ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
FACULTY OF INFORMATICS

THESIS

OE-NIK	Student’s name:	**Barta, Zoltán Kevin**
2024	Student’s registration number:	**T-006056/FI12904/N**

STUDENT'S DECLARATION

Dated: Budapest, November 30, 2024

[illegible]

....

student's signature

Development and Deployment of a Simple Ecommerce Website

Kevin Zoltán Barta

Contents

1 Introduction

1.1 Short Description of the Task

The goal is to produce a simple yet robust and extendable ecommerce website which includes a frontend/backend part and a database. The website should have the core functionalities of an ecommerce website, such as creating an account, logging in and placing orders. Additional functionalities may be added; however, the focus is on the core functionalities and their stability.

1.2 Definition of Ecommerce

Ecommerce stands for electronic commerce. Commerce refers to the trade of goods and the activities associated with this. [1] Although ecommerce has several definitions [2], in this paper we accept the simplified definition of trading goods through electronic communication and activities related to that. [3] Ecommerce may be business-to-business (B2B), business-to-consumer (B2C) and even consumer-to-consumer (C2C). [1, 3] In our case, we will be focusing on the business-to-consumer model, since this website intends to sell goods to the broad public. As opposed to brick-and-mortar stores, ecommerce sites typically do not have physical premises.

1.3 History of Ecommerce

1.3.1 Early Attempts

A core concept to cover in the advent of ecommerce is the Electronic Data Interchange (EDI), which allows the automated exchange of documents between businesses. This technology appeared in the middle of the 1970s. Even though the functionalities EDI offered, it did not spread rapidly, more than 20 years past its appearance, by the late 1990s, less than 1% of the companies in Europe and the United States had adopted this technology, due to its costliness and complexity. [3]

Not long after the beginning of EDI, in the late 1970s, EFT (Electronics Funds Transfer) appeared, which enabled electronic transfers. In case of the United States, these transfers happen under the Automated Clearing House (ACH) system. According to data from 2019, the ACH system handles 24 billion transactions every year. [4] In 1979, Michael Aldrich invented teleshopping, which can be considered as a predecessor of today's ecommerce. [4, 5]

One of the first attempt of ecommerce as we know today was in 1984, when CompuServe, at that time the main provider of bulletin boards, introduced the Electronic Mall, which offered the goods of more than 100 stores. The service did not become popular. [6]

1.3.2 New Generation of Ecommerce

In the early 1990s, with the appearance of the graphical user interface, the Internet started to gain traction with the general public. This process led the Internet, once a tool for researchers and engineers, to start growing into a business-oriented technology. A good illustration of this is the fact that in 1997 the commercial (.com) domain took over the educational (.edu) domain in terms of popularity, as the most widely used top-level domain. From an ecommerce perspective, the 1991 lifting of commercial restrictions imposed on the network by NSFNET marked an undoubtably important event, allowing the establishment of non-governmental ISPs. [2]

Once ecommerce began its spread, the next obstacle for further growth was limited Internet access and security concerns. With the help of new legislation and agreements, the security concerns were (at least partly) addressed. [3] In 1993, the Secure Socket Layer (SSL) protocol was created, which laid the foundation of secure data transfer online. Not long thereafter appeared the first third-party credit card processing companies, which marked a significant event in the development of ecommerce. [6]

In 1995 Amazon appeared in the ecommerce scene as a bookstore, followed by eBay, a hybrid consumer-to-consumer and business-to-consumer trading platform, two months later. The rapid growth of ecommerce can be illustrated by the fact that the market in the United States more than doubled in its value between 1997 and 1998, growing from $2.6 billion revenue to $5.8 billion. An even more startling example is Amazon's growth in sales from just $16 million in 1996 to $1.6 billion in 1999. Even during the dot-com crash, ecommerce sales still increased. [3]

In the 2010s, ecommerce experienced rapid growth. In 2013, China exceeded the United States in online sales, making it the world's largest ecommerce market. China experienced a 33.3% growth of expenditure in online sales in 2015, the same year, the United States experienced a 14.6% growth. [5] The role of social media also became more prominent for businesses. Companies started to develop more direct connections with their consumers, to give a personal touch to their products, in an attempt to gain and retain more customers. [4]

2 Literature Review

2.1 UI Best Practices

To learn about the best practices in UI, I am drawing inspiration from two of the most famous UI-related books: Steve Krug's *Don't Make Me Think, Revisited* and Adam Wathan & Steve Schoger's *Refactoring UI*.

2.1.1 Takeaways from "Don't Make Me Think"

In case of Steve Krug's Don't Make Me Think, Revisited, the author focuses on usability and emphasizes that the principles he lays down may already be known by the reader, however they may not be actively applied during their workflow.

It is crucial to keep the design intuitive, so that the user can recognize what action they need to take without having to actively think about it. Using this approach can save the user both time and effort. We should aim to make the website self-evident and where that is not possible, we should aim to make it at least self-explanatory. [7] An example I could think of where self-evident approach may not work in case of an ecommerce site, is when we need to integrate a third-party service, such as payment or package tracking. In this case, we are dependent on a service that is not managed by us; thus, we may have to explain its usage in advance if it happens to have bad design or if possible, write some form of a wrapper around it, keeping the original business logic, with our custom UI (this however could be challenging due to legal reasons).

Users try to avoid reading and just skim through the websites as fast as possible with a goal in their mind. [7] For this reason, we should not turn our website into a reading assignment. This is often due to the consideration that clicking the wrong button has lower opportunity cost (potentially a few more clicks) than reading the website carefully (inherently time-consuming activity). The most important takeaway is to avoid ambiguity as users tend to click on the first button that appears to be matching their target. A classic source of ambiguity in case of ecommerce sites is between signing up and logging in. Our goal with a website is typically not to teach the user how to use it, since that is not what they come for, instead we must focus on keeping it obvious so that the user does not have to read a novel in order to find their target.

An important principle of design is that there is no need to reinvent the wheel, it is better to follow conventions where it makes sense. In some specific situations it can be beneficial to break conventions but in the vast majority the cases, it is better to stick to them. Creating visual hierarchy is a great way to group elements and draw attention to the most important parts of the site (e.g., by using a larger font or a color that stands out). This technique existed even before the Internet, for example in newspapers. Segmentation of the text is also important; people prefer reading the same text broken down to paragraphs instead of all at once. [7] This not only groups related thoughts together but also allows the reader to take a break, if necessary, between two paragraphs.

Krug calls against being verbose on the Internet. Needless words must be eliminated, since people typically hurry on websites, minimizing their time to read. Navigation on the Internet compared to the physical world can be more challenging, due to the lack of scale, sense of location and direction. This can be tackled to some degree by establishing site hierarchy. Some parts are displayed on every page of the website, typically those located in the header and the footer, e.g., the site ID (name and logo of the website), navigation section and utilities (such as the search bar, the login button or the about page). The page must indicate clearly where we currently are on the site. Sub-levels can add complexity to the site and designers of the website often fall into the pitfall of not giving enough significance to the lowest level, albeit the users spending an equal amount of time on the lowest pages as the home page. Displaying the current page is also really important to provide the user a good sense of location on the site and the possibility to easily return next time. [7]

The homepage has to establish the identity of the site. Relevant content is also important, since outdated content indicates that the site has been abandoned. Juggling between the balance of what the user needs and what we want to showcase may prove tricky, but it is essential to maintain usability while also encouraging the user to discover more. Once visiting the home page, the user needs to quickly understand what the page is about. For this purpose, Krug suggests using a tagline, which is a short description that describes what the site is about, typically next to the site ID. [7]

Throughout the book Krug emphasizes the importance of usability. Usability tests provide incredibly valuable insights from the users, that the designers and developers might miss throughout the development process. This process consists of a user trying the website and the development and design team (or a usability engineer) taking notes of user’s remarks and what they are doing on the website. It is recommended to conduct these tests periodically throughout the development cycle, to notice and fix design mistakes as soon as possible. The goal of this qualitative test is typically not finding all the flaws of the website but the most prominent ones. [7]

Krug starts off talking about UI design for mobiles by stating that the same basic principles apply here too. Development for mobile devices (responsive design) can be challenging, due to the space constraints. The most important features must be the most easily accessible ones. It is important to note that certain actions, such as hover, are not available on mobile devices. Due to these limitations, the user interface may not be as intuitive as on the desktop version, the goal is to make it easy to learn and memorable, so that the user does not feel discouraged opening the website on their phone. [7]

Unfortunately, it is common to forget about accessibility, but on a larger website, it is inevitable to include in the design. Krug mentions that the most common reasons why accessibility gets overlooked is: not being affected by a disability, not being in a social circle where people are affected by any disability, due to the extra work and compromises in the design. While it is true that adding accessibility features can be complex, it is not always the case. Simply adding alt texts for images, which indicate what can be seen on the image for blind users (and also serve as the text in the tooltip of the image), using the label element in case of forms to associate an input field with the appropriate text label and adding a high contrast theme are relatively simple steps that can be taken towards a more accessible website. [7]

2.1.2 Takeaways from "Refactoring UI"

Similar to Krug, Wathan & Schoger recommend functionality-driven UI design in their book. They also emphasize the importance of global thinking, and only focusing on the details in the later stages of the design. The authors suggest that designing too much in detail without having the ability to try the interface will cause frustration, so they suggest jumping straight to the implementation phase, once the designer is happy with the basic design, then repeat this process in cycles for all the different features. [8] I experienced the same while designing the wireframes, I realized, if too focused on the details, this process can be really time consuming, so I decided to limit the number of wireframes I create and try to minimize the time spent on them.

Wathan & Schoger also emphasize the importance of building a personality for the website, which is highly affected by the chosen font, colors, border-radius and the tone of the language. The authors also recommend setting some constraints to restrict the number of choices of colors, fonts, scaling, etc., allowing easier decision making and ensuring more consistency. [8]

The authors also bring up the importance of visual hierarchy. They mention that font size alone for this purpose is not ideal and offer other alternatives, such as font weight and color. Besides emphasizing important parts of the website, it is also important to deemphasize the less important details, to create a better contrast. Another issue pointed out is redundant labels, since many pieces of data, such as a phone number or an email address, already stand out to the user, therefore there is typically no need to label them. [8]

To avoid having a cluttered layout, the authors suggest adding plenty of whitespace while designing the component and then gradually reducing that to the level that feels appropriate for the element. They suggest this technique because it is easier to notice when whitespace has to be removed as opposed to when it needs to be added. Similar to the color scheme, a systema has to be created for the layout as well, to constrain the number of possibilities. [8]

The authors suggest taking a mobile-first approach, designing the mobile UI first and then readjusting that to wider displays. They also warn against needlessly filling up the screen, overusing grid layouts and using relative sizing in all situations. Wathan & Schoger also state that relative sizing does not immediately mean that the UI will scale appropriately and recommend a more fine-tuned approach observing what feels more suitable, with the basic principle that large elements should scale down faster while already small elements should scale down slower. [8]

When it comes to fonts, it is once again crucial to build a scale and stick to that. The authors discuss multiple approaches to build this scale, including simple mathematical ratios but in the end, they conclude that hand-crafted scales tend to be a better choice. It is also important to choose a non-relative measurement such as pixel or if we insist on choosing a relative measurement, we should use something that is relative to a fixed value (e.g., font size of the root element in case of rem) not something that is relative to the font size of a non-fixed value (e.g., relative to the font size of the immediate parent element in case of em). As for choosing the font, the recommended approach is to be on the safe side and select from popular fonts, since they have already been tested by the public. [8]

One common pitfall people fall into is spreading the text over a wide layout to utilize all the empty space; however, this ruins the reading experience on wider displays. Line spacing is also very important and has the purpose of clarifying which line comes next once the reader reaches the end of a line. Due to this, narrow content may have smaller spacing while wide content should have greater spacing. The same applies to font size, larger font sizes help with recognizing the next line, while smaller font size requires more spacing. [8]

While discussing colors Wathan & Schroger state their preference for HSL representation over RGB/hex, as it proves more intuitive to humans according to the authors. For building a color palette, the recommended approach is to find colors in three categories: grays, primary colors and accent colors, storing different shades of the selected colors. To come up with the shades, one must start with the color in the middle, find the darkest and lightest shades of it, then fill the remaining gaps on the palette. Instead of adjusting the lightness, we can also make use of the perceived brightness of the colors and adjust the hue based on that. [8]

Contrast between the text and the background allows the text to pop out and it is also important for accessibility. The easiest way to achieve higher contrast between the text and the colored background is by using a dark colored font on a light background or vice-versa. Color should only reassure what the design already communicates but it should never be the sole element to convey a certain piece of information. [8]

To achieve a raised/inset effect, one might mimic how light reflects on objects with raised/inset surfaces in real life. Simulating the light coming from above is a good idea, since most people stare at their screens slightly downwards. Shadows are a great way to position elements on the z-axis (since screens are two dimensional). Once again, the authors advise creating a systema, in this case for the different depth of the shadows. Ambient shadows (most visible during lower elevation) can be simulated by adding an additional tighter and darker shadow around the object. [8]

Depth can be conveyed not only through shadows and gradients. Lighter colors feel closer to the person while darker ones feel more distant. Using overlapping layers is another way of creating depth. [8]

Working with images raises some challenges. For example, making text on an image background readable. This can be achieved by adding a shadow with a large blur area around the text and lowering image contrast or adding an overlay in front of it. It is important to pay attention to the intended sizes of icons, if we scale up icons (even vector images) that were originally designed to be small, often times they end up looking too plain. [8]

The final part of the book focuses on final touches. The most important takeaway for me was to stay consistent, plan in advance smartly (do not over-plan but plan everything fundamental) and avoid common pitfalls. I found the book very helpful in solidifying the principles to consider while designing an application and only on rare occasions I encountered certain designs presented as better that may be up to debate.

2.2 Cloud Deployment and Security

2.2.1 Security Concerns

Common security concern customers have with ecommerce is about the confidentiality of their transactions, unauthorized use of their credit cards, fraud, misuse of their data and many others. There are four main requirements for data safety: [9]

- Privacy – only authorized parties should have access to the information.
- Integrity – the message cannot be altered.
- Authentication – the sender and the recipients must identify themselves.
- Non-repudiation – proof that the target received the message.

Kalamkar mentions two more important aspects: [10]

- Confidentiality – prevention of unauthorized access.
- Availability – the user should be able to access the resources they are authorized to.

Hussain also mentions access control in this list, stating that only authorized persons should have access to the resources, [11] which coincides with the definition used by Padmannavar for privacy. [9] Sun et al. also mention these principles in their own words. [12] Yang et al. also mention most of these principles while adding a few others, such as fine-grained access control, secure data sharing in a dynamic group, leakage-resistance and complete data deletion. [13]

It is important to comply with these principles throughout the data life cycle, which Chen and Zhao describe as follows: [14]

Generation → Transfer → Use → Share → Storage → Archival → Destruction

Chen and Zhao mention valuable considerations regarding cloud security, namely that the multi-tenant nature of the cloud makes it difficult to determine what resources have been compromised in case of a security risk. In case an organization uses multiple providers for provisioning different resources, it is difficult to unify security measures. The multi-tenant nature of the cloud poses a threat of unauthorized users accessing the data of other tenants. They also emphasize that cloud security measures must meet the needs of massive information processing. [14]

2.2.2 Common Security Threats

Hussain outlines seven main threats for ecommerce security: [11]

- Authentication Attacks – when an unauthorized user alters the system. This may happen through shared passwords or unattended open sessions. Brute-forcing the passwords on resources could be another way of getting access.
- Integrity Attacks – when the data is tampered with during transition. An example when this can be exploited is if a program does not check the buffer limit, allowing the attacker to add arbitrary data after the limit. This allows the attacker to slow down or completely shut down system by flooding it with data.
- Confidentiality Attacks – an example would be getting access to data using a packet sniffer program.
- Virus - malicious computer programs that are designed to replicate themselves and spread to other computers when triggered by a certain event. A network can get infected by a virus from an outside source. Once a computer is infected within the network, the other computers are also at a high risk of being affected.

- Trojan Horse – these malicious pieces of code are disguised as useful programs and require the user to download them. Once installed, the system of the user is compromised.
- Worms – malicious programs that spread through computer networks. They are designed to replicate themselves, e.g., by sending copies of themselves in an email attachment. They can create security risks by opening TCP ports, or flood the network, initiating a Denial-of-Service attack.
- Database Threats - some database systems store the username/password pair in a non-secure way, if an attacker gets access to these credentials, they gain access to private information in the database.

Al Ladan breaks down the security threats of ecommerce platforms to three different levels: [15]

- Client level
- Frontend server and application level
- Network and backend server level

One of the vulnerabilities on the client level is devices connected to wireless networks. Password-protecting the network can help mitigate the risks, however wired networks are still considered more secure. Possible attacks on wireless networks include: [15]

- Replay attack – the attacker can capture and retransmit a message, even to a different destination.
- Eavesdropping – the attacker can capture and read sensitive data if it is not encrypted.
- Pull attack (on cellphones) – the attacker has full control over the mobile device.
- Push attack (on cellphones) – the attacker plants malicious code on the mobile device which spreads that to other devices in the network.
- Lost device – although not an attack, it is still an important aspect to consider, that in case a device gets lost, malicious actors should not be able to access sensitive data.

Server exploits are the most dangerous threats on the frontend server/application level, wherein attackers can gain administrator access. Attacks the hackers can launch on this level include: [15]

- Buffer overflow – when the allocation of storage is not handled properly by a program, the attacker can make use of this vulnerability tricking the server into executing some malicious code.
- Software bugs – a common reason behind security holes is unskilled or careless developers, who leave mistakes in their code, creating a new vulnerability. Solving

this problem is not trivial, since ecommerce platforms should be interoperable with external systems, adding more complexity. Following standards could provide a solution, although it is worth noting that even the standards and protocols can keep changing rapidly, which makes staying up-to-date a difficult task.

- Viruses and malicious software – viruses and malicious software can infect ecommerce platforms various ways, causing significant losses, steal sensitive data or make the platform, or a part of it, inaccessible.

Networks are dependent entities; they depend both on the private network, which is often owned and managed by others (e.g., in a cloud environment), and the public network, where control over security is lacking. These factors contribute to security issues, where the most common issues are: [15]

- Session interception – the session can be obtained by an attacker, e.g., in case of a man in the middle attack, where the attacker places a malicious host between the client and the server.
- Cross-site scripting (XSS) – attackers can place malicious code inside the content served on a website. Because the browser detects the ecommerce site as a trusted source, the inserted code operates with the same permissions, which the attackers can exploit.
- Firewall loophole – firewalls separate the backend servers from the corporate network. Since they are typically implemented at a network level, they do not protect against attacks on a higher level (e.g., HTTP which is in the application layer). This means that, for example, a buffer overflow attack could be carried out in this layer.

2.2.3 Mitigation of The Security Risks

With the help of encryption, it is possible to address privacy concerns. [9, 14] One method for this is using PKI (public key infrastructure), which consists of a public/private key-pair, where the public key encrypts the message and the private key decrypts it. The public key, as the name implies, is publicly distributed, while the private key is kept hidden by the recipient. This technology forms the bases of the RSA cryptosystem and the PGP encryption. PKI is not fit for encrypting large amounts of data, for this reason it is typically only used for agreeing upon a key, thereupon using symmetric encryption. [9] Symmetric encryption can be really fast and have a low complexity [11, 14], allowing easy implementation, however it requires a shared secret that the parties agreed on previously. [11] Unfortunately, encrypting stored data is not always a viable option. In case of static data, it can lead to problems with indexing and querying. [14]

Although credit card details can be transferred through SSL safely, the server storing credit card details poses a security risk. MasterCard and Visa developed a protocol, SET (Secure

Electronic Transaction) [9, 15], which authenticates the three parties involved in the transaction, namely the bank, the merchant and the customer. This protocol prevents the merchant from storing any sensitive information on their server. There are three transactions happening during a credit card payment: [9]

- Credit card details transferred to the merchant or payment gateway
- Credit card details forwarded to the bank from the merchant
- Order and customer details provided to the merchant from the payment gateway/credit card company

Digital signatures serve as a way to verify the origin of online transactions. It is best compared to a physical signature. Besides providing proof of the origin, the digital signature also assures the recipient that the data has not been altered. [11]

Digital certificates allow parties to prove their identity. They use an electronic key to encrypt and sign digital information. These certificates are issued by a Certificate Authority and signed by the authority's private key. [11]

Smart cards are plastic cards with similar dimensions to traditional debit and credit cards. There are two main categories of these, microprocessor cards and memory cards. The purpose of these cards is helping with authentication. They are called smart, due to their capability of data processing and the algorithms stored on them. [11]

Electronic money is payment on the Internet resulting in money transferred from one account to another. The transaction contains data, encrypted by the merchant's private key, about the amount spent, a serial number, the identity of the buyer and expiry. The transaction is recorded to ensure that the same money is not spent twice. Once the issuer verifies the serial number, to ensure that the money is not double-spent, they add the amount on the merchant's bank account. [11]

2.2.4 Cloud Storage

Cloud computing is the on-demand offer of applications and resources as services over the Internet. The National Institute of Standards and Technology (NIST) defines cloud computing as "*cloud computing enables ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction*". [12]

NIST differentiates three service models (also called the SPI model):

- Software as a Service (SaaS)
- Platform as a Service (PaaS)

- Infrastructure as a Service (IaaS)

Based on access scope we differentiate three cloud categories: [12]

- Public – property of a service provider, it may be accessed by everyone.
- Private – property of a company, only accessible by authorized parties.
- Hybrid – a mix between public and private.

NIST differentiates an additional deployment model, called community cloud. [14]

Cloud storage allows the storage and sharing of data in the cloud. It has the advantage of unlimited space, convenience, safety, possibility of backups and low cost. Cloud platforms typically offer three types of storage: [13]

- Block storage – data broken down to fixed blocks and only reassembled once the user retrieves them. It is the default storage method on hard drives and can be used on Storage Area Networks (SANs). Block storage is not capable of storing metadata, making it unfit for unstructured data. It is also a popular choice for deploying virtual machine file system (VMFS). [16]
- File storage – data saved in files, e.g., network-attached storage (NAS) and even an ordinary hard drive uses file storage. It uses a hierarchical structure of folders. [16]
- Object storage – large volumes of unstructured data, all the objects are at the same level. It can also store metadata associated with the object. Each object can be accessed using a unique number. The files can only be managed via an API. [16]

Cloud storage consists of four layers: [13]

- Storage layer
- Primary management layer
- Application interface layer
- Access layer

2.2.5 Data Security in the Cloud

Data security has been a major concern in the IT field. Data security in the cloud is more complicated than in a traditional environment since the data is scattered over different machines and storage devices. Protecting the data of the users is crucial to gain their trust. [12, 15] While cloud computing can save time and money for a company, they must trust the service that they are using, since the most important asset of a company is the data that they manage. [12] Surveys from the early days of cloud support this: in a 2009 survey conducted by IDCI 74% IT managers and CIOs stated that the primary challenge for the adaptation of cloud computing was security issues. Another survey from 2009, by Gartner, further proves this, where more than 70% CTOs mentioned privacy and security concerns as the main preventive factor of cloud adaption. [14]

Sun et al. focus on four different data security aspects: integrity, confidentiality, availability and privacy. [12]

Data integrity refers to the protection of the data from unauthorized altering, fabrication or removal. In case of a standalone system, this is not a complex task and can be achieved simply using database constraints and transactions. Most importantly these transactions must have ACID (atomicity, consistency, isolation, and durability) properties. In a cloud context the key component of data integrity is properly managed authorization. To verify the integrity of the data, Proofs of Retrievability can be adopted, which uses a combination of error correction code and spot-checking. The HAIL (high-availability and integrity layer) builds on top of Proofs of Retrievability, which provides a way to check data in different clouds, ensuring availability and integrity. [12] Another technique is provable data integrity (PDI, developed by NEC Labs) [14], where the client fingerprints the data and uploads the fingerprint along with the data to the cloud platform, then whenever the client wants to check the integrity of the data, it sends a “challenge” to the server and compares the received data with the expected output. [17]

Data confidentiality is pivotal for users to store private and confidential data in the cloud. Sun et al. call against storing sensitive data in the cloud, even if encryption is in place, due to the possibility of human errors, such as improper key management, while also criticizing the lack of fine-grained authorization. [12] Chen and Zhao also emphasize the importance of key-management, while bringing up the issue that users lack expertise, which makes them rely on the key-management of the cloud provider, that in turn makes the cloud provider’s responsibility more complex due to the increasingly large volume of keys they need to manage. Access control is also crucial, the mobility of the employees is relatively high, meaning that systems should be able to quickly provision and deprovision accounts, to avoid unauthorized access by employees who left the organization. [14]

Homomorphic encryption ensures that the algebraic operations on the clear text remain the same even after encryption, which means that the text does not have to be decrypted first. [12, 14] This technique can help preserve the confidentiality of the data while undergoing data operations in the cloud. Fully homomorphic encryption allows any operation on the data without having to decrypt, however the computational complexity and the amount of storage it requires still makes it impractical as of 2014. [12]

Due to the inefficiency of full homomorphic encryption, researchers started focusing on limited homomorphic encryption, where only the most common operations, such as the search operation, are possible without decryption. In-memory database encryption utilizes a synchronizer which provides the key to the client, to decrypt the data it wants to access. The drawback of this approach is the delay that the synchronizer adds however this issue can be addressed by minimizing the communication between the nodes and the synchronizer. [12]

Distributive storage can help with ensuring the integrity of the data while also providing data privacy. One such technique is breaking the data into chunks, encrypting these parts and uploading them to different databases. With the help of tailored measurements, it is possible to adjust the provisioned resources according to the user's need based on the outgoing and incoming traffic. [12]

A hybrid technique can be applied to achieve both data confidentiality and data integrity. It utilizes both key sharing and authentication techniques. RSA can be used for the key exchange between the user and the cloud provider. A three-layer model may be followed, in which the first layer handles authentication, the second layer is responsible for encryption of the data and the third layer is used for data recovery. [12]

Data concealment is a technique to achieve data confidentiality, by mixing real data with fake data, which only the authorized users can differentiate. This increases the overall volume; however, it enhances security. Watermarking can be used to differentiate real data. [12]

Deletion confirmation means that once the user confirms that they want the data to be deleted, it is actually deleted and not recoverable. This means all the copies of the data have to be deleted at the same time, in a way that it is not possible to recover the deletion. Encryption also helps here, e.g., using the FADE (file assured deletion) system, data is encrypted before it is uploaded to the cloud and it assures that the file is unrecoverable after deletion. [12]

Data availability refers to the availability and recoverability of data in case of accidents (e.g., natural disasters or hardware issues) or network failure. Cloud computing adds a level of complexity to data storage, since cloud clients have to comply with local laws in the country where the servers are based. [12]

2.2.6 Data Privacy in the Cloud

The Canadian Institute of Chartered Accountants (CICA) defines privacy in the Generally Accepted Privacy Principles (GAPP) as *"The rights and obligations of individuals and organizations with respect to the collection, use, retention, and disclosure of personal information."* [14]

Privacy is the idea of control over one's private information and the ability to reveal information selectively. Sun et al. outline four main categories of privacy issues: [12]

- Enabling users to have control over their data and preventing data theft, selling and misuse.
- Avoiding data loss and keeping consistency during data replication.
- Clarifying the responsible party for the legal requirements.
- Finding the extent cloud providers are involved in processing the data.

Service abuse is an important issue in the world of cloud computing. A good example of this is deduplication, when the same data is shared by different users, and the cloud provider stores only one copy of it, to save on storage. Another form of service abuse is when the attacker tries to deplete the resources of the cloud provider, by overloading the resources, incurring extra costs for the client or the cloud provider. [12]

2.3 Analyzing Existing Ecommerce Platforms

This section focuses on three main ecommerce platforms: Alibaba, Amazon and eBay. The choice is not arbitrary, the intention was to showcase the differences between B2B (Alibaba), B2C (Amazon) and C2C (eBay) ecommerce sites. For the sake of completeness, a self-hosted platform (WooCommerce) will also be included. It is worth noting that while Amazon and Alibaba have many other services, the focus will be on their core websites: A mazon.com and Alibaba.com.

2.3.1 Alibaba

Alibaba was founded by Jack Ma in 1999 who served as the chairman and chief executive officer of the company until 2013. The company's growth coincides with the spread of internet in China. [18]

The Alibaba Group has many subsidiaries, including Chinese retail markets such as Taobao, Tmall, Juhuasuan, a Chinese wholesale market called 1688.com, the global version of the site called Alibaba and the retail version of Alibaba, AliExpress. [18]

Besides ecommerce sites, the Alibaba Group also has payment services (Alipay), a logistics information system (China Smart Logistics), online marketing services (Alimama) and even a cloud computing provider (Alibaba Cloud Computing). [18]

Alibaba assists small exporters, primarily based in China, in finding businesses who can source their products. While Alibaba does not have commission fees, they do make a profit by membership fees (membership determines the number of products sellers can showcase [highlight] and the number of RFQ responses [request for quotation] per month). A custom pricing model is used to determine the membership fees. [18, 19] Most of the company's revenue comes from advertising. According to data from 2014, Alibaba reported a gross merchandise volume of $296 billion, exceeding eBay and Amazon combined. [20]

A typical deal is made with the following workflow from the buyer's perspective:

Sign up → Set up business details → Find the needed products → Request a quote from the sellers/ask for clarification (e.g., about shipment) → Evaluate quotes and select the best one → (Optionally) Create a trade assurance contract if possible (if the seller is a Trade Assurance supplier, for handling disputes) → Pay for the product/deposit (in case the product is not readily available, the rest will be paid later). This process can be done through online banks, credit card, telegraphic transfer or Payment Terms (offered by Alibaba in the United States) with different transaction fees. → Shipping → Inspect the received products (optionally conduct pre-shipment product inspection) [21]

Shipping is handled by a freight service. Alibaba offers the following shipping methods: [21]

- Ocean Freight – buyers can choose from two container options, FCL (full container load) or LCL (less than container load), the former fit for high volume orders, where an entire container can be filled with the order, and the latter for low volume orders, so that the buyers can save money.
- Air Express & Air Parcel – a cost effective airborne shipping method that is available for products of suppliers based in China.
- Air Freight – also known as air cargo, it is designed for orders between 150 and 500 kg. It is a time-sensitive and more expensive shipping option. The shipping rate is calculated from the cargo's actual weight and dimensional weight, where the larger one becomes the chargeable weight.

Dimensional weight is calculated using the following formula: $(length \times width \times height) \div (dimensional\,weight\,factor)$, where the dim factor (dimensional weight factor) is the volume of package allowed per unit of weight. For example, if the minimum weight allowance per m^3 is 50 kg, then the dim factor can be calculated dividing the volume by the weight: $\frac{1\,m^3}{50\,kg} = 0.02\,m^3/kg$. If the order in question has a weight of 20 kg with the following dimensions (in meter): $0.5 \times 0.5 \times 2$ then the volume is 0.5 m^3. In this case, even though the actual weight is only 20 kg, the dimensional weight is $(0.5 \times 0.5 \times 2) \div 0.02 = 25\,kg$, meaning that rate will be calculated based on the dimensional weight.

2.3.2 Amazon

Amazon was founded in 1994 by Jeff Bezos. In the early days, the company was focused entirely on the sale of books, which is why it was headquartered in Seattle, due to the proximity of the Oregon book distribution center. Soon, Amazon grew in popularity and started selling other products. The company got publicly listed in 1997. Since 2000, they have allowed other businesses to use the portal for selling their own goods. [22]

The original vision of Amazon was to provide an online catalog. Bezos pointed out the dependence on the location, and the costs incurred by that, in case of traditional brick-and-mortar stores, and that is what he intended to eliminate, since technology was getting cheaper, as opposed to real-estate, which kept increasing in price. Albeit this statement by Bezos, during the early days of Amazon, later on, he ended up investing a lot, including in warehouses, to serve the market better. [22] Amazon's key resource is its technological infrastructure, which they built over more than two decades. [23]

Amazon operates with a low profit margin, which makes it attractive to buyers. This is possible thanks to the lack of a physical store and keeping a small inventory. Amazon.com accounts for more than 65% of Amazon's income (23% from third-party sellers) according to data from 2022. Amazon has over 8 million sellers worldwide. From the sellers' perspective, Amazon's revenue stream consists of advertising fees, which the sellers pay for product promotion, and commission on sales. Another source is Amazon Prime, a subscription-based service for buyers, which guarantees free two-day shipping, access to the video and music streaming catalog and many other benefits. Since Amazon has their own delivery service, it is also a potential source of their profit. Besides the Amazon Marketplace, Amazon also generates income from other services, such as Amazon Web Services (AWS), a cloud provider, Amazon Kindle, an e-reading service, Amazon Advertising and many others. Even though most of their revenue comes from Amazon Marketplace, AWS is Amazon's most profitable service. [23]

2.3.3 eBay

In 1995, French-born programmer, Pierre Omidyar founded AuctionWeb, which was rebranded as eBay in 1997. Although initially a portal for individuals selling items to each other directly, today eBay is not only a C2C but also a B2C platform. In 1998, eBay became a publicly traded company. In 2002, eBay acquired PayPal, a popular online payment platform, which they integrated into website as a payment gateway for purchases. The company bought Skype in 2005. [23] In 2007, eBay took a $1.4 billion write-down of Skype, essentially acknowledging that the acquisition did not go as planned, followed by divestment and selling the majority of their stake to an investor group in 2009, then selling their remaining 30% stake to Microsoft in 2011. [24]

Low fees were a key reason behind eBay's success. eBay offered various advantages compared to brick-and-mortar auction houses, e.g., lower commissions and geographic location, thanks to the fact that the site was online, was not a boundary for customers anymore. Besides commission, eBay is profitable thanks to advertising fees, listing fees, final value fees, eBay Plus (a subscription service that provides customers with benefits, such as free shipping) and other revenue streams from its subsidiaries. [25]

Final fees are a significant source of eBay’s revenue. The fee can widely vary, depending on the seller’s status. Sellers gain better status by selling a larger volume of products and by preserving good reviews. Advertising generates revenue on a per-click basis, if a customer clicks on the promoted ad, the seller gets charged a fee. [25]

2.3.4 WooCommerce

WooCommerce (along with Shopify) is one of the most popular ecommerce platforms for building an online store. It is actually a plugin for WordPress, with many integrated tools, developed by Automattic, the same company that owns and operates WordPress. Its popularity can be illustrated by the fact that it has more than 5 million active installations. [26] WooCommerce attracts people with more expertise, as opposed to Shopify, due to its steeper learning curve. The reason why the focus is on WooCommerce instead of Shopify is due to its open-source, highly customizable nature, and the fact that it can be self-hosted, [27] unlike Shopify. [28]

While WooCommerce can be self-hosted, they do offer hosting solutions, which is one source of their income. Another source is the WooCommerce marketplace, where custom themes and extensions are sold. For shipping, WooCommerce offers a free extension with no fees, with basic functionalities, however they also offer more advanced paid shipping extensions developed by themselves. They also offer their own payment gateway, which has no subscription fee but a fee must be paid after every transaction. [29]

From a security perspective, WooCommerce warns that installing third-party extensions and themes can introduce new vulnerabilities. There are also security related extensions, which can take care of automated backups, provide brute-force attack protection, malware and vulnerability scanning, detailed logging and anti-spam technologies. An SSL certificate is also a basic but inevitable requirement of a secure site, which typically hosting platforms provide for free. WooCommerce also offer software development support for their customers at a paid rate. They also provide a list of trusted developers and agencies that can help with development (possibly another source of their income via commissions). [29]

3 Evaluation Criteria

Before creating a detailed specification, it is important to set the evaluation criteria of the website, that is technical criteria that the website must follow. These criteria consist of functional and nonfunctional requirements.

3.1 Nonfunctional Requirements

Nonfunctional requirements are the description of properties or characteristics that the system must display or adhere to. [30]

3.1.1 Performance

With today's processing power performance is only a major issue when too many concurrent processes are running or the time complexity of the running algorithm is too high. For this reason, the main performance requirement of the website is to use algorithms with low time complexity wherever it is possible and avoid using too many microservices. Optimization from time to time is also key to reaching higher performance.

3.1.2 Security

As the literature review concludes, security is paramount for an ecommerce site. Dependencies must be kept up-to-date to patch the known vulnerabilities. User entered data must always be handled with extra caution and as a potential threat. Sensitive data such as password must be protected appropriately, for example by hashing them or even adding salt to them first, to prevent the possibility of cracking the hashes of common passwords.

The principle of least privilege must apply and wherever it is feasible, the components should be kept within a private network. Appropriate password requirements must be enforced, to prevent brute-force attacks. The user must be authorized every time they take an action.

3.1.3 Usability

The interface should be intuitive and easy to use. Later on, this could be evaluated by conducting a usability test. Usability should come before design as a priority. Navigation throughout the website should be self-explanatory, returning to a previously visited page should not cause any trouble for the user. The website should be at least somewhat responsive; however, a fully-fledged mobile interface is not a requirement.

3.1.4 Availability

Since we assume that the website is targeting consumers all around the world, it is important to guarantee high availability. For this reason, the website should be served from a reliable cloud platform. As an example, Amazon Compute's SLA guarantees 99.99% uptime (which means a maximum downtime around less than 5 minutes per month) for EC2 instances deployed in multiple availability zones in the same region and pays back a certain percentage in service credits if they fail to adhere to that. [31] Using a cloud platform like AWS will enable easy scalability, both vertically and horizontally.

3.1.5 Extensibility

The backend should provide an API with endpoints for user and admin actions. Wherever possible a modular approach is preferred. The whole system needs to be designed with future extensibility in mind.

3.2 Functional Requirements

Functional requirements are the description of behaviors that the system must follow under certain conditions. [30]

3.2.1 User Registration and Authentication

Users should have the ability to sign up and login to their accounts. In case they forgot their password, they should be able to reset it, by receiving an email with a special link where they can set a new password. Once the user is logged in, they need to receive a token that authorizes them to take certain actions (e.g., modifying their data or placing an order).

3.2.2 Product Catalog

Products should be categorized and be displayed in the product catalog. Promotions from the product catalog are visible on the home page, while the product catalog can be browsed by either opening the page of a category or performing a search using a keyword. The product cards should provide some basic information about the product, such as their name and a thumbnail.

3.2.3 Product Page

The product page should display information about the product, such as title, description, price, number of items left in stock and quantity to order with an option for the users to put it in their cart. The page should prevent the user from putting more of the product in their cart than what is available in stock.

3.2.4 Cart

The cart should display the products and the quantity of them that the users put in there. Here, the users have the ability to check out and place their order. The user should be able to easily discard items from the list that they no longer want to buy.

3.2.5 Checkout

While placing the order, the user should be prompted for their personal information. Once the order is placed, the user should get a confirmation email about their order.

3.2.6 Search Engine

At least a very minimalistic search engine should be available that is capable of carrying out a search among the product titles and display the results in a formatted manner. The search engine should also look for the keyword within the description but the results should be sorted by relevance (first the products where the keyword was found in the titles, then those where it was found in the description).

3.2.7 Order Tracking

The latest status of the order should be summarized and displayed (e.g., processed, shipped, delivered, etc.). The different statuses (e.g., statuses sent by the courier) along with their timestamps should be trackable on the page of the order.

3.2.8 Admin Page

For the sake of simpler product and order management, an admin page could be created. This page facilitates all the actions that would otherwise be done directly in the database. The admin page should have access to basic CRUD operations on the tables in the database, for example adding, deleting and updating products, deleting users and orders and updating the statuses of orders.

4 System Design

4.1 Frontend Website

As suggested by many, I decided to start with the frontend part of the website. I came to this decision for several reasons, namely to discover the functionalities I intend to incorporate in the website, to have a visual idea of what I plan to achieve and also due to the sheer depth of frontend development, I realized this part might take the longest to develop.

4.1.1 Chosen Frontend Technologies

Choosing the right technology for a website can be a challenging undertaking. The vast number of technologies competing each other makes it especially difficult for the developers to settle for one. I decided to approach the question more generally than gradually get more specific.

There are many frontend frameworks out there in the market, such as React, Vue.js, Next.js, Svelte, etc. According to StackOverflow trends, React is the most popular framework, followed by Angular and Next.js. [32] Google Trends outlines similar statistics. [33] Since I have experience in Angular, I chose this framework for my website. The strongly typed nature of TypeScript helps with building more stable apps, with the slight disadvantage of the need for compilation.

Once I selected the framework, I looked for a UI component library. After some research, I narrowed down the potentials to 3 libraries: PrimeNG, Angular Material and Taiga UI. Due to my work, I have experience with Angular Material, however I am not too satisfied with its appearance, since it tends to give the website looks akin to Google products. Taiga UI seemed quite promising; however, it is still in an earlier stage of its development and has a smaller userbase, which also means less documentation. With PrimeNG, I had some experience, thanks to personal projects and also work. PrimeNG offers the widest range of components and although there are reports of occasional bugs with some components, I decided to choose this framework, since it is easy to use and visually appealing.

For styling, I contemplated using Tailwind, since it allows rapid development speed, however it also has its own learning curve and I figured that without a strong CSS background, Tailwind may actually cause more problems than it solves. For this reason, I decided to proceed with the project using pure CSS and potentially PrimeFlex, since it is a utility library optimized for PrimeNG.

4.1.2 Wireframes

Once settled with the frontend technologies, I decided to create a blueprint of the website I was imagining. Wireframes seemed perfect for this purpose. I decided to use Figma, since I had experience with it from before and it is an easy-to-use tool for designing. I created the design of different pages, namely the home page, the page of a subcategory, the page of a product and the page of the items added to the cart. Some constant elements include the header and the footer, which are displayed on all pages.

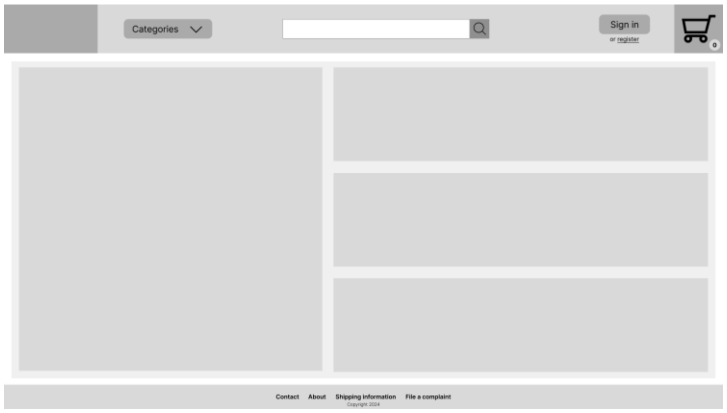

Figure 4.1 - Home page

Figure 4.1 - Home page depicts the home page of the website, which includes some promotions, the header and the footer. The header includes the site id (the logo of the website with a link to the home page), a multi-level dropdown for selecting the appropriate category and subcategory, a search field for looking up products, sign in and register buttons and the user’s cart. The footer includes the most typical links for an online store, such as contact, the about page, shipping information and a page for handling complaints.

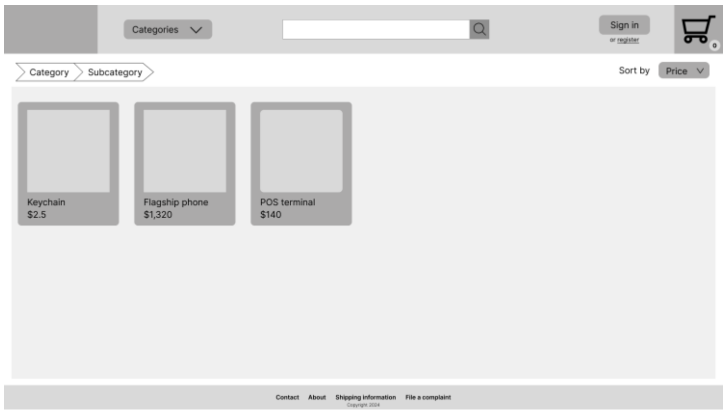

Figure 4.2 - Subcategory Page

Figure 4.2 - Subcategory Page includes breadcrumbs, that show which category and subcategory the user selected, product cards with the names and the prices of the products on them.

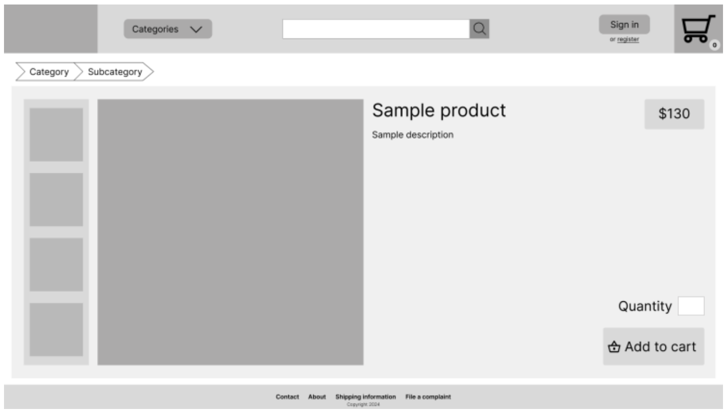

Figure 4.3 – Product Page

Figure 4.3 – Product Page shows images of the product, along with a photo carousel, name and description of the product, the price of it, the quantity to order (which defaults to 1) and a button to add the required quantity to the cart. It also includes the breadcrumbs that show the category and the subcategory of the product.

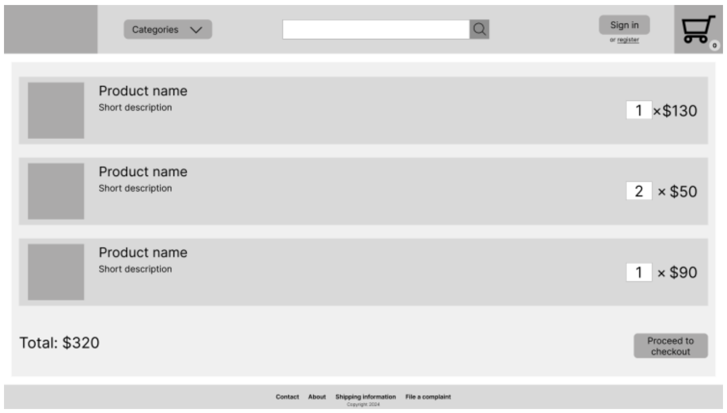

Figure 4.4 - Cart Page

Figure 4.4 - Cart Page shows the items that the user has added to the cart. A thumbnail is shown of the item, along with its name, a short description, the price and the quantity the user specified and a proceed to checkout button, which once clicked takes the user to the page where they are prompted for their personal information and shipping address.

4.2 Database

4.2.1 Chosen DBMS

In order to store persistent data efficiently a database is inevitable for an application. Since I am not dealing with large volumes of unorganized data, I decided to use a relational database. The next step was selecting the database management system (DBMS). In the past, I had done research on different database management systems and at that time came to the conclusion that PostgeSQL and MySQL are the best choices for most use-cases. These two systems were my chosen options, thanks to the fact that they are free and open-source, scalable, they boast a really large user base and both systems rank high in performance benchmarks. Considering that there is no huge gain from choosing one over the other, I made my decision based on my own observation of tendencies, namely that PostgreSQL is gaining traction over MySQL.

4.2.2 Schema

Once the DBMS has been chosen the next step was to create a database schema. I tried to determine what possible entities may be needed for an ecommerce website and came up with an initial schema. Unsure about my schema, I decided to revisit it once I already designed the initial state of the frontend. I decided to wait, to gain first-hand insights about possible missing entities or unexpected constraints that I might have failed to consider. Waiting has proven to be a good decision, because in the end I came up with a more refined schema, with a lot of considerations.

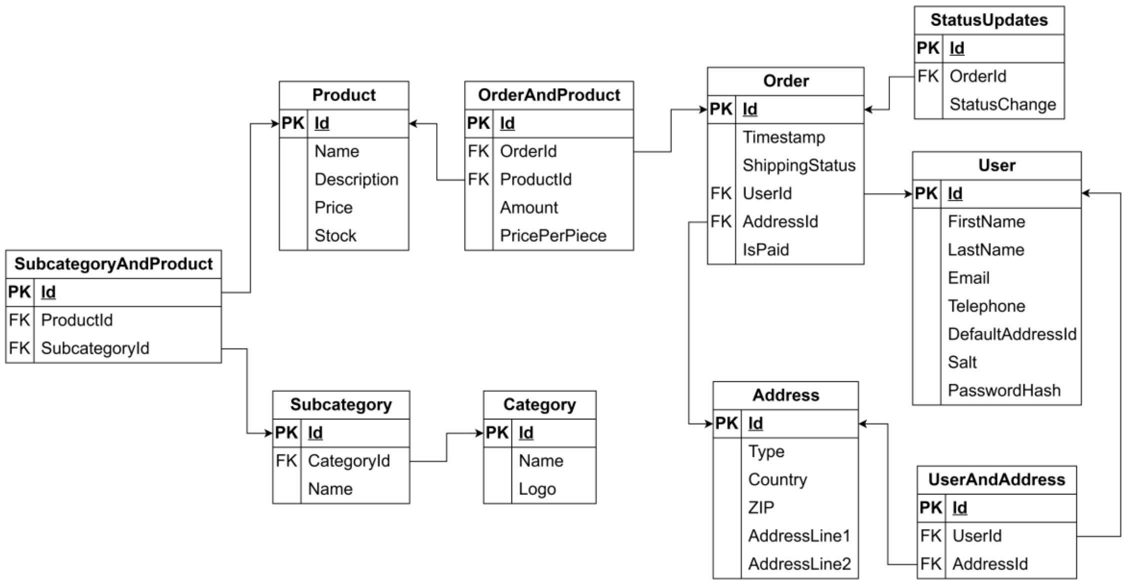

Figure 4.5 ERD of the initial state of the schema

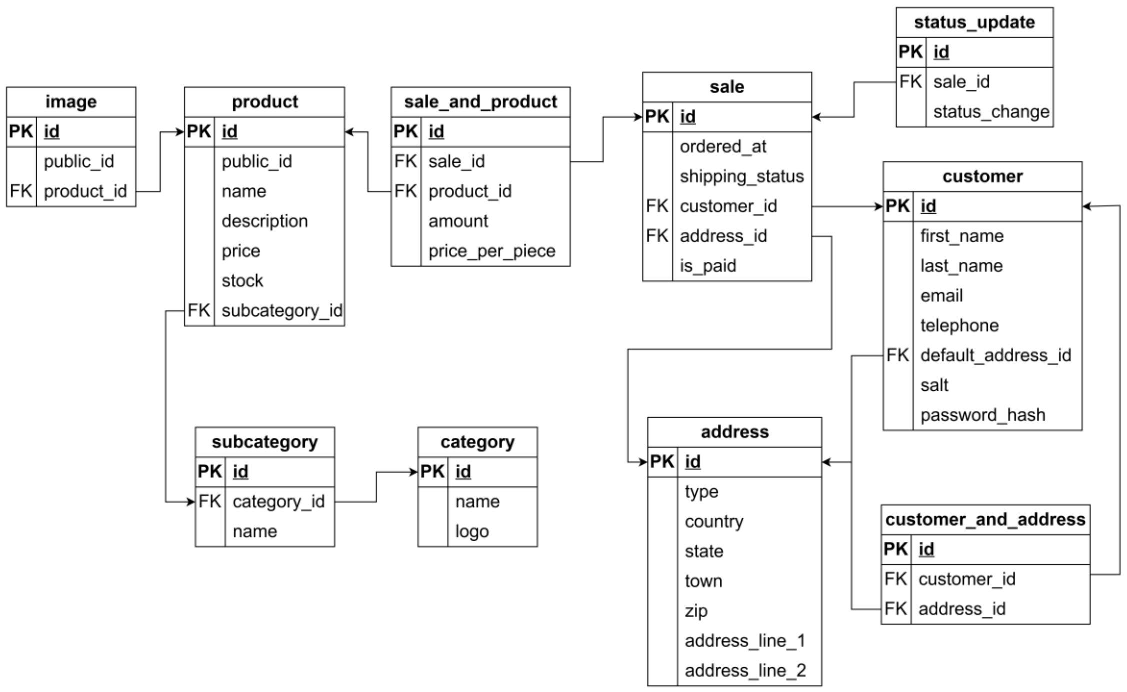

Figure 4.6 ERD of the improved version of the schema

As seen in Figure 4.5 ERD of the initial state of the schema, I used title case, which I later regretted, since I find that the case-insensitive nature of SQL renders it useless, so I replaced it with snake case, which is more appropriate for table and field names in my opinion. In the newer schema, Figure 4.6 ERD of the improved version of the schema, I also managed to eliminate the SubcategoryAndProduct connector table, since I realized I should make one product belong to only one subcategory, to keep the site simple (this becomes obvious from looking at the wireframes, since the breadcrumbs on the product page points to the appropriate subcategory of the product, which is only possible to implement, if a product only belongs to one category). An Image table has been connected to the Product table, so that images can be associated with the products. I also paid attention to more subtle details, e.g., not using reserved words such as order and user as the names of the tables.

4.3 Backend API

The purpose of the backend API is to securely enable data retrieval and mutation in the database. The API can be broken down to two parts: private and public API. The public part of the API is accessible by everyone, good examples to this is retrieving products, categories or the login endpoint. The private API is only accessible by authenticated users, (e.g., the endpoint for retrieving orders) or admins (e.g., creating products in the database.

4.3.1 Public Endpoints

Public endpoints that need to be created, along with the action that needs to be taken:

Customers

- Create customer – create a customer in the database.
- Authenticate – authenticate the user using the combination of the user’s email and password. Issue a JWT token which can be stored by the user’s browser.

Products

- List products – list all the products or filter them by a keyword if any was specified. Further filters and sorting can be implemented, based on price or amount in stock.
- Product – get information about a specific product.

Categories

- List categories – list all the categories.
- List subcategories – list all the subcategories.
- List products in category – list all the products within the specified category.
- List products in subcategory – list all the products within the specified subcategory.

4.3.2 Customer Endpoints

The following endpoints are specific to authenticated customers:

- Forgot password – send an email to the user that allows them to reset their password in a limited timeframe.
- Reset password – the endpoint that can be used by the unique link sent in the password reset email.
- Place order – purchase products bundled together in an order.
- List orders – list all the orders associated with the user.
- Order – shows information about an order placed by the customer (including shipping updates).
- Add address – add a new address.
- Update profile – update information such as first or last name, phone number or default address.
- Update email – update the email address of the authenticated user while prompting them for their password for security reasons.
- Update password – update the password of the authenticated user, while prompting them for their old password for security reasons.
- Logout – revokes the previously issued JWT token therefore logging out the user.

- Delete customer – deletes the user’s account.

4.3.3 Admin Endpoints

These endpoints should not only be private but also only users with specific authorities should be able to access them. The endpoints could be called from the dedicated admin page to execute CRUD operations on the different tables. These operations require careful planning, since the tables are related to each other, meaning that a simple deletion or addition could end in a foreign-key conflict.

5 Implementation

5.1 Frontend Website

I started the implementation of the frontend website by setting up the appropriate environment. I reviewed whether I had the necessary programs installed, such as Angular CLI version, Node.js and the npm package of PrimeNG. Initially I focused on my biggest weakness, which was CSS, by practicing it independently of the project and later on gradually building components within the project.

Basic responsiveness (responding to zoom and different screen sizes) was an important aspect in development process, however my initial approach to adjust the size of the components based on the screen with and height, later on using viewport, proved to be inaccurate. Later on, I realized I could use rem as a unit of measurement, which sets the size of the components relative to the root font size. This resulted in significant improvement in responsiveness in case of zooming albeit this alone does not make the website mobile-friendly. Since building a working website first was more urgent, I decided to lower the priority of making it mobile friendly and focus on the development of the main functionalities of the website and only once that is finished, then make the website responsive. Typically, it is recommended to take a mobile-first approach and develop the mobile version of the website first (since it is easier to create a desktop version from there, unlike the other way around), however in my case the main focus was on the desktop version.

Unsurprisingly, due to the fact that I did a lot of trial and error, the folder structure got cluttered quickly, which I decided not to bother, until I reach a stable state of the website. Once I designed the subsections that appear on every page (the header and the footer), I decided to create a systema for my components. I decided to hierarchically and logically group the components together, e.g., the components used by the navigation bar would appear under its folder (unless it is used within other components too, in which case it should be placed in a folder designated for shared components). Although I was not sure whether I can settle for this structure for the entire duration of the project, having organized the folder structure significantly simplified overviewing the project.

5.1.1 API Client

After the foundation had been laid down, it was time to load some data to test the different pages. Initially, I loaded the data from JSON files using a test data loader service. This way, I could work on the design on the website while the API was still in development. Once the API reached a working state, I decided to create an API client that is responsible for communicating with the API. I could reuse the model classes that I created during the testing phase (e.g., Product, Customer). The API client just stores the base URL of the API, injects an HttpClient and is mainly used other services as a base class. Different components each have their own service (e.g., ProductService, CustomerService). Each endpoint in the backend API has its corresponding method in one of these services which together form the client API. These methods return observables, to which we can subscribe inside the components that call the methods.

5.1.2 ImageService

The ImageService is responsible for retrieving the images of different products. The service stores the endpoint of the S3 bucket where the product images are hosted. Since each of the images has their own unique public ID, we can use that to retrieve them by simply appending “.jpg” to the end of their IDs. In case a product does not have any image associated with it, the service also has an option to show a blank image which displays a typical placeholder image.

5.1.3 Observables

Handling observables was one of the most challenging part of frontend web development. With the help of RxJS, it is possible to make an observable depend on another, reducing the need for a complex chain of subscriptions. Albeit the robust framework RxJS offers for observables, mastering it is a challenging task. Communication between different components can become difficult over time, to overcome this challenge, a SubjectBroken can be created, that stores different subjects which can emit their values as their values change and other components can subscribe to them. Normal observables are not multicasting, whereas subjects are, enabling us to subscribe to them from multiple components.

5.2 Database

As I mentioned previously PostgreSQL was the database of my choice.

5.2.1 Primary Key

An important question to consider was the ID to be used as the primary key, either a sequential ID (INT/BIGINT) or a UUID. The biggest downside of UUID is performance, which is partly due to the fact that UUIDs are not sequential. This issue was addressed in UUIDv7, however that version of UUID is not supported by the latest version of Postgres (version 17) as of October 2024.

One of the biggest advantages of the sequential integer IDs is readability and the fact that they are much shorter and easier to remember, which helps with testing and debugging. UUIDs are more typically meant to be used in NoSQL databases, where larger chunks of data are grouped together in objects. For these reasons, I decided to settle for BIGINT as my primary key type.

5.2.2 Permission Management

Initially I used the superuser (postgres) to create and manage the database. Later on, I revisited this and created a user (shopadmin) with CREATE rights. Then as the shopadmin user created the database (ecommerce) and everything (tables, sequences and functions) within the database. This way, since the shopadmin user is the owner of all the entities, it has access to everything within the database, but nothing outside of it.

5.2.3 Password Security

Passwords should never be stored in the database. Instead, a cryptographic hashing function should be applied to the password the user enters and the digest should be stored in the database. I decided to use SHA256, since other common hashing algorithms such as MD5 and SHA-1 are both proven to be affected by collision attacks.

To enhance security, I decided to generate salt before the hash calculation, append the salt to the password and calculate the digest after that. Following this, the hash and the salt would both be stored in the record of the customer. I incorporated this in a function called create_customer, which is responsible for creating a new customer record. A check_password function has also been created, which can authenticate the user based on their email and entered password. This is carried out the same way as the password generation, salt is appended at the end of the password, the whole text is hashed and compared with the actual hash.

5.2.4 Product ID Obfuscation

Following the principle of security through obscurity the ID (for example a product id) should not give away details about the database (e.g., the number of products stored). For this reason, if a sequential ID is used, preferably that should not be displayed in the URL of the frontend. The same problem would not arise with UUID (especially before version 7) since it does not retain sequence information.

To solve this problem, I decided to introduce two new fields in the product table. The first field is called public_id, the public identifier of the product, a 12 character long randomly generated base32 text. The generation of this id is handled by a custom user-defined function called generate_base_32. This function attempts to generate a base32 ID of a given length (by default 12) until it successfully finds a unique ID. Using a simplified form of the birthday problem, we can approximate that it would take about $\sqrt{M}$ products, where M is the number of possibilities, before we face a 50% chance of collision during ID generation. Plugging in 32^{12} for M, we get $\sqrt{32^{12}} = 1073741824$, which means it would take more than a billion products to reach that point. This is an acceptable compromise between trying to keep the length of the ID short while also ensuring that the system is capable of handling larger volumes of products.

5.2.5 Automation and Image Management

During development, the database may have to be recreated numerous times. For this reason, I decided to automate this process. Initially I did this manually running sequentially multiple SQL files, to create the shopadmin user, the ecommerce table as the shopadmin user and the necessary tables/functions, then populate those tables. As the complexity grew, I realized I should further automate this process. I created a simple CMD script that can easily be rewritten in Bash too if needed, using the psql command, to run the necessary scripts sequentially. Previously I used psql commands, such as \c within the SQL scripts, however since this is not allowed when loading SQL scripts using the psql command, I had to add additional psql commands for changing the user or the database. In the SQL files I also made use of prepared statements and functions to further help with the population of the tables (e.g., in case of customer table, since the salt and the password are connected, the users must be created through a function).

After the simpler part of the automation has been done, I decided to populate the image table by uploading images from a specific folder to an Amazon S3 bucket. I felt like PostgreSQL is not the right tool for this purpose, so the image uploader script has been written in Python instead while still interacting with the database. The folder has images with names following the format {product_id}_{sequence}.jpg, where sequence stands for the sequential id of the image to distinguish images of the same product. The script iterates through the images in the folder and extracts their product_id. Then, a unique 20-character-long public_id is created for the image and the product_id extracted from the name connects the image to the appropriate product. Once the record has been created, the image is subsequently uploaded to the specified S3 bucket. The public URL to this S3 bucket is then used by the frontend to load the images (once their public_ids have been retrieved using the backend). The whole process is really convenient, since it typically takes less than 15 seconds.

5.3 Backend API

In order to do CRUD operations and dynamically load the data, an API is needed. The website will facilitate the API the user's visit, namely for retrieving the products, logging in, placing orders, tracking orders and logging out.

5.3.1 Choosing The Technology

The first step was to settle for a technology. Since I only have experience in a handful of languages and not all languages are suitable for building an API, I had to make a decision between Python and Java. Python offered two main libraries, Flask and FastAPI. I used the former numerous times, but had no experience with the latter. Since FastAPI is still a recent technology, which has not even reached its first stable release (currently at version 0.115 as of November 2024), I decided to rule that library out.

Even though, lately I have been more involved in Python, from my perspective, an object-oriented language like Java seemed more fit for an API for a number of reasons. In my personal experience, I found that projects written in object-oriented programming languages are generally easier to navigate in. Java Spring Boot also boasts a robust environment that is highly specialized for building APIs. Another important aspect is serialization, which is easily achieved through attributes in Java, without much manual work. For these reasons, I decided to choose Java Spring Boot for my backend.

5.3.2 Model Classes

The backend API has to communicate with the Postgres database; this is achieved with the help of Java Persistence API and PostgreSQL's JDBC Driver. First, I had to ensure the most important base functionality: loading the product catalog and product pages. Java Spring Boot offers a robust range of tools to easily load data from the database into the application. By creating the entity class, with the appropriate names as annotations (if the variable/table name differs from the one in the database), the data loading process is quite straightforward. It is also important to mark the primary key with the appropriate attribute; this is not only important for identification, but also because of key generation, as in case of my project, the key is always automatically by the database.

Another challenge to tackle is to mark the relations between the different tables. In case of a table with a one-to-many relationship, the table must contain a collection of the related table as a member of its class, with the appropriate foreign key in an annotation. The opposite is true in case of many-to-one relationships, where the table has to contain one instance of the related table as its class member. As one could infer, this means that in case of many-to-many relations, both tables need a collection of tables as their class members.

Instead of function-based folder structure (e.g., grouping the repositories, services and models together), I opted to use a feature-based folder structure (e.g., CustomerRepository, CustomerService, Customer all go in the same Customer folder). While both approaches have their own pros and cons, for a larger project such as an API for an ecommerce website, I found the latter to be more suitable.

5.3.3 Repositories

In general, repositories were the easiest component to create, due to the lack of much logic involved inside them. This is thanks to the fact, that a JPA provides a lot of abstraction with its built-in features, so much so that simpler queries can be automatically built by simply declaring method names with the appropriate keywords, e.g.: for finding a Customer by email, one could write a method: *Customer findByEmail(String email);* and JPA would automatically build the query once the method is called. This, of course, only applies to simpler queries, for more complex queries native queries can be used using the Query annotation. With the help of native queries, it is also possible to call functions inside the database, this way outsourcing part of the logic (and thus some of the computational overhead) to the database.

5.3.4 Services

Services are where most of the business logic happens. In case of the core features of an ecommerce website, we cannot talk about really complex logic (with the possible exception of authorization and authentication), however some trivial logic was still written within the services. One example would be the product search functionality, which I achieved with the help of regular expressions (matching case insensitively, using a fallback strategy, by showing those products first where the keyword can be found in the title and then those products where it can only be found in the description).

While reading data is usually straightforward, inserting can become trickier due to the dependencies of the tables and automatically calculated values. An example of this is customer creation, where a salt needs to be generated and the hash value of the password + salt combination needs to be calculated and subsequently stored, however the logic of this has been outsourced to a function within the database.

5.3.5 Controllers

Controllers are responsible for directly processing the requests and returning the appropriate response to them. Parsing to JSON is handled by Java Spring Boot internally. With the help of Lombok, I created getters for all the models. In some cases, including all the attributes during serialization/deserialization can cause issues. In this case, I used JsonIgnore or JsonBackReference in case of foreign key relations, which makes the engine exclude these attributes from the serialization process.

5.3.6 Authentication and Authorization

Authentication and authorization are one of the most complex parts of the API. They are implemented with the help of Spring Security 6 and the JJWT API (for generating JSON Web Tokens).

By implementing security, the API has been split to two parts: public and private endpoints, with /api/public/, /api/user respectively. The reason for choosing user instead of private was to enable possible future extension for admin authentication, where admin actions, such as user and product creation/modification can only be conducted by admins.

In the configuration, CSRF protection has been disabled, since it is an issue that only affects stateful application, but since this ecommerce application is stateless thanks to JWT, it is unnecessary to enable CSRF protection. Everyone is allowed to access the public part of the API (/api/public/) while all the other parts of the API require authentication.

A custom authentication manager has been created to enable password-based authentication with the help of an authenticator function in the database. The authenticator is called from the login controller. It first ensures that both the username and the password have been provided, then attempts to authenticate the user. If the authentication succeeds, a new UsernamePasswordAuthenticationToken is issued, otherwise the method throws an exception.

A JWT filter has been created, which checks all the requests on the private endpoints. Its purpose is to extract the JSON Web Token from the authorization header of the request, then extract the claims (such as username, expiry date) from the token, then call the JWT service to validate the user based on the claims, which returns an authentication token if the authentication succeeded. After successful authentication, the SecurityContextHolder's context is set to the authentication token (so that while the request is processed, the user can be identified) that the JWT service returned. Following this, the request goes through the rest of the filter chain, then reaches the appropriate controller.

Users are managed using an implementation of the UserDetails interface (CustomerDetails). This class helps with the retrieval of the user's authorities, username (in my implementation email), password (in my implementation the hash associated with the user) and other important properties, such as whether the user's account locked or expired. Users can be found using UserDetailsService (CustomerDetailsService), which normally manages the UserDetails objects.

6 Deployment

In case of an ecommerce application, the project does not end after development. If we decide to handle deployment on our own, it often requires complex underlying infrastructure. Thankfully, we can facilitate many cloud providers that provide Infrastructure as a Service (IaaS). One such cloud platform is AWS, which is my chosen platform, due to the fact that I have most of my experience with this platform and the two other major cloud platforms (GCP and Azure) do not offer anything over AWS that would make switching to these platforms worthwhile in my case. In this section, I will present how the application can be deployed on AWS.

6.1 Infrastructure as Code (IaC)

Building the infrastructure manually may look like a good idea in theory, however issues arise when something goes wrong or the company decides to move to another platform. To properly document infrastructural changes, it is better to use IaC, so that the changes can be tracked, monitored and even supervised. The most popular IaC technology today is Terraform, which is a cloud-agnostic tool for provisioning infrastructure. With the help of Terraform, we can easily build, scale and dismantle the infrastructure as necessary.

6.2 Setting up the Infrastructure

6.2.1 Setting up AWS and Terraform

For the sake of simplicity, let us assume that we are working with a brand-new account on AWS. The first step is to enable multi-factored authentication (MFA) for the root user account. As of November 2024, this step is mandatory as it should be, since the lack of MFA poses a great risk, due to the amount of leaked credentials online. The next step is to create an IAM user for Terraform and copy its credentials. This is the recommended approach, to avoid accidentally leaking the root credentials, which would pose much greater risk to the user's account, since the root account controls all the IAM accounts. Once the credentials have been copied, AWS CLI needs to be installed. Once installed, a profile needs to be set up for the Terraform IAM user along with its credentials. The simplest way to test whether the setup succeeded is by running *aws sts get-caller-identity*, which returns information about the active AWS CLI profile. If this step was successful, Terraform can be installed.

6.2.2 Initial Infrastructure

After the AWS credentials have been stored, Terraform should work out of the box. AWS separates its resources by regions and inside regions availability zones. We need to select the region we would like to create our infrastructure in. From the available regions Hungary lies closest to Frankfurt (the only Central-European region), thus our selected region should be eu-central-1.

The first step to set up the infrastructure is creating a virtual private cloud (VPC). Once the VPC has been created, the next step is subnetting. We will need to create private and public subnets in each of the availability zones. The region eu-central-1 has 3 availability zones (eu-central-1a, eu-central-1b and eu-central-1c), so all together there will be 3×2 subnets. As an example, the VPC could be created with 10.0.0.0/16 CIDR and the subnets inside could go from 10.0.0.0/20 to 10.0.80.0/20 (with 4094 possible hosts each), leaving the 10.0.96.0/19 and 10.0.128.0/17 CIDR blocks unallocated for possible future needs for further subnets.

After subnetting, an internet gateway needs to be created and attached to the VPC. Then, two route tables need to be created (public and private) and in the public route table traffic needs to be routed to the internet gateway. If the private subnet needs outbound traffic to the internet, a NAT gateway could also be created in that subnet.

6.2.3 Database

To set up the database, we can create a database server in RDS. In case of this project, we would need to select PostgreSQL for its engine. Benchmarking should be carried out to determine the necessary instance type. The password for the server can be generated by AWS and stored in the secrets manager. Another alternative, is to manually enter the password and store it encrypted in the SSM Parameter Store. For the sake of security, its best to place the database in the private subnets. Setting up regular backups is also crucial to avoid accidental data loss. We can also enable “Auto minor version upgrade” to automatically receive the latest security updates. Major upgrades could be carried out using blue-green deployment to avoid downtime. To ensure high availability, multi-AZ deployment should be used with a read-replica in another availability zone.

6.2.4 EC2

While microservices are steadily gaining popularity, I opt to use a more traditional architecture for running the frontend and the API servers. Two autoscaling groups will be needed, one for the frontend server and another for the API server. The triggers to scale out can be set up based on computational demand (e.g., once CPU usage reaches 75% for a longer period). The autoscaling group requires a launch template. In the launch template, we can specify an Amazon Machine Image (AMI) as our base image (e.g., Ubuntu is a simple choice). We can also specify the instance type (which will determine the CPU and memory of the instance), storage and user data (a script that will be executed on the instance after launch). For configuring the instances, we could use a service such as Ansible, however in our case user data should be suitable enough. In the user data a web server (such as Apache) could be installed, the source could be pulled and built (or a better option is to download the latest build artifact from the GitHub repository and unpack it in the root directory of the web server). Configuration files of the web-server could also be stored remotely, pulled and then applied to the web server. The base URL of the API could be stored in an environment variable that is set in the user data with the help of Terraform. The same process applies both to the frontend and the backend.

