Final Year Project Report

Full Unit - Interim

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

December 13, 2024

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 Where other published and unpublished source materials have been used, these have been acknowledged.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: 7510

Student Name: Edward Instance

Date of Submission: 14/12/2024

Signature: einstance

Table of Contents

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. The report concludes with a status update on the development process and next steps toward the project's completion.

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 primarily to track the progress of 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 though a series of goals and achievements. Then after an outline of how the problem will be solved there will be a chapter on what has been done so far and how the project got there. Alongside the project specific information there is also a couple of sections about state of the art web development as a whole and other aspects related to web development.

There are 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 survey

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. Because of this there are gaps in the report and this will be addressed in the future. Meanwhile the literature used adds value to the report and project by guiding decisions that were made and by showing why others were made.

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.

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 fount but also user trends.
- Prepare for scalability so that when deployed, spikes of users will not damage the application or affect other users.
- Enable communication between buyers and sellers: Create messaging and communication features to allow buyers and sellers to negotiate prices and clarify item details before completing a purchase.
- 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.

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. 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. 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. 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. As an administrator I want to understand the activity the application is receiving as well as compare it to the past.
5. 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. As a user I want my transactions to be secure so that I can feel confident that I will not lose any money.
7. 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. As an administrator I want to have access to what subscriptions people are getting so that I can judge what is wanted.
9. As an administrator I want the application to be available to all users that want to access it.
10. As an administrator I want the application to be quick for all the users so that they do not get frustrated.
11. 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.
12. As a User I want to be able to ask for help if any issues with the application occur.
13. As a user I want to be able to view all of the past items I bought.
14. As a seller I want the ability to easily change or cancel my subscription based on my needs.
15. As a seller I want to choose how I am paid by the platform.

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.

2.1 Technologies

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.

2.1.1 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 [7], 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) [8], interact with external systems (e.g., fetching data) [8], and share data between components without passing it explicitly as props (e.g., via context) [8]. 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 [9] as it performed well in the benchmark section, and is also the most used and well-known framework [9]. 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 [9].

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 [10]. 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.

2.1.2 TypeScript

TypeScript is a strongly typed programming language that builds on JavaScript, giving you better tooling at any scale [11]. 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 [12] and helps developers write code that is easier to maintain and extend over time [12].

2.1.3 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 [13].

Java's architecture encourages scalability thanks to built-in support for multi-threading and concurrency [13]. 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.

2.1.4 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 [14]. 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 [15]. As well as that PostgreSQL is always ACID compliant [15] whereas MySQL is not always ACID compliant. Secondly, PostgreSQL has more features, better performance and scalability when compared to SQLite [16]. Overall PostgreSQL was the best choice for database as it has a rich set of features and outperforms its competitors.

2.1.5 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.

2.1.6 GraphQL

GraphQL is a query language for APIs and a runtime for fulfilling those queries with your existing data [17]. 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 [18]. 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.

2.1.7 Terraform

Terraform is an infrastructure as code (IAC) tool that lets you define both cloud and on-prem resources [19]. 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.

2.1.8 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.

2.1.9 GitLab CI/CD

GitLab CI/CD is a continuous method of software development, where you continuously build, test, deploy, and monitor iterative code changes [20]. 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.

2.2 Frameworks

This section contains all of the frameworks that this project is using. These 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.

2.2.1 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 [21]. 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 [22]. This shows that it is trusted and can be used to create applications with high performance and high loads.

2.2.2 Tailwind CSS

Tailwind is a utility-first CSS framework [23], 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 [23], 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.

2.2.3 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 [24].

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.

2.2.4 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 [25]. 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 [25].

Spring Boot provides Convention over configuration [25] this means that developers only write code for the unconventional aspects of the app they're creating [25]. 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 [25]. 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 [25], Actuator for monitoring and management, and starter dependencies make it easier to set up, test, and deploy applications efficiently.

2.2.5 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 [26].

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 [27], 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.

2.3 Platforms

2.3.1 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.

2.3.2 Amazon Web Services

Amazon Web Services (AWS) is the world's most comprehensive and broadly adopted cloud [28], AWS is the leading cloud and they have significantly more services, and more features within those services, than any other cloud provider [28]. 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% [29]. 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.

