Camilla Corsetti
Berenike Hegedus

ARTDEC

Bachelor Thesis
Supervisor: Santiago Donoso

January 2020
Copenhagen School of Design and Technology
Web Development

Table of Content

Summary

The aim of the report is to outline the whole development process of the online marketplace called ArtDec, from research and planning until implementation and development. ArtDec is specifically designed for creating a community-based platform for designers to merchandise and market their artwork. The website is providing a convenient and satisfactory place to sell their design products.
The report is divided into five main analysis sections.

In the first section, the focus is on gathering enough information in order to make decisions towards the best solution. The users were researched by interviewing the primary target group, design students at the Copenhagen School of Design and Technology. Based on their answers the team could create a persona that helps to personalize and empathize with the primary user. This led us to have an overview of what functionalities the website should provide. The next research topic was cyber attacks and security design. We examined the most probable cyber attacks and risk analysis was made to find out which are the exact vulnerabilities that are threatening the ArtDec website based on probability, severity and the control over the mitigation. The main design principles were chosen and the team prepared for securing the website from the biggest risks. By the end of this section, there is a concrete plan available with interactive wireframes to plan the structure and behaviour, as well as styled mockups. The research phase with all its methods can speed up the development process and leaves time on the logic, proper code structure and security issues.

The second section is a mathematical decision-making process for the selection of the best technology stack to use for this specific use case. In the Analytical Hierarchy Process (AHP), we can define our main criteria for the project: scalability, performance and experience. The technology stack is divided into three parts, the frontend, the backend and the database solution. We gave each topic a primary stack selection that was examined based on our three criterias. The analysis concluded in the final tech selection: Node.js, PostgreSQL and general HTML, Javascript with Handlebars. This process is time-consuming and requires a lot of research on different topics. However, it allowed us to find the best fitting long-term solution while considering our short term limitations.

The AHP is followed by the main development section, Architecture and Design. This chapter aims to give a closer look at our code structure, the libraries that were used and how we build up the website to solve the requirements of the project. There is a list of the main libraries used for different features such as Socket.io for a live chat or Sequelize for setting up and communicating with PostgreSQL. The code is structured based on a Model-View-Controller scheme and forms on Object-Oriented Design Thinking principles in order to create maintainability and reusable code. Lastly, the database design is presented including a normalized ER-diagram as well as some examples of development issues such as how to create a many-to-many table with Sequelize or how to store images.

In the Quality Assurance chapter, we show the ways of how we avoided mistakes to prevent failures of the website. The main focus is to provide consistency and reliability. These were achieved by using a Bootstrap template that improves the user experience, providing flexibility, consistent design and good usability. Reliability is taken care of the principle of failing securely. This means that even if the website crashes at any point, transactions can provide data integrity. Moreover, unit tests were applied to investigate and prove the success of some main features of the system.

The next chapter is Security, where some topics are highlighted to show the implementation of the results of the research phase. The section contains details concerning the authorization with the use of cookies and tokens. It explains how the input fields are secured by multi-level sanitizing and validation to prevent malicious code injection. Furthermore, it outlines the design of the signup and login system that aims to ensure the safety of our users and to eliminate the risk of sensitive data exposure.

Considerations should be given to some of the possible future updates and extending the features of the website. This can be read in the last chapter of the report in Further Considerations and Development issues. It draws the conclusion that more steps need to be taken for better security, especially if some of the discussed extra features are added such as a payment system. The created customer support live chat leaves space for improvement in terms of creating a social chat with several chat rooms and the possibility for general users to reach out to artists.

The project resulted in a high fidelity prototype built with a well-selected tech stack providing the most important needs in terms of scalability, performance and security. It gave a steep learning curve to the development team and met the requirements set by the educational institute within the scope and limitations of the project.

Introduction

ArtDec is a marketplace platform that aims to create a solution for independent digital artists to merchandise and sell their designs and products online in only a few easy steps. The website gives access to a wide range of products with a unique design.

Our goal is to help upcoming local artists and digital designers to sell their products and designs. By using the website, they can focus on the creation and they don't have to deal with the hassles of technicalities and sales. We provide them with a simple platform where they can upload images of their products, allowing broad audiences to purchase their products.

We are building a website with a user-friendly approach to attract more and more users to explore the artists and their products and to be part of this newly forming digital art community. Therefore it is important that the UX is well designed and the website has a high performance. The growth of the users means that the website has to be able to increase in scale, so scalability is one of the main criteria when it comes to designing and choosing the tech stack. Another main principle is to be able to develop a safe website, leaving no vulnerabilities unsecured, so the users are protected and they can trust our product. Lastly, it is important to consider our desired learning curve as developers.

1.

Upload Your Work

Create and account and upload your designs

2.

Start Selling

Manage your products and promote your work

3.

Sit Back

We take care of printing and shipping

Creating a community and supporting local artists

The website creates an online space for upcoming designers and buyers. In the future, the website will have its own chat with the possibility to create groups. This initiative strengthens and shaping the art community.