6.2.5 Load Balancers

Two application load balancers (ALB) need to be created, a private (internal) and a public (internet-facing) ALB. The private ALB will be used by the instances in the autoscaling group of the frontend, to reach the API, while the other ALB will be reached by the end-users. Thanks to the stateless nature of JWT, we do not have to worry about the load balancer switching between different servers, since there is no active session that the servers need to maintain with the client.

6.2.6 Route 53

Once the infrastructure has been set up, it is now accessible from the load balancer. No credible ecommerce site would use its load balancer's DNS however, thus a domain needs to be purchased. Once a domain has been purchased, its name servers need to be set to those of provided by AWS's Route 53 after a hosted zone has been created. If the domain has been purchased via AWS, this step is not needed, since it is already in the realm of AWS, otherwise it is necessary to ensure that AWS can manage the domain's DNS records. Once managed by AWS, with the help of the AWS certificate manager (ACM) a TLS certificate can be requested, after that, issuing happens quite quickly.

6.2.7 CloudFront Distribution

The CloudFront distribution (CDN) connects the domain, the ALB and the TLS certificate. We need to select the TLS certificate; we intend to apply to our website and the load balancer where the traffic should be routed. Once created, an endpoint will be associated with the CDN. From here, the only step to do is routing our root domain to the endpoint of the CDN by creating a CNAME record.

6.2.8 Security Groups

Security groups are necessary to ensure that only the authorized parties can communicate with each other. By default, AWS blocks all access and the security groups specify which parties are able to communicate with each other. The backend instances need to be able to communicate with the database, while backend's ALB needs to be able to communicate with the frontend autoscaling group's instances and the end-users need to be able to reach the domain.

7 Evaluation

Self-assessment can be conducted with the previously outlined evaluation criteria. In this section, I will linearly walk through the requirements specified in the evaluation criteria section.

7.1 Nonfunctional Requirements

High performance has been ensured by using a compiled language efficiently for the backend API. Although I did not carry out any benchmarking, I also did not notice any components that would have an issue with their loading time on the frontend. A possible improvement after deployment could be to continuously monitor the performance of the application using some monitoring tool such as Prometheus with Grafana, Datadog or even the built-in tools on AWS console.

Security is guaranteed by authenticating the user and then using JWT token for authentication/authorization. Further improvements could be done here by a more fine-grained permission scheme for the private endpoints of the API. Another potential improvement could be adding a CI/CD pipeline for automatically updating packages (this could be done with the help of GitHub Actions and enabling and configuring Dependabot).

In terms of usability, I did not experience any issue. To me, the UI seems intuitive and easy to use. However, further tests should be conducted (e.g., a usability test or surveying users) to ensure that the UI really is intuitive and whether there could be any improvements added.

With the cloud infrastructure I previously outlined, availability should not be an issue (with the possible short-term exception of a really sudden spike in traffic). A potential improvement could be containerizing the services and moving the application into an Elastic Kubernetes Service (EKS) Cluster.

Extensibility is arguably hard to evaluate as it highly depends on how we intend to extend the application. Throughout development, I used a modular approach to ensure future extensions can be carried out smoothly.

7.2 Functional Requirements

While nonfunctional requirements may be challenging to evaluate, due to their vague nature, functional requirements are straightforward, thus they can be evaluated more objectively.

User authentication has been implemented and users can be created with the help of the backend API, however on the frontend, the user registration page is still missing.

The product catalog is working as expected and the products can be browsed by different categories.

The cart, checkout and order tracking functionalities are missing. This is mainly due to the fact that even more crucial parts of the application took up more time than expected and authentication has only been implemented in the latest stage of the project, leaving room for the future implementation of these features.

A minimalistic search engine has been implemented which looks for the keyword in the name and description of the products. Further improvements could be added by adding sorting and filtering functionalities.

An admin page has not been created but admin functionalities have been added to the backend API. In my opinion the admin page should come as lowest priority, as it is not something that affects user experience, it merely simplifies mutating entities in the database.

Overall, it is clear that many functionalities are missing from the functional requirements, while the nonfunctional requirements have fulfilled. This is largely due to the limited timeframe I had for the project and the fact that I was unwilling to compromise on the quality of the application in exchange for developing more functionalities.

Bibliography

[1] D. N. B. Gajjar, "Commerce, E-Commerce and Trade," *International Journal for Research in Management and Pharmacy (IJRMP),* vol. 2, no. 1, 2013.

[2] V. K. Yadav, "Global Prospect of E-commerce," *International Journal of Social Science & Interdisciplinary Research,* vol. 3, no. 1, 2014.

[3] Y. Tian, History of E-Commerce, 2007.

[4] K. Nguyen, Technology Contribution in Electronic Commerce, 2019.

[5] D. Zhe, E-commerce trend and E-customer analyzing, 2017.

[6] V. Simakov, "History of Formation of E-commerce Enterprises as Subjects of Innovative Entrepreneurship," *Three Seas Economic Journal,* vol. 1, no. 1, 2020.

[7] S. Krug, Don't Make Me Think, Revisited, New Riders, 2014.

[8] A. Wathan and S. Schroger, Refactoring UI, 2018.

[9] M. S. S. Padmannavar, "A Review on Ecommerce Security," *International Journal of Engineering Research and Applications (IJERA),* vol. 1, no. 4, pp. 1323-1327.

[10] M. D. Kalamkar, "A study of Ecommerce Security," *IJCTA,* vol. 10, no. 9, 2017.

[11] D. M. A. Hussain, "A Study of Information Security in E- Commerce," *International Journal of Computer Engineering Science,* vol. 3, no. 3, 2013.

[12] Y. Sun, J. Zhang, Y. Xiong and G. Zhu, "Data Security and Privacy in Cloud Computing," *International Journal of Distributed Sensor Networks,* 2014.

[13] P. Yang, N. Xiong and J. Ren, "Data Security and Privacy Protection for Cloud Storage: A Survey," *IEEE Access,* vol. 8, pp. 131723-131740, 2020.

[14] D. Chen and H. Zhao, "Data Security and Privacy Protection Issues in Cloud Computing," in *International Conference on Computer Science and Electronics Engineering,* 2012.

[15] M. I. Al Ladan, "E-Commerce Security Challenges: A Taxonomy," *Journal of Economics, Business and Management*, vol. 4, no. 10, 2016.

[16] IBM Cloud Education, "Object vs. File vs. Block Storage: What’s the Difference?," [Online]. Available: https://www.ibm.com/blog/object-vs-file-vs-block-storage/. [Accessed 6 May 2024].

[17] K. Zeng, "Provable data integrity verifying method, apparatuses and system". Patent US20090171878A1, 2008.

[18] KraneShares, "Alibaba 101: An Overview of the World’s Largest E-Commerce Company," 2014. [Online]. Available: https://kraneshares.com/resources/2014_12_alibaba_overview.pdf. [Accessed 7 May 2024].

[19] Alibaba, "Alibaba Pricing," [Online]. Available: https://seller.alibaba.com/pricing. [Accessed 7 May 2024].

[20] BBC, "Alibaba: What exactly does it do?," 4 September 2014. [Online]. Available: https://www.bbc.com/news/business-29077495. [Accessed 7 May 2024].

[21] Alibaba, "Alibaba.com sourcing guide," [Online]. Available: https://www.alibaba-na.com/hubfs/ustradeshows/supplements_nutrition/alibaba_com_sourcing_guide.pdf . [Accessed 8 May 2024].

[22] N. Maio and B. Re, "How Amazon's E-Commerce Works?," *International Journal of Technology for Business,* vol. 2, no. 1, 2020.

[23] D. Pereira, "Amazon Business Model," Business Model Analyst, [Online]. Available: https://businessmodelanalyst.com/amazon-business-model/. [Accessed 9 May 2024].

[24] eBay Inc. Staff, "eBay Inc. Reiterates 'The Truth About Skype'," eBay, 3 May 2014. [Online]. Available: https://www.ebayinc.com/stories/news/ebay-inc-reiterates-truth-about-skype/. [Accessed 9 May 2024].

[25] D. Pereira, "eBay Business Model," Business Model Analyst, 2023. [Online]. Available: https://businessmodelanalyst.com/ebay-business-model/. [Accessed 9 May 2024].

[26] Automattic, "WooCommerce," Automattic, [Online]. Available: https://wordpress.org/plugins/woocommerce/. [Accessed 8 May 2024].

[27] S. Trovato and R. Watts, "WooCommerce Vs. Shopify (2024 Comparison)," Forbes, 2024. [Online]. Available: https://www.forbes.com/advisor/business/software/woocommerce-vs-shopify/. [Accessed 8 May 2024].

[28] T. Wingfield, "Can I Host Shopify On My Own Server?," Kanteneo, [Online]. Available: https://kanteneo.com/blog/can-i-host-shopify-on-my-own-server/. [Accessed 8 May 2024].

[29] K. Marr, "WooCommerce pricing: How much does it cost to run a store?," WooCommerce, [Online]. Available: https://woocommerce.com/posts/woocommerce-pricing/. [Accessed 8 May 2024].

[30] K. Wiegers and J. Beatty, Software Requirements.

[31] AWS, "Amazon Compute Service Level Agreement," 25 5 2022. [Online]. Available: https://aws.amazon.com/compute/sla/. [Accessed 19 9 2024].

[32] StackOverflow, "StackOverflow Trends - Web Frameworks," StackOverflow, [Online]. Available: https://insights.stackoverflow.com/trends?tags=reactjs%2Cvue.js%2Cangular%2Csvelte%2Cangularjs%2Cvuejs3%2Cnext.js. [Accessed 26 6 2024].

[33] "Google Trends - Web Frameworks," Google, [Online]. Available: https://trends.google.com/trends/explore?date=2012-01-01%202024-06-26&q=%2Fm%2F012l1vxv,%2Fg%2F11c6w0ddw9,%2Fg%2F11h4q9rcf3,%2Fg%2F11bc69_ykv,Svelte&hl=en. [Accessed 26 6 2024].