2.3.3 Stripe

Stripe is a fully integrated suite of financial and payment products [30]. 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 Stipe 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.

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 [31]. Layers are a way to separate responsibilities and manage dependencies. Each layer has a specific responsibility [31]. 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 [31]. 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 [32]. 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 [32]. 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 [33]. 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 [34]. 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 [34].

Currently, this project does not implement an event-driven approach, but future developments may incorporate event processing for actions like user sign-up, checkout, and others. Adopting an EDA model could offer several advantages, including better scalability. Since event consumers can scale independently, the application would be better equipped to handle increasing traffic and workloads

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) [35]. 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 [35] 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.

3.2.3 Component-based architecture

Component-based architecture is a framework for building software based on reusable parts [36], 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 [37]. 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 have been considered for this project so far. During the project the OWASP top 10 Web Application Security Risks [38] will be considered as these are all likely risks that could be exploited.

3.3.1 Authorization

Authorisation is the process of granting or denying access to resources based on a user's identity and privileges.

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 [39]. 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 SQL Injection

SQL injection was the top of the OWASP top 10 for years and has recently been to third [38], 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.

Currently there is not much protection against these attacks in the project but this will definitely be addressed in the future. One protection that is in the project is that when interacting with the database all of the queries are parametrized which helps prevent injection attacks as the data being processed interacts with the database differently as a parameter.

3.3.3 Infrastructure misconfiguration

Infrastructure misconfiguration can cause serious vulnerabilities and insecure design and security misconfiguration are on the OWASP 10 [38]. 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 [40]. 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.

3.4.2 User consent

There will be a consent warning when joining the page so that the user can what details and power they are giving the platform. This is a plan to increase transparency with the users so that they can trust the application. Secondly there will be a privacy policy which will highlight all of this above while also showing users how their data will be protected, once again showing them that we can be trusted with their privacy.

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 soon 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.

Currently all that has been made on the DevOps front is the testing and linting pipelines, and next should be the infrastructure for deploying the application as well as setting up a end to end monitoring solution that can work both locally and when deployed.

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 [41]. Whereas some of the benefits are that toy 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 will be part of this project, this will be done using AWS and some of their services are already in use. The reason cloud deployments are being considered over on-premises deployments is because they are infinitely scaleable, we do not need to manage servers, we can avoid paying expensive capital expenditure 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 [42] 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.

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.