There are many advantages of shopping local and supporting local designers and artists. It can generate decent work and economic growth and help shape a more sustainable community.

By purchasing from locals, you're not only supporting them, but you support your local economy. Significantly more money stays in a community, simply because locals are most probably spending their profit on local businesses such as banks and service providers.

Problem Formulation

→

How can we develop a well-performing, scalable and safe local online marketplace for digital artists that strengthens the digital art community in Copenhagen?

What technology stack is required in order to develop a user-friendly and scalable webpage?

What methodologies and security design principles do we have to implement to make sure that the website is secure from the most possible cyberattacks?

Requirements, Scope and Limitations

→

The goal of the project is to develop a functional high fidelity prototype for the ArtDec project. The team is left with specific requirements and deadlines set by the educational institute, so it is important to have an overview of the limitations. Therefore, the team has to find a solution that meets all requirements and deadlines.

The project has to meet a specific deadline, the latest January the 10th of 2020 the prototype of the website has to launch with the main functionalities.
Cover at least two subjects of Web Development course and demonstrate the workflow.
Result in a product with backend solutions.

Requirements set by the ArtDec project
The project has to provide a dashboard for the artist to be able to manage their profile and their products
Security
Consistency in the design and good UX and usability
The team needs to find the most suitable tech stack
Live chat for community shaping and social aspects

Due to the scope of the project, there are some features that need further development. First of all, artDec is a marketplace, however, the payment system is not included in the first prototype, since it wasn't the focus of the project nor the course. The live chat, for showing purposes, is limited to a version that serves as a customer support chat. The community chat is not accessible yet. Other small features such as an overview of monthly sales, saving items to wishlists and following favourite artists are considered in the future of the project as well.

Methodology

→

// Users and interaction
Interviews
Personas
User Stories
Use Case Diagram
User Flow

//Planning the website
Sitemap
Wireframing
Mockup
Interactive Prototype
Analytical Hierarchy Process

// Development
Database Normalization
Object-Oriented Design Thinking
MVC
Unit testing

// Security
Risk Analysis
Least of Privilege Psychological Acceptability
Securing the weakest link

SQL Injections
Cross-Site-Scripting (XSS)
Cross-site-request-forgery(CSRF)
Sensitive Data Exposure
Broken Access Control
Broken Authentication

1. Research and Planning

Getting to know the right segment that the website targets is the first crucial step in the development process. We will examine the users, their behaviours and their needs. Based on these findings it is possible to further discuss the main functionalities that the website has to cover and some security issues, to find the most threatening vulnerabilities that we have to protect the users from. At the end of this section, we can create a solid foundation of the product with a clear overview in terms of design, structure and direction.

Users

There are two main user groups - the artists and the general users. The project primarily focuses on creating a solution for independent digital artists to sell their designs and products online in only a few easy steps. Besides, there are the buyers, being the general users of the website. (We call them “buyers” in this report, however that term also covers those users who just stumble upon the website and might not end their interaction with an actual purchase.)

Therefore, the primary target group is the independent digital artists [artists] and the buyers [general user]. The goal is to create an end product where both segments’ needs are satisfied, but the main principle is to focus on the artists, therefore they were researched to find their specific needs.

