managed cache delivering real-time, cost-optimized performance for data-driven applications [48], it is being used to host a cluster of Redis caches for the backend. ElastiCache Serverless means zero infrastructure management, zero downtime maintenance, and instant scaling to match any application demand [48].

**Managed Streaming for Apache Kafka (MSK)** Amazon MSK is a streaming data service that manages Apache Kafka infrastructure and operations, making it easier for developers and DevOps managers to run Apache Kafka applications [49]. Amazon MSK operates, maintains, and scales Apache Kafka clusters, provides enterprise-grade security features out of the box, and has built-in AWS integrations [49]. This service allows for easy and secure Kafka clusters to be used with the backends service as discussed above which is why it was chosen over other solutions.

**Relational Database Service (RDS)** Amazon Relational Database Service (Amazon RDS) is an easy-to-manage relational database service [50]. Amazon RDS automates undifferentiated database management tasks, such as provisioning, configuring, backing up, and patching [50]. This was decided as the best way to host the Postgres Database on AWS.

## Global Services

**Elastic Container Registry (ECR)** ECR is a fully managed container registry which is storing the frontend and backend docker images after they have been built in CodeBuild.

**Simple Email Service (SES)** SES is an AWS service that can be used for sending emails to clients, and internally, this project uses it to send the get in contact emails from users, it has a Javascript integration so it was easy to add to the frontend.

**S3** Amazon Simple Storage Service (Amazon S3) is an object storage service offering industry-leading scalability, data availability, security, and performance [51]. It is being used in this project to store and host users item's images so that they can be used within the application. You can store virtually any amount of data with S3 all the way to exabytes with unmatched performance [51] which means that the developers do not have to worry about the amount of images the users are uploading.

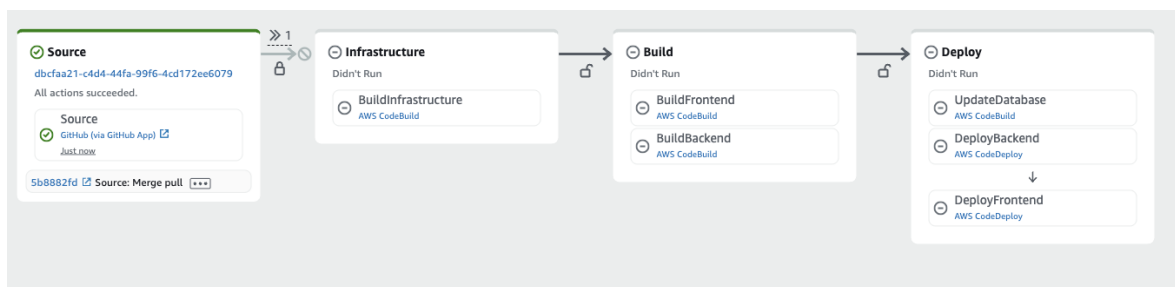
**Route53** Route53 is AWS's Domain Name System (DNS), while any DNS could have been used as they are all very similar, Route53 was chosen due to its integration with other AWS services and how easy it was to point DNS records at Load balancers inside AWS which made Route53 ideal for this project.

**DynamoDB** Above in technologies section under DynamoDB, the reasons why DynamoDb was chosen were outlined and some more of the benefits are that DynamoDB scales to zero, has no cold starts, no version upgrades, no maintenance windows, no patching, and no downtime maintenance. DynamoDB offers a broad set of security controls and compliance standards [52].

**Cognito** Cognito is a developer-centric, cost-effective service that provides secure, tenant-based identity stores and federation options that can scale to millions of users [53], in this project it is used for authentication, to control user accounts as well as JWT authorization and more.

## CodePipeline

AWS CodePipeline is a continuous integration and continuous delivery (CI/CD) service for fast and reliable application and infrastructure updates [54]. It has multiple integrations with source code providers which means that the pipelines can be ran on code updates. This project uses it to control the majority of the CI/CD and it has been configured to use Codebuild and CodeDeploy to manage building new releases and deploy them, below is the pipeline.

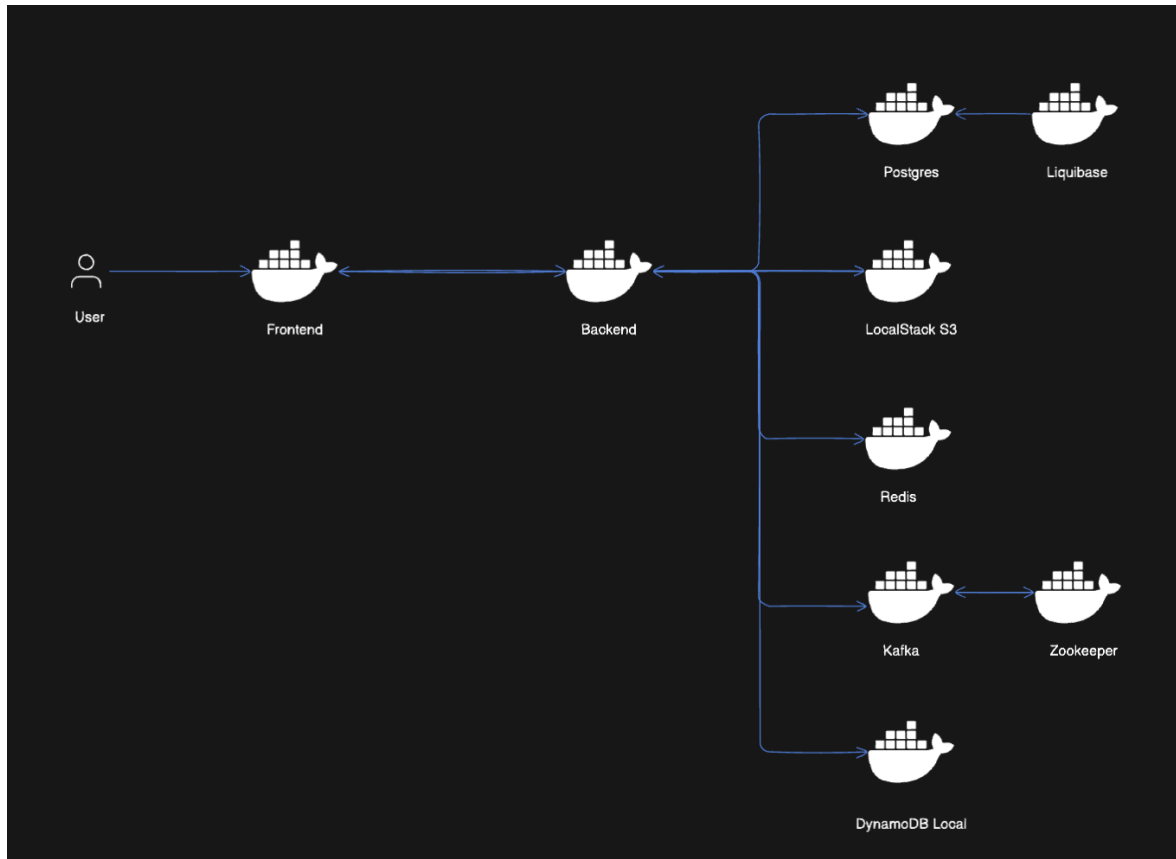


This image shows the steps in the pipeline and how it is configured, first the source code is retrieved, then in the Infrastructure step a Codebuild project is ran which updates the deployed infrastructure. Then the frontend and backend are built and pushed to ECR, and finally the database is updated and the services are updated using a blue green canary

deployment. If any of these stages fail the next steps will not run which means prevents faulty updates being deployed.

## 5.1.2 Docker Architecture

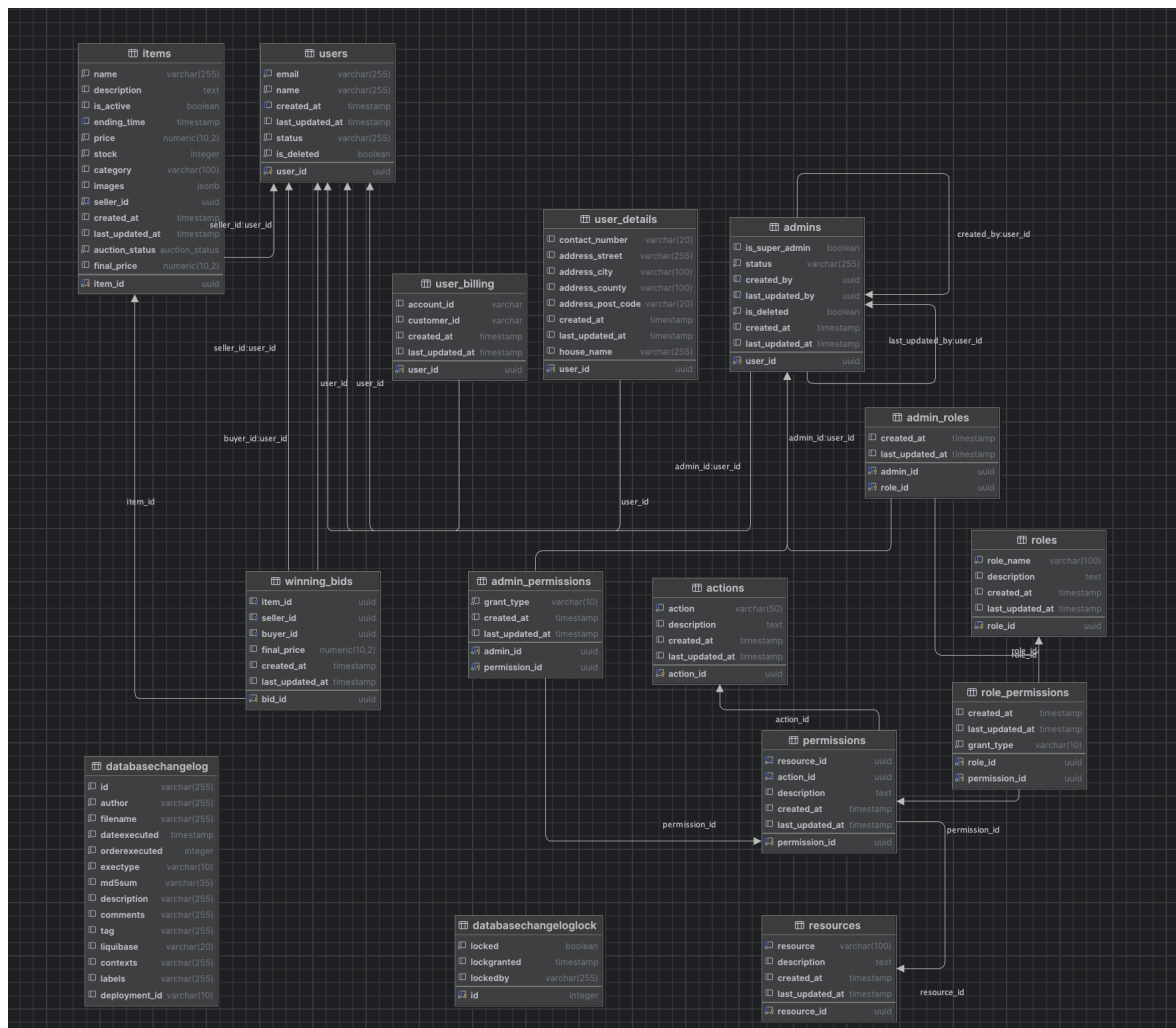
When deploying this using the Docker stack on a server there are differences to how it is deployed on AWS, below is a diagram on how the containers are ran.



The main difference is the simplicity of the architecture and how there is less restrictions around how the services communicate with each other, there is less architectural security like there is with AWS security groups. Secondly, it is missing these: load balancing, CI/CD, Cognito and SES, while it is possible to add these to the project later on there is no local version or alternative of Cognito and it is essential to this application so this must be hosted on AWS no matter how the application is deployed.

## 5.2 Database Design

### 5.2.1 Relational Database



The initial database model was designed based on the required information about users, and on research of online shops and what information they show, then as the application progressed and evolved so did the data model. Above is the Unified Modelling Language (UML) diagram of the final model and it shows how the tables are linked together. Using UML throughout development helped identify potential design issues early in on, such as unnecessary dependencies or missing attributes.

Every table contains a last updated at and created at value which are used for auditing purposes, these values update based on triggers so that they are always updated. Finally to confirm the database designs and query functionality there is a separate test schema that contains procedures for testing all of the functions.

### 5.2.2 NoSQL Database

The DynamoDB NoSQL database has a minimal schema that is flexible, it includes table keys and indexes. For example below is the schema of the development tables, in the dev chats table there are the hash and range keys which are the conversationId and the createdAt

and then there are two global secondary indexes (GSI's) which are the chatId and the userId. The keys allow the application to retrieve all chats where the conversationId is know and or all chats in a specific time period and the indexes allow the application to query the data on these different columns (This requires extra read and write capacity).



dev-chats	dev-bids	dev-admin-access-logs
chatId S	itemId S	adminId S
userId S	bidId S	timestamp S
conversationId S	createdAt S	
createdAt S		

## 5.3 Features

This section contains some of the most important features that were completed.

### 5.3.1 AI Chatbot

```

1      const optimisticChat: Chat = {
2        __typename: "Chat",
3        chatId: optimisticChatId,
4        conversationId: currentConversationId,
5        message: newMessage,
6        sender: "User",
7        createdAt: new Date().toISOString(),
8      };
9
10     // Optimistically update the conversation state
11     setConversation((prev) => [...prev, optimisticChat]);
12
13     createChatMutation({
14       variables: {
15         conversationId: currentConversationId,
16         message: newMessage,
17       },
18       onCompleted: (data) => {
19         if (data?.createChat) {
20           const { response } = data.createChat;
21
22           // Remove optimistic chat and replace with actual chat and response
23           setConversation((prev) => [
24             ...prev,
25             ...[response].filter(
26               (msg): msg is Chat => msg !== null && msg !== undefined,
27             ),
28           ]);
29         }
30       },
31       onError: (error) => {
32         console.error("Error sending message:", error);
33         // Remove optimistic chat on error
34         setConversation((prev) =>
35           prev.filter((msg) => msg.chatId !== optimisticChatId),
36         );
37       },
38     });

```

CodeBlock 5.1: JavaScript code for optimistic UI update in chat.

The AI Chat bot allows users to ask for help and the response is generated by a large language model (LLM), this is important as it allows the users get the help they need while also allowing the team behind the project to not be swamped by responding to these messages.

The chatbot works using GraphQL queries and mutations but there are also some interesting parts of code that enhance the user experience. Firstly, there is optimistic updating of the chats, this means that the second a user has sent a message, their message is updated on the UI even before the mutation has been fired. The code for this is above 5.1:

```

1      OpenAIClient client = OpenAIHttpClient.builder()
2          .apiKey(openAiApiKey)
3          .project(openAiProject)
4          .organization(openAiOrganization)
5          .build();
6
7      String prompt =
8          "You are a friendly, helpful assistant for an "
9              + "online shopping platform called SubShop. "
10             + "Your name is SmartShop Assistant. "
11             + "Respond in a warm, conversational tone while being concise and
12             ↪ accurate. "
13             + "Address the user directly and personalize your responses. "
14             + "If you cannot answer based on your knowledge, politely "
15             + "explain you're unable to help "
16             + "and suggest contacting customer service. "
17             + "End your response with a follow-up question or "
18             + "offer of additional help when appropriate. "
19             + "User's Question: "
20             + message;
21
22      ResponseCreateParams params =
23          ↪ ResponseCreateParams.builder().input(prompt).model(ChatModel.GPT_40_MINI).build();

```

CodeBlock 5.2: Java code for creating chats with OpenAi.

Secondly, there is the way that the chat is created by OpenAI and the prompt that OpenAI receives, the code is above 5.2. This code shows how interactions with third party SDK's but also how the LLM is prepared to answer the questions it receives. There is a better solution to this problem that would involve hosting an independent LLM that uses Retrieval-Augmented Generation (RAG) to get application specific information so that the answers created are much better.



### 5.3.2 Searching

```

1  IF NOT EXISTS (
2      SELECT 1
3      FROM pg_attribute
4      WHERE attrelid = 'items'::regclass
5            AND attname = order_by
6            AND attnum > 0
7            AND NOT attisdropped
8  ) THEN
9      RAISE EXCEPTION 'Invalid column name for ordering: %', order_by;
10 END IF;
11
12 RETURN QUERY EXECUTE format(
13     'SELECT
14         i.item_id,
15         i.name,
16         i.description,
17         i.is_active,
18         i.ending_time,
19         i.price,
20         i.stock,
21         i.category,
22         i.images,
23         i.seller_id,
24         i.final_price
25     FROM items i
26     WHERE similarity(i.name, $1) > 0.3
27     AND is_active = true
28     ORDER BY %I %s
29     LIMIT $2 OFFSET $3',
30     order_by,
31     order_direction
32 )
33 USING search_text, page_size, page * page_size;
34 END;
35 $$ LANGUAGE plpgsql;

```

CodeBlock 5.3: SQL for searching for items.

This part of the code shows the database searching for items method, it uses the pg\_trgm extension to get a similarity score of how close the search text is to the names of the items in the database. Currently the similarity score is set to be over 0.3 to try and match lots of items and the items are ordered on how closely matched they are which means that users will get the best results first. The inner function returns a query so that the correct table data can be returned into the returns table query. A plpgsql function is being used so that a more of the logic can be added into the schema rather than duplicating logic in the layers above.

As mentioned earlier in the SQL Injection sub section it was discussed why this function is good from a security standpoint but there are also risks. The order by column name is as input from the client, while there is a check for verifying it is a valid table there are still security issues with this.

The benefits aside from searching, the order by feature and pagination are important user features as they can help a user interact with the searched data as there will be less information on the UI and they can order it whatever way they want. Secondly the pagination helps the server load as not all of the information is returned every query.

### 5.3.3 User payments

User to User payments was enabled thanks to Stripe Connect, but the implementation of it is an essential part of this project so it should be highlighted. First, the payment flow must be decided, for security reasons the payment information should all be handled by Stripe and when a user places a bid, the payment needs to be authorized upfront to guarantee the availability of funds and allow the user to resolve any potential card issues (like 3D Secure authentication) immediately. This is achieved using a Stripe SetupIntent, the SetupIntent is created on the Frontends server using the code below 5.4:

```

1  const setupIntent = await stripe.setupIntents.create({
2    customer: customerId,
3    payment_method_types: ["card"],
4    usage: "on_session",
5    metadata: {
6      item_id: itemId,
7      bid_amount_gbp: (amount / 100).toFixed(2),
8      reason: `Bid authorization for item ${itemId}`,
9    },
10   description: `Authorize card for bid on item ${itemId}`,
11 });
12
13 return NextResponse.json({
14   success: true,
15   clientSecret: setupIntent.client_secret,
16 });

```

CodeBlock 5.4: Frontend API for setup intents.

Once the SetupIntent is created the client secret is returned so that it can be linked in the Stripe elements for confirmation of the SetupIntent and it is confirmed by using the code below 5.3.3. If successful, Stripe securely saves the payment method details associated with the user's Stripe Customer ID and links it to the SetupIntent. It is linked to the created Setup Intent by the Stripe elements. The SetupIntent is returned so that the paymentMethodId

```

1  const paymentMethodId = setupIntent.payment_method;

```

can be sent to the backend for finalisation.

```

1  const { error, setupIntent } = await stripe.confirmSetup({
2    elements,
3    confirmParams: {
4      return_url: `${window.location.origin}/shop/item/${itemId}`,
5    },
6    redirect: "if_required",
7  });

```

CodeBlock 5.5: Frontend confirmation of setup intents.

Now that the payment has been authorized, once the auction has completed the winning payment needs to be confirmed and processed. This process is being completed in the backends closed auction consumer, it reads the closed auctions, completes the payments and updates the database. It gets both the buyer and sellers stripe information from the database and then uses the PaymentMethodId that was generated in the previous step . The transfer information is set so that stripe confirms the payment without immediate approval as the user is off session and has already confirmed it. Below is the code described

```

1  PaymentIntentCreateParams.Builder paramsBuilder = PaymentIntentCreateParams.builder()
2    .setAmount(amountInPence)
3    .setCurrency("GBP")
4    .setCustomer(customerId)
5    .setPaymentMethod(finalBid.getPaymentMethod())
6    .setConfirm(true)
7    .setOffSession(true)
8    .setCaptureMethod(PaymentIntentCreateParams.CaptureMethod.AUTOMATIC)
9    .setTransferData(
10      PaymentIntentCreateParams.TransferData.builder()
11        .setDestination(sellerId)
12        .build()
13    )
14    .putMetadata("item_id", item.getId().toString())
15    .putMetadata("bidder_user_id", finalBid.getUserId().toString())
16    .putMetadata("seller_user_id", item.getSeller().toString());
17
18  PaymentIntent.create(paramsBuilder.build());

```

CodeBlock 5.6: Backend creation of payment intent.

### 5.3.4 Redis Leader Election Service

This code implements a leader election mechanism using Redis as a distributed lock provider. The goal is to ensure that among potentially multiple instances of the application running concurrently, only one instance holds the "leadership" status at any given time. The leader instance is responsible for getting items from completed auctions and publishing them to a Kafka topic.

Firstly each instance checks if they are the leader and if they are then they will try and extend their leadership. Otherwise they will try and become the leader and if there is an existing leader they wait until the scheduled heartbeat, which is a function that runs every

30 seconds, and then try again. If there are any errors during the leader election the instance checks if they are the leader and if they are they will remove themselves as leader so that the other instances can take over while the error is being investigated. The code for this is below

.

```

1  private final AtomicBoolean isLeader = new AtomicBoolean(false);
2
3  @Scheduled(fixedRate = LEADER_EXPIRY_SECONDS * 1000 / 3)
4  public void leadershipHeartbeat() {
5      attemptToAcquireOrRefreshLeadership();
6  }
7
8  private void attemptToAcquireOrRefreshLeadership() {
9      try (Jedis jedis = jedisPool.getResource()) {
10         LocalDateTime now = LocalDateTime.now(ZoneOffset.UTC);
11         String timestamp = now.format(FORMATTER);
12         String hostInfo = getHostInfo();
13
14         AppLogger.info("Attempting to acquire leadership for host: " + hostInfo);
15         if (isLeader()) {
16             // Already the leader, try to extend the lease
17             String currentLeader = jedis.get(LEADER_KEY);
18             if (instanceId.equals(currentLeader)) {
19                 // Still the leader, reset expiry
20                 jedis.expire(LEADER_KEY, LEADER_EXPIRY_SECONDS);
21                 AppLogger.info("Instance {} leadership renewed at {}", instanceId, timestamp);
22             } else {
23                 // Lost leadership
24                 isLeader.set(false);
25                 AppLogger.info("Instance {} lost leadership at {}", instanceId, timestamp);
26             }
27         } else {
28             // Try to become the leader using SET NX EX (only set if not exists with expiry)
29             String result = jedis.set(LEADER_KEY, instanceId,
30                                     SetParams.setParams().nx().ex(LEADER_EXPIRY_SECONDS));
31
32             if ("OK".equals(result)) {
33                 isLeader.set(true);
34                 AppLogger.info("Instance {} acquired leadership at {} on {}",
35                               instanceId, timestamp, hostInfo);
36             }
37         }
38     } catch (Exception e) {
39         AppLogger.error("Error in leader election: {} at {}",
40                        e.getMessage(), LocalDateTime.now(ZoneOffset.UTC).format(FORMATTER));
41
42         // On error, assume not the leader for safety
43         if (isLeader()) {
44             isLeader.set(false);
45             AppLogger.warn("Instance {} relinquishing leadership due to error", instanceId);
46         }
47     }
48 }

```

CodeBlock 5.7: Backend redis leadership service.

### 5.3.5 Infrastructure shared between local and remote states

The final piece of code being discussed is how infrastructure that is to be applied to AWS can also be deployed to Docker containers and the best ways for this. Firstly, code duplication is unwanted, it would be easiest to create local and remote version of the code and update both each time an update was needed.

This problem was solved by creating a shared and a local directory of terraform code alongside the remote directory. This means that the remote configuration can stay the same and the local provider can be altered to use the local endpoints as seen in the code block below .

```

1 terraform {
2   required_version = ">= 1.9"
3   required_providers {
4     aws = {
5       source = "hashicorp/aws"
6       version = ">= 5.60.0"
7     }
8   }
9 }
10
11 provider "aws" {
12   region = "eu-west-2"
13
14   access_key          = "test"
15   secret_key          = "test"
16   skip_credentials_validation = true
17   skip_metadata_api_check   = true
18   skip_requesting_account_id = true
19
20   s3_use_path_style = true
21
22
23   dynamic "endpoints" {
24     for_each = toset(["dynamodb", "s3"])
25     content {
26       dynamodb = "http://localhost:8000"
27       s3         = "http://localhost:4566"
28     }
29   }
30
31   default_tags {
32     tags = {
33       Environment = "local"
34       Project      = "SubShop"
35     }
36   }
37 }

```

CodeBlock 5.8: Local terraform provider.

Now that both the local and remote providers are setup, the same IaC can be added to the

shared directory and used like any other module, this is shown below where in the remote main.tf there is a s3 module and a shared\_s3 module. This approach creates one source of truth for the shared infrastructure which enhances the development experience.

```
1 module "s3" {
2   source = "../modules/s3"
3
4   environment      = var.environment
5   codepipeline_iam_role_arn = module.iam.codepipeline_role_arn
6   codebuild_iam_role_arn   = module.iam.codebuild_role_arn
7   codedeploy_iam_role_arn  = module.iam.codedeploy_service_role_arn
8 }
9
10 module "shared_s3" {
11   source = "../shared/s3"
12
13   environment      = var.environment
14   allowed_origins = [var.domain]
15 }
```

CodeBlock 5.9: Remote Terraform shared and private modules..

## Chapter 6: Installation and User manual

This chapter will cover the high level steps to install the application and a user manual so that potential users can look at this to run and use the application. There is a more in-depth installation and running instructions inside the source code if there are any issues.

### 6.1 Installation

There are some prerequisites to running the application on both Docker and AWS, first you need to setup Stripe <https://stripe.com/gb> so that you can accept payments from subscriptions as well as enable payments between users, and for this you need to sign up for Stripe connect. Then you need to create a subscription and get the priceId of it so that it can be used inside the application.

Whether deploying with either method, Cognito must be setup, and for this you will need to create an AWS account and setup the correct IAM roles so that you can get AWS credentials with access to the resources needed to create the infrastructure. To setup Cognito, open a command line, ensure terraform is installed, setup your AWS access keys using the AWS CLI. Then navigate to the remote infrastructure directory and run:

```
1 terraform apply -target=module.cognito
```

This will setup Cognito, then you can use the AWS console to get the required environment variables for it.

For more detailed instructions follow the README's inside the source code.

#### 6.1.1 Docker

To run the application using Docker, you first need to ensure Docker is installed on the platform that you are building the application on. If it is then you can proceed to setting it up. Open a Command line and navigate to the docker directory in the product, then setup the environment variables by following the README and finally run the containers using:

```
1 docker compose -f docker-compose-local.yml up -d
```

Once this is done then to apply the local infrastructure go to the local infrastructure directory and run

```
1 terraform apply
```

like how it was done for setting up Cognito. After all of that the application will be ready.



## 6.1.2 AWS

When deploying to AWS the steps are slightly different, it is recommended that you initially build the Frontend and backend Docker images, to do so go to their respective directories and follow the instructions.

Next you want to build the infrastructure using terraform, to do so navigate to the remote terraform directory, setup the environment variables using the example files and the instructions in the README, then setup the terraform backend to use GitLab terraform states as discussed above or use a different terraform backend, then run:

```
1 terraform apply
```

This will start to create the infrastructure, but there are some manual steps that need completing these are also contained in the main infrastructure README, first you need to complete the codeconnections authentication so that the CI/CD pipelines will be able to run correctly, then you need to enable production mode for SES. These steps both have guides from amazon for doing this. Finally you need to get the name servers of the Route53 hosted zone and set your domain to point at them.

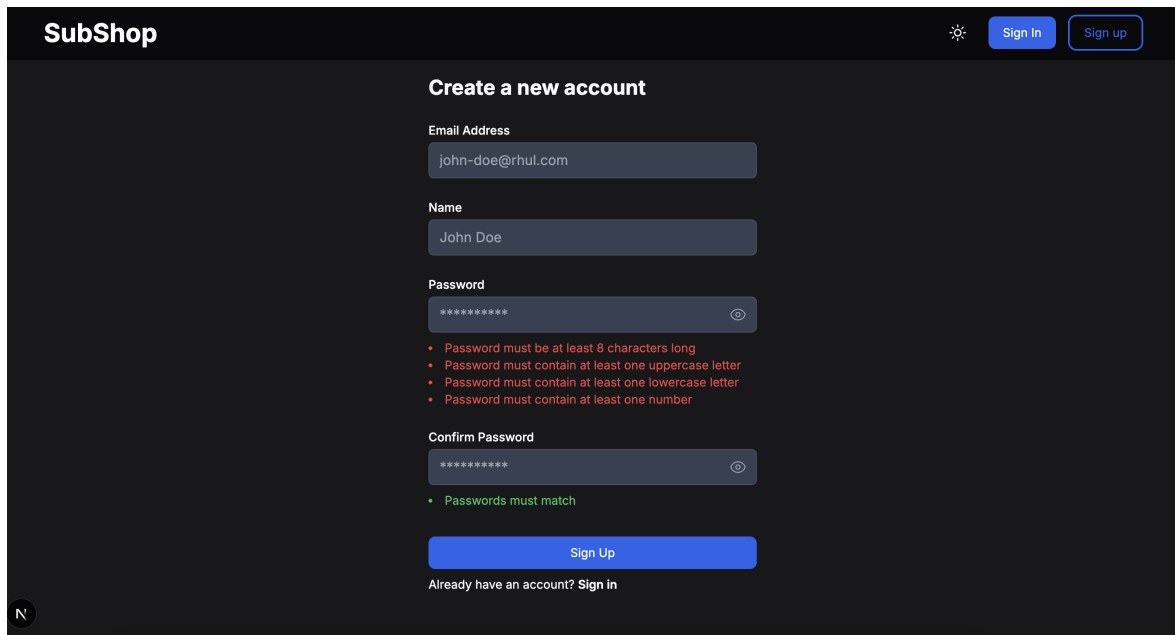
Another tip would be to push the Docker images to ECR with the latest tag before all of the deployment is completed, this will ensure the ECS services are failing to deploy while waiting for the CI/CD pipelines to finish as there will be an existing image in the repository. The steps for this will be inside ECR under view push commands.

After the infrastructure is built and the CI/CD pipeline is completed the application will be ready to receive traffic.

## 6.2 User manual

### 6.2.1 General Use

Once the code is up and running these steps will help you navigate the application. The first step is going to the application either by using the domain you configured or the address of it. Then you want to sign up as a new user using your details:



The image shows the 'Create a new account' form in the SubShop application. The form is set against a dark background. At the top left is the 'SubShop' logo, and at the top right are 'Sign In' and 'Sign up' buttons. The form fields are: 'Email Address' with the value 'john-doe@rhul.com', 'Name' with the value 'John Doe', 'Password' with masked characters and a visibility toggle, and 'Confirm Password' also with masked characters and a visibility toggle. Below the password field, there are four red error messages: 'Password must be at least 8 characters long', 'Password must contain at least one uppercase letter', 'Password must contain at least one lowercase letter', and 'Password must contain at least one number'. Below the confirm password field, there is a green message: 'Passwords must match'. At the bottom of the form is a blue 'Sign Up' button and a link 'Already have an account? Sign in'.

SubShop

Create a new account

Email Address  
john-doe@rhul.com

Name  
John Doe

Password  
\*\*\*\*\*

- Password must be at least 8 characters long
- Password must contain at least one uppercase letter
- Password must contain at least one lowercase letter
- Password must contain at least one number

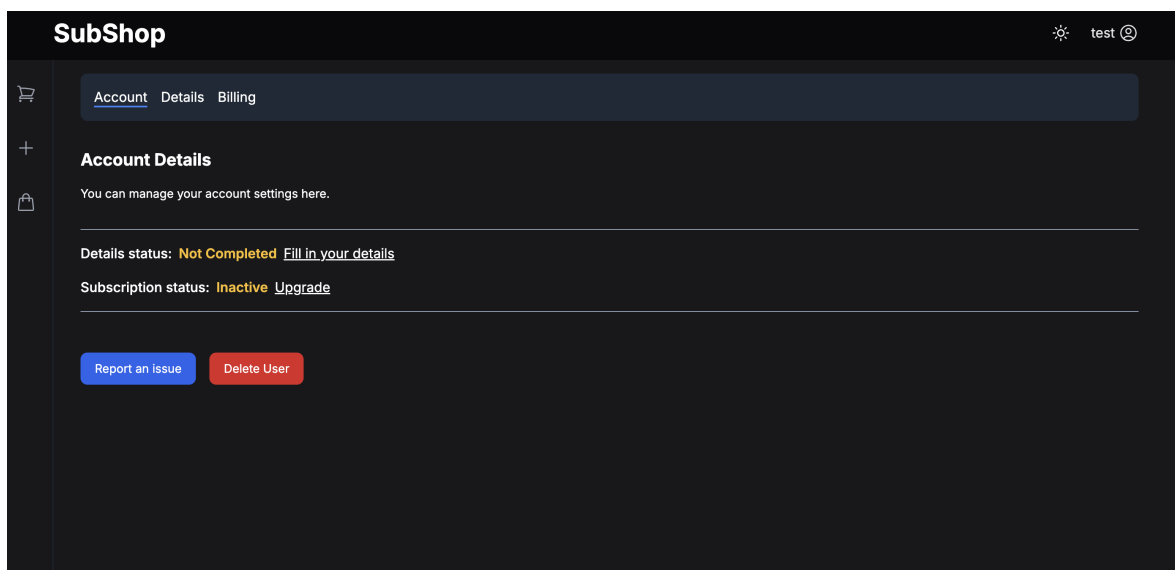
Confirm Password  
\*\*\*\*\*

- Passwords must match

Sign Up

Already have an account? [Sign in](#)

Once you have created an account you will be redirected to the accounts page, once here you will need to follow the highlighted steps to complete your account setup. After that you are free to use the application or register as a seller. There are also buttons here that allow you to delete your account or report an issue, the delete account button will remove your account and revoke access, and the report button will create a ticket in the configured Jira account with all the details from the user.



The image shows the 'Account Details' page in the SubShop application. The page has a dark background. At the top left is the 'SubShop' logo, and at the top right are a settings icon and a 'test' button. The page has a sidebar with a shopping cart icon, a plus icon, and a bag icon. The main content area has a header with 'Account', 'Details', and 'Billing' tabs. Below the header is the 'Account Details' section with the text 'You can manage your account settings here.' Below this is a section with 'Details status: Not Completed' and a link 'Fill in your details'. Below that is a section with 'Subscription status: Inactive' and a link 'Upgrade'. At the bottom are two buttons: 'Report an issue' and 'Delete User'.

SubShop

test

Account Details Billing

Account Details

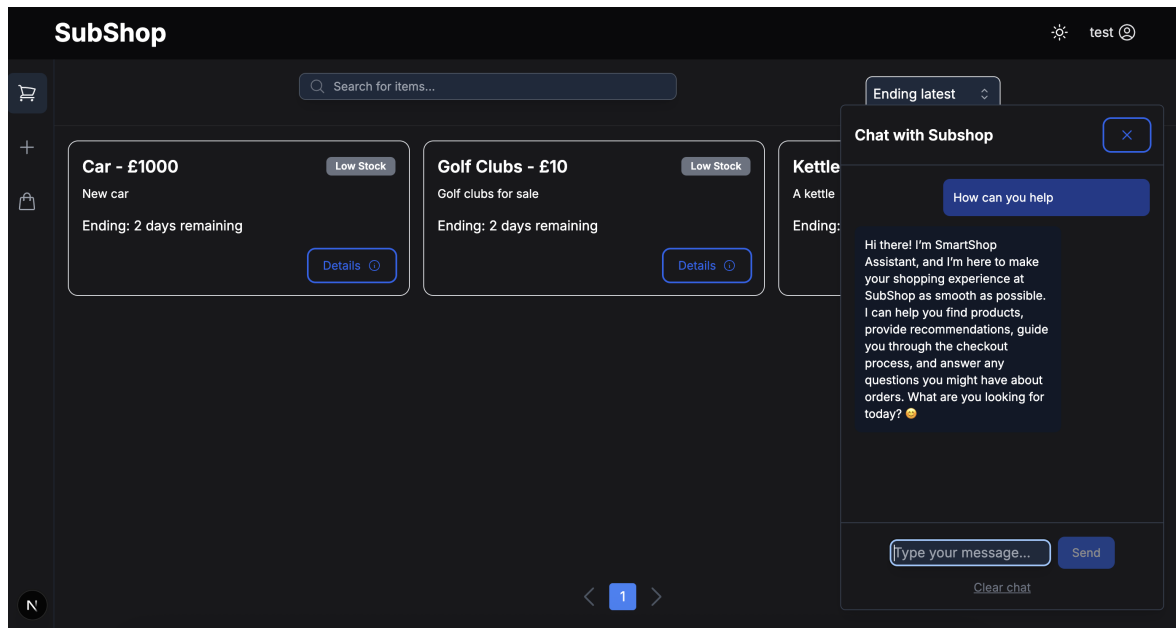
You can manage your account settings here.

Details status: **Not Completed** [Fill in your details](#)

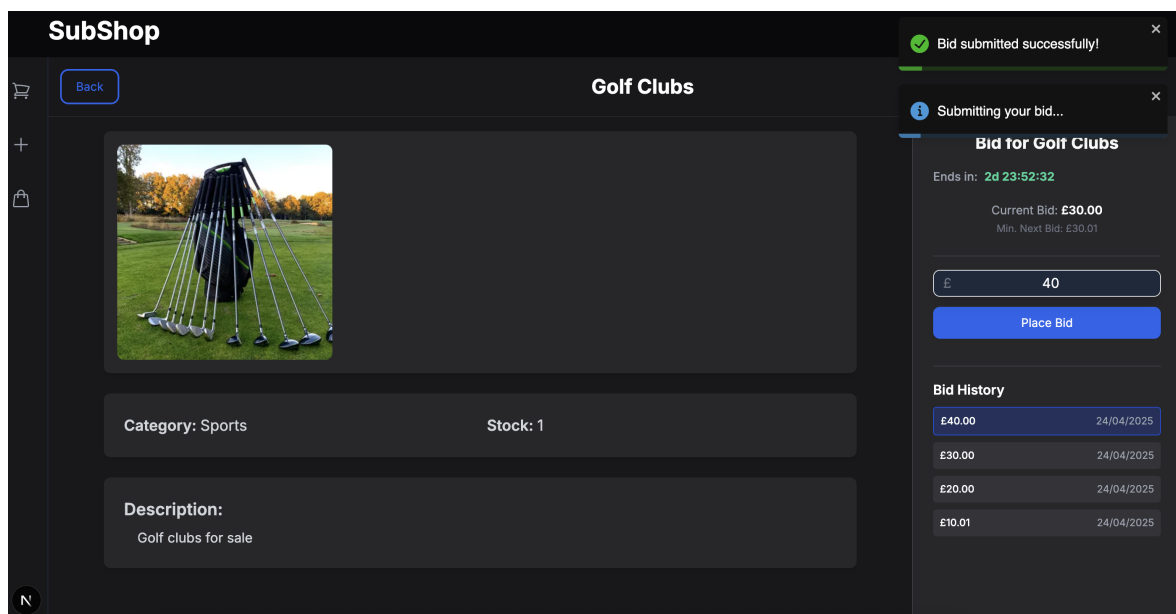
Subscription status: **Inactive** [Upgrade](#)

Report an issue Delete User

After you have set up an account you can go create new items, or browse other peoples items in the shop. As seen below there are some items already in the shop and there is a helpful chat window for you to ask questions with. The shop page contains all of the active auctions on the platform as well as give you a way to search for specific items you want.



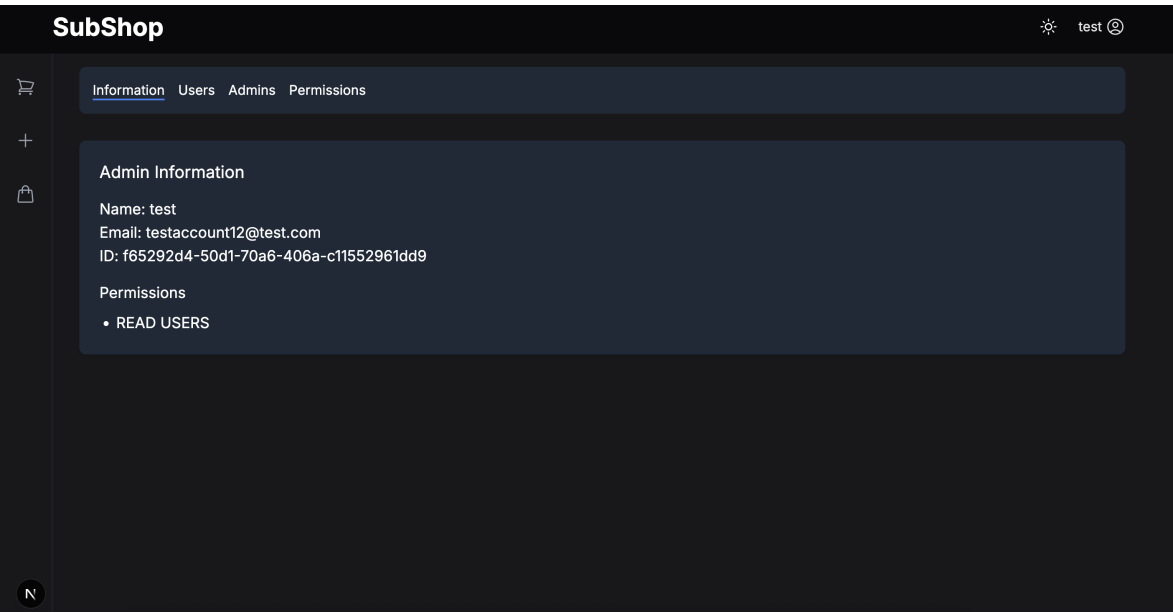
If you see an item that you might be interested in you can press the details button which shows more information about an Item as well as allows you to bid against other people for the item. The payments here do not take any money they just authorise the funds being taken once the auction is finished.



Finally, if you win the auction, the payment will go through and the item will be sent to your account.

## 6.2.2 Admin Use

To create the first admin you need to add the Cognito Group to the specific user and update the database admins table manually, after that you can use the admins dashboard shown below to promote users to admins and assign them permissions. As well as that if you have the correct permissions then you can view all of the admins, users and permissions.



## Chapter 7: Reflection

Overall the project progression was very good but there were some things that need to be reflected on. Firstly, while the harder features such as cloud deployment, CI/CD and event driven architecture with streaming are important, they did slow down progression as they were new technologies to learn so they did require more time to develop, this meant that some other features were missing important features such as the logins error handling.

Secondly, as discussed in the testing section above, the frontend and integration tests were slightly neglected as development speed was a priority so it was harder to keep these all updated.

Different subscriptions with different tiers weren't added and this was one of the main features that was wanted, although all of the supporting features were almost completed so it would not be too hard to add in this feature.

The handling of items after auctions has not been fully completed and there is no setup for the postage of items has not been setup or implemented.

While the backend was created in one service, which at the time was great, there are plans to separate it into smaller, independent services that can utilise the gateway and create a GraphQL Federation. This will have great benefits but the initial creation of it as one service was the right idea.

Finally, the diary inside the git repository was not updated as regularly as it should have been, while the progress was tracked in separate documents, it was not frequently updated inside the git repository and it would have been better to do so.

## Chapter 8: **Future plans**

The first things to be done would be to go over the reflection and try to resolve what was found there, then the plan is to break the code up into separate services and potentially repositories so that it is easier to onboard developers and maintain the code. As this all was made with a tight deadline it will be important to take a step back and try and find any mistakes made during development. Finally, reaching out to users and trying to gain traction, improve the application based on their needs and wants as well as continued feature development.

# Bibliography

- [1] F. Zhu and Q. Liu, “Competing with complementors: an empirical look at amazon.com,” *SSRN Electronic Journal*, 2014. Available: [https://www.hbs.edu/ris/Publication%20Files/amazon\\_2018-06-05\\_4a83c515-af0c-4366-9fba-8fb059d0b4f6.pdf](https://www.hbs.edu/ris/Publication%20Files/amazon_2018-06-05_4a83c515-af0c-4366-9fba-8fb059d0b4f6.pdf).
- [2] “E-commerce users in the united kingdom 2025.” Available: <https://www.statista.com/forecasts/477128/e-commerce-users-in-the-united-kingdom>.
- [3] G. Laatikainen and A. Ojala, “Saas architecture and pricing models,” 06 2014. Available: <https://ieeexplore.ieee.org/document/6930585?denied=>.
- [4] W. Cheng, Z. Yue, L. R. Ye, and D.-D. Nguyen, “Wang et al.: Subscribe to fee-based web services subscription to fee-based online services: What makes consumer pay for on-line content?,” 2005. Available: [http://ojs.jecr.org/jecr/sites/default/files/paper4\\_20.pdf](http://ojs.jecr.org/jecr/sites/default/files/paper4_20.pdf).
- [5] “What’s the best pricing model for your business?.” Available: <https://www.business.com/articles/pricing-license-vs-subscription/>.
- [6] M. Rehkopf, “User stories,” 2019. Available: <https://www.atlassian.com/agile/project-management/user-stories>.
- [7] S. Counter, “Search engine market share stat counter.” Available: <https://gs.statcounter.com/search-engine-market-share2>.
- [8] S. Counter, “Google lawsuits.” Available: <https://www.pbs.org/newshour/politics/google-faces-off-against-u-s-government-attempt-to-break-up-company-in-search-monop>
- [9] M. O. Source, “React,” 2024. Available: <https://react.dev/>.
- [10] “Built-in react hooks – react.” Available: <https://react.dev/reference/react/hooks>.
- [11] M. Levlin, “Dom benchmark comparison of the front-end javascript frameworks react, angular, vue, and svelte,” 2020. Available: [https://www.doria.fi/bitstream/handle/10024/177433/levlin\\_mattias.pdf?sequence=2&isAllowed=y](https://www.doria.fi/bitstream/handle/10024/177433/levlin_mattias.pdf?sequence=2&isAllowed=y).
- [12] “Top 30 companies using reactjs development,” 04 2023. Available: <https://jaydevs.com/top-companies-using-react-js/>.
- [13] “Typescript - javascript that scales..” Available: <https://www.typescriptlang.org>.
- [14] S. Islam, “Javascript alternative (typescript) and its effectiveness in web development degree programme software engineering,” 2023. Available: [https://www.theseus.fi/bitstream/handle/10024/795522/Islam\\_Saiful.pdf?sequence=2&isAllowed=y](https://www.theseus.fi/bitstream/handle/10024/795522/Islam_Saiful.pdf?sequence=2&isAllowed=y).
- [15] Z. Fehervari, “Astro vs next.js - zoltan fehervari - medium,” 04 2024. Available: <https://medium.com/@fhrvri.mmxiv/astro-vs-next-js-3e9b3d57462d>.
- [16] “Showcase — next.js.” Available: <https://nextjs.org/showcase>.
- [17] T. Labs, “Tailwind css - rapidly build modern websites without ever leaving your html,” 2023. Available: <https://tailwindcss.com/>.
- [18] “Introduction to apollo client,” 2024. Available: <https://www.apollographql.com/docs/react>.

- [19] “Java in web development: Why is it the language of choice for enterprise applications?.” Available: <https://www.itmagination.com/blog/java-in-web-development-why-is-it-the-language-of-choice-for-enterprise-applications>
- [20] “What is java spring boot?—intro to spring boot — microsoft azure.” Available: <https://azure.microsoft.com/en-gb/resources/cloud-computing-dictionary/what-is-java-spring-boot>.
- [21] “Home - dgs framework.” Available: <https://netflix.github.io/dgs/>.
- [22] M. Iqbal, “Netflix revenue and usage statistics (2024),” 10 2024. Available: <https://www.businessofapps.com/data/netflix-statistics/>.
- [23] Kafka, “Kafka.” Available: <https://kafka.apache.org/>.
- [24] P. G. D. Group, “Postgresql: About,” 2019. Available: <https://www.postgresql.org/about/>.
- [25] “Mysql vs. postgresql - comparing relational database management systems (rdbms) - aws.” Available: <https://aws.amazon.com/compare/the-difference-between-mysql-vs-postgresql/>.
- [26] A. Eesuola, “Sqlite vs postgresql: A detailed comparison,” 06 2024. Available: <https://www.datacamp.com/blog/sqlite-vs-postgresql-detailed-comparison>.
- [27] GitLab, “Gitlab ci/cd — gitlab.” Available: <https://docs.gitlab.com/ee/ci/>.
- [28] Hashicorp, “What is terraform — terraform — hashicorp developer,” 2024. Available: <https://developer.hashicorp.com/terraform/intro>.
- [29] AWS, “What is aws? - amazon web services,” 2024. Available: <https://aws.amazon.com/what-is-aws/>.
- [30] F. Richter, “Infographic: Amazon dominates public cloud market,” 04 2024. Available: <https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/>.
- [31] Stripe, “Online payment processing for internet businesses - stripe,” 2024. Available: <https://stripe.com/gb>.
- [32] T. G. Foundation, “GraphQL: A query language for apis,” 2012. Available: <https://graphql.org/>.
- [33] K. Schrade, “Why use graphql? — apollo graphql blog.” Available: <https://www.apollographql.com/blog/why-use-graphql>.
- [34] “N-tier architecture style - azure architecture center.” Available: <https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/n-tier>.
- [35] AWS, “What are microservices?.” Available: <https://aws.amazon.com/microservices/>.
- [36] IBM, “What is serverless computing?.” Available: <https://www.ibm.com/topics/serverless>.
- [37] IBM, “Event-driven architecture — ibm.” Available: <https://www.ibm.com/topics/event-driven-architecture>.
- [38] GeeksforGeeks, “Mvc design pattern - geeksforgeeks,” 02 2024. Available: <https://www.geeksforgeeks.org/mvc-design-pattern/>.



- [39] P. Gillin, “What is component-based architecture?,” 04 2022. Available: <https://www.mendix.com/blog/what-is-component-based-architecture/>.
- [40] GeeksforGeeks, “Singleton method design pattern,” 08 2024. Available: <https://www.geeksforgeeks.org/singleton-design-pattern/>.
- [41] O. Foundation, “Owasp top ten,” 2024. Available: <https://owasp.org/www-project-top-ten/>.
- [42] GeeksforGeeks, “Json web token (jwt) - geeksforgeeks,” 08 2024. Available: <https://www.geeksforgeeks.org/json-web-token-jwt/>.
- [43] IBM, “Ibm role based access control.” Available: <https://www.ibm.com/think/topics/rbac/>.
- [44] I. C. O. (ICO), “8. data minimisation,” 05 2023. Available: <https://ico.org.uk/for-organisations/uk-gdpr-guidance-and-resources/childrens-information/childrens-code-guidance-and-resources/age-appropriate-design-a-code-of-practice-for-online-services/8-data-minimisation/>.
- [45] AWS, “What is infrastructure as code?.” Available: <https://aws.amazon.com/what-is/iac/>.
- [46] AWS, “Six advantages of cloud computing - overview of amazon web services,” 2023. Available: <https://docs.aws.amazon.com/whitepapers/latest/aws-overview/six-advantages-of-cloud-computing.html>.
- [47] AWS, “Aws security groups.” Available: <https://docs.aws.amazon.com/vpc/latest/userguide/vpc-security-groups.html>.
- [48] AWS, “Aws elasticache.” Available: <https://aws.amazon.com/elasticache/>.
- [49] AWS, “Aws msk.” Available: <https://aws.amazon.com/msk/>.
- [50] AWS, “Aws rds.” Available: <https://aws.amazon.com/rds/>.
- [51] AWS, “Aws s3.” Available: <https://aws.amazon.com/s3/>.
- [52] AWS, “Aws dynamodb.” Available: <https://aws.amazon.com/dynamodb/>.
- [53] AWS, “Aws cognito.” Available: <https://aws.amazon.com/cognito/>.
- [54] AWS, “Aws codepipeline.” Available: <https://aws.amazon.com/codepipeline/features/?nc=sn&loc=2>.

# Appendix

# Appendix A: **Diary**

## A.1 Week 1

- I met with my supervisor to discuss what the project entails and to also discuss my initial plan.
- I am struggling to decide on what product I want to build but I am thinking about creating a store application that allows for individual users, teams and enterprise users.
- I have been researching technologies that I am planning on using both now and In the future.
- After researching the technologies I wanted to use I found some issues with how to test them so I am evaluating what would be the best option, I also asked my supervisor what they thought would be a good approach, with the advice I got and the research I completed I will be able to narrow down what to use.
- Finally I have been working on my project plan and have started the abstract, created an initial timeline and started thinking about the risks my project has.

## A.2 Week 2

- I continued working on my project plan, I had some areas where I was unsure so I asked my supervisor to read over one of my drafts.
- I continued to research web technologies and I am planning on setting up the initial application.
- I updated my plan based on the recommendations I received and finished it off.
- I submitted the project plan and merged it into the `main` branch of this repo as well as creating a tag for it so that I can easily track when it was completed.
- I had planned on setting up the initial application and I will have some of it set up but it will be hard to fully understand the requirements until my project plan has been marked and I have some feedback on it.

## A.3 Week 3

- I spent the majority of this week planning and researching.
- I have decided to use a `Java Spring Boot` application with a `PostgreSQL` database for the backend and a `Typescript Next.js` Frontend.
- I created an initial setup for both the frontend and backend and I plan to continue working on the setup over the next week.
- I did lots of research on all of these frameworks as well as authentication providers and payment providers.

## A.4 Week 4

- I setup the authentication provider that I am planning to use. To do this I used the research I did on **AWS Cognito** and **Terraform** so that I could provision it using Infrastructure as Code (IaC).
- I also setup **Apollo Client** on the frontend which I will use to manage the states of my data and interact with the backend of the application.
- Finally I setup **GraphQL Codegen** which will enable me to generate type safe code from my **GraphQL** schema.

## A.5 Week 5

- I setup authentication on the frontend which now allows for a user to login if they have a **Cognito** user account.
- I then setup the use of **JWT** for authorisation. This was only setup to send the **JWT** to the backend.
- I started adding **JWT** verification on the backend but I started getting lots of issues with **CORS** once I added **Spring Security**. This meant that the issue is rolling over into Week 6.

## A.6 Week 6

- I met with my supervisor this week and discussed my current progress, I identified that I need to start my reports as soon as possible so I am planning on doing that.
- I finished off the initial backend security that I was working on and merged it.
- I have been building the user interface out.
- Created different security profiles for the backend.
- I created a database and an initial users table in it.

## A.7 Week 7

- Created unit and integration tests for the backend.
- Added **Checkstyle** to the backend and fixed existing issues.
- Started to create **GitLab CI** pipelines for the project.
- Started my interim report.

## A.8 Week 8

- Continued to work on the pipelines but had lots of issues with the images they would run on.
- Added a `userDetails` table which contained basic information about a user.
- Created the UI for the user details.
- Researched `Netflix DGS` to fix errors and created custom datafetchers for the user details.

## A.9 Week 9

- Finalised the user details UI, backend and the database schema and functions.
- I updated the security config, specifically the `JWT` decoder.
- Fixed the backend pipeline as the new changes were causing errors.
- Continued to work on my interim report.

## A.10 Week 10

- I added items to the database and backend.
- I setup initial searching of the items.
- Created and completed the presentation slides for week 11.
- I added end-to-end and component testing to the frontend.
- Added and configured `Knip` for the frontend.

## A.11 Week 11

- Updated the frontend security info.
- Setup initial `Stripe` subscriptions.
- Added steps to setup the `Stripe` subscriptions.
- Presented my presentation and watched peer presentations.
- Finished the interim report.

## A.12 Week 12

- Assessed status of project and made a plan on what to do going forward.

- Created SQL tests for the database.
- Setup Redis as a cache.
- Setup Cache connection from backend.
- Implemented initial caching with Redis.
- Setup Cognito in the integration tests.

## A.13 Week 13

- Setup a delete user functionality that soft deletes users so that some important information is not lost.
- Started to work on the AWS infrastructure.
- Changed the setup of the IaC.

## A.14 Week 14

- There was lots of progress with the infrastructure this week but it was all works in progress so there is not much in the diary.

## A.15 Week 15

- The infrastructure overhaul was completed this week but there are still things to work on with it.
- A production merge was made after configuring the code for it.
- The User interface for searching was created and the implementation was started.

## A.16 Week 16

- There was more infrastructure overhaul this week as the load balancers were setup and SSL.
- Work on SES (Simple Email Service) and the implementation was started here and hoping to be completed by the end of the week.

## A.17 Week 17

- The SES infrastructure and implementation were completed like planned.
- An integration with Jira for bug reporting was created.

- A backend logging solution was added.
- Fixes for the integration tests were completed.

## A.18 Week 18

- Started the final report.
- Lots of new backend features were created this week.
- Queries and mutations for items were made.
- Pagination was added and worked on.
- The create item UI was made.

## A.19 Week 19

- Queries to get items by users were made.
- The backend was restructured and refactored.
- The GraphQL schema was restructured as well.
- Pages for the users items were created.
- A production merge was made.

## A.20 Week 20

- Frontend pagination, sorting and filtering were added where possible.
- The shops items got an overhaul.
- Admin features were started to be worked on.

## A.21 Week 21

- Admin features were created and worked on heavily.
- There were lots of fixes to do with the admins.
- Especially to do with how the caching worked here.
- Production merge.

## A.22 Week 22

- More infrastructure updates.
- This mainly was to do with the CI/CD.
- Test, build and deployment stages were created inside a pipeline.

## A.23 Week 23

- An integration with OpenAI was started.
- A AI chatbot was added.
- Work for an Apollo Gateway was started.

## A.24 Week 24

- There were lots of updates to the gateway and it was completed.
- But it was parked for now as the backend is still one service.
- The infrastructure was also setup and parked.

## A.25 Week 25

- The bidding service was setup and completed.
- Kafka was configured and added.
- Final infrastructure updates were made.
- The final changes were added.
- The final report was finished.