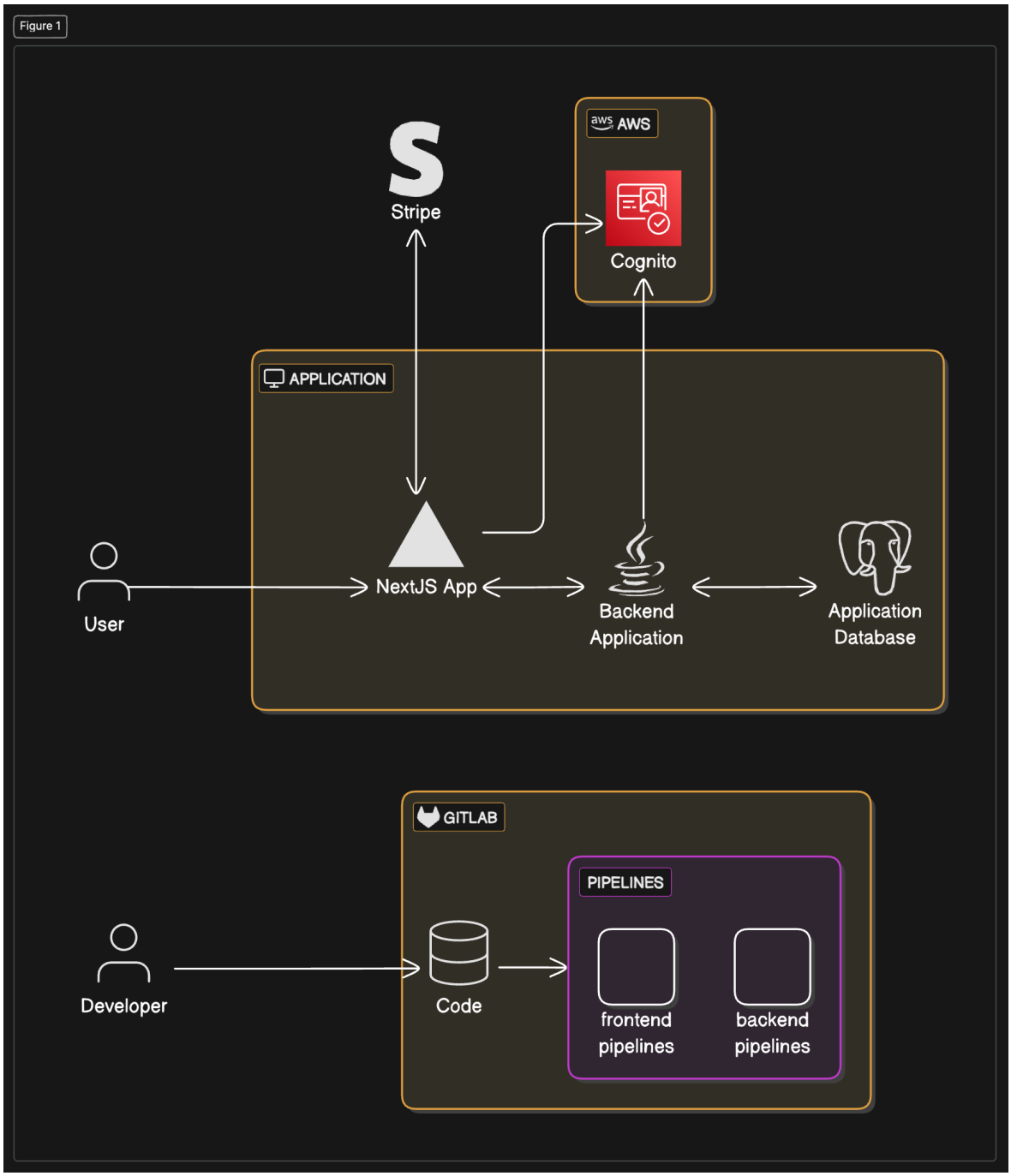
3.5.4 User experience

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: **Progress**

This final chapter contains all the information about what has been completed so far, it has been split up into the architecture and infrastructure, the database design, the features and finally the future steps of the application development.

4.1 System architecture and Infrastructure



The diagram above shows the architecture of the application and how each service interacts with the other services. To start you can see the user on the left interacting with the

Next.js application which is the frontend, the frontend interacts Stripe for payments and AWS Cognito for identity management.

Next you can see that the backend interacts with the frontend, database and AWS Cognito, the interactions with the database stores the data sent from the frontend and also retrieves data and sends it to the frontend. The backend's interaction with cognito is for verifying the JWT tokens.

Finally, at the bottom there is the developer experience, once code is committed and pushed to GitLab, the automated pipelines for running tests and linting is ran.

The details of all of these interactions are below, but first there is the code on how Cognito is setup using IaC. This first section sets up the versions of terraform and the AWS provider.

```
terraform {
  required_version = ">= 1.9"

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = ">= 5.60.0"
    }
  }
}
```

Next the user pool is setup it is setup to use emails as the primary username, then a password policy is setup as a basic security measure, finally the user's name is added to the cognito schema.

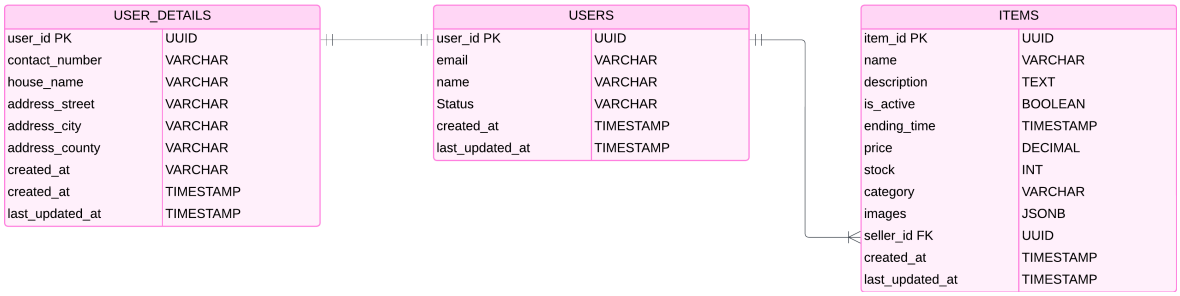
```
resource "aws_cognito_user_pool" "user_pool" {
  name = "final-year-project-user-pool"

  # Allow users to sign in with their email address
  username_attributes = ["email"]

  #   Alow users to sign up with their email address
  admin_create_user_config {
    allow_admin_create_user_only = false
  }

  # Allow users to recover their account using their email address
  account_recovery_setting {
    recovery_mechanism {
      name     = "verified_email"
      priority = 1
    }
  }

  # Configure the password policy for the user pool
  password_policy {
    minimum_length    = 8
    require_lowercase = false
    require_numbers   = true
```

```
    require_symbols   = false
    require_uppercase = true
  }

  schema {
    attribute_data_type = "String"
    name                = "name"
    required            = true
    mutable             = true
  }

  tags = {
    Name : "final-year-project-user-pool"
  }
}
```

Then the client is setup, with the correct user pool, OAuth, authentication flows and the token refresh. This is setup so that the client can authenticate users with username and passwords as well as refreshing the JWT tokens. With that Cognito is setup, and once the configuration is applied the client and user pool id's are outputted.

```
resource "aws_cognito_user_pool_client" "cognito_client" {

  name = "final-year-project-client"

  # Set the client's user pool to be the one created above
  user_pool_id = aws_cognito_user_pool.user_pool.id

  # Enable and configure OAuth authentication with JWT tokens
  supported_identity_providers = ["COGNITO"]

  # Enable the following flows for client authentication
  explicit_auth_flows = [
    "ALLOW_REFRESH_TOKEN_AUTH",
    "ALLOW_USER_PASSWORD_AUTH",
    "ALLOW_USER_SRP_AUTH"
  ]

  # Keep the refresh token valid for 30 days
  refresh_token_validity = 30

  # Don't generate a client secret
  generate_secret = false
}
```

4.2 Database design



The database is designed based on the required information about users, and on research of online shops and what information they show. The schema is also normalised when new tables and information is added to the schema. Finally, Unified Modeling Language (UML) diagrams are being used to diagram the schema and show the relationships. Using UML also helps identify potential design issues early in development, such as unnecessary dependencies or missing attributes.

4.3 Features

This section contains some of the most important features that have been completed and implemented.

4.3.1 Schema

This is the GraphQl schema, it shows the types, inputs, queries and mutations that are used, the schema also works as the API documentation as it describes all of the information that can be requested as well as the functions that can be called. The UML diagram above shows the current state of the database schema and it will be used as an example on how the schema evolves throughout the project.

```
type Mutation {
    createUser(userInput: UserInput): User
    saveUserDetails(id: String!, detailsInput: UserDetailsInput): User
    saveItem(itemInput: ItemInput): Item
}

type Query {
    getUser: User
    searchForItems(searchText: String): [Item]
}

input UserInput {
    id: String!
    email: String!
    name: String
}

input UserDetailsInput {
```

```
    contactNumber: String
    houseName: String
    addressStreet: String
    addressCity: String
    addressCounty: String
    addressPostcode: String
}

input ItemInput {
    id: String
    name: String
    description: String
    isActive: Boolean
    endingTime: String
    price: Float
    stock: Int
    category: String
    images: [String]
}

type User {
    id: String!
    email: String!
    name: String
    status: String
    details: UserDetails
    items: [Item]
}

type UserDetails {
    id: String!
    contactNumber: String
    houseName: String
    addressStreet: String
    addressCity: String
    addressCounty: String
    addressPostcode: String
}

type Item {
    id: String
    name: String
    description: String
    isActive: Boolean
    endingTime: String
    price: Float
    stock: Int
    category: String
    images: [String]
    seller: User
}
```

4.3.2 Searching

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.

```
CREATE OR REPLACE FUNCTION search_for_items(
    search_text VARCHAR
)
RETURNS TABLE (
    item_id UUID,
    name VARCHAR,
    description TEXT,
    is_active BOOLEAN,
    ending_time TIMESTAMP,
    price DECIMAL,
    stock INT,
    category VARCHAR,
    images JSONB,
    seller_id UUID
) AS $$
BEGIN
    RETURN QUERY
    SELECT
        i.item_id,
        i.name,
        i.description,
        i.is_active,
        i.ending_time,
        i.price,
        i.stock,
        i.category,
        i.images,
        i.seller_id
    FROM items i
    WHERE similarity(i.name, search_text) > 0.3
    ORDER BY similarity(i.name, search_text) DESC;
END;
$$ LANGUAGE plpgsql;
```

4.3.3 Running

This section explains one of the ways to run the whole application together, Docker is used because of the reasons discussed in the technologies section, and a compose file is used so that all of the application can be ran from one file. This also allows all of the application to be on the same docker network, and for some services to depend on others so that they are ran in the correct order. Finally healthchecks are added here so that Docker can better monitor the running services.

```yaml
services:
  final-project-frontend:
    build:
      context: ../frontend/
      dockerfile: Dockerfile
    container_name: final-project-frontend-local
    ports:
      - 3000:3000
    healthcheck:
      test: curl --fail http://localhost:3000/api/health || exit 1
      interval: 60s
      retries: 5
      start_period: 20s
      timeout: 10s

  final-project-backend:
    build:
      context: ../backend/application/backendService/
      dockerfile: Dockerfile
    container_name: final-project-backend-local
    ports:
      - 8080:8080
    healthcheck:
      test: curl --fail http://localhost:8080/details/health || exit 1
      interval: 60s
      retries: 5
      start_period: 20s
      timeout: 10s

  postgres:
    container_name: final-project-database-local
    image: postgres
    environment:
      POSTGRES_DB: ${POSTGRES_DB}
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
    ports:
      - "5432:5432"

  liquibase:
    image: liquibase/liquibase
    container_name: final-project-liquibase-local
    command: --url=jdbc:postgresql://${POSTGRES_URL}:5432/${POSTGRES_DB} --username=${POS
    volumes:
      - ../backend/database/src:/liquibase/src
      - ../backend/database/changelogs:/liquibase/changelogs
    depends_on:
      - "postgres"
```

4.3.4 Security

This is an example of some of the backend security config, this is the API key filter which is used within the authentication chain. It takes in a valid API key in the constructor and uses

this when comparing API keys from requests.

```
public class ApiKeyFilter extends GenericFilterBean {

  private final String validApiKey;

  public ApiKeyFilter(final String apiKey) {
    this.validApiKey = apiKey;
  }
```

The filter method takes in a request, response and the filter chain, the request it retrieved from the method and from that the X-API-KEY header is retrieved. If there is no API key the rest of the security chain is checked, but if there is a key and it is valid then the security context is set to a authentication object which is approved, then the filter is ran with the new authentication and is approved. Finally, if the API key is invalid then an exception is thrown, and it is caught by the try catch and a 401 error is returned.

```
  @Override
  public void doFilter(final ServletRequest request,
                       final ServletResponse response, final FilterChain chain
  ) throws IOException, ServletException {

    HttpServletRequest httpRequest = (HttpServletRequest) request;
    HttpServletResponse httpResponse = (HttpServletResponse) response;
    String apiKey = httpRequest.getHeader("X-API-KEY");

    try {
      if (apiKey != null) {
        if (validApiKey.equals(apiKey)) {
          Authentication auth = new
                  UsernamePasswordAuthenticationToken(
                  apiKey, null, Collections.emptyList()
          );

          SecurityContextHolder.getContext().setAuthentication(auth);
        } else {
          throw new BadCredentialsException("Invalid API Key");
        }
      }
      chain.doFilter(request, response);
    } catch (BadCredentialsException ex) {
      // Set 401 status and include an error message in the response
      SecurityContextHolder.clearContext();
      httpResponse.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
      httpResponse.getWriter().write("Unauthorized: " + ex.getMessage());
    }
  }
}
```

4.3.5 Subscriptions

This code deals with the interaction between the frontend and Stripe to create the subscriptions, the API route starts by getting the user id from the request and an error is thrown if it is not present. Then the price id that is being used for the subscription is retrieved from the .env file and again if there is not one an error is thrown.

```
export async function POST(request: NextRequest) {
  try {
    const { userId } = await request.json();

    if (!userId) {
      return NextResponse.json({ error: "Missing userId" }, { status: 400 });
    }

    const priceId = process.env.STRIPE_PRICE_ID;
    if (!priceId) {
      return NextResponse.json(
        { error: "Missing STRIPE_PRICE_ID" },
        { status: 500 },
      );
    }
```

Once the correct id's have been retrieved the API checks gets the customer details and then if the user has an existing subscription by using one of the Stripe helper functions we created. If there is a subscription then the payment intent for the next invoice is retrieved.

If the user's existing subscription was cancelled then a new subscription is created using the Stripe SDK with the price details. The payment behaviour is set to default_incomplete as the user needs to pay for their new subscription. and the subscription request is extended to have the latest payment intent, the payment intent's client secret is what is returned.

```
    const customer = await findCustomerByUserId(userId);

    if (!customer) {
      return NextResponse.json(
        { error: "Customer not found" },
        { status: 404 },
      );
    }

    // Check for existing subscription
    const existingSubscription = await findExistingSubscriptionByUserId(userId);

    if (existingSubscription) {
      // Handle specific statuses of the existing subscription
      const latestInvoice =
        existingSubscription.latest_invoice as Stripe.Invoice;
      const paymentIntent =
        latestInvoice?.payment_intent as Stripe.PaymentIntent;
```

```
  if (existingSubscription.status === "canceled") {
    const newSubscription = await stripe.subscriptions.create({
      customer: customer.id,
      items: [{ price: priceId }],
      payment_behavior: "default_incomplete",
      payment_settings: { save_default_payment_method: "on_subscription" },
      expand: ["latest_invoice.payment_intent"],
    });

    const updatedInvoice = newSubscription.latest_invoice as Stripe.Invoice;
    const updatedPaymentIntent =
      updatedInvoice.payment_intent as Stripe.PaymentIntent;

    return NextResponse.json({
      client_secret: updatedPaymentIntent?.client_secret,
    });
  }
```

If the user's existing subscription was incomplete then the payment intent client secret is returned immediately. Secondly, if the subscription is still active, then a that message is returned.

```
  if (existingSubscription.status === "incomplete") {
    // Return the client_secret to complete the subscription
    return NextResponse.json({
      client_secret: paymentIntent?.client_secret,
    });
  }

  // If the subscription is already active
  return NextResponse.json({ message: "Subscription is already active." });
}
```

Finally if there is no existing subscription a brand new one is creating using the same settings as the one above and then the payment intent's client secret is returned.

```
// No existing subscription, create a new one

const newSubscription = await stripe.subscriptions.create({
  customer: customer.id,
  items: [{ price: priceId }],
  payment_behavior: "default_incomplete",
  payment_settings: { save_default_payment_method: "on_subscription" },
  expand: ["latest_invoice.payment_intent"],
});

const latestInvoice = newSubscription.latest_invoice as Stripe.Invoice;
const paymentIntent = latestInvoice.payment_intent as Stripe.PaymentIntent;
```

```
    return NextResponse.json({
      client_secret: paymentIntent?.client_secret,
    });
```

If there are any errors they are caught and an error message is returned.

```
  } catch (error) {
    console.error("Error in subscription handler:", error);
    return NextResponse.json(
      { error: error.message || "Unexpected error occurred" },
      { status: 500 },
    );
  }
}
```

4.3.6 Pipelines

Finally this is the configuration for the frontend pipeline, it uses node:23-alpine as a base image and throughout the whole pipeline it is the only image used. First the lint and test stages are defined, then a cache and before script. The before script installs the frontend dependencies and then caches them for usage in the next steps.

```
image: node:23-alpine

stages:
  - lint
  - test

default:
  cache:
    key: $CI_COMMIT_REF_SLUG
    paths:
      - .npm/
  before_script:
    - cd ./product/frontend/
    - echo "Installing dependencies..."
    - npm ci --cache .npm --prefer-offline
```

Next the two jobs in the lint stage are ran, first the cache is checked and the dependencies are retrieved from the cache and if they are not there then they are installed. These steps run eslint and knip.

```
lint_frontend:
  stage: lint
  script:
    - echo "Running frontend linting..."
    - npm run lint
```

```
knip_frontend:
  stage: lint
  script:
    - echo "Running frontend knip..."
    - export API_KEY="Test" # Added mock api key for knip
    - npm run knip
```

Finally, the before script is ran again before the final job which is the component tests.

```
component_test_frontend:
  stage: test
  script:
    - echo "Running frontend component tests..."
    - npm run cy:run:ct
```

4.4 Future plans

The future plans for this project after this report has been completed is to continue working on the core functionality of the application, as well as fixing the issues that were found while doing the research for this report, such as the gaps in the literature review. The next big steps for this project is setting up the production and test enviroments on AWS, as well as updating the GitLab pipelines to setup CI/CD for these new environments. One last issue to address would be to update the searching functionality to use a system that is more advanced than similarity matching, tools like OpenSearch and Elasticsearch .

Bibliography

[1] F. Zhu and Q. Liu, "Competing with complementors: an empirical look at amazon.com," *SSRN Electronic Journal*, 2014.

[2] "E-commerce users in the united kingdom 2025."

[3] G. Laatikainen and A. Ojala, "Saas architecture and pricing models," 06 2014.

[4] L. R. Y. Wang Cheng, Zhang Yue and D.-D. Nguyen, "Wang et al.: Subscribe to fee-based web services subscription to fee-based online services: What makes consumer pay for online content?," 2005.

[5] "What's the best pricing model for your business?."

[6] M. Rehkopf, "User stories," 2019.

[7] M. O. Source, "React," 2024.

[8] "Built-in react hooks – react."

[9] M. Levlin, "Dom benchmark comparison of the front-end javascript frameworks react, angular, vue, and svelte," 2020.

[10] "Top 30 companies using reactjs development," 04 2023.

[11] "Typescript - javascript that scales.."

[12] S. Islam, "Javascript alternative (typescript) and its effectiveness in web development degree programme software engineering," 2023.

[13] "Java in web development: Why is it the language of choice for enterprise applications?."

[14] PostgreSQL, "Postgresql: About," 2019.

[15] "Mysql vs. postgresql - comparing relational database management systems (rdbms) - aws."

[16] A. Eesuola, "Sqlite vs postgresql: A detailed comparison," 06 2024.

[17] T. G. Foundation, "Graphql: A query language for apis.," 2012.

[18] K. Schrade, "Why use graphql? — apollo graphql blog."

[19] Hashicorp, "What is terraform — terraform — hashicorp developer," 2024.

[20] GitLab, "Gitlab ci/cd — gitlab."

[21] Z. Fehervari, "Astro vs next.js - zoltan fehervari - medium," 04 2024.

[22] "Showcase — next.js."

[23] tailwindcss, "Tailwind css - rapidly build modern websites without ever leaving your html.," 2023.

[24] "Introduction to apollo client," 2024.

[25] "What is java spring boot?—intro to spring boot — microsoft azure."

[26] "Home - dgs framework."

[27] M. Iqbal, "Netflix revenue and usage statistics (2024)," 10 2024.

[28] AWS, “What is aws? - amazon web services,” 2024.

[29] F. Richter, “Infographic: Amazon dominates public cloud market,” 04 2024.

[30] Stripe, “Online payment processing for internet businesses - stripe,” 2024.

[31] “N-tier architecture style - azure architecture center.”

[32] AWS, “What are microservices?.”

[33] IBM, “What is serverless computing?.”

[34] IBM, “Event-driven architecture — ibm.”

[35] GeeksforGeeks, “Mvc design pattern - geeksforgeeks,” 02 2024.

[36] P. Gillin, “What is component-based architecture?,” 04 2022.

[37] GeeksforGeeks, “Singleton method design pattern,” 08 2024.

[38] OWASP, “Owasp top ten,” 2024.

[39] GeeksforGeeks, “Json web token (jwt) - geeksforgeeks,” 08 2024.

[40] ICO, “8. data minimisation,” 05 2023.

[41] AWS, “What is infrastructure as code?.”

[42] AWS, “Six advantages of cloud computing - overview of amazon web services,” 2023.

Appendix