

**By Maliha Ahmed**  
**Supervisor: De Vera, Erjill**  
**Year 3 – Final Year Project**

# **ADVANCED WEBSITE DEVELOPMENT**

## **(2024)**

**Word count: 14,134**

## Contents

<b>Abstract</b> .....	3
<b>Chapter 1: Rationale</b> .....	4
<b>Section 1.1: Introduction</b> .....	4
<b>Section 1.2: Overview</b> .....	4
<b>Section 1.3: Aims and Objectives</b> .....	4
<b>Chapter 2: Literature Review</b> .....	6
<b>Chapter 3: Background Reading</b> .....	14
<b>Section 3.1: State of the art Web Development</b> .....	14
<b>Section 3.2: Architectural Paradigms</b> .....	17
<b>Section 3.3: System Use Cases</b> .....	19
<b>Chapter 4: Software Engineering Process</b> .....	21
<b>Section 4.1: Planning</b> .....	21
<b>Section 4.2: Design</b> .....	24
<b>Section 4.3: Development</b> .....	32
<b>Section 4.3: Testing</b> .....	37
<b>Section 4.5: Tools</b> .....	43
<b>Chapter 5: Technical Decision Making</b> .....	45
<b>Chapter 6: Critical Analysis and discussion</b> .....	47
<b>Section 6.1: Project Process</b> .....	47
<b>Section 6.2: Presentation of Findings</b> .....	47
<b>Section 3: Deliverable analysis</b> .....	48
<b>Chapter 7: Professional Issues-UF</b> .....	50
<b>Chapter 8: Bibliography and Citations</b> .....	51
<b>User Manual</b> .....	52
<b>Demo</b> .....	53
<b>Project Diary</b> .....	54

## Abstract

This document presents a development of the advanced web development for creating a sustainable website the aims to minimise textile pollution across the world. This is executed through the creation of a fully developed website, including key features such as an authentication system, listing of websites, searching, adding to cart, and including a payment system. This document consists of 8 Chapters. Chapter 1 consists of a brief discussion of the aims and goals of the website, a rational and a quick overview of the requirements. Chapter 2 consists of a critical analysis of all the relevant background materials such as research papers, books, and other web pages. Chapter 3 consists of background reading which discusses the state of art of the website, evaluating the different programming languages that is ideal to use during Web Development, Architecture paradigms, discussing different methods to be used during the development, and lastly System Use Cases, which analyses existing websites and depicts their functionalities and UX designs. Chapter 4 discuss the software engineering process including the planning which swiftly discusses the project timeline, the design which depicts a series of potential designs of the final UI, the development which highlights the most prominent aspect of the code further showing ways and techniques of implementing the code, testing, which tests and functionality of the website and lastly the tools that have been utilised to ensure the excellence of the website. Chapter 5 discuss the technical decisions that have been made throughout the development of the website, including the chosen programming language, the design and other essential decision during the development of the project. Chapter 6 discusses a critical self-evaluation, outlining the personal goals and requirements that have been met, the goals of the project and lastly carrying out an intensive deliverable analysis. The document will then conclude with professional issues, particularly on plagiarism and ways on how avoiding plagiarism has been a prominent topic throughout the project, ensuring all the research and citations are cited in Section 8. This report will also include the User Manual, Demo Video, and the Project diary at the end of the file to demonstrate the functionality and the timeline of the project being created.

# Chapter 1: Rationale

## Section 1.1: Introduction

In the emerging of new fashion and clothing as the time passes, more and more clothing is being produced. This has a significant effect on the consumers behaviour. Due to the distribution of social media, consumers are more aware of the environmental consequences caused by the mass production of clothing. My commitment lies on developing a website that is not only fully functional, but also prioritise security and the safety of buying and selling clothing. A successful website consists of creating a website that is both visually appealing, with the minimum visual fog, and maximising the simplicity of the website, so that its optimal in the simplicity, diversity, colourfulness, intensity, and interactivity. [1] All these features allow the end user to familiarize themselves with the look and feel of the website, allowing them to engage and interact more, overall having a positive experience. A fully engaging website should include the use of persuasive texts, and light colours, matching the theme of the website, for example, in the ecommerce website, the use of light and autumn colours will give a feeling of comfort, overall enhancing a positive experience. [2]

## Section 1.2: Overview

Important features such as the use of listing, adding to cart, viewing the items for sale, checkout and payment shapes a fully functional website, such features enhance the usability and successfully shapes the purpose of the website [3]. Before developing a website, it is essential to carry out comprehensive research on small aspects, such as programming languages, as all languages have strengths and weaknesses, doing research on these aspects will ensure that the chosen languages will adhesively tailor to the user's needs. Carrying out research on Architectural Paradigms and analysing similar functionalities of other works would play a part on understanding way to optimise a website without carrying out trial and error, allowing the quicker execution of the project with the best possible outcome. The development should always follow the rules of software engineering, through the process of design and planning, carried out by the developing and testing, ensuring the documentation of each step, which gradually depicts the development on paper which allows the ease of reflection down the line. Lastly, the use of critical analysis of technical decisions and self-reflection allows to see the positives and negatives, allowing an adhering learning experience for future projects.

## Section 1.3: Aims and Objectives

The main aim of this project is to create a fully functioning ecommerce website that ensures the safety and efficient functionality of the website, including advanced features. It is required to make sure that the website is visually appealing, user-friendly, furthermore carries out the best user practice, adhering to human computer interaction. The aims also include the use of proper documentation that depicts the website development, documenting each step initiating from the research stage, through the potential deployment of the website. It is necessary to include any professional factors that has been carried out throughout the development, to ensure the best practices.

## The objectives

To meet these aims, the following section lists the objectives that need to be completed:

To ensure a *fully developed UI*:

- **User registration and Authentication:** Ensuring the user can create an account and log into their accounts.
- **Listing products for sale:** To ensure that the users can list their product for sale by adding a title, description, category, price, and image of the product.
- **View the listed products:** The users should be able to view the product that they have just listed and must be able to click on the listing on its designated page.
- **Category filter:** The listing should be separated into each designated section, E.g. Men, Women and Children's should only have listing that represents their category.
- **Navigation through pages:** The user should be able to navigate through all the pages and always have a way to return to the home page.
- **Cart:** Ensuring the adding of items into the cart if they choose to purchase the item. Furthermore, they should be allowed to navigate to the cart and remove any unwanted items from the cart.

*Advanced features* to be implemented:

- **ACID transactions:** The user must be able to check out of the cart, which then displays the total amount and allow the users to purchase the item securely.
- **Relational Database:** Ensuring the implementation of a database that contains all the listing and user information that has been submitted.
- **Use of encryption (Advanced Security Mechanisms):** Implementing encryption for the passwords that are saved into the database during the sign-up process to protect against SQL injections.

To ensure a *user-friendly* website:

- **Use of simplicity designs:** Ensuring that the components are not overlapping each other, causing visual pollution while navigating through the pages. Each page needs to be simple and clear.
- **Clear use of buttons and indications:** Ensuring each feature is marked and labelled so that the user acknowledges each component and understands the functionality of the feature. Often done using labels and clear and visible buttons. E.g. Having a bold "Sell" button which makes it clear that the following components allow the users to sell their items.

To ensure the *safety* of the website:

- **Restricted modules until authorisation:** Implementing the user authentication and ensuring that features such as "Sell", and "Cart" can only be accessed after the user has logged into their accounts.

*Documentation* aspect of the website:

- **Literature review:** Including a comprehensive literature review, criticizing the research, ensuring the information to be as accurate and reliable as possible through the best sources,
- **Background research** – Include all the relevant research that has been carried out throughout the project that enhances the quality of the project.
- **Software engineering:** To depict the planning, designing, developing, and testing of the project chronologically, to demonstrate a clear thought process of the website.
- **Technical Decision making:** To ensure the reasoning behind the decisions with fully critical arguments for each decision.
- **Self-evaluation:** To reflect and analyse the final products, and evaluating alternatives that could be made, stating the advantages and limitations for each valuation.

## Chapter 2: Literature Review

### Section 2.1: Enhancing websites.

Before initiating the adhesive documentation of the process, it is essential to carry out research on the fundamentals of a successful website, therefore in this section, the main topic would be to look into the fundamentals on what makes a successful website, including how to make a website look aesthetically pleasing which encourages engagement from the users, a deep dive into the customer's point of view and lastly, the most important of all, what makes a successful website. Different references have been utilised in this section but here are the main one:

#### Source 1:

*'What makes a beautiful website? factors influencing perceived website aesthetics?'*

Noponen, S. 2017, What makes a beautiful website? Factors influencing perceived website aesthetics. Available from:

<https://jyx.jyu.fi/bitstream/handle/123456789/53109/URN%3aNB%3afi%3ajyu-201702271533.pdf?sequence=1&isAllowed=y>

The main objective of this article is taking about how important certain factors are during web development to ensure the aesthetics of a website, which includes simplicity, diversity, colourfulness, craftsmanship, unity, complexity, intensity, novelty, and interactivity. It then discussed each fact in detail, especially the importance of the aesthetic of the website, promoting loyalty. Although the source is slightly older, since it was published dating in 2017, 7 years ago, the thesis addresses the main issues that still are relevant today. Simplicity, being the most relevant factor for the ecommerce website, where having minimalistic design elements, it allows the costumers to familiarize themselves with each feature, as each feature are clarified. The thesis is scholarly standards as the research is from the University of Jyväskylä, each new statement is backed up with solid referencing, removing any potential bias regarding the topic as it's a scholar standard research.

#### Source 2:

*'What makes a successful website?'*

Scott-Parker, S. 2003, What makes a successful website?, BA (Hons) Multimedia Design and Digital Animation dissertation, Cumbria Institute of the Arts.

The main objective of this thesis discusses the main aspects of the website development phase, which investigates the branding, the potential audience that would attract users into using this website, the response time of the website, accessibility and many other factors in consideration that makes a website meaningful and successful. There is a huge emphasis on branding and how it is to carry out ongoing research, to ensure the relevancy of the website and ensuring the optimisation of the website, where the website is fully functioning, with minimal overwhelming information and elements, keeping the users engaged; ensuring the loading time of the website is not overbearing, keeping the customers engaged. This particularly is relevant to the ecommerce website in my project, as I am always looking for new ways to program my websites ensuring the relevancy of my website in the long term. Furthermore, looking at various case studies of failed and successful websites and taking it into accountability. Although the source is old, as it is dated back to 2003, the topic is highly relevant to this day as it discusses the branding, accessibility and usability which is a crucial part of developing a successful website. The source also contains many references and sources, most of which were from research and other thesis, which eliminates the sense of bias as the article provides an objective analysis.

### **Source 3:**

‘Understanding of website usability: Specifying and measuring constructs and their relationships’

Lee, Y. and Kozar, K.A. 2011, 'Understanding of website usability: Specifying and measuring constructs and their relationships', Decision Support Systems. Available from: <https://doi.org/10.1016/j.dss.2011.10.004>.

This source talks about the importances of the Website’s usability, especially in the e-commerce aspect, identifying ways to ensure that the usability of the website is optimal, which includes the use of studies, surveys, and other research methods, which ensures that the website is optimal. Although this is the least relevant source for me, it does depict clear advice such as generate a mass questionnaire on ways to improve a website, including inquiries on what a user might like in an ideal e-commerce website, and red flags to avoid. Although the release of the thesis dates to 2011, the concept and the main principals of the article still remain the same, and due to the academic level of the publication, the article is less likely to be biased and have credibility enhancement, furthermore, the thesis have appropriately discussed all the references and cited them accordingly.

### **Conclusion for Source 1, 2 and 3**

The main points of the first source discusses the importance of having an aesthetic website, and also talks about all the ways to make a website more aesthetically pleasing, and how it contributes to user’s engagement and loyalty to a website, furthermore, the scholar research was incredibly optimal, however it does not mention much about the technical aspect of the website, allowing the information to be less relevant to my project.

The second source discussed various aspects of the website, including branding, audience targeting, response time, and accessibility, discussing the importance of having up to date systems and always optimal, furthermore, researching new ways to improve the websites, however, since the source is 17 years old, the techniques and ways may be slightly outdated.

The last source discusses the importance of carrying out public surveys, allowing a proper understanding of customer's behaviour. This source depicts good practice in carrying out primary research, however, the research they are discussing can be unrealistic, especially during a smaller size project.

Overall, source 2 was the best source of information since it contained comprehensive coverage on ways to enhance a website, including ways to ensure the accessibility of a website, so that it is easily accessible to everyone, furthermore, being the most relevant topic for my project.

## **Section 2.2: Backend programming languages / Frameworks**

This section discusses about different programming language's functionalities and the roles they play in a successful backend programming language, discussing the strength and weakness of each language and how they effect the development of the backend programming.

### **Source 4**

*'Guest Editor's Introduction: Python: Batteries Included'*

[4] Dubois, P.F. 2007, 'Guest Editor's Introduction: Python: Batteries Included', Computing in Science & Engineering, vol. 9, no. 3, pp. 7-9. Available from: <https://doi.org/10.1109/MCSE.2007.51>.

This source discusses the role of python programming language in projects, highlighting the key features such as the standard libraries, third-party tools, modules that allows a wide range of tasks, especially in web development. It emphasizes on batteries included which explains the inbuilt capabilities. This source gives a meaningful evaluation of what python as a programming language carries out, however it does not have much information about Django, as a result decreasing the relativity for the background reading section. The source is quite old as it was published on 29<sup>th</sup> May 2007, however it came from the official institution of electrical and electronic engineering, as a result increasing the accuracy of the content.

### **Source 5**

*'How does the Python MVC Framework Work? What are the Benefits?'*

Pratap, M. 2023, 'How does the Python MVC Framework Work? What are the Benefits?', Blog post, 6 March. Available from: <https://supersourcing.com/blog/how-does-the-python-mvc-framework-work-what-are-the-benefits/>.

This source discusses ways that Django follows the Model-View-Template (MVT) architecture, where the model represents the database, view represents the layer of business logic execution and template represents the HTML rendering. It discusses the ease of workflow that Django pursues using mapping of user requests, furthermore, discussing more about the efficiency of



managing databases using Object-Relational-Mapping. This is the most relevant out of the Django background reading as it depicts proper information about the framework itself and gives an insight into the advantages of the framework overall. However, the information about the challenges is quite limited, which makes it difficult to evaluate whether this is the right choice in framework during the technical decision-making section. Overall, the reliability of this source is somewhat on the edge since it is a web article created by an author with little background information of the author, which may mean that some of the information could be inaccurate or biased, as well as limited lists of references. However, the recency of the paper would imply that the research is up to date.

#### **Source 6:**

*'Why Choose NestJS As Your Backend Framework?'*

Romik, T. 2023, 'Why Choose NestJS As Your Backend Framework?', Blog post, 17 August. Available from: <https://selleo.com/blog/why-choose-nest-js-as-your-backend-framework>.

This source discusses the building blocks of NestJs, where it uses the Modules, providers and controllers which organises code and handles the complexities and logics accordingly, it also discusses the companies that uses NestJS and discusses the features. It gives a good insight on the positives of the framework, ending the article with a summary. However, this source does not discuss the negatives of the frameworks, which makes the source seem slightly biased. Furthermore, the reliability of the source is unknown as it is written by an author's information is limited but similarly to source 5, it is written quite recently, around 17<sup>th</sup> August 2023, as a result it ensures the recency of the information.

#### **Source 7:**

*'FastAPI vs Flask: what's better for app development?'*

Mendes, A. & Ferreira, R. 2024, 'FastAPI vs Flask: what's better for app development?', Blog post, 12 March. Available from: <https://www.imaginarycloud.com/blog/flask-vs-fastAPI/#flaskpros>.

This source discusses the comparison between two frameworks, but it gives an especially good discussion on flask. It presents with a balanced argument about flask, stating that flask's syntax is easier to grasp and support intensive unit testing through the inbuilt servers, because of a simpler testing process. It also discusses the disadvantages of flask, where it discusses the lack of inbuilt session management and its HTML oriented which is designed more for web development rather than APIs. These balanced arguments make it simpler to determine which programming framework to choose between the other frameworks in the technical decision-making section. It is also a comparison article which compares two frameworks, as a result it eliminated the change of bias. Lastly, this article is recent as it was last updated on 6<sup>th</sup> March 2024, therefore it is consistently updating based on the patches being out. However, like source 5 and 6, it is not clear who wrote the article, furthermore, it was written by more than 3 authors, as a result there may be inconsistencies in the information.

#### **Conclusion for Sources 4, 5, 6 and 7**

While source 4 is written by an authentic institution, due to its minimal information on Django, the framework that was needed to be researched in the Background reading, it was not as relevant to the research. However, it gave a fair insight on components that Django and other python frameworks uses.

Sources 5 and 6 were similar in terms of the reliability, as they were both created by authors that did not have a clear background and they both were stating heavily positive aspects of the frameworks, which makes it sound suspiciously biased. However, they both provided with intensive research on both NestJS and Django. Furthermore, both sources came out recently, as a result it indicates that the information is still relevant till date, increasing the reliability of the source.

Lastly, although source 7 had its limitations, where there were three authors scripting the same article, resulting in possible inconsistencies, source 7 is the most reliable and the best source out of the four listed as it is a clear comparison between two frameworks, which eliminates chances of bias, as well as giving a clear positive and limitation aspect of flask, followed by a summary of each which allows the technical decision making process simpler.

## **Section 2.2: Frontend programming languages / Frameworks**

### **Source 8:**

*'What is Angular Used For and When Should You Use it Instead of Other Frontend Technology?'*

Luzniak, K. 2021, 'What is Angular Used For and When Should You Use it Instead of Other Frontend Technology?', Blog post, 23 December. Available from: <https://neoteric.eu/blog/what-is-angular-used-for-and-when-should-you-use-it/>

This source provides a good overview of Angular, discussing the features, overview, benefits, and other aspects of Angular that is necessary to comprehend before choosing the frontend frameworks. This source states that Angular overall is optimal in terms of the flexibility and efficiency in writing code, specifically useful for web development, the main uses are for appealing UI animations and other Web applications. Overall, this source is ideal for understanding more about Angular and why developers should use them, the recency of the article, last published on 23<sup>rd</sup> December 2021, allows the source to be up to date. However, like source 5 and 6, the source is written by an author whose background is not known, therefore there are chances of misinformation, and also due to its lack of disadvantages regarding the framework, allows space for bias, furthermore, it does not show a proper form of referencing.

### **Source 9:**

*'The Pros and Cons of Swift Programming Language'*

Share IT. 2024, 'The Pros and Cons of Swift Programming Language', Share IT Solutions, Available from: <https://www.shareitsolutions.com/blog/swift-pros-cons/>.

This source discusses the overall “Pros and cons of Swift”, where it discusses the recency of the programming language, resulting in a faster, and efficient language, and discussing how due to its recency it does not have a large group of developers among other discussion. This source is like Source 8 in the sense that the source does not have a known author, therefore it challenges

the reliability of the content and information given, however due to its balanced argument, where it presents the argument with both “pros and cons”, it eliminated any chances of bias.

#### **Source 10:**

*‘Performance, Modularity and Usability, a Comparison of JavaScript Frameworks’*

Ockelberg, N. & Olsson, N. 2020, Performance, Modularity and Usability, a Comparison of JavaScript Frameworks, Degree project in Computer Engineering, First Cycle, 15 credits, Stockholm, Sweden. Available from: <https://www.diva-portal.org/smash/get/diva2:1424374/FULLTEXT01.pdf>.

This source is a comparison of all the JavaScript frameworks in one document, it has much information about 3 other frameworks, Angular, Vue and React. It discusses the components of each of the frameworks, with related tests and simultaneously comparing each of the frameworks together. This source gives a great insight of all the JavaScript frameworks, now how being from the same programming language, it is vastly different. I focused on React specifically as it gave a good evaluation of the framework, how it is easily used for creating UI and how it is flexible and versatile. It also depicts the challenges of the framework and how it is a hard framework to get used to for developers who are new to the syntax. Overall, it gives a balanced view to the framework, being the most relevant out of all other sources. The source is filled with reliable references, resulting in more credibility in the source, and the recency, published on February 18<sup>th</sup>, 2020, allowing in a more recent and relevant analysis. However, the source focuses more on the comparison between other languages than the language itself, as it has less information about React (only around 3-4 sections excluding the comparison paragraphs). Therefore, it makes it slightly difficult to navigate through to scavage for more information.

#### **Conclusion for Sources 8, 9 and 10**

In conclusion, both sources 8 and 9 were similar in terms of reliability, as they both have been written by unknown authors, furthermore, both sources had a lack of citation and references, decreasing the credibility of the source. Although Source 9 had a balanced argument, stating both positives and negatives of Swift, source 8 only depicts Angular in a positive light, raising suspicions on the biasness. However, the recency of both sources increases the relevancy since the information is up to date for the current frameworks. The best Source however was source 10, as it accurately gives a balanced view and displays pages of citations, despite the lower amount of information on React.

### **Section 2.3: Relational Database Management Systems**

#### **Source 11:**

*‘What is PostgreSQL? Introduction, Advantages & Disadvantages’*

Peterson, R. 2024, 'What is PostgreSQL? Introduction, Advantages & Disadvantages', Guru99, Updated 16 March. Available from: <https://www.guru99.com/introduction-postgresql.html>

This source gives an overview of the PostgreSQL, where it discusses the advantages, disadvantages in PostgreSQL. It discusses all aspects of the database, including its history and the key features, including how it allows administrators to build a tolerant free environment and

it supports location-based services. Overall, this source provides all the basics and key information a user should know before using PostgreSQL, keeping everything concise and displaying a comparison table against other database management systems. Similar to other sources, the lack of citations and references makes the source slightly unreliable as it is unclear whether the information in the source presented is accurate. However, the recency of the source, where the last update was made on March 16<sup>th</sup>, 2024, means that the information presented is relevant and still up to date, furthermore, the balanced arguments and the clear comparison to other database management systems implies that the source is not biased.

#### **Source 12:**

*'The Basics of Relational Databases Using MySQL'*

Blansit, B.D. (2006) 'The Basics of Relational Databases Using MySQL', Journal of Electronic Resources in Medical Libraries, 3(3), pp. 135-148. DOI: 10.1300/J383v03n03\_10. Available from: [https://doi.org/10.1300/J383v03n03\\_10](https://doi.org/10.1300/J383v03n03_10). Published online: 03 Oct 2008.

This source provides intensive research on every aspect of MySQL, starting with the simple definition of what the RDMS is to be displaying how to create classes tables. It shows the functionality of most of the features, clearly displayed with pictures from the MySQL query browser. Everything is accurately cited and provides a balanced argument on the benefits and limitation of using this RDMS. This article has been published by an authentic source, Electronic Resources in Medical Libraries, as a result enhancing the credibility of the source. However, upon further inspection, MySQL's query browser looks drastically different from the pictures that were displayed on the book, furthermore, the source is old, as it was first published on the 3<sup>rd</sup> of October 2008, therefore, there is a possibility that the information displayed here is outdated or no longer a feature regarding the MySQL.

#### **Source 13:**

*'What is SQLite? An overview of the relational database solution'*

Unknown author. 2023, 'What is SQLite? An overview of the relational database solution', 27 June. Available from: <https://www.ionos.co.uk/digitalguide/websites/web-development/sqlite/>.

This source provides an extensive information about SQLite, discussing about how the database works to the security of the database, covering all the aspects giving balanced arguments to why developers should use the database and the specific type of projects the database is optimal for. This source is relevant to my project as it fully analyses the all the aspect of the database, as a result enhancing background reading. Its also recently published, implying that the information displayed on this source is still relevant. However, due to its lack of citations and no clear author, questions the credibility of the information on the source.

#### **Conclusion for Sources 11, 12 and 13.**

Source 12 was the best source out of all these sections as it carries out in-depth research on the database, showing clear analysis of MySQL with plenty of visual demonstrations, as well as having authentic sources, with many citations, increasing the credibility of the source. It was also published by an authentic author, as it was published through "Journal of Electronic

Resources in Medical Libraries”, as a result it was likely gone through extensive peer review to ensure the reliability of the source, despite being slightly old. Source 11 and 13 on the other hand, while they both provide with in depth research, the lack of citations and authentic authors makes the sources unreliable and increase the likelihood of bias. As a result, Source 12 is the best source.

## Section 2.4: Architectural paradigms

### Source 14:

*'Object-Oriented Analysis & Design'*

thesunpandev 2024, 'Object-Oriented Analysis & Design', GeeksforGeeks, Last Updated 14 March. Available from: <https://www.geeksforgeeks.org/object-oriented-analysis-object-oriented-analysis-design/?ref=lbp>.

This source discusses the Object-Oriented Architecture, discussing the analysis and the design, where it states the important aspect of OOAD, how it has reusable solutions and the use of UML and Use cases describes the user to understand the requirements. It also discusses the benefits and limitation of using OOAD and accurately concluding with real word applications that may use Object Oriented Architecture. This source is relevant to my background reading as it gives an insight into the architecture, allowing key information, nicely formatted. It is also from a reliable source as Geek for Geeks is a great website to refer to for programming/computer science enquiries. The article also gets updated regularly as this article was last updated on the 8<sup>th</sup> March 2024. Despite all these benefits, the article does not present with any citations or references which questions the accuracy of the source.

### Source 15

*'Service-Oriented Architecture (SOA)'*

Barney, N. & Nolle, T., 'Service-Oriented Architecture (SOA)', TechTarget, Available from: <https://www.techtarget.com/searchapparchitecture/definition/service-oriented-architecture-SOA>.

This source gives an accurate in-depth information on what Service-Oriented architecture is, where it discusses the ways that the model allows communication across different platforms and how it benefits business. It is relevant to my project due to its amount of intensive research and information that enhances my background reading. The clear balanced arguments, furthermore, eliminates any chances of bias. However, due to the lack of active date of which the source has written, there is a chance that the source may be outdated.

### Source 16 and 17

*'The Model View Controller Pattern – MVC Architecture and Frameworks Explained'*

*'What is MVC? Advantages and Disadvantages of MVC'*

Hernandez, R.D. 2021, 'The Model View Controller Pattern – MVC Architecture and Frameworks Explained', FreeCodeCamp, April 19. Available from: <https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/>.

Unknown author. 2016, 'What is MVC? Advantages and Disadvantages of MVC', InterServer Tips, Posted on October 28. Available from: <https://www.interserver.net/tips/kb/mvc-advantages-disadvantages-mvc/>.

Both Source 16 and 17 displays accurate information about MVC, where they both highlights the advantages and the disadvantages of Model-View Architecture, explaining the components and how they work together, allowing a faster development and collaboration amongst the developers. However, Source 16 provides a more in-depth explanation of the MVC pattern, breaking down each component and showing real code examples, whilst source 17 only discusses the overview, benefits, and limitations. Source 16 is from a reliable source as 'Freecodecamp' is the go-to website for most computer science/ programming related inquiries. However, due to the lack of references and citations, questions the credibility of the information.

### **Conclusion for Sources 14, 15, 16 and 17**

Out of the 4 sources, the best source is source 16 as it gives a clear and in-depth explanation of the MVC pattern, breaking down each of the components allowing a better understanding of the topic. Furthermore, it gives a balanced argument on MVC, therefore, eliminating potential bias in the information. The source also comes from a reliable website, unlike the other sources as a result it enhances the credibility of the information. Whereas despite having good selection of information, the other sources did not have references, or were not written from a known source, as a result source 16 was the most impactful source out of the rest.

## **Chapter 3: Background Reading**

Background Reading is the most important part of the project development as it allows comprehensive research on each component, which ensures the best possible materials and outcome of the final project.

### **Section 3.1: State of the art Web Development**

This section consists of extensive research about three programming languages that is best suited for backend, frontend, and the database.

#### **3.1.1 Backend:**

##### **Django:**

Python is a very popular and flexible programming language which is widely used for website development due to its readability, simplicity and a wide range of frameworks and libraries that it contains. One of the frameworks which is most popular in the Website development in Python includes Django. Django ensures that “batteries-included” feature which implies that the users do not have to use up much time and effort to set up the development environment installing

extra libraries and dependencies. It instead focuses on building the application and uses many in built features, such as ORM or other template engines which supports many databases. [4] Django MVC (Model-view-controller) uses a clean layout that uses pattern to create a maintainable website application, the main complements being the Model, View, and controller, which essentially is to implement the user interface, data, and control logic. [5] Django's ORM (Object-Relational-Mapping), allows easy backend functionalities such as interacting with the database syntax instead if dealing with raw SQL queries.

#### **NestJS:**

NestJS is an Open-source, progressive Node.js framework that allows developers to create demanding backend systems, making it testable, scalable, supporting databases like PostgreSQL, MongoDB, MySQL. Being heavily influenced by React and Vue, NestJS lies in their dependency injection system, which allows to create a loosely coupled program, making it easier for developers to maintain applications. Nest JS uses modern JavaScript and TypeScript features which allows developers to modify the behaviour of the functionality by using controllers, services, and middleware. The syntax allows routing and request handling easier, allowing an easier way of developing RESTful APIs. It has a powerful command line interface, which boosts productivity and easier to develop the backend sided website. [6]

#### **Flask:**

Flask is a light and versatile web framework for python, and it's incredibly popular for its simplicity, flexibility, and its ease of use. It allows developers to prototype and develop application with minimal code, making it more efficient, providing freedom to choose which libraries use developers would like to use unlike other frameworks, enhancing flexibility and customization. Flask's routing system is quite straightforward, allowing the developers to clearly use URL routes and other functionalities, making easier to map HTTP requests to an endpoint and allows ease of handling data. One of the prominent features of Flask includes its micro-framework which allows developers to be low-level and customise layers of abstraction, allowing a greater performance. Furthermore, Flask supports unit testing, allowing users to simulate conditions and test their application's functionality. [7]

### **3.1.2 Frontend:**

#### **Angular:**

Angular frontend framework is a wide choice among many website developers, where it features a component-based architecture, where each components handled a distinct part of the user interface, encouraging reusability of code. Angular was created using Typescript as a result it uses modern approaches the management of a complex UI, allowing it to split the application in a way that makes it easily maintainable and testable. This is done by breaking the application into smaller testable features which allows users to test each item in detail, ensuring there are no faults. Angular allows users to interact with components, allowing them to get data, making the code more usable and isolated. It also supports wide range of features such as templating, two-way binding, modularity, RESTful API access, dependency injection and AJAX. [8]

#### **Swift:**

Swift a popular choice amongst many programmers due to its ability to create high performance and secure applications. Swift also consists of modern syntaxes security features and quick performances which is praised for reliability and the consistency of code without clutter. Swift is said to be faster than Python 2.7 and includes LLVM tools which allows the assembly code to comply to machine code. It also runs machine code which accelerated the development through value types, storing the header (.h) and implementation (.m) files in a single swift file. [9] However, due to its recency, it had fewer number of developers who use swift, as a result, less information about the debugging process, which would result in a slower development. Swift may also have various compatibility constraints where project carried out in older versions of swift would not be able to run in Swift 5.0, for example.

#### **React:**

React is also a popular framework amongst many developers, due to its ability to write intuitive UI components within the JavaScript code. React's wide community with several weekly downloads ensures the ease of debugging through research as well as its efficient in handling, fetching, and rendering of data. The dynamic form and iteration also allow ease of rendering components with iteration over arrays and Object. However, react has many complex syntaxes which makes it challenging for developers who may not be familiar with the syntax. [10] Additionally, its inability to change the elements, instead having new elements that are rendered every time may lead to inconsistencies in management.

### **3.1.3 Database:**

#### **PostgreSQL:**

PostgreSQL is an open source and enterprise-class database management system that supports both SQL and JSON, for both relational and non-relational queries, supporting advanced datatypes and optimisation. It assists developers to create applications allowing admins to build environments through the protection of data. PostgreSQL also can run dynamic websites as LAMP stack option, furthermore, being low maintenance and beginner friendly. [11] However, despite all these features, PostgreSQL is designed to adapt to large scale applications, as a result it is slower than other database management systems.

#### **MySQL:**

MySQL is widely used and is one of the most popular relational database management systems across different applications. It is open source and has a large community of users who share knowledge with each other. It has excellent performance and is highly optimised, being compatible with most operating systems and the scalability allows the distribution of workload through the support of cluster and replication. [12] However, MySQL is complex due to its configuration and tuning where parameters like buffer sizes and storage engines can require advanced knowledge, making it difficult for beginners who are not familiar with such features.

#### **SQLite:**

SQLite is a lightweight and easy to use relational database that is ideal for smaller projects, having a small footprint, which takes up little disk space and memory as a result it is ideal for



small scale application and compatible with multiple devices with smaller specifications. It can be easily integrated with nearly every framework and programming languages; however, its scalability poses a huge challenge as its not suited for multiple users accessing the database simultaneously due to its small project limitation. Furthermore, it has limited functionality, where it does not offer centralised control over the file-based databases, as a result developers may need extra tools to manage the database. [13]

## **Section 3.2: Architectural Paradigms**

Architectural design plays an important role in the software design process, displaying its foundation for the structure and the behaviour of the system. This defines the basic organisation of a system, that is within its components and their relationship with the environment. Architecture designs provides frameworks and guidelines the allows an ideal development and maintenance which helps to manage the complications and meet the requirements. Furthermore, different paradigms, fundamentally to find different approaches that build solutions to a specific type of problems. The architectural paradigms that will be addressed would include the object-oriented architecture, service-based architecture, and model-view- controller.

### **Section 3.2.1: Object-Oriented Architecture**

Object oriented software methods has been widely used across different developments, as its approach to programming solves problems by using all the computations carried out using objects, which are component of a program that knows how to perform certain actions, interacting with other elements of the program. Objects are essentially data structures in memories that get manipulated accordingly by software or hardware systems through user interface. Objects have descriptors which provides information about the type of the object.

OOA analyses a problem domain that represents using objects and their interactions and then designing a scalable solution, helping systems that are easy to understand maintain and extend the system by organising the functionality so that it is reusable. [14]

The importance of object-oriented architecture involves modelling real-world object a software with methods that represents certain behaviour of an object using the design and implementation approach. It also uses design patterns that helps developers create a more maintainable and sufficient software, using diagrams such as UML and Use Cases which represent the interaction that different components have with the software, and to further understand the requirements.

A fundamental aspect of object-oriented programming stresses the creation and interaction of objects, which comes with various benefits, including the modularity, the ability to hide information and promotes code reuse. However, a few challenged include the complexity as each OOA needs to be modelled and managed, and due to its complexity, it is presented with a steep learning curve, [14] where it requires a strong understanding to grasp the concepts and techniques.

Some real-world Applications for Object Oriented Architecture include Ecommerce since ecommerce require complicated uses of online platforms such as profiles, listing of items, shopping carts, payment, and processes.

### **Section 3.2.2: Service-oriented architecture:**

Service-oriented architecture is a development model in which allows services to be reusable, allowing communication and formation of new languages across multiple platforms and providing self-contained units to carry out tasks. Software-oriented architectures allows the exchange of information between different services, ensuring the scalability of applications while reducing costs of development, which is ideal for applications which require many systems that require interactions between one another.' [15]

Service oriented architecture simplifies complex systems so that can be used by other service consumers through the reusable systems, which can be used to create new applications carrying out specific tasks such as service's input and output, as well as communication required. [15]

Service-oriented architecture uses systems that involves the client staying independent of a system however still communicating with the various services, allowing a complicated software that can be used as a single unit through other applications.

Service-oriented architectures allows reusability amongst various applications, providing standard approaches to carry on business process consistently, simultaneously ensuring the security of the business. Furthermore, the independence of the services, it allows ease of update and modifications, without effecting other services, allowing an ease of maintenance. It is also highly available, as a result it can be accessed to whoever requests.

However, due to the ability of managing multiple services, it is complex, especially when dealing with a high volume of messages. Furthermore, input of parameters for each service can decrease the overall performance due to its load time and response time being increased.

### **Section 3.2.3 Model-View Controller:**

Model-View controller is an architectural patter that transforms complicated applications into manageable processes through the separating frontend and backend code into separate components which allows the ease of management as it keeps them separate and changeable without interfering with each other, which means it can be easily updatable, modifiable, and easy to debug. [16]

Model is the backend which has all the data and the logic stored, View is the frontend which has the main graphic and user interface, and Controller is the controls of how the data is being displayed in the application. The controller usually initiates the user interface, and also modifies the data from the Models, The View pulls or updates the data through setter's getter and event handles, lastly, Model gets modified via setters and getters from controllers. [17]

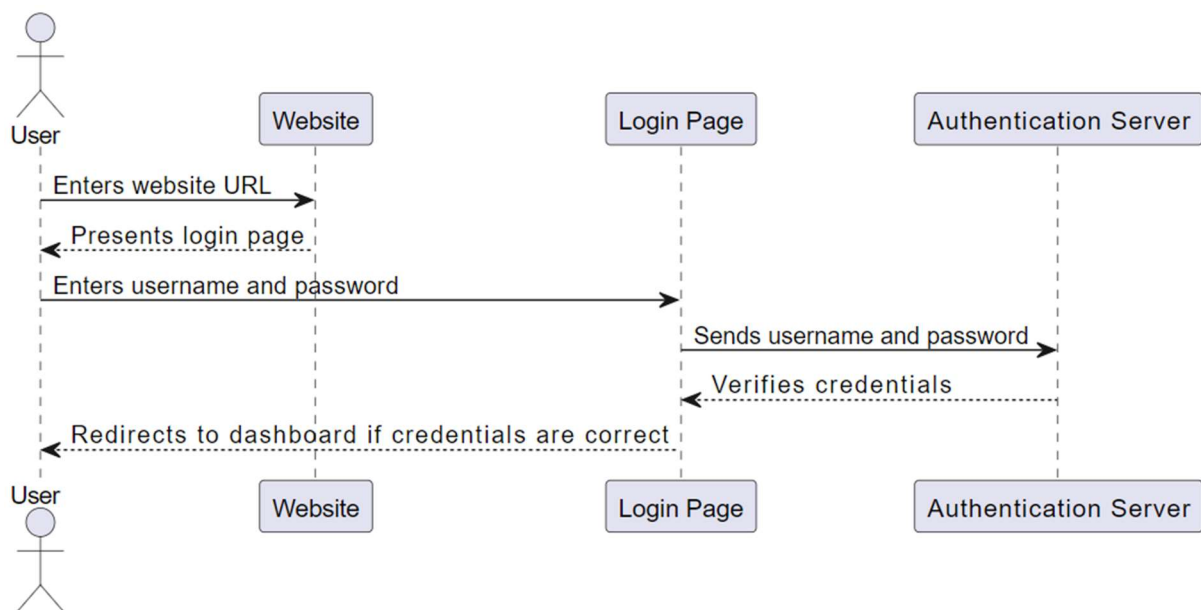
Although the separation of the structure, it allows a faster development as it allows multiple developers working on different modules separately, however due to its factor, it is complex as the architecture is difficult to grasp for users who may be beginners and are not familiar with the

separation aspect. Furthermore, the strict rules can be limiting especially regarding the functionality of the application.

### Section 3.3: System Use Cases

Before creating a e-commers website, it is essential to evaluate other ecommerce website first and then plan out the website concept of the websites. Despite the difference in the websites, the websites all hold a common ground. Below there are various finding in the form of system use cases that are required for a successful e-commers website:

#### Section 3.3.1: Login



In this use case, it demonstrated how a user would log into a website, it would first navigate into the website by using URL, it will then be presented with a login page. The user would then enter their username and password in the field that was provided. The credentials will then be sent to the authentication server, there the server verifies if the username and password match. If the credentials are correct, the server send a confirmation back to the users allowing them to successfully log into their account.

#### Section 3.3.2: Sign Up



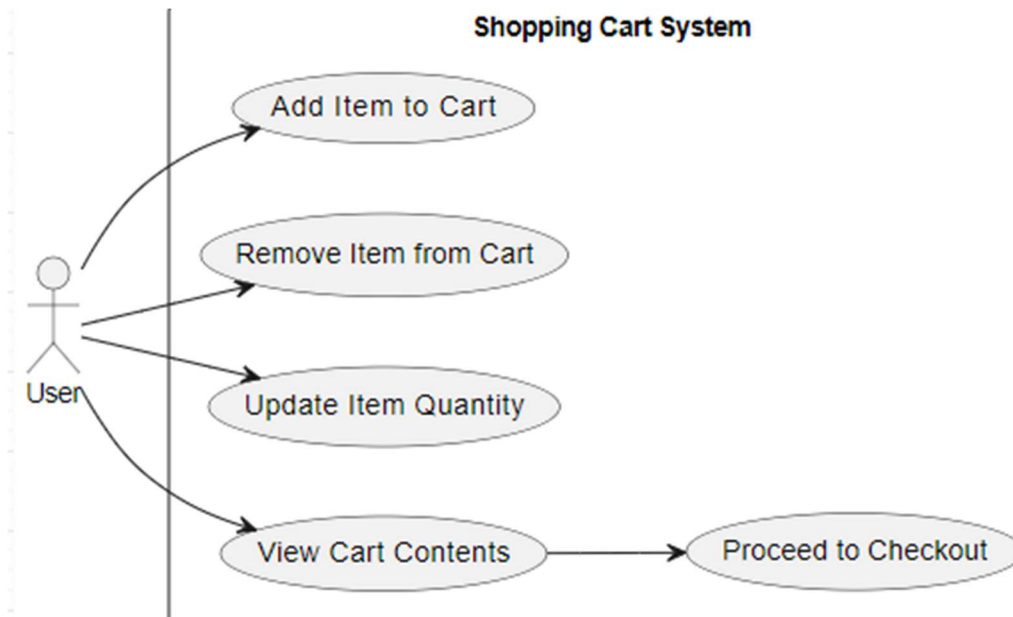
In this use case, it demonstrated how a Sign-up system works, how a user enters their username, email and password, ensuring that the email's formatted where it must include an "@" sign. It then checks against the database for availability, and then stores the information into the database, proceeding the user into successfully creating their accounts.

#### Section 3.3.3: Listing Items



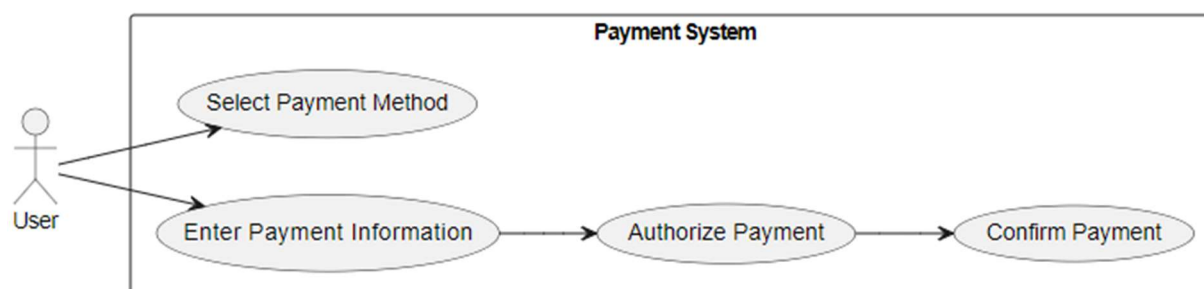
In this use case, the user enters the details of the items, such as the title, description, category, and images, they then set the price and the quantity of the items and submits for listing, which then can be viewed in the front page.

#### Section 3.3.4: Shopping Cart



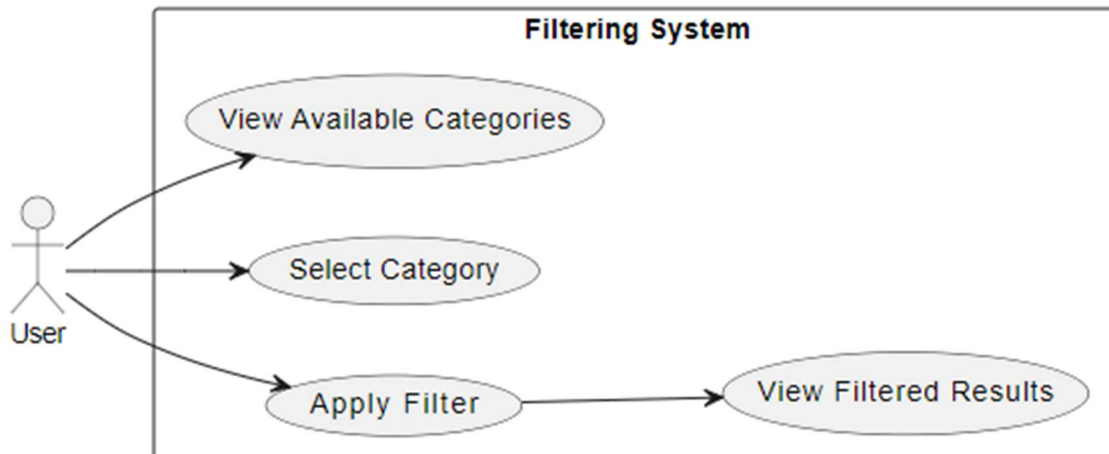
This use case depicts the human interaction between the users and the shopping cart system, where the user adds an item of their choice in the cart, the user can also remove the item from the cart, update the quantity, view the item in the cart, and then checkout once the user is content.

#### Section 3.3.5: Payment



This use case demonstrates the payment process, where once the user checks out of the shopping cart, it then selects their payment method. After selecting their payment method, the payment gets authorized and then the user confirms the payment.

#### Section 3.3.6: Filtering



This use case demonstrates the filter process, where the user views the available filters, such as “Men, Women, Children” and then they select a category and then apply the filter, this then displays all the listings that is based on the selected category.

### Section 3.3.1: Search feature



This use case demonstrates the search feature, where the user will enter a search prompt into the search bar, it will then display all the titles of the listing based on the first character of the search prompt as “suggestions”, and the user can either click on the suggestion, or they can perform the search, which then will display all the results based on the search query.

## Chapter 4: Software Engineering Process

### Section 4.1: Planning

It is necessary to have a proper planning before carrying out the project as it will help meet the deadlines and outlines the requirements, making it easier to carry out both the documentation and the development process.

The requirements for my projects are as follows:

- **User friendly and Aesthetically pleasing UI:** The UI must be simplistic, both visually and functionally, to ensure simplicity and avoiding visual pollution.
- **User Registration and Authentication:** It is required to have a user registration and authentication page, where the user can create their own accounts, by entering their username, email, and password, and then log into their accounts securely.
- **Relational database:** There should be a database in place that contains the product and user information based on the listed products and the sign up.
- **Encryption:** The password saved into the database must be encrypted to protect against SQL injection attacks’

- **Restricted modules until Authorisation:** Features such as “Cart” and “Payment” must be protected behind a login wall, where the user must create their accounts and login to access these features.
- **Listing Products for sale:** It is required to include a “Sell now” button, which redirects the user to a form, where they can enter the title, description, category, price, and the image of the item they would like to list, furthermore, there should be a “List” button which puts the item out for listing.
- **View Listed products:** The listed products should be displayed in the home page, furthermore, it should be clickable, where when the user clicks on the product, it must redirect them to a different page where it shows the product.
- **Category filter:** The listing should be filtered based on their category. For example, if the category of the listing is “Women’s” clothing, then the listing should be filtered so that it is displayed on the Women’s page.
- **Navigation through pages:** The user should be able to navigate through each page, making sure that there are always ways for the user to return into the home screen without having to press “go back to the previous page” button on their browsers.
- **Search function:** The user should be able to search their items based on the first letter they input; the suggestions must display all the listing based on the first letter of the search query. The user should then be able to click on the title and be redirected to the item page.
- **Cart:** The user should be able to add and remove items from their cart, furthermore, it should display a total price on the cart page.
- **Acid transaction:** There should be a payment section displayed, where once the user checks out, it redirects them to a payment page, where they can either pay through PayPal or enter their card details.

Below is the timescale of when to get these requirements done by:

#### Term 1:

##### Week 1 & 2:

- Research React.js and Flask
- Collect materials that would be useful for the implementation of the websites.
- Start making a rough design on how the website is going to look like. E.g. The placement of the buttons, the window, the layout.
- Create user stories and UML diagrams of the webpages.

##### Week 3 & 4:

- Setup the frontend and the backend.
- Create the frontend layout aesthetically, e.g. Navigation bar, Men, Women, Children, About us page.

##### Week 5 & 6:

- Implement a relational database.
- Create a navigation bar, search bar, and implement button functionality.

- Enable navigations between pages.

**Week 7 & 8:**

- Create a form for the sell page, which takes in the title, category, price, description, and image.
- Allow data transmission between the UI and the database, so that it stores the listing data.

**Week 9:**

- Display the items listed in the “Sell” page into the home page.
- Allow category filtering where the listing is places in their own categories.

**Week 10 & 11:**

- Enable search functionality, where the users can search for their items.
- Documentation and prepare for presentation.

**Term 2:**

**Week 1 & 2:**

- Allow listing interaction where the user can click on their posts.
- Create the listing page, where the page is generated based on the Item’s ID.

**Week 3 & 4:**

- Add an “add to cart button” in every listing card across all the pages.
- Add a basket system to the website which allows multiple items to be stored.

**Week 5 & 6:**

- Add a Sign-up page where the user must enter their Username, Email and Passwords
- Add a Login page, where the user must enter their Username and Password to log in

**Week 7:**

- Adding encryption so that when the user enters their passwords, it is encrypted.
- Add a checkout page, where the users can checkout and are presented with a payment system, where the users can pay for the items that they would like to purchase.

**Week 8 & 9:**

- Add restrictions to the websites so that it can only be accessed once the user log in.
- Add a log out button replacing the login in button once the user is logged into their website.

**Week 10 & 11:**

- Finish the documentation.

## Section 4.2: Design

### Section 4.2.1 Competitor's analysis

Before starting the design process, it is necessary to investigate competitor's websites and analyse their choice in layout. Due to the E-commers aspect of my project, I have investigated "Vinted" (Vinted.co.uk) and "Ebay" (Ebay.co.uk).

#### Vinted:

When exploring Vinted's website, one striking feature that appeared is its inviting homepage, especially since when the users click on the homepage, it is presented by a bold statement that states: "Ready to declutter", corresponding to another button directly below it, linking the users to their listing form, which allows the users to list their items easily. This approach to the website is incredibly effective as it shows a clear purpose, since Vinted is a clothing site which lets users resell their old clothing, the use the bold statement helps the users feel included and persuade them into listing their items. Moreover, the choice of the background behind the statement features women holding a shirt, this complements the colour scheme of the website as well as keeping to the same theme of the website, selling clothing.

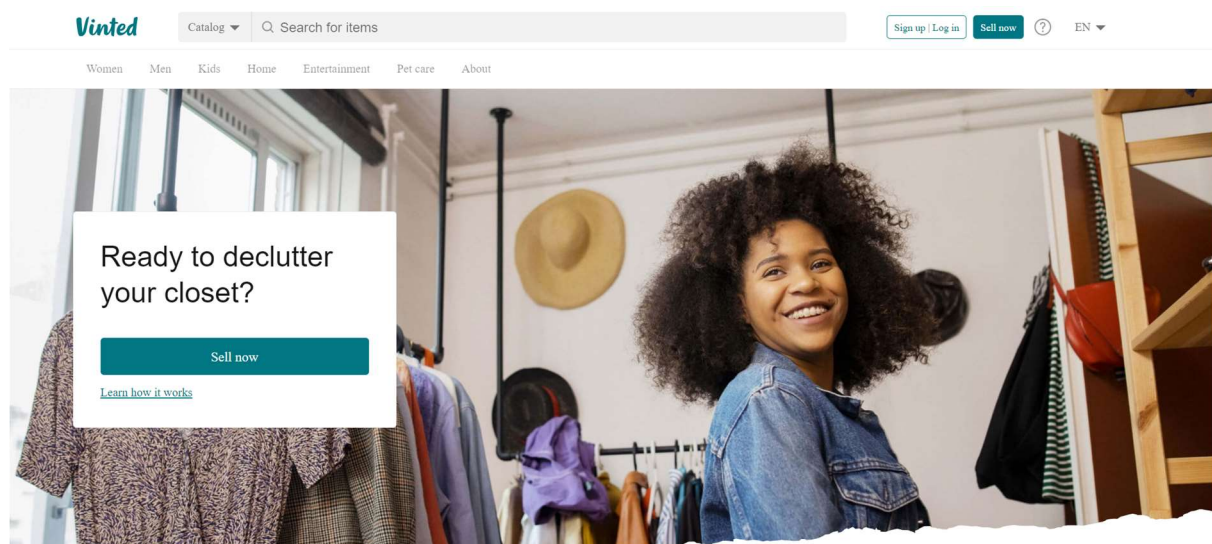


Figure 1 from <https://www.vinted.com/>

#### Ebay:

When first vising the Ebay's website, users may be overwhelmed by the overload of information. The homepage seems cluttered with images, texts and information that does not seem necessary for the users. Furthermore, the navigation bar does not seem to be attracting attention, as it seems hidden from the rest of the page, especially the "hello, sign in, prompt that the user encounters, which is presented with a hyperlink in a simple aria text. However, eBay's placement for the search bar determines the priority of the platform's functionality and encourages the users to search for items they need, as the search bar was beneath the navigation bar. Despite the page being overwhelming, it encouraged users to still use the site and purchase items.



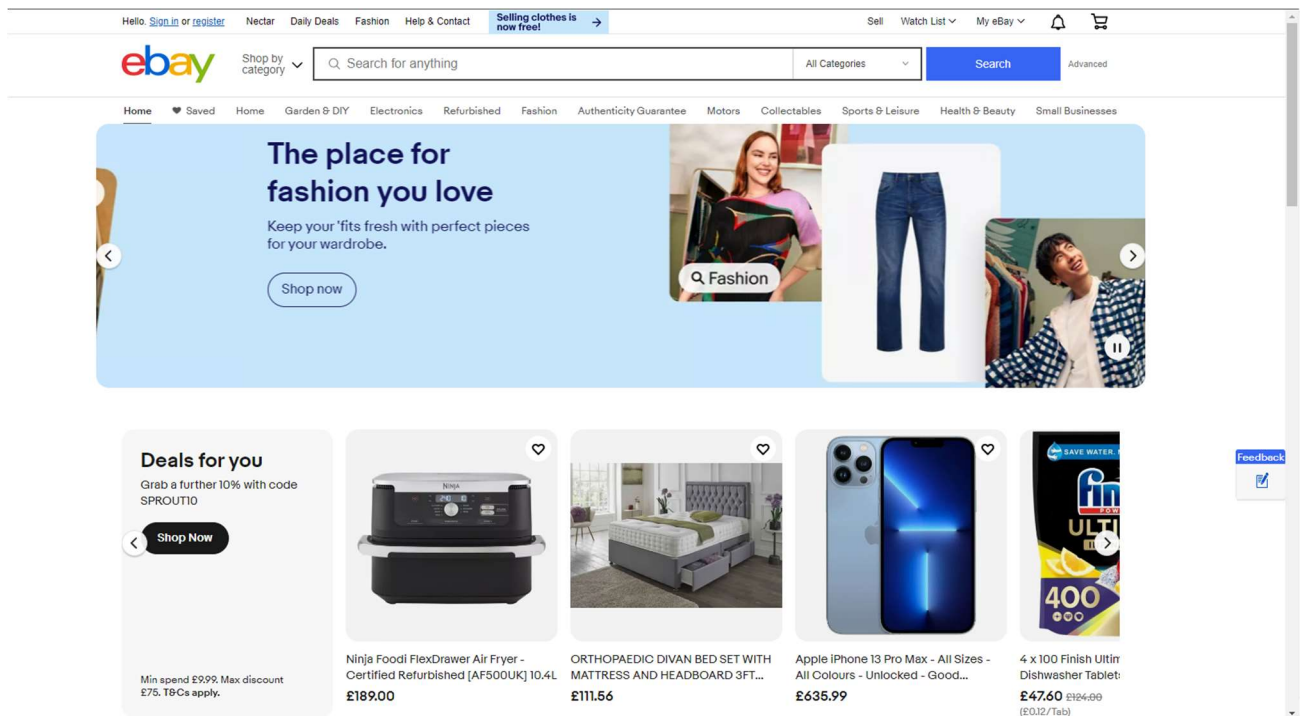


Figure 2 from <https://www.ebay.co.uk/>

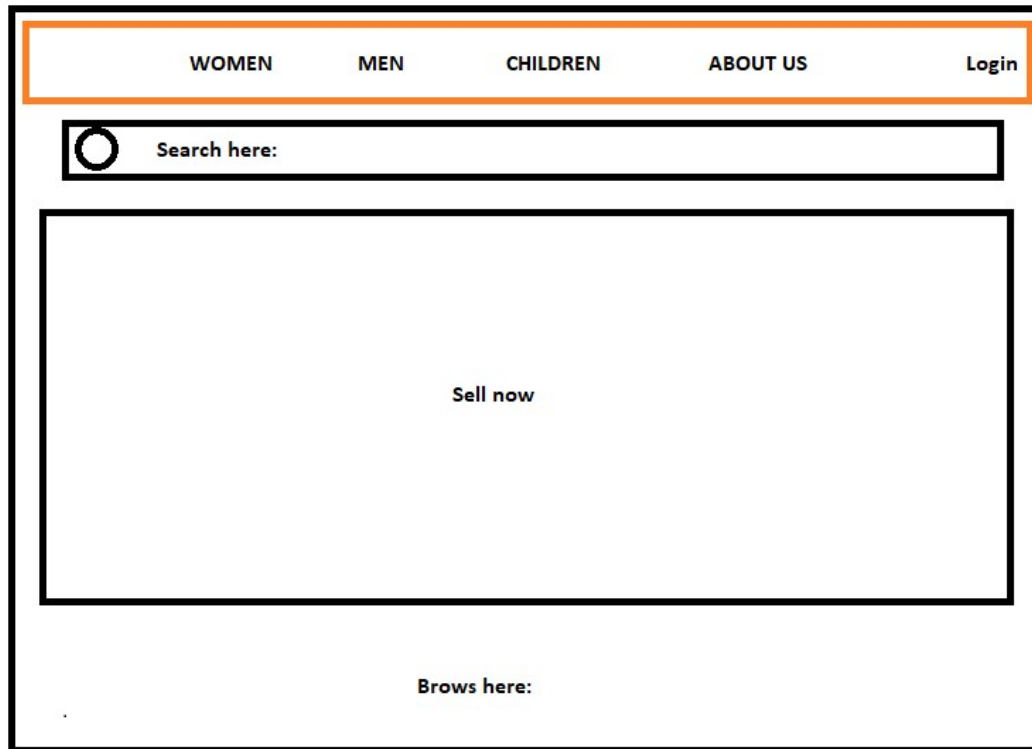
## Section 4.2.2: Low Fidelity Prototype

### Home page:

Based on the competitor's analysis, I have incorporated a similar feature into my website's design. On my homepage, I have integrated a background with a consistent colour scheme and have used a looped video which are of users cycling through the wardrobe of clothing, to enhance familiarity within the website. I have also placed a bold "Sell now" button right in the middle of the webpage, allowing the button to be the first thing the user's see when they click on the website, encouraging them to sell their items.

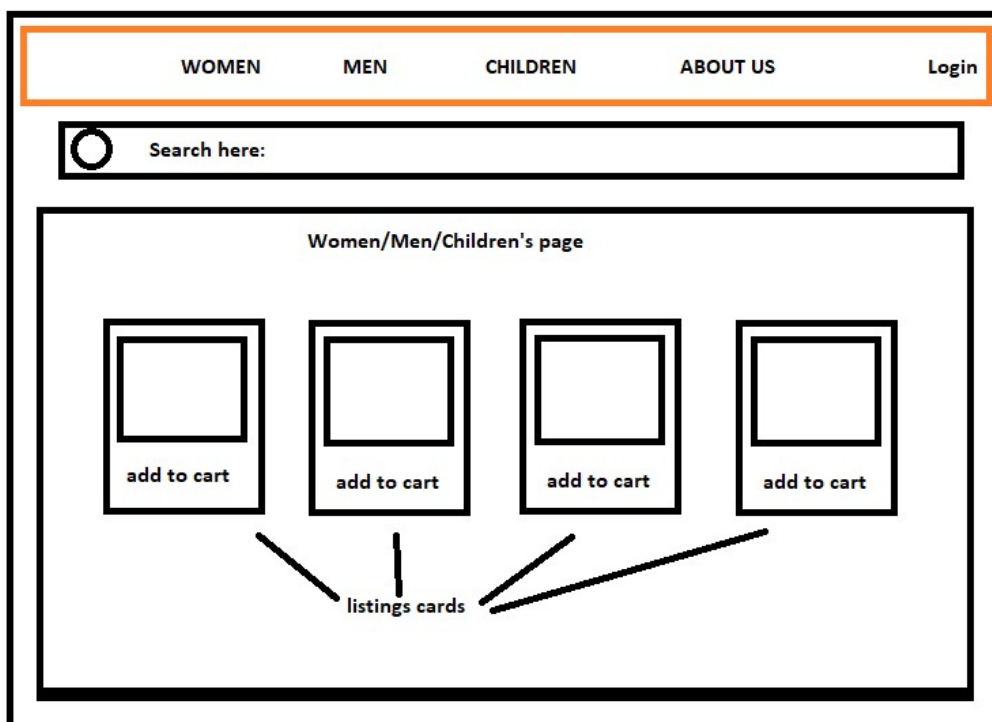
I have also decided to take a simpler approach to my website by implementing a navigation bar that is bold and visible clearly, accessible to the users, and presenting each button in a distinct manner. Additionally, I made the navigation bar is larger in size, enhancing the visibility and the usability, ensuring users can see each button avoiding overwhelming visual pollution.

Lastly, I have put emphasising the search option by positioning it underneath the navigation bar, and included the browse option beneath the search function, allowing the users to scroll through each product and select the products that they prefer the most:



### Men's, Women's, and Children's page:

In the Men's, Women's, and Children's page, I have maintained a consistent layout to home page where the navigation bar is at the top and so is the search bar. The contents in the Men's, Women's and Children's page are the same, except the listing. The listing will be displayed based on the categories the user has listed.



### Sign up/ Login page:

Similarly, I have kept the login page the same, ensuring the consistency of the website, with the like the home page, where I have kept the same colour scheme and the navigation bar. However, I have removed the search bar as the users should not be able to search something in the login page.

The wireframe shows a navigation bar at the top with links: WOMEN, MEN, CHILDREN, ABOUT US, and a Login link. Below the navigation bar is a central box titled "Login / Sign Up". Inside this box are two input fields labeled "username" and "password", followed by a "Login/ Sign up" button.

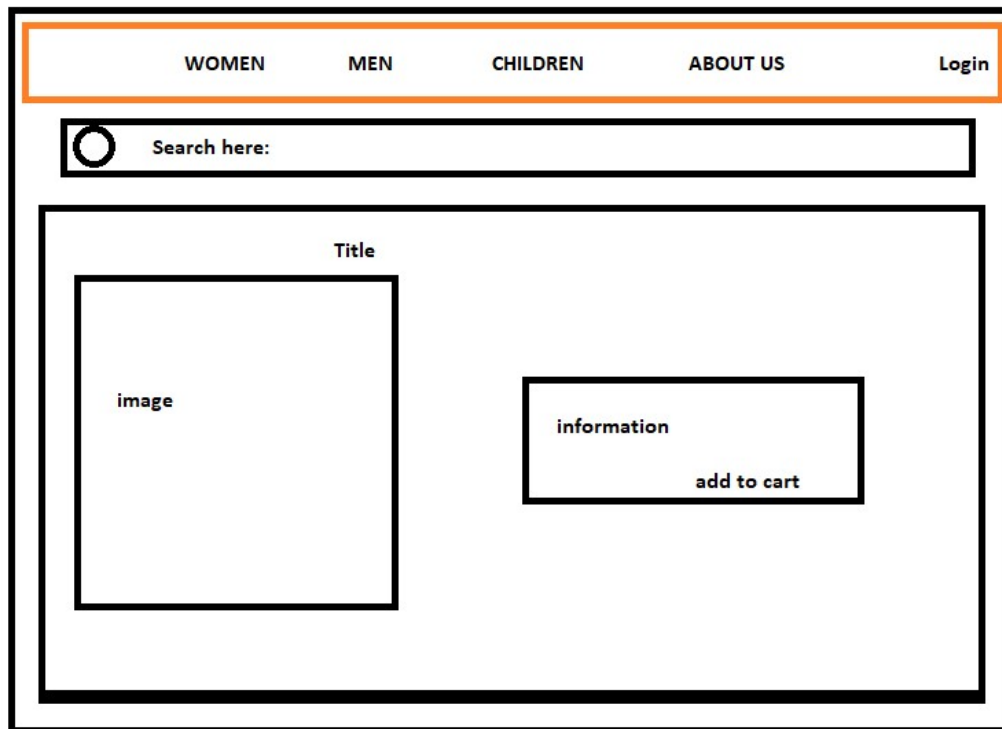
### Cart:

I have made sure that the items in the cart are smaller and concise, so that it only displays the title and the price. I have also made sure to add a remove button. However, similar to the login page, I have removed the search feature.

The wireframe shows a navigation bar at the top with links: WOMEN, MEN, CHILDREN, ABOUT US, and a Login link. Below the navigation bar is a central box titled "Cart". Inside this box, there are three items, each represented by a small square icon and a "remove" button. To the right of these items is the text "items in the cart:". Below the items is a "total:" label and a "Checkout" button.

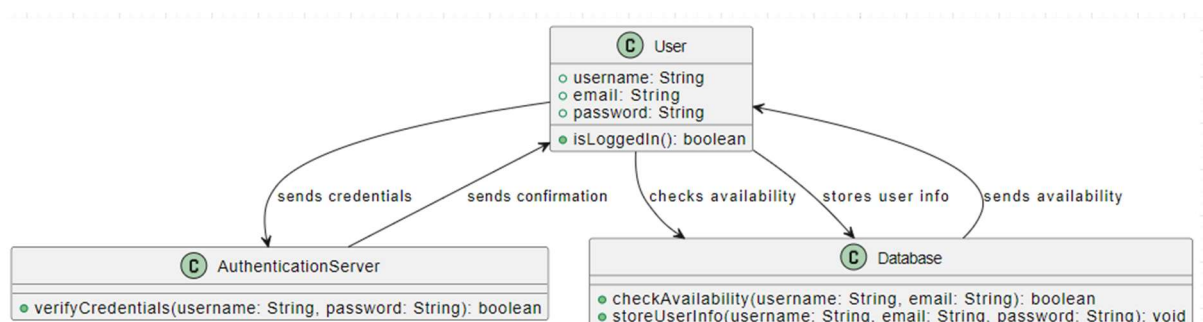
### Viewing listing:

Like the other pages, I have kept these pages consistent, where I ensured that the navigation and search bar are still present. I also made sure they have the image section larger than everything else, so that it draws more attention to it.

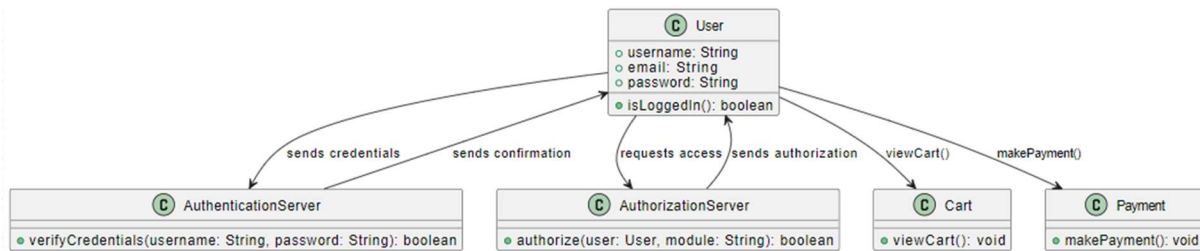


### Section 4.2.3: UML diagrams

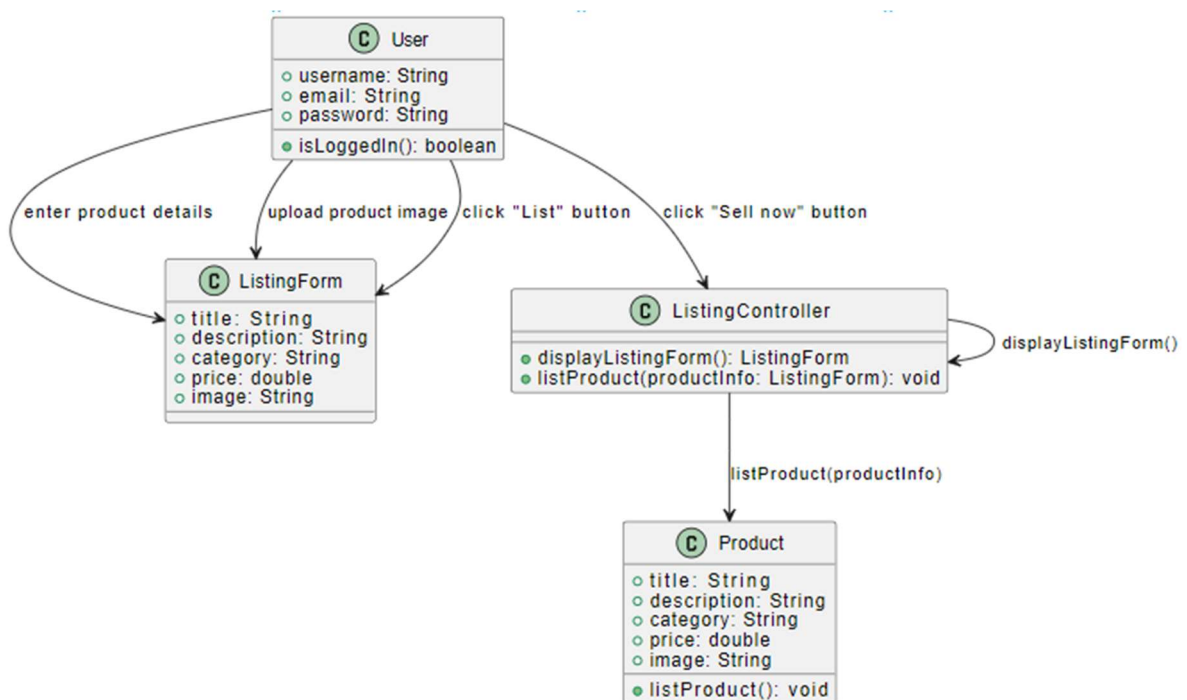
#### User Authentication:



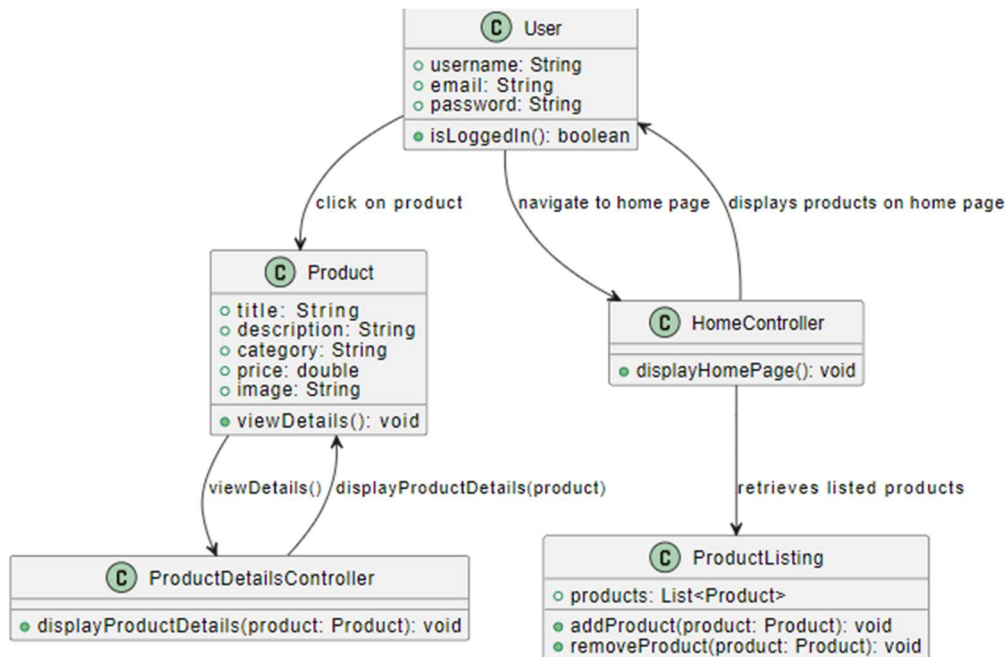
In this diagram, it shows those who have the attributes “username”, “email” and “password”, which then gets sent to the Authentication Server, which validated the credentials, and then sends a confirmation to the users, allowing the user to log in. To sign up, the user enters the user information, which then checks for availability, i.e. if the username is taken. And then it returns back the availability message.

**Restricted modules until Authorisation:**

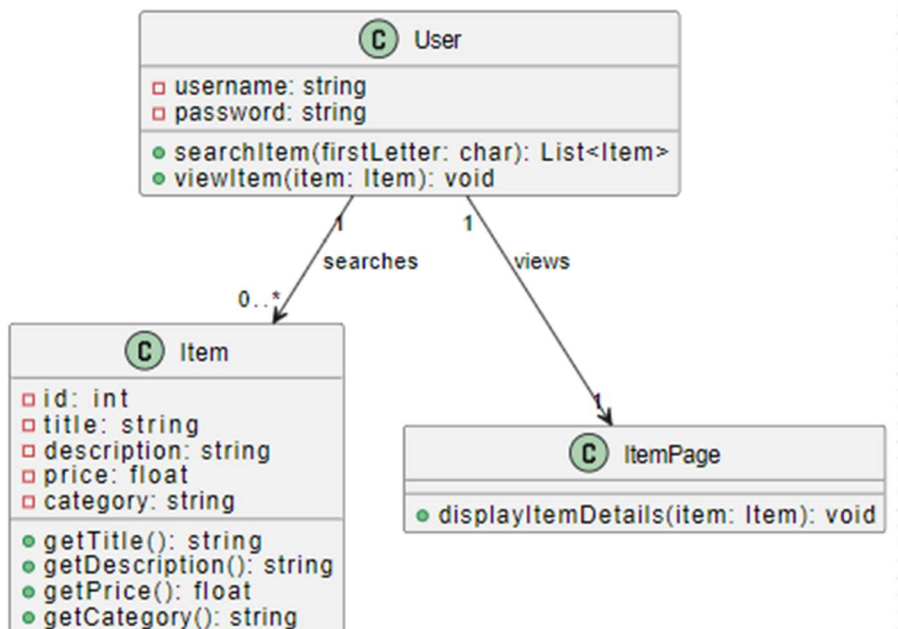
This UML shows that once the user is authenticated, by entering their credentials, and successfully logged into their accounts, they are able to view “Cart” and “Payment”.

**Listing Products for sale:**

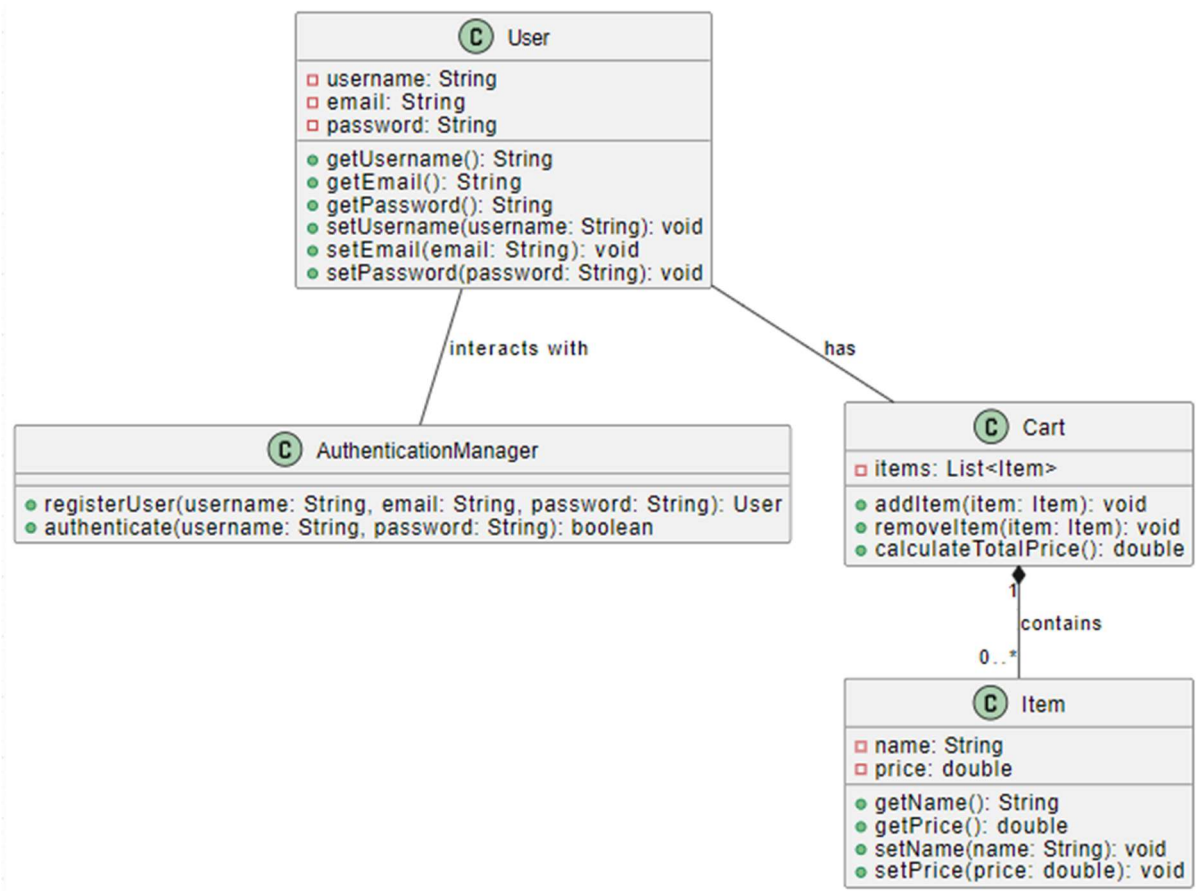
This UML diagram shows the process of listing an item, where once the user is logged in, they have access to the “Sell now” form, a form, where they can enter the title, description, category, price, and the image and then submit using the “List” button. The product is then sent to the database as `ListProduct`.

**View Listed products:**

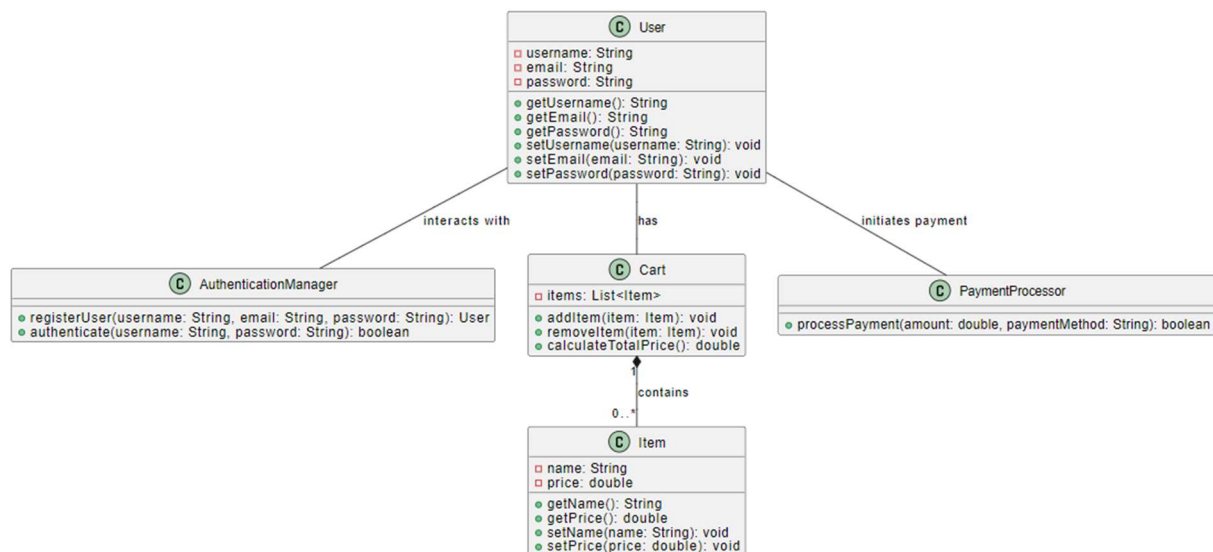
This UML shows that once the user is logged in, the listing that they have submitted should be displayed in the home page, furthermore, it should be clickable. Once the user clicks on the product, it must redirect them to a different page where it shows the product.

**Search function:**

This UML shows the search function, where the user searches their items based on the first letter they input, it then displays the item accordingly.

**Cart:**

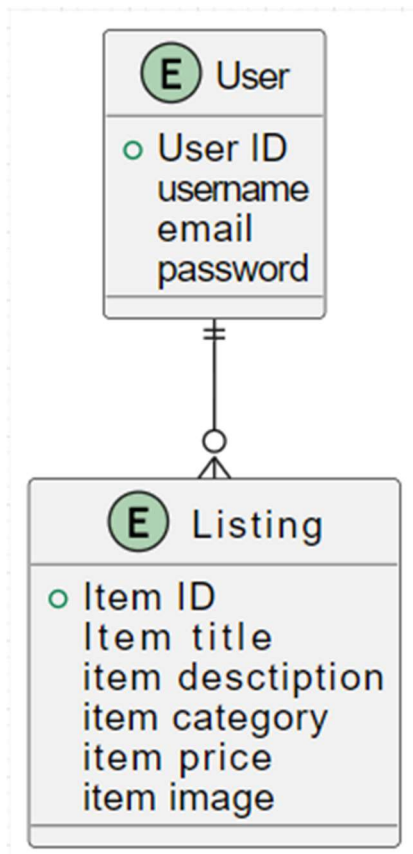
This UML shows the user should be able to add and remove items from their cart, furthermore, the item displayed on the cart should only show the title and the price. All of this is only possible until the user has logged in.

**Acid transaction:**

This UML demonstrates the user carries out the payment if there are items in the cart and the user is logged in.



### Section 4.2.3: Database Schema Diagram:



This database Schema diagram demonstrates the Users, who utilise the platform and they contain their own identifiers. Listing demonstrates the item that the user has set up for sale, with the ID being the primary key, and having other attributes such as Item title, item description, item category, item price and item image. A user sells multiple items; however, each listing can only be associated with one user.

## Section 4.3: Development

### Section 4.3.1 Payment:

To implement the payment feature, first I had to ensure that once the user clicks on the “checkout” button on the Cart page, it should redirect them to the checkout page. To implement this, I have used ReactDOM and updated the App.jsx so that I can turn Checkout into a navigable page.

```
<Route path="/checkout" element={ < Checkout /> } />
```


I have then changed the added another button that says the following:

```
<button onClick={() => setCheckout(true)}>Click here to pay</button>
```

Which then is supposed to redirect the users to the Payment page. The reason to why I added 2 pages to pay is to ensure the security of the website.

I then logged into Developer.paypal.com, and created a fake personal account, which can be used later for testing that the payment system works.









	Home	<b>Apps &amp; Credentials</b>	Testing Tools	Event Logs
---	------	-------------------------------	---------------	------------

---

You're in sandbox mode.

### REST API apps

App name	Client ID	Secret
<a href="#">Default Application</a>	AVM3EnkX2pCzGLbhdWMI4...	 .....  
<a href="#">Platform Partner App - 4811794303583989725</a>	AeNWP5JJwk9UiNJ3XdOTa...	 .....  

Through this I have obtained the client's ID which is important for later.

After doing so, I imported the PayPal NPM package by installing react-paypal-js npm. After doing so I had gained the PayPal's context provider, which I then imported into my Index.jsx:

```
import { PayPalScriptProvider } from "@paypal/react-paypal-js";
```

This is in place so that I can successfully manage the PayPal JavaScript SDK script.

I then set up an initial configuration for PayPal SDK, where it ensures that the application can communicate with the PayPal servers and immediately receive any funds. I have implemented this by doing the following:

```
const initialOptions = {
  "client-id": "AeNWP5JJwk9UiNJ3XdOTaKfwn4h9S48HsEou3r62ER1VxPx8eSmwM6ZcBhx69f8iK3Cr_5phJqCwM9-",
  currency: "GBP",
  intent: "capture",
};
```

Where I have entered the ClientID that I had received during the PayPal sandbox. I have changed the currency to Pounds and the intent to capture.

After doing so, I wrap the the PayPalScriptProvider around my App, to ensure that the payment system is available throughout my application:

```
<PayPalScriptProvider options={initialOptions}>
  <App />
</PayPalScriptProvider>
```

I then create a Payment component where I pass totalPrice, onSuccess and onError as a prop, doing so allows me to reuse these components, and also to pass on the component from the parent (Checkout) to the child (Payment).

I then use the useRef hook to create a reference, which will be used to display the Paypal buttons through the PayPal SDK.

```
function Payment({ totalPrice, onSuccess, onError }) {
  const paypalButtons = useRef();
```

I then used the `useEffect` hook, which is useful to initialise the PayPal SDK, ensuring that the buttons are only rendered after the PayPal SDK is available and has been mounted. To check if the PayPal SDK has been properly loaded, I do the following command:

```
useEffect(() => {
  if (!window.paypal) {
    console.error("PayPal SDK not loaded.");
    return;
  }
}, [])
```

Before proceeding with the initialisation, it is necessary to ensure that `window.paypal` exists, if it doesn't, then the console should return with an error, and the `useEffect` hooked is stopped.

The PayPal buttons are then initialised using the `window.paypal.Buttons()` which then configures for payment, and allows the creation of order on the PayPal platform, through the function `createOrder()`, which then renders purchase units that stores the data that contains the information, and actions that provides PayPal API

```
window.paypal
  .Buttons({
    createOrder: (data, actions) => {
      return actions.order.create({
        purchase_units: [{
          amount: {
            value: totalPrice.toFixed(2),
            currency_code: "GBP"
          }
        }]
      });
    },
  });
```

It then specifies the amount of the purchase, through the purchase unit that is set to `totalPrice.toFixed(2)`, which takes the total price from the cart, and then sets the currency to British pounds.

```
onApprove: async (data, actions) => {
  const order = await actions.order.capture();
  onSuccess(order);
},
onError: (err) => {
  onError(err);
}
```

This is where I implement the `onSuccess` and `onError` callbacks, this allows me to know whether the payment was successful or not.

Lastly rendering the Paypal buttons:

```
return <div ref={paypalButtons}></div>;
```

This component is then imported to the Checkout page, where it renders the `<Payment>` component only if the checkout is true, however if the checkout is false, it must return the button "Checkout".

### Section 4.3.1: Authentication:

In order to create an authentication, I created an AuthContext.jsx file, rendering its props as a children so that it can provide context to its children, which initialises the “isAuthenticated” to using useState to false. So that the user the user is not authenticated upon first entering the website.

```
export const AuthProvider = ({ children }) => {
  const [isAuthenticated, setIsAuthenticated] = useState(false);
```

I then defined logic as an async function, that takes formData as a parameter, which will then handle the authentication. I then used the try method which sends a POST request to the API login for formData. The code then pauses the execution until the fetch is completed, converting the form data object into JSON.

```
const login = async (formData) => {
  try {
    const response = await fetch('http://localhost:5000/api/login', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(formData)
    });
```

If the response is valid, it displays success, and turns the authentication into true, or else it returns an error message.

```
    if (response.ok) {
      setIsAuthenticated(true);
      return true;
    } else {
      return false;
    }
  } catch (error) {
    console.error("Error:", error);
    return false;
  }
};
```

In the Logout function, I have based it so that it removes the authorisation token from the storage, and then sets the isAuthenticated to “false” effectively logging out of the website once the Logout button is pressed.

```
const logout = () => {
  localStorage.removeItem('token');
  setIsAuthenticated(false);
};
```

The `AuthContext.Provider` is wrapped around its children's components so that it has access to auth state and functions.

```
<AuthContext.Provider value={{ isAuthenticated, login, logout }}>
  {children}
</AuthContext.Provider>
```

After doing so, I then create a “`PrivateRoute.jsx`” component and import `AuthContext`. Where then I use using the `useContext` hook.

```
const PrivateRoute = ({ children }) => {
  let authContext = useContext(AuthContext);

  let location = useLocation();
```

I tack the current location using “`useLocation`” hook, so that it tracks where the user is.

If the user is in a certain location, and is not authorised, I used the `(!authContext)`. It should automatically redirect the users to the `/login` page.

```
return <Navigate to="/login" state={{ from: location }} />;
```

I have then wrapped the `/Sell` page and the `/Cart` page around the `PrivateRoute` component so that both the sites get redirected to `/login`.

```
<Route path="/sell" element={<PrivateRoute><Sell /></PrivateRoute>} />
<Route path="/cart" element={<PrivateRoute><Cart /></PrivateRoute>} />
```

I then import `AuthContext` into `Navbar` so that “`isAuthenticated`” state is available. If so, then the navbar replaces the login button with the Logout and Cart button.

```
</nav>
<div className="auth-buttons">
  {isAuthenticated ? (
    <>
      <button className="auth-button" onClick={handleLogout}>Logout</button>
      <Link to="/cart" className="auth-button cart-button">
        <FaCartShopping />

```

or else, the user is presented with the Login button.

```
</>
  ) : (
    <Link to="/login" className="auth-button">Login</Link>
  )}
```

## Section 4.3: Testing

It is necessary to carry out testing as it ensures that the product successfully meets the end requirements, detecting bugs early on and improving client's satisfaction. Due to these factors, for testing I have carried out both White and Black box testing.

### White Box testing:

Since I have both frontend and backend, to test the backend side of the code, I decided to carry out white box testing as this involves complete knowledge of the application. I have created 9 sets of unit testing. That tests the following:

#### Test home page:

This tests the home page and whether accessing the home page returns code 200 with the expected console message:

```
def test_home_page(self):
    print("Testing Home Page...")
    response = self.app.get('/')
    self.assertEqual(response.status_code, 200)
    self.assertEqual(response.data.decode('utf-8'), "enter /api after the link to see all the items in the database")
    print("Home Page test passed successfully.")
```

The test had passed.

#### Test\_get\_listing:

This test checks if the API receives a listing and returns the correct data.

```
def test_get_listing(self):
    print("Testing Get Listing...")
    new_listing = Listing(title='Test Title', description='Test Description', category='Test Category', price='Test Price', image='test_image.jpg')
    with app.app_context():
        db.session.add(new_listing)
        db.session.commit()
        response = self.app.get('/api/listing/1')
        self.assertEqual(response.status_code, 200)
        data = response.json
        self.assertEqual(data['id'], 1)
        self.assertEqual(data['title'], 'Test Title')
        self.assertEqual(data['description'], 'Test Description')
        self.assertEqual(data['category'], 'Test Category')
        self.assertEqual(data['price'], 'Test Price')
        print("Get Listing test passed successfully.")
```

The test had passed.

#### Test\_invalid\_listing\_id:

This test if trying to retrieve non-existing listing from the API returns with a 404 error message with the corresponding error message.

```
def test_invalid_listing_id(self):
    print("Testing Invalid Listing ID...")
    response = self.app.get('/api/listing/999')
    self.assertEqual(response.status_code, 404)
    data = response.json
    self.assertEqual(data['error'], 'Listing not found')
    print("Invalid Listing ID test passed successfully.")
```



This test has passed.

### Test\_add\_listing

This test checks of adding new listing through the API would return with a (201) message

```
def test_add_listing(self):
    print("Testing Add Listing...")
    data = {
        'Title': 'New Test Title',
        'Description': 'New Test Description',
        'Category': 'New Test Category',
        'Price': 'New Test Price',
    }

    response = self.app.post('/api/listing', data=data)
    self.assertEqual(response.status_code, 201)
    self.assertEqual(response.json['message'], 'Listing added successfully')
    print("Add Listing test passed successfully.")
```

This test has passed successfully.

### Test\_user\_signup:

This test checks if the user signup works correctly, by registering as a new user and then verifying its status code (201)

```
def test_user_signup(self):
    print("Testing User Signup...")
    data = {
        'username': 'test_user',
        'email': 'test@example.com',
        'password': 'testpassword'
    }

    response = self.app.post('/api/signup', json=data)
    self.assertEqual(response.status_code, 201)
    self.assertEqual(response.json['message'], 'Signup successful')
    print("User Signup test passed successfully.")
```

It passed successfully.

### Test\_user\_login:

This test checks if the login allows logging in with valid credentials and verifies the status code as (200).

```
def test_user_login(self):
    print("Testing User Login...")
    new_user = User(username='test_user', email='test@example.com', password='testpassword')
    with app.app_context():
        db.session.add(new_user)
        db.session.commit()

    data = {
        'username': 'test_user',
        'password': 'testpassword'
    }
    response = self.app.post('/api/login', json=data)
    self.assertEqual(response.status_code, 200)
    self.assertEqual(response.json['message'], 'Login successful')
    print("User Login test passed successfully.")
```

This test has passes successfully.

### Test\_invalid\_login:

This test checks if the user login is invalid and if it returns with (401) status code.

```
def test_invalid_login(self):
    print("Testing Invalid User Login...")
    data = {
        'username': 'invalid_user',
        'password': 'invalidpassword'
    }
    response = self.app.post('/api/login', json=data)
    self.assertEqual(response.status_code, 401)
    self.assertEqual(response.json['error'], 'Invalid username or password')
    print("Invalid User Login test passed successfully.")
```

This test has passed successfully.

### Test\_get\_all\_users:

This test checks if the API retrieves all the users and returns a list of users.

```
def test_get_all_users(self):
    print("Testing Get All Users...")
    new_user = User(username='test_user', email='test@example.com', password='testpassword')
    with app.app_context():
        db.session.add(new_user)
        db.session.commit()

    response = self.app.get('/api/signup')
    self.assertEqual(response.status_code, 200)
    data = response.json['users']
    self.assertTrue(len(data) > 0)
    print("Get All Users test passed successfully.")
```

This test displayed an error.

### Test\_get\_all\_listings:

This test checks if the API retrieves all the listing data.

```
def test_get_all_listings(self):
    print("Testing Get All Listings...")
    new_listing = Listing(title='Test Listing', description='Test Description', category='Test Category', price='Test Price')
    with app.app_context():
        db.session.add(new_listing)
        db.session.commit()

    response = self.app.get('/api/listing')
    self.assertEqual(response.status_code, 200)
    data = response.json['listings']
    self.assertTrue(len(data) > 0)
    print("Get All Listings test passed successfully.")
```

This data has passed successfully.

## Black box testing:

Due to the nature of Black Box testing, where the user testing has no knowledge of the system's functionality, I have used this to test the frontend of the code. By making a family member who are not aware of any of the functionality regarding the code. I have conducted 7 UI tests and see if it passes:

### Trying to log in without access to an account:

For Login, I asked the user to try and login without having an account:

**Login First!**

Maryam

.....

The username or password does not match

Login

---

Don't have an account? [Sign Up](#)

Pass as it does not let the user through.

### Signup to an account:

To test this feature, I let the user sign up to their accounts

Thank you for submitting. Click here to Login into your account.

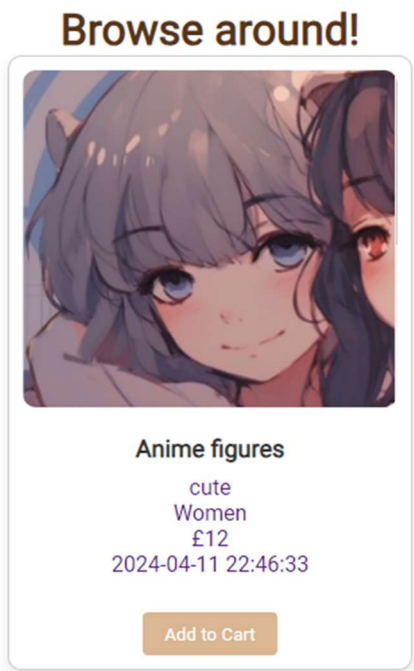
Login

Pass as it creates the account and redirects the user into a the login page



**Listing an item:**

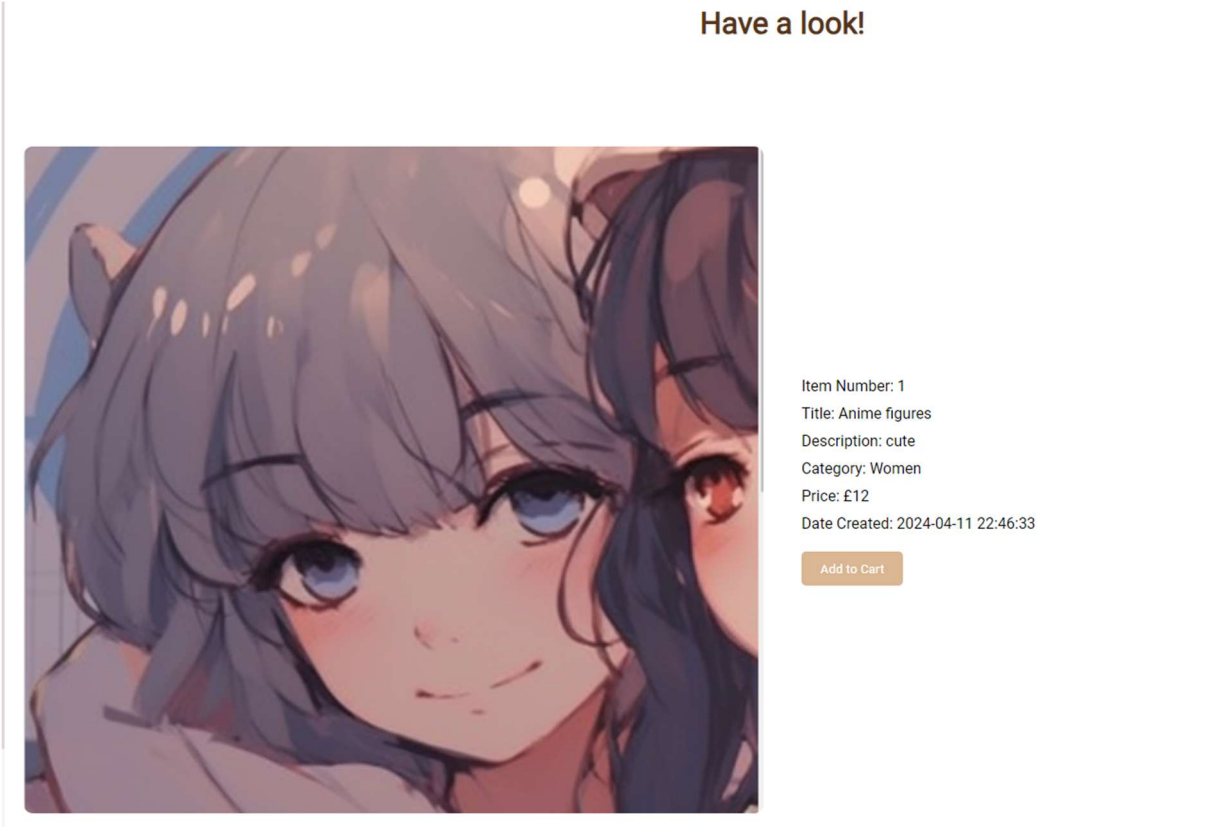
For this test I had asked the user to see if they can add a listing of their choice.



Passed as it allows the user list an item.

**Click on listing:**

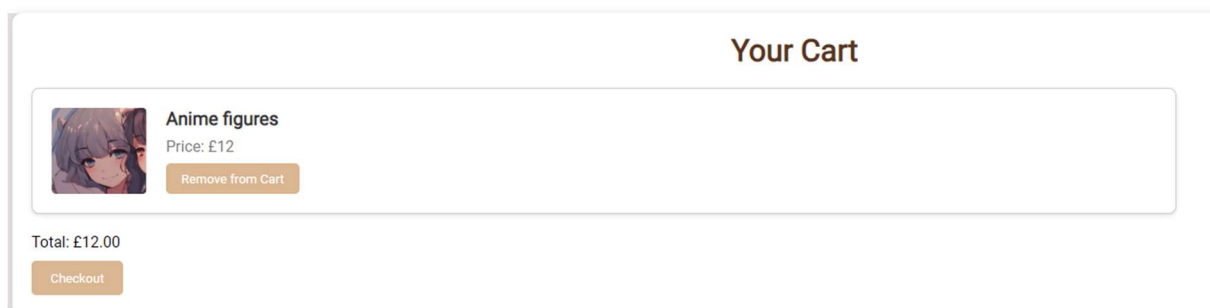
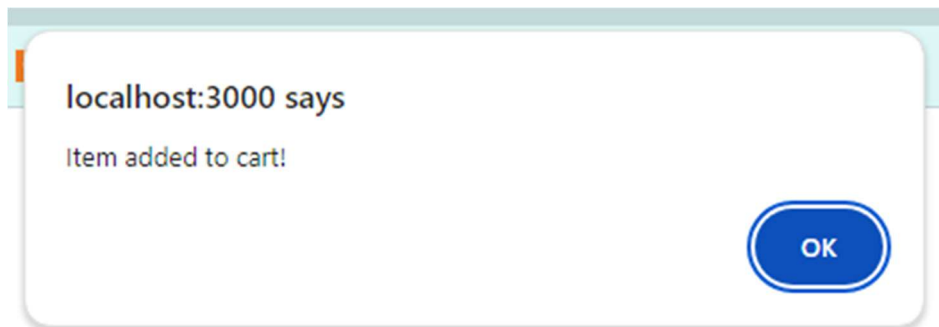
For this test I had asked the user if they can click on their listing.



Passed as the user was able to navigate to their listing by clicking it

**Add to cart:**

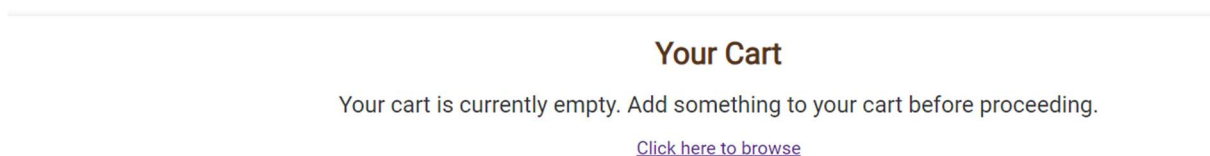
For this test I had asked the user if they could add their listing to cart:



Passed as when the user clicks “add to cart” it displays an alert message stating that the item has been added to cart, and upon navigating to cart, the item is displayed on the cart.

**Remove to from cart:**

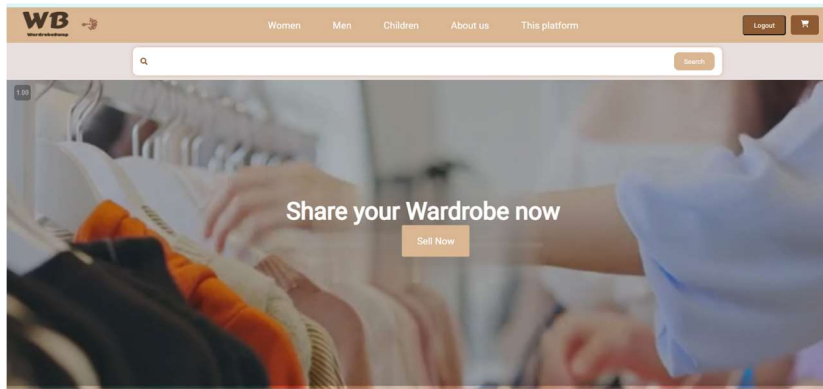
The user was asked if they could remove listing for the cart:



Passed as the user successfully removed the listing for the cart, which then displayed an empty cart message, redirecting the user to browse further.

**Clicking the logo allow navigation to home page:**

The user was asked to click on the logo from the cart to see if it takes the back to the home page:



Passed as it allows the users to successfully navigate to the home page as soon as they click the logo.

### Test evaluation:

Although most of the tests carried out in this report had passed, including a more diverse type of testing would have been more ideal as a website is complex, as a result it has many features and attributes that needs to be tested thoroughly. Furthermore, if I had more time, I would have used TDD to carry out future testing, as TDD allows a forced break down of a system due to its components, as well as creating better quality code, which would have improved the quality of the code further. I have not used TDD for my testing as it required to write the tests before writing the code, and due to the volume of the project, it was slightly difficult to carry out tests before the development. However, in future projects, I will be utilising TDD earlier on so that it allows fasted debugging.

## Section 4.5: Tools

I have used various tools that assisted me during the development of my website, this includes, planning out the days each component must take me, tracking the progress that was made throughout the development, and lastly resting the backend APIs using posting and sending data. The main tools that I have utilised are Gitlab, Trello and Postman.

### Gitlab

Gitlab is one of the tools that has been used the most throughout the development of the website, as I have used it to first clone the repository multiple times into my IDE, by copying the repository URL and then logging in using “git clone [repository URL]”, allowing me to change and update my code in different devices. After every change, I have carried out various features such as “git commit” followed by a commit message and then staged the changes and ushing into the GitLab using the “git push” command. I have also utilised the different branches to store certain code, such as I have a branch for Payment, cart, listing, and other features. Gitlab was especially useful as it provided with a clear version control as it allows us to track the changes, especially if the code is accidentally lost, it can be easily retrieved. It also helped the supervisors to track the work progress made, allowing a clear understanding of how close/far the user is when it comes to meeting the client’s requirements.

Name	Last commit	Last update
📁 .vscode	fixed images	3 weeks ago
📁 backend	update database	1 week ago
📁 frontend	Changed css	3 weeks ago
📄 Term1 Intern Report.pdf	adding image	2 months ago
📄 book.txt	committing progress	1 month ago

Figure above is the GitLab's home page.

### Trello:

I have used Trello through creating cards where each card lists each requirement, grouped into 5 sections, User registration and authentication, Product Listing and Viewing, Shopping cart and Advanced Features and Implementation. Using these cards, I was able to filter and track the progress of my project, following the requirements and manually ticking each requirement upon completion. Trello's simple interface allowed easy drag and drop features. Trello's simple UI allowed ease of use, resulted in easy tracking of progress as I was able to add more requirements if needed. Furthermore, the visual layout also made the requirements clearer as each card are spaced out, with contrasting colours.

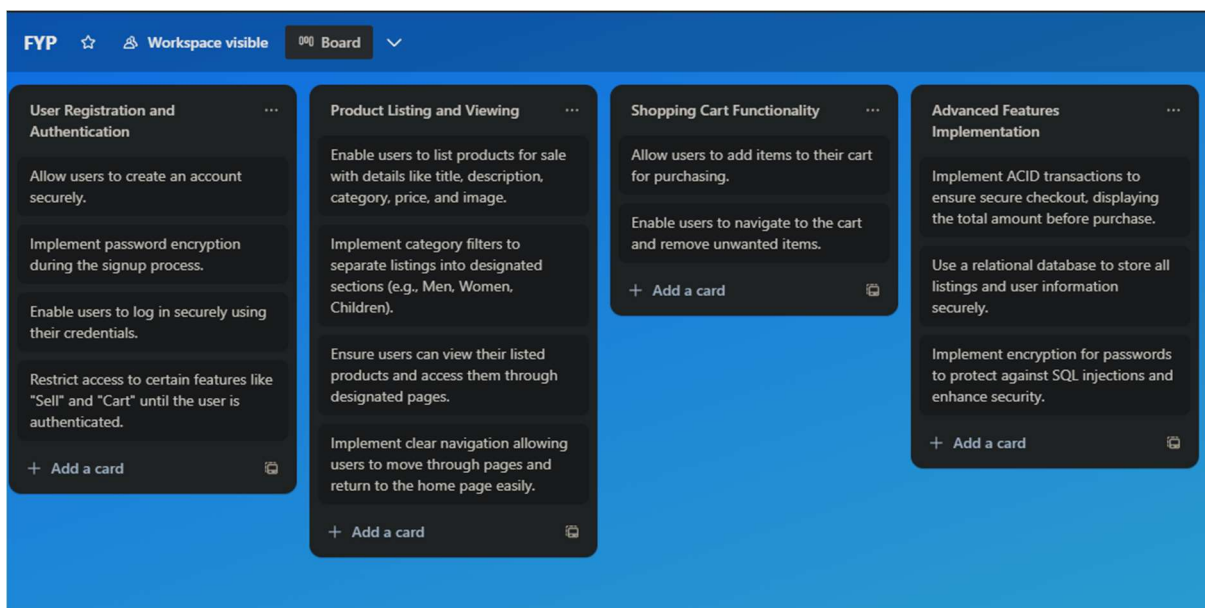


Figure above demonstrates the requirement layout based on the project requirements.

### Postman:

I have used Postman to send http request to my API servers and gets a response back. This is vital for testing my APIs during the development stage, I had to ensure that my API was working according to plan, allowing ways to document the progression of my APIs. With Postman, a user can generate comprehensive documentation for their requests, allowing an easier development especially since it allows comprehensive understating of the API. Postman, furthermore, allow me to organize your request into collections, which makes it easier to manage all the API requests. Postman allows grouping related requests, where the request can be shared between team members, giving monitoring capabilities to keep track of APIs performance uptimes. A

user can set monitors to periodically send requests to the API and receive alarms if there are any problems regarding the API, furthermore, Postman provides with the choice for mock servers where the users can simulate the API responses, which is useful for when the backend server is not available yet.

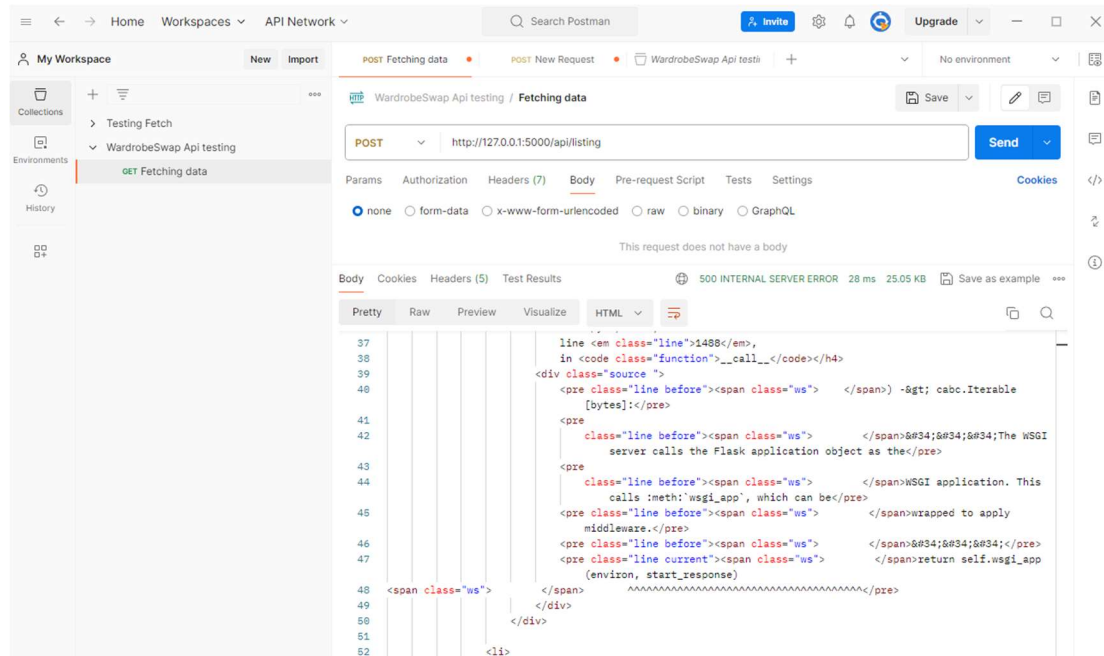


Figure above shows a screenshot of the POSTMAN send POST request to my API.

## Chapter 5: Technical Decision Making

### Section 5.1: Backend

The framework that I chose: **Flask**.

I have decided to use flask due to its simplicity and minimalism, as someone who is relatively new to web development, something that offers lightweight framework and helps to start a small-medium project with ease, where the flexibility and Rapid development is prioritizes, it allows new developers to quickly grasp the framework and utilise its features with ease. The flexibility and customization of Flask allows me to use the libraries and tools that I prefer unlike the other frameworks with the libraries embedded in their frameworks, which allows me to be more creative and flexible with my development. As a result, Flask is the best choice for my project.

### Section 5.2: Frontend

The framework that I chose: **React**.

React is a component-based framework, where it allows users to reuse the UI components, despite this, reacts minimalistic approach allows there to be a low learning curve, especially for users who are familiar with JavaScript, unlike other languages such as Angular and Swift, where both languages may be slightly challenging to grasp due to its complex components and dependencies. Reacts Dom increases its performance, allowing a faster rendering and updates

speed. Unlike Angular, React does not get effected by its complex architecture, unlike Swift. Furthermore, its performance in web developments context could depend on elements like server-side processing and network latency. Reacts component-based architecture provides stability and scalability, allowing web developers to robust UI components and scale applications efficiently, which is perfect for areas whereas Angular has a more complex architecture, which limits flexibility, as a result, React is the best choice for my project.

### **Section 5.3: Database**

The RDMS that I chose: **SQLite**.

SQLite takes the least amount of skill to use, as it requires no server set up, therefore, it is ideal for smaller scale servers. On the other hand, MySQL and PostgreSQL are more complicated to set up and configure, specifically regarding optimisation and its performance whilst handling large databases. Due to PostgreSQL and MySQL's scalability, these relational databases can handle large data sets and high traffic loads with features, especially for high number of workloads. Although SQLite isn't suited for heavy load usage, it is still the best relationship database for my project as my e-commerce website is designed to handle smaller scale of data. In addition to this, due to its file-based nature and lack of centralised control, SQLite performs well for a small-scale application where the database size is limited. MySQL and PostgreSQL are both optimised for performance, but PostgreSQL usually outperform MySQL due to its advanced optimisation potential. SQLite being lightweight, it is best suitable for simple data storage and relevant tasks.

### **Section 5.4: Architecture Paradigms:**

The Architecture Paradigm that I chose: **MVC**.

MVC's clear separation in the modules, allow for an easier guidance of complicated applications. By breaking the applications into distinct factors, each responsible for its own specific features (such as data, presentation, and user interaction), MVC encourages modularity and an ease of maintenance, furthermore, having a faster development. Object Oriented Architecture on the other hand, despite it having a easier to model, it is quite complicated to manage and as it required a deeper understanding of principle. Service Oriented Architecture also offers easily communicable services across different platforms and languages; however, it deals with a high message volume making it more complicated for users. MVCs separates the concerns into Models data and logic. Views which are presentation and controller, making the application more usable and allows faster development through the same line of work on different platforms. MVC has been very well-established across various platforms, allowing flexibility between platforms and components where developers can choose different platform, based on its project requirement. This level of flexibility is ideal for an e-commerce website, and considering all these elements, the best choice would be MVC as it has a faster development and manageability.

## Chapter 6: Critical Analysis and discussion

### Section 6.1: Project Process

Looking back at my progress throughout the project, I can identify places where I have limited and could have conducted myself better. The first constraint is my poor time management. I often tried to rush into meeting the deadlines and pushed the existing deadlines a later date, not only increasing my levels of stress, but it also compromised the quality of work, since some parts of my project has not been tested yet, or it is still broken. Had I allocated my time to a more efficient manner, with clear deadlines and strict work ethics, I would have avoided the unnecessary pressure during the execution, furthermore, having a better-quality delivery. For future projects, to improve on this poor time management, I will be trying to allocate a daily schedule of working towards a project, for example an hour every weekday. This way, the habit would turn the poor time management into a more relaxed way of meeting the requirements, enhancing the quality of work.

Additionally, I had not used the resources around me to conduct myself better when in time of need. Throughout my project, I had encountered the same error every time, and each fix would be countered by another error regarding the same issue, this caused me to lose a lot of valuable time trying to figure out the error on my own. For example, I have encountered the same CORS error where it states “Cross-Origin Request is blocked”, another example of this includes my database, as my database kept having issues regarding the storage of data. If I had asked help from supervisors that were allocated to me, they could have effectively guided me into a solution. In future projects, I will ensure to consistently ask for assistance whenever I require it, as these assists can help me immensely, as a result accelerating my work,

Lastly, I had not taken account for potential setbacks such as illness and data losses. I had issues with my devices where my laptop had crashed multiple times, as a result I had lost a huge amount of work, furthermore, due to the technology issues, I had lost marks. If I had managed my time more and carried out proper risks and mitigations analysis in advance, it would have allowed a swifter damage control to get back on track.

Despite all the challenges I had faces, there are several positive aspects, as I managed to complete most tasks on time, despite the struggling. I had also managed to prioritise my work to meet the deadlines accordingly. Furthermore, I had demonstrated resilience in overcoming the obstacles faced during the illnesses and loss of information, where I had effectively gained back all the lost information and managed to rewrite it all within a few hours.

Moving forward, I will ensure to build towards these strengths and use it to address areas in the future.

### Section 6.2: Presentation of Findings

Reflecting on my choices throughout my projects are generally positive, especially my choice in backend Flask. Flask offered several advantages, for example, it easily allowed me to create a working database, and using Flask, I was about to implement features that would have been difficult to implement otherwise, such as Password encryption, Flask’s bcrypt feature allowed



me to safely encrypt the user passwords in the database, which helped with the security aspect of my website, as it protects against SQL attacks. Furthermore, the ease of the framework allowed me to learn and implement all the Getters and setters that shaped the API.

However, upon closer reflection, I realised that there were areas where I could have carried out more research, especially regarding relational databases, and investigated alternatives to SQLite. Despite the simplicity of the database, I was presented with many challenges, where the database was not able to hold and store some of the information displayed. It also did not have many advanced features to solve the issue, as a result affecting the quality of the overall work. Similarly, while I had opted for React, upon closer inspection, exploring Angular as an alternative may have been more beneficial for my project. This is because React's complexity slowed down the process of the development as I was spending more time on researching on React's components than actual development a lot of the time.

Furthermore, I acknowledged that I could have used more tools available to me for this project, such as using the lecture materials on how to implement TDD would have been useful to create intensive testing, allowing an ease of development.

In conclusion, while I am mostly satisfied with my decisions, however doing some extra research on certain aspects of the languages, and furthermore, learning from these experiences, I can try and develop enhance my background reading by reading about more languages, as well as using all the resources that was presented to me.

### Section 3: Deliverable analysis

Reflecting upon the quality of my code, I am mostly pleased, with the structure of my code. In my opinion the code seems to be clearly structured, with necessary comments with each functionality clearly demonstrated and depicted.

I have ensured that all the requirements are met using checklists, where I have listed each requirement, and ticked each of the requirements off the list once the requirements have been completed ensuring that all the requirements.

especially since I have managed to meet most of my initial requirements,

I have ensured that I created a *fully developed UI*:

- **User registration and Authentication:** The user can now successfully create their own accounts and log in.
- **Listing products for sale:** The user can now list items for sale.
- **View the listed products:** The users can now view the product that they have just listed and are able to click on the products.
- **Category filter:** The listing is filtered based on the category of the Men, Women and Children's section.
- **Navigation through pages:** The user can now navigate through each page and always find a way to reach the home page.
- **Cart:** The users can now add to cart and remove the item from the cart. There is also a total and a "checkout" button on the cart.



I have included *Advanced features* such as:

- **ACID transactions:** The user can now checkout of the basket and are presented with a PayPal transaction page.
- **Relational Database:** There is a relational database that contains all the information about the Listings and the Users.
- **Use of encryption (Advanced Security Mechanisms):** There is an encryption in passwords to protect against SQL injections.

I have ensured to create a *user-friendly* website:

- **Use of simplicity designs:** I made sure that the website is simplistic and does not overwhelm the users.

I have ensured the *safety* of the website:

- **Restricted modules until authorisation:** I have implemented restricted modules, where the user needs to be logged in before entering certain modules, such as the sell page and the cart page.

Overall, I am generally satisfied with my project, however, I recognise that I should have carried out more intensive testing. Furthermore, if I had more time, I would have implemented a comment feature, where the user can comment under each other's post, and also give the items a rating.

## Chapter 7: Professional Issues-UF

### Plagiarism

Plagiarism comes in many forms, this includes the direct copying of code from an online source without proper acknowledgement of the person's code that has been used, pretending other's work as yours. This is especially unethical as it violates the original user's intellectual property, and it challenges the integrity of the developers, including the project. It is important to address the ethical issues surrounding plagiarism in Website Development, by providing correct references that noted back to the original author. Plagiarism also damages the mutual trust that there is in the developer community, further affecting a person's reputation. Plagiarism as a developer could potentially lead to severe consequences such as removal of academic or work environment due to it and legal actions and laws of copyright.

A few things the users should consider if they had used code from an external source is, by ensuring the user give clear acknowledgement and attribution to the original creator, including their names, URL, title, and other related information that would identify the user so no misunderstanding happen if there are any situation where the original creator could press copyright charges. This will help other programmers understand the context that the code had derived from. The developers should also seek permission for extensive use, as it demonstrates respect for the original code and ensures that the code has been used fairly and objectively.

When crafting my report and my code I have used methods in avoiding plagiarism. One of which being a plagiarism checker. I used a software called justdone.ai, which is developed by Turnitin, the same software my university uses for submissions. This minimalizes the chances of plagiarism strikes. For example, I have also the plagiarism check while carrying out background reading, ensuring that I have not accidentally plagiarised any background reading. This is often a feasible way of avoiding plagiarism, however, at times plagiarism detectors can be inaccurate, and give false information.

I have also reviewed multiple sources and put them in clear quotation once I have used it in my report. These sources have been listed in the bibliography below. I used 17 sources on the background reading section, and each citation has been clearly mentioned and discussed accordingly. Another way I have ensured to not plagiarise is the use of extensive literature review. Each review depicts a clear overview of the citations and dedicated each section to the corresponding authors.

Another way I have avoided plagiarism is through uses many sources, making it difficult to plagiarise, as there are too many sources to plagiarise from. I have also ensured that most of the information, especially regarding my project was original, and if any references have been used, I have clearly cited them in the Bibliography.

Overall plagiarism should not be practiced, as it is unethical, further causing legal actions strikes.

## Chapter 8: Bibliography and Citations

[1] Noponen, S. 2017, What makes a beautiful website? Factors influencing perceived website aesthetics. Available from:

<https://jyx.jyu.fi/bitstream/handle/123456789/53109/URN%3aNBN%3afi%3ajyu-201702271533.pdf?sequence=1&isAllowed=y>

[2] Scott-Parker, S. 2003, What makes a successful website?, BA (Hons) Multimedia Design and Digital Animation dissertation, Cumbria Institute of the Arts.

[3] Lee, Y. and Kozar, K.A. 2011, 'Understanding of website usability: Specifying and measuring constructs and their relationships', Decision Support Systems. Available from:

<https://doi.org/10.1016/j.dss.2011.10.004>.

[4] Dubois, P.F. 2007, 'Guest Editor's Introduction: Python: Batteries Included', Computing in Science & Engineering, vol. 9, no. 3, pp. 7-9. Available from:

<https://doi.org/10.1109/MCSE.2007.51>.

[5] Pratap, M. 2023, 'How does the Python MVC Framework Work? What are the Benefits?', Blog post, 6 March. Available from: <https://supersourcing.com/blog/how-does-the-python-mvc-framework-work-what-are-the-benefits/>.

[6] Romik, T. 2023, 'Why Choose NestJS As Your Backend Framework?', Blog post, 17 August. Available from: <https://selleo.com/blog/why-choose-nest-js-as-your-backend-framework>.

[7] Mendes, A. & Ferreira, R. 2024, 'FastAPI vs Flask: what's better for app development?', Blog post, 12 March. Available from: <https://www.imaginarycloud.com/blog/flask-vs-fastAPI/#flaskpros>.

[8] Luzniak, K. 2021, 'What is Angular Used For and When Should You Use it Instead of Other Frontend Technology?', Blog post, 23 December. Available from: <https://neoteric.eu/blog/what-is-angular-used-for-and-when-should-you-use-it/>

[9] Share IT. 2024, 'The Pros and Cons of Swift Programming Language', Share IT Solutions, Available from: <https://www.shareitsolutions.com/blog/swift-pros-cons/>.

[10] Ockelberg, N. & Olsson, N. 2020, Performance, Modularity and Usability, a Comparison of JavaScript Frameworks, Degree project in Computer Engineering, First Cycle, 15 credits, Stockholm, Sweden. Available from: <https://www.diva-portal.org/smash/get/diva2:1424374/FULLTEXT01.pdf>.

[11] Peterson, R. 2024, 'What is PostgreSQL? Introduction, Advantages & Disadvantages', Guru99, Updated 16 March. Available from: <https://www.guru99.com/introduction-postgresql.html>.

[12] Blansit, B.D. 2006 'The Basics of Relational Databases Using MySQL', Journal of Electronic Resources in Medical Libraries, 3(3), pp. 135-148. DOI: 10.1300/J383v03n03\_10. Available from: [https://doi.org/10.1300/J383v03n03\\_10](https://doi.org/10.1300/J383v03n03_10). Published online: 03 Oct 2008.

[13] Unknown author. 2023, 'What is SQLite? An overview of the relational database solution', 27 June. Available from: <https://www.ionos.co.uk/digitalguide/websites/web-development/sqlite/>.

[14] thesunpandev 2024, 'Object-Oriented Analysis & Design', GeeksforGeeks, Last Updated 14 March. Available from: <https://www.geeksforgeeks.org/object-oriented-analysis-object-oriented-analysis-design/?ref=lbp>.

[15] Barney, N. & Nolle, T. [2023], 'Service-Oriented Architecture (SOA)', TechTarget, Available from: <https://www.techtarget.com/searchapparchitecture/definition/service-oriented-architecture-SOA>.

[16] Hernandez, R.D. 2021, 'The Model View Controller Pattern – MVC Architecture and Frameworks Explained', FreeCodeCamp, April 19. Available from: <https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/>.

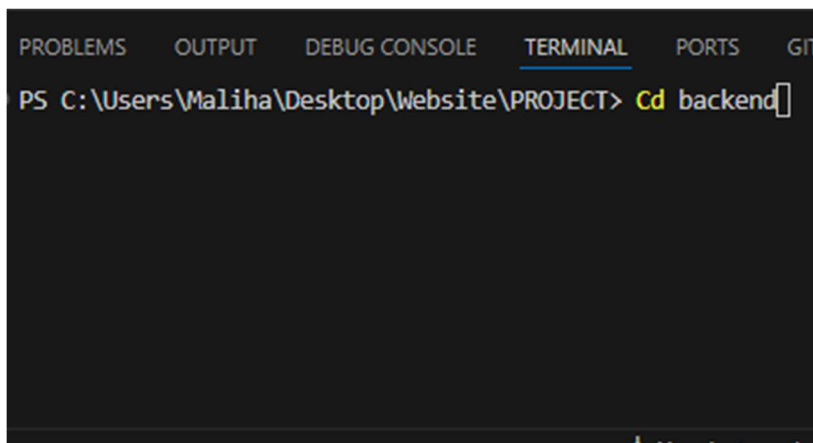
[17] Unknown author. 2016, 'What is MVC? Advantages and Disadvantages of MVC', InterServer Tips, Posted on October 28. Available from: <https://www.interserver.net/tips/kb/mvc-advantages-disadvantages-mvc/>.

## User Manual

To run the code, the user must do the following step:

- 1) Must have python installed.
- 2) Must support React JS

There should be 2 terminals.



Terminal 1: *CD Backend*

The users should then install the following components in terminal 1 (Backend):

*pip install Flask-SQLAlchemy*

*pip install flask-cors*

*pip install Flask-Migrate*

```
pip install Flask-Bcrypt
```

The user should then activate the Virtual Machine, by doing the following command:

```
Venv/Scripts/Activate:
```

However, if the running script is disabled:

You can try this:

```
\Venv\Scripts\Activate.ps1
```

After the Virtual Machine is running, do the following command depending on your Python version.

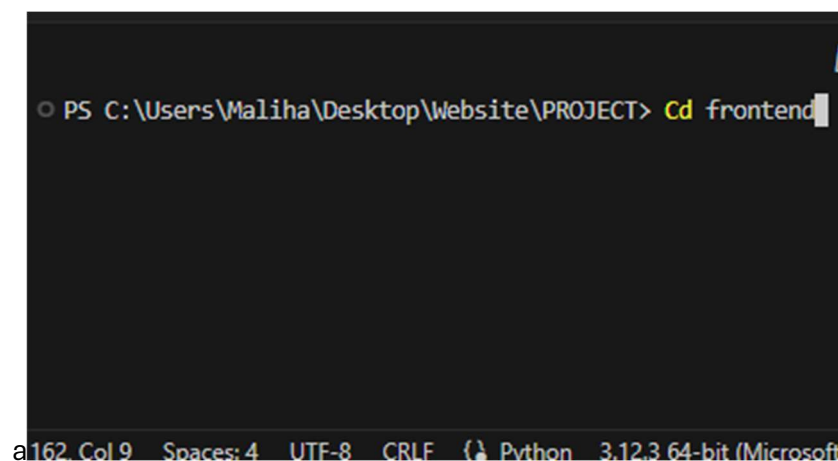
Older version of python:

```
python app.py
```

However, if the user is using a newer version they should do:

```
python3 app.py
```

The user should be presented with a link which is the navigation to the API. They must click as the website does not run without the API.



Terminal 2: *Cd Frontend*

add this into the terminal:

```
npx eslint --cache
```

And then do

```
Npm start
```

## Demo

[https://www.youtube.com/watch?v=YCCTK6m\\_nKU](https://www.youtube.com/watch?v=YCCTK6m_nKU)

## Project Diary

11/10/23 - 12/10/23

- 1) tried to connect to gitlab on vscode but couldnt due to computer issues
- 2) finally managed to by creating an personal access token and using that as a password instead of my own password.
- 3) used vscode clone to clone this into my repository

14/10/23

- 1) researched and found out that using flask for backend is the best choice
- 2) set up server.py and created a server where it gives me a url and a link
- 3) created html file and connected to the server
- 4) connected to the database

15/10/23

- 1) looked at the file and noticed how disorganised it is and rearranged it to make it look more presentable and less confusing
- 2) imported react js frontend and editing the files accordingly and keeping the ones i need
- 3) the server.py was no longer working so i created a virtual machine by using backend\Scripts\activate and installed flask within the virtual machine

16/10/23

- 1) errors as it does not run the frontend server.
- 2) this was because my functions in <div> was wrong. when i got rid of it, it worked

18/10/23

- 1) rearranged the components in the front and backend - unable to commit due to existing errors

20/10/23

- 1) trying to fix the errors caused by installing flask and react - still cannot commit due to existing errors

26/10/23

1)the issues fixed regarding the installation of flask and react, however the contents on server.py do not display on the webpage

2)managed to display the contents by hanging the the name of the @app.route to @app.route('/home'). and changing the name of the url to /home - still cannot commit due to existing errors

28/10/23

1)fixing issue where the contents arent being fetched from backend

2)got rid of the code that is supposed to fetch backend code and display it on frontend preparing it to commit

02/11/23

1)could not do much due to other assignments however, researched about how to make the website look presentable

2)looked into the colour scheme and the type of website

3)settled for pastel colours

4)decided on how to make the navigation bar and collected all the materials for after the due of assignments

05/11/23

1)started implementing the navigation bar

2)created a new file called Navbar

3)error faced, where it mentions "the file is not found in components" while the file is clearly in the componenets

06/11/23

1)due to some personal issues, i had to switch devices and finding issues setting up gitlab on new device, therefore, i am also unable to commit (back up on USB)

08/11/23

1)unable to work on the project due to other coursework

2)fixed the bug of "file not found", by adding "export function Navbar()"

9/11/23

1)fixed the issue with gitlab not being able to connect to my IDE

10/11/23

- 1)started working on implementing search bar
- 2)working on allowing users to search items from the database
- 3)resumed working on navbar, which was put on halt due to errors
- 4)fixed bugs regarding gitignore and node modules, and committed changes

11/11/23

- 1)managed to implement navbar, however a new bug appeared where the page links do not appear when we click on "more" option
- 2)trying to implement the database for the search function,

13/11/23

- 1)fully implemented the navbar and other minor issues with Navbar, and committed the official Navbar
- 2)implemented searchbar feature without connecting to the database
- 3)when looking into commits, i realised that it says that the commits were committed by 2 people, Maliha Ahmed (anonymous), and Maliha Ahmed ZKAC334 (Uni account). I logged into VSC using microsoft
- 4)test submitting book to see if it fixes the submission of Maliha Ahmed (anonymous).
- 5)added a search button
- 6)added a video background that is slightly dimmed, that allows users to sell their clothing

14/11/23

- 1)spend the whole day trying to implement the website so that when i click on buttons it would redirect me to a different file
- 2)keep getting error messages saying "export 'AbortedDeferredError' (reexported as 'AbortedDeferredError') was not found in 'react-router'"

15/11/23

- 1)had to recreate the whole workplace since the .json files were mixed up. i had 2 .json files and a few things uninstalled. i also had 2 react-dom and react-router-dom installed, which clashed with my files overall.
- 2)added a logo
- 3)committed new workplace



27/11/23

- 1) The navigation wasn't working. It kept showing 'You cannot render a <Router> inside another <Router>'. You should never have more than one in your app.
- 2) removed BrowserRouter, and the navigation works
- 3) made it so that when the logo is pressed it's redirected to the home page

28/11/23

- 1) connected to the frontend to the backend, so that the port to fetching and posting data is open
- 2) implemented the GET and POST method

29/11/23

- 1) Attempting to create database using `db.create_all()`, however there are some issues regarding python on recognising the error

30/11/23

- 1) created a form in the sell option
- 2) refined all the pages so that it looks more organised

2/12/23

- 1) Successfully created the database

3/12/23

- 1) successfully posted the data from the form into the server,

6/12/23 - 7/12/23

- 1) managed to implement a filter the listing by category
- 2) committed everything

25/01/24 - 13/02/24

- 1) implemented the image filter so that it displays on the home page
- 2) trying to fix CORS error

20/02/24 - 26/02/24

- 1) attempted to fix the image
- 2) attempted to fix the database bug where the listing was not saving into the database anymore

28/02/24 - 03/03/24

- 1) fully implemented the image column in the database
- 2) attempted to fix the database with the new image column
- 3) tried rendering the image so that it shows on the listing card

04/03/24 - 07/03/24

- 1) had continues CORS error and tried fixing it through the use of CORS function
- 2) implemented a basic form for login

09/03/24 - 10/03/24

- 1) restructured the Sell page so that it handles the image uploads
- 2) implemented a simple cart page

13/03/24 - 18/03/24

- 1) came across CORS error again, and managed to fix it
- 2) created the Sign up page

18/03/24 - 20/03/24

- 1) Added functionality to the login page so that it handles the username and password
- 2) added a functionality to sign up page so that it handles username, email and password
- 3) linked the pages together

21/03/24 - 23/03/24

- 1)fixed the css for the login
- 2) added suited authentication to the

24/03/24 - 29/03/24

- 1) implemented the cart function
- 2) fixed the database as it was not storing the user information into the database
- 3) initiated the search function

30/03/24 - 01/04/24

- 1) Added a display listing page
- 2) Implemented the search feature
- 3) initiated the payment

02/04/24 - 06/04/24

- 1) worked on the search feature
- 2) fixed the cart feature
- 3) Started fixing authentication

07/04/24 - 09/04/24

- 1) finished the search feature
- 2) completed authentication
- 3) implemented the payment feature

09/04/24 - 10/04/24

- 1) fixed a few bugs
- 2) updated css
- 3) fixed payment bug

10/04/24 - 12/04/24

- 1) Added more css
- 2) fixed bugs regarding payment and search
- 3) added javadoc comments