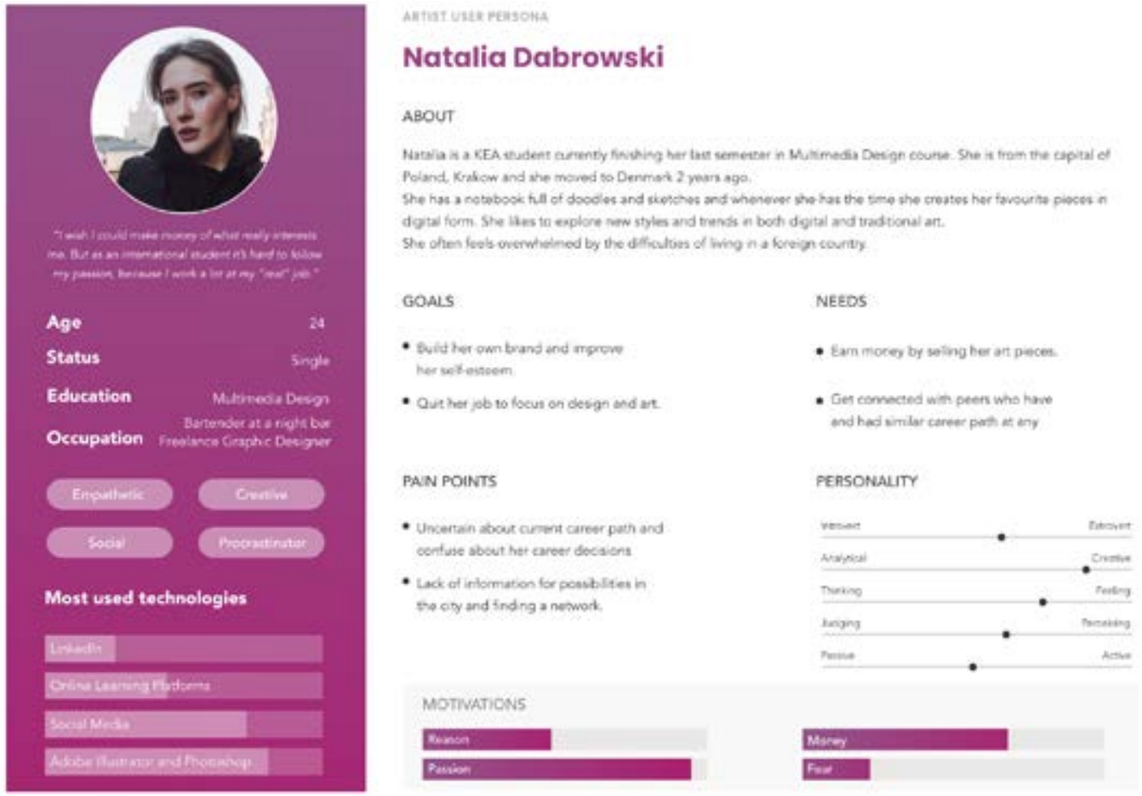
User stories

Regular Users

- As a buyer, I would like to be able to browse through and filter the products, to easily select which product to buy.
- As a buyer, I would like to be able to manage and have an overview of my cart, to proceed to checkout.
- As a buyer, I would like to be able to receive fast and friendly customer support about my purchases whenever I need, to be able to rely on the website.

Artists

- As an artist, I would like to be able to signup and upload my designs to start selling my art.
- As an artist, I would like to be able to see a dashboard where I can manage my profile and products.
- As an artist, I would like to be able to receive fast and friendly customer support about my sales whenever I need, to be able to rely on the website.



User stories

Persona analysis was made based on the interviews (See Appendix), to keep important factors of the target groups in focus throughout the project. The persona helps the development team personalize and empathize with the customer, including their behaviours, goals, and expectations. [1]

Our primary target group is independent digital artists, hobbyists and designers who would like to make some extra money by selling their art. Our website can provide them with an easy way to set up their own webshop, so they don't have to work too much with the technicalities and the logistics. Once the artists sign up, they can upload their designs printed on different products, set a price and publish it. On their dashboard, they can easily manage their selling products and update their profile.

The products are available for the general users, who can browse the webshop to find any hidden gems the local art community can provide. A regular user persona is someone, who is interested in home decor and stylish accessories for their home or office and they would like to support local upcoming artists instead of buying decorations from big retail companies.

Security

At this point in the planning process, it is clear how artDec should function and what are the exact user roles and their capabilities to interact with the website. Having this overview allows us to start focusing on the security, to figure out which security design principles should be applied, what are the weakest point of the website and how to secure it from the most possible attacks.

A thorough risk analysis was made in a spreadsheet, which is available in the Appendix. The summary and the most important points are discussed in the following paragraphs.

Security Design Principles

Least of Privilege

Users should be strictly limited to the processes they are supposed to do. For example, a regular user should not be able to access the artist dashboard, or an artist should not be able to delete other users, they are limited to managing their own account and products.

In order to protect the website from such vulnerability, the system should make use of accounts with different privileges for the different types of users. To achieve this it is a good practice to create a precise and clear use case diagram to plan the user roles and apply them in the development process.

Psychological Acceptability

This principle highlights the importance of user experience and reminds developers not to ruin the UX with a lot of restrictions and rules towards the users because of security reasons. Meaning, we can't expect the user to provide a password with 50 characters and have a 5 level login process. It is up to the team to decide how deep and thorough the defence should be on the website, however, the bar is highly dependent on the business case. For instance, websites working with a lot of sensitive data such as bank platforms should put security first. On the other hand, more user-oriented sites with less sensitive data should focus on making the UX as seamless as possible to attract more users and keep existing users satisfied.

Securing the weakest link

Any system is only as strong as the weakest link. Every case will have different vulnerabilities and strengths based on the user interface and the data stored and passed through the website. This principle reminds us to analyze our system to have a great understanding of the most possible attacks and to make sure the system is secured from as many angles as possible. This is where the risk analysis comes in.

Risk Analysis

The risk analysis is based on the 2017 Top 10 vulnerabilities by OWASP[2]. These 10 most potential vulnerabilities are discussed on how they apply to our website in the perspective of the probability, severity and our control over avoiding them. After making sure that all the weakest points and the most potential risks are secured in the system, we can move forward other security issues that were lower on the list.

The full risk analysis is available in the Appendix.

Risk	Description	Affected systems (the system containing the risk)	Probability [1-10]	Probability desc	Severity [1-10]	Severity desc
Cross-Site Scripting	The attacker inserts script into unsanitazied input form on client side. The script modifies how the website is displayed which can trick users into executing the attacker's code.	search, login, signup, adding new product, update user profile (INPUT FIELDS)	6	Not sanitazing the input field from client and server side will lead to XSS attact. Nowadays browsers implement protection against cross-site scripting. XSS is a fearly easy attack and very popular amongst hackers.	8	The attacker will have acess to user personal data and abuse it.
Broken Access Control	Flaws in logic and authentication. The attacker can access functions and data if the users' access is not restricted properly	whole application	5	These can be easy overlooked, it is easy to detect the disfunctionality but hard to verify in present. In our case even if the attacker gets access to artist profile, won't be able to do much harm.	7	the attacker can access to functions that a normal user shouldn't, or exploit personal data. The attacker can act as an admin and access unauthorized data
Injection	The attacker inserts database commands into input fields and executes them inside the database.	search, login, signup, adding new product, update user profile (INPUT FIELDS)	9	It's the most common threat for 2017.	9	The damages can vary from just revealing some data to destroying all data, or denial of access.
Insufficient Logging & Monitoring	Not being able to track down attackers and gain back control over your system as early as possible	maintainance of the application	5	It is about implementing monitoring and logging functionality; using monitoring tools.	2	It doesn't protect from attacks but it could result in great delays in the discovery of attacks.

Risk Analysis

SQL Injection

An SQL injection is when the attacker inserts database commands into input fields and executes them inside the database.

Despite the ease of control and prevention, it is still a popular vulnerability that constantly appears on OWASP's Top 10 list [2]. The severity of a SQL injection attack is high since the attacker can obtain access to the database [3]. However, it is easy to prevent these attacks by following good practices for querying the database and using a trusted library that can avoid the problem at the heart of injection attacks. Both on the backend and the frontend, we can make sure that the data passing through the system is sanitized.

Sensitive Data Exposure

The risk of Sensitive Data Exposures applies once the data is accessed by an attacker. Sensitive data should not be available and only the necessary data should be stored in the database [2].

"This type of attack is a data breach and it's directly affecting customers and the business. The stakes can be high, but it's important to recognize to what extent the application has to fight against it. The questions that we needed to ask was "What information we must protect? What information can be intimate?"" [3] Based on this principle we must only store data that is necessary, furthermore, all the intimate data such as address must be encrypted, passwords must be transformed with strong adaptive and salted hashing functions.

Cross-Site Scripting

Cross-Site Scripting means the attacker inserts script into unsanitized input fields on client-side, modifying the output of someone else's site. The script modifies how the website is displayed or behaving which can trick users into executing the attacker's code. [2]

The probability of someone trying to expose this risk is relatively high because it is easy to perform. Fortunately, careful sanitization and validation of user-generated input and output can mitigate the risks.

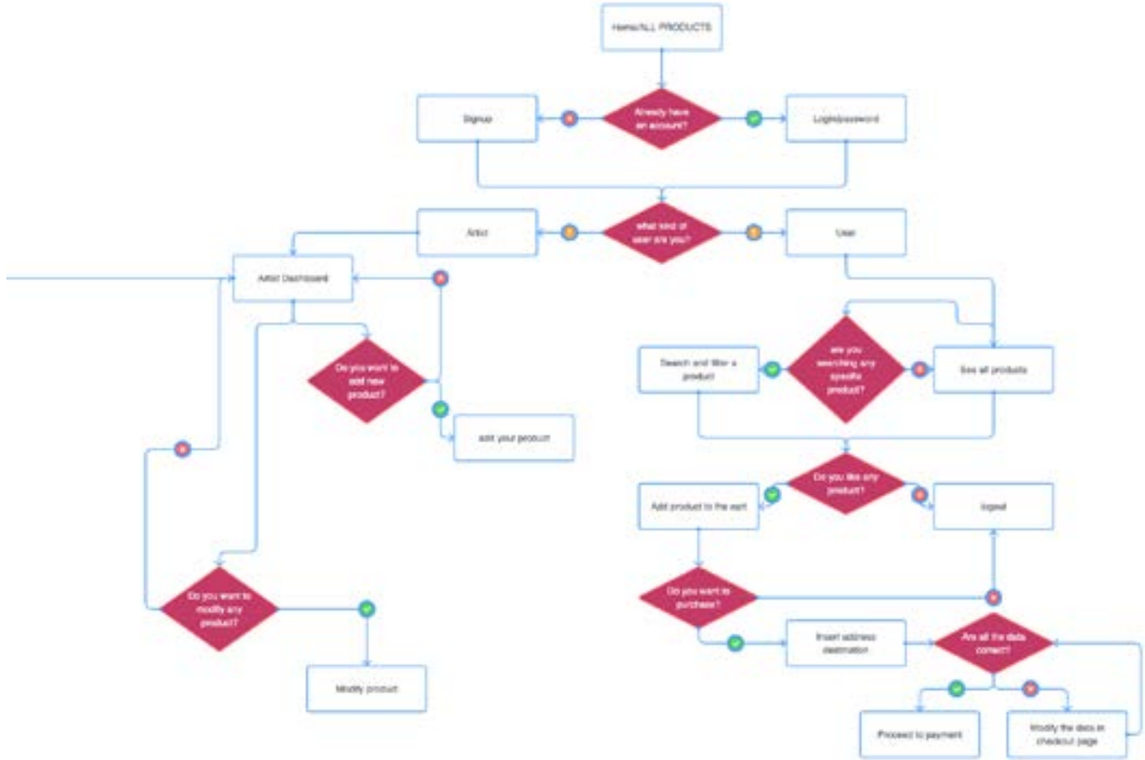
Foundation of the website

There are two main user groups - the In this section we create the foundation of the website and use methods such as creating a sitemap, making a user flow and high-fidelity wireframes and interactive mockups. All of these will make the development process much faster as they serve as guidelines for production and implementation.

Sitemap

The site map is a visual representation and model of the whole website page hierarchy starting from the homepage including all the different subpages. This helps the developers to have an overview of the different pages. It makes it easier to see how the users will reach the different points of the website. Furthermore, it allows the team to outline the different route restrictions - since there are different types of users, not every user will have the same permission for the pages.

Sitemap



User Flow Chart - a visual representation of the user experience between the pages

Use Case Diagram and User Flow Chart

Use case diagrams and user flow charts give the developers an understanding of the website's dynamic behaviour. The diagram specifies what each type of user should be able to do, while the user flow chart shows all the different events during the execution of the main tasks of the users. [4]

The use case diagram has two main characters - the basic user and the artist.

The basic user should be able to

- Search product
- Purchase product
- Signup / login / logout

The artist users should be able to perform everything that a basic user can do, plus:

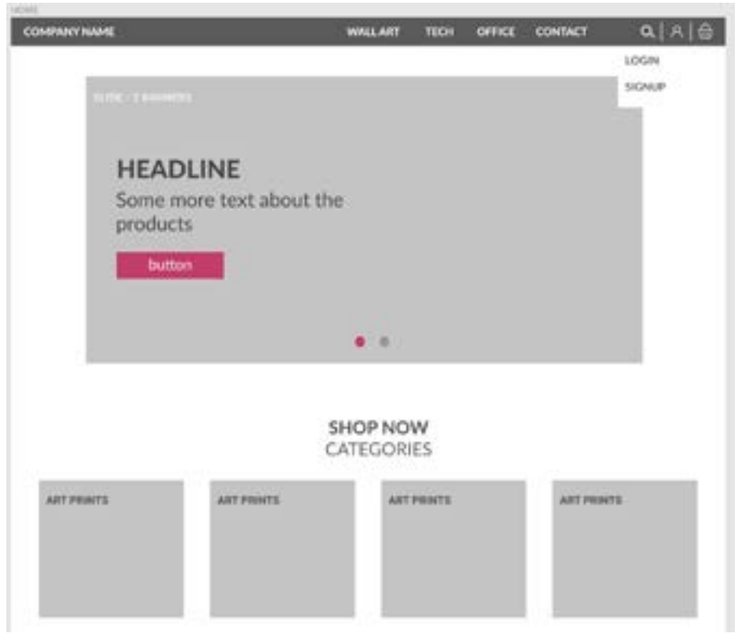
- CRUD of their products
- CRUD of their profile - deleting, in this case, means that they deactivate themselves

Wireframes and Mockups

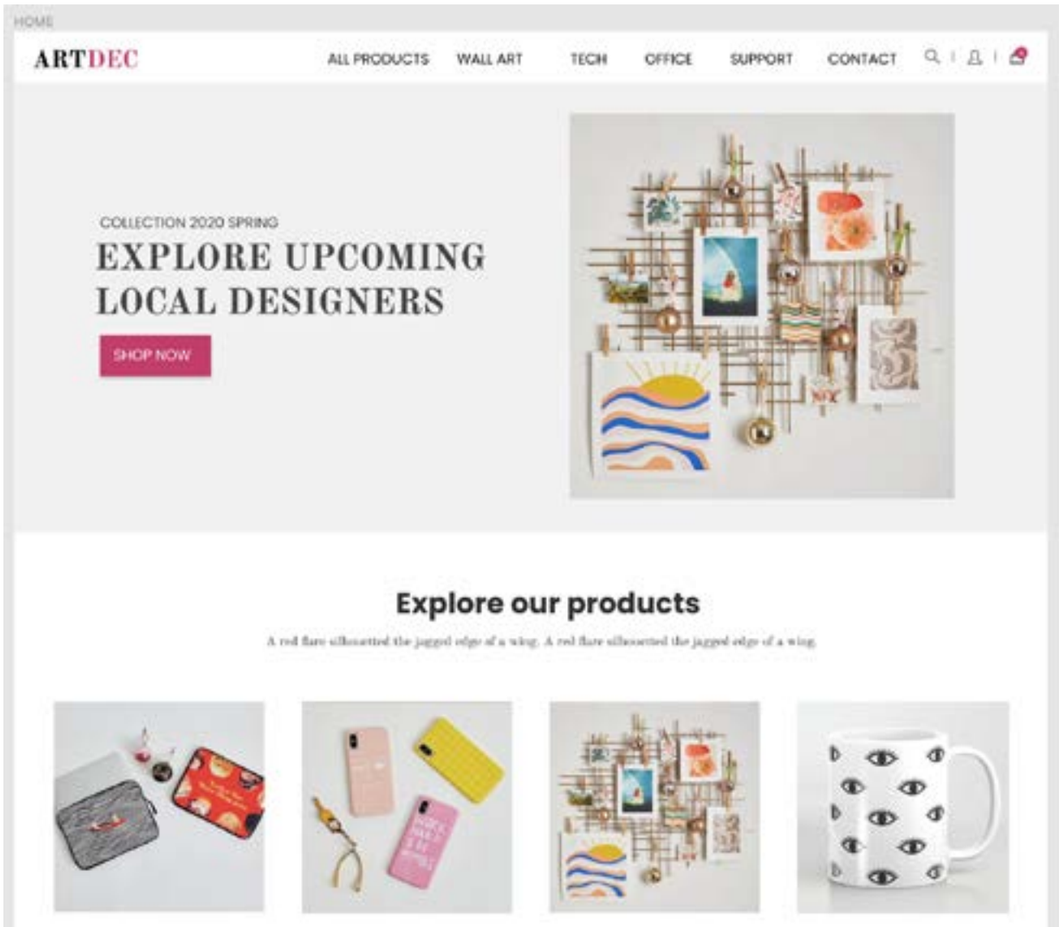
At this point in the early development phase is now possible to create the structural level of the website. The previous planning methods made it possible to create the wireframes fast and accurate, avoiding later changes as much as possible. These wireframes are serving as the main guide for how the website should be structured and dynamically behave. In addition to the wireframes, the team added an interactive flow to them to recreate how the interaction on the website will look like. Later in the process, after choosing the tech stack and the development methods, we created the mockups based on our chosen style template. The finalized version of the design is helping our development process.

Figma.com was used for both methods. [5] Figma has the advantage of allowing us to translate our design choices into CSS properties, making the styling during coding much faster.

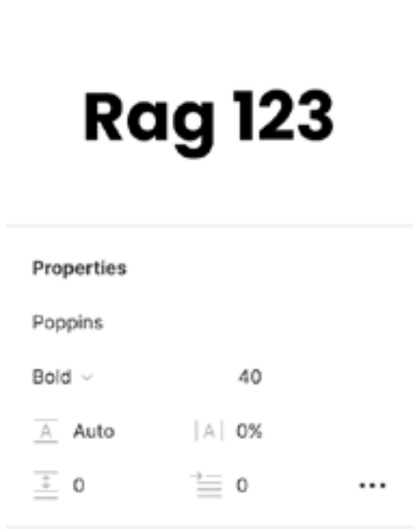
All the wireframes are available in the Appendix, for the interactive prototype please click here.



Wireframe



Mockup



Figma - translating design into code

2. Analytical Hierarchy Process

→

The following section aims to select the most suitable tech stack for the project use case, considering the information gathered in the previous phase, including the website requirements, the user needs and the security principles.
Analytical Hierarchy Process (AHP) is used for the selection of the tech stack. AHP is a mathematical tool for problem-solving and decision-making. [6] First, the main criterias are defined and weighted. Then, the tech stack is divided into three parts: frontend, backend and database. Each part is given a primary tech selection that is analyzed based on the previously defined criterias. The process concludes in the final technology for the ArtDec project.

All the mentioned calculations and tables are in the appendix.

Criteria	Scalability	Experience	Performance		Priority
Scalability	1	0,1428571429	0,3333333333	0,3624601243	0,080961232
Experience	7	1	5	3,27106631	0,7306446714
Performance	3	0,2	1	0,8434326653	0,1883940966
				4,4769591	1

Criterias and their priorities

→

Criterias

Experience

Experience is the most important criteria because it dictates the technologies that we can develop within the given deadline to deliver a high fidelity prototype. Therefore other technologies may get better scores in performance and scalability. The experience will, however, be the determining criteria to ensure that we deliver a prototype that fulfils the requirements on time.

Therefore, the strength of experience is 7 over scalability and 5 over performance.

Performance

It is important to ensure the fast performance of the application because it can easily become an issue given the many picture uploads. The application has to respond fast to provide a good user experience. If the application is slow, users will get annoyed which most likely results in a decrease in the number of overall users and purchases.

It is important that the application has fast performance given any number of users. It is better to start out with a fast application, rather than starting out with a slow application that can, however, scale well. We, therefore, assess the strength of performance to be 3 over scalability.

Scalability

The project is a web application that aims to have a lot of users. Therefore scalability is an important criteria to ensure that the application can increase in scale. If the application fails to be scalable it will be limited to extend the overall user base in the future. The biggest downside of not having a scalable application is the potential extra costs of adapting or changing the existing technology, therefore this criteria is important to consider since the very beginning, so the project won't be forced to change the stack when scaling up and thereby spending unnecessary time and money upon doing so.

Considering our other criterias (performance and experience) we do however assess the strength of scalability to be the lowest determining criteria.

Backend

The backend alternatives were rather limited because of our lack of experience. PHP is an obvious backend technology to consider since all team members have experience with it. Laravel is a PHP framework it would most likely be natural to work with based on the fact we have experience with PHP, but we do not have any experience with the actual Laravel framework. Based on the fact that both of us has worked with NodeJS during their internship, it might be a solution to develop the project as well as practising the newly learnt stack. NodeJS uses Javascript syntax which we also have a lot of experience working with. It makes the development much faster because we can write the same syntax both frontend and backend. Furthermore, it has a lot of benefits such as the possibility of implementing WebSocket protocol, allowing real-time bidirectional communication.

Our preliminary selection, therefore, ended up being PHP, Laravel and NodeJS.

Scalability

Node.js is designed to be highly scalable, using single-threaded asynchronous architecture, it has moderate importance over PHP. [7] Laravel defines a lot of functionality that would otherwise have to be written in PHP, therefore it also gets moderate importance over PHP. Laravel in relationship to Node.js is comparable.

Performance

Between the 3 backend solutions, NodeJS is a bit faster and more lightweight. [8] Therefore we gave the same weight to PHP and Laravel while NodeJS has a weight of 2.

Backend	Scalability	Experience	Performance	Score
Node.js	0,0323844928	0,3321475197	0,09419704832	0,4587290609
PHP	0,0161922464	0,3083384437	0,04709852416	0,3716292143
Laravel	0,0323844928	0,09015870789	0,04709852416	0,1696417248
				1

Backend coenlusion - Node.js 46%

Experience

Our knowledge of core PHP is greater than Laravel, even though Laravel is written in PHP, much of the Laravel functionalities are Laravel specific. PHP, therefore, has moderate to strong importance over Laravel and is thereby given the weight of 4.

As a group, our knowledge of PHP compared to NodeJS is almost the same, the whole team knows some level of PHP and NodeJS. Therefore PHP and NodeJS has equal importance and is thereby given the weight of 1.

NodeJS was our primarily stack during our internship, which means our knowledge is more current, while none of the team members ever dealt with the Laravel framework, therefore NodeJS has strong importance over Laravel and is given the weight of 5.

Conclusion

Based on our chosen criteria and weighing the selected stack options against each other, we have come to the conclusion that NodeJS wins with a score of 45%. PHP comes in at second place with a score of 37% and Laravel third with a score of 16%.

Frontend

For the frontend solution in our preliminary selection, we discussed using a framework versus plain Javascript, HTML with handlebars. There are many advantages of using frontend frameworks that can improve the workflow and code structure and they include a good level of security by default. On the other hand, a frontend framework would take longer time to implement and the scope of the project does not necessarily require the framework provided benefits. Therefore, we chose to select two frontend frameworks - Angular and React - to hold up against plain Javascript, HTML with handlebars.

Scalability

Handlebars generates pages dynamically, thus slightly improving the scalability and maintainability of the website. [9]-[10]

Angular, React and Handlebars all have the advantage of the ability to reuse components, but React and Angular allows you to use more complex components. Therefore both React and Angular have moderate to strong importance over Handlebars solution resulting in a weight of 7.

React is a library-framework where we can include libraries based on the application needs. Meanwhile Angular is a framework that by default contains a lot of libraries which makes the framework much heavier compared with React, however, we found that Angular is more scalable than React because it is a complete framework whereas React is just a library. Angular has equal to moderate importance over React and is thereby given the weight of 2.

Experience

As a group, we have much more experience with native HTML/JS/Handlebars with very strong importance over React and is thereby given the weight of 7.

Our knowledge about Angular has improved during an elective project in the previous semester but is still not on the same level as HTML/JS/Handlebars. HTML/JS/Handlebars has strong importance over Angular and is thereby given the weight of 5.

Due to the above mentioned elective course in Angular, our experience in Angular is better than React. Angular has moderate importance over React and is thereby given the weight of 3.

Frontend	Scalability	Experience	Performance	Score
HTML/JS	0,005265650965	0,5338416358	0,1393351611	0,6784424478
React	0,02925544959	0,05915389275	0,01767374959	0,1060830919
Angular	0,04644013145	0,1376491428	0,03138518597	0,2154744602
				1

Frontend coenlusion - HTML/JS/Handlebars 67%

Performance

Handlebars compiles templates into JavaScript functions. This makes the template execution faster than most other template engines. Handlebars isn't meant to create large scale applications but it can be handy when it comes to the creation of a prototype. [11] On the other hand, React is a front-end library very light which allows the developer to include only the needed libraries to properly develop a high-secure and highly scalable application without losing power in terms of performance.

React might be a fair choice for this criteria, but considering the scope and size of the project, HTML/JS/Handlebars has very strong importance over React and is thereby given the weight of 7.

Like React, Angular also needs to be compiled and is therefore not as performant as HTML/JS/Handlebars, in addition, the fact that it is a framework makes it overall heavier. HTML/JS/Handlebars has strong importance over Angular and is thereby given the weight of 5. [10]

Angular has equal to moderate importance over React and is thereby given the weight of 2.

Conclusion

Based on our chosen criteria and weighing the selected stack options against each other, we have come to the conclusion that HTML/JS/Handlebars wins with a score of 68%. Angular comes in at second place with a score of 21% and React third with 11%.

Database

For the frontend solution in our preliminary selection, we discussed using a framework versus plain Javascript, HTML with handlebars. There are many advantages of using frontend frameworks that can improve the workflow and code structure and they include a good level of security by default. On the other hand, a frontend framework would take longer time to implement and the scope of the project does not necessarily require the framework provided benefits. Therefore, we chose to select two frontend frameworks - Angular and React - to hold up against plain Javascript, HTML with handlebars.

Database	Scalability	Experience	Performance	Score
MySQL	0,008995692444	0,3409675133	0,01671923083	0,3666824366
PostgreSQL	0,008995692444	0,3409675133	0,1053245543	0,45528776
MongoDB	0,06296984711	0,04870964476	0,06635031151	0,1780298034
				1

Database conclusion - PostgreSQL 46%

Scalability

MongoDB is a non-relational database whereas MySQL and PostgreSQL are both relational databases. NoSQL is inherently better at scaling, as it can scale horizontally. Furthermore, a schema/model doesn't have to be defined, so if the data changes MongoDB can adapt. [12]-[13]-[14] Therefore, MongoDB has a strength level of 7 compared to PostgreSQL and MySQL, which are equal to each other.

Experience

The team has more experience in query languages. MySQL and PostgreSQL are using the same query language which we have been working since the very beginning of our education. Therefore, they both have very strong importance (7) in comparison to MongoDB. When it comes to MongoDB, the team could easily agree on the lack of confidence in working with it.

Performance

In terms of performance, PostgreSQL has a lot of advantages. Postgresql includes most functionalities of an SQL database with some possible advantages of a NoSQL database. [12]-[13]-[14] It gets moderate importance over MongoDB with a weight of 2 and strong importance over MySQL with a weight of 5. MongoDB is more fitting to NodeJS and faster than MySQL so it has strong importance over MySQL with a weight of 5.

Conclusion

Based on our chosen criteria and weighing the selected stack options against each other, we have come to the conclusion that PostgresSQL wins with a score of 46%. Not far from behind there is MySQL with 37% and lastly MongoDB with a score of 18%.

3. Architecture and Design
→

Node.js

Node.js is a Javascript runtime environment, a relatively new platform for writing network and web applications. In the past 10 years, it has rapidly gained popularity in the coding community thanks to its primary innovation, a little change in handling requests, and the ability to use JavaScript on the server-side development. [16]

Node.js is built entirely around an event-driven non-blocking model of programming. This method solves the struggle against server resources and the limits on the number of requests they can process. [15] Meaning, when the web application runs a database query, it runs the request and tells Node.js what to do next when the response is received. In the meantime, the code is free to start processing other incoming requests, while freezing the browser, allowing the user to interact with the user interface. [17]

The other huge benefit of Node.js is the countless number of libraries available through the npm module system. There are core modules defined within Node.js's source - including the bare minimum functionalities such as HTTP, URL, path - third party modules and custom-built modules. Some modules are included globally, such as console, while most of them have to be installed and imported. [20]

Node.js with Express

Express is a web framework built upon Node.js. It offers the basic functionalities as well as some powerful new functionalities and tools. Writing pure Node.js without a framework is not commonplace, because it is time-consuming, error-prone and insecure. [15]-[16]-[17] Having a framework allows developers to focus on business logic, makes the coding fast, optimized and secure. Some of the main areas where Express really makes it easier for developers are: routing and layers, REST API design and modules, middlewares and static file handling, templating engines. [15]

When installing express-generator with npm, it automatically creates the folders: [19]

App.js - This is the main file that starts the application, containing all the configuration information
Node_modules - contains all the modules that have been installed
Package.json - gives information about the application, such as modules and their versions, dependencies that should be installed etc.
Public - this is the folder for the browser, containing everything front-end related, such as stylesheets, images, and Javascript files
Routes - in this file we can define all the pages that the app should respond to
Views - the layouts for the UI

Important libraries

In this section, we are introducing the libraries that played an important role in the development of the product. They are briefly explained to give an overview of the project environment, while some topics will be further discussed in upcoming sections.

Database

Sequelize

Sequelize is a promised-based Node.js Object Relational Mapping tool for relational databases such as PostgreSQL, MySQL, SQLite and MariaDB. It reduces development time by letting the developers focus on coding and logic. It provides a database synchronization mechanism with the possibility to create the database structure by specifying the model structure and their relations in the backend. [21]-[22]

Pg

Sequelize is independent of specific dialects of databases, which means that the project must include the respective connector library as well. In order to get Sequelize nicely working together with PostgreSQL database, we need to install two additional libraries: pg and pg-hstore. [23] After installation, the dialect must be declared to 'postgres'. In our case, all of the values needed for the database connection are stored in the config file, therefore in the connection, the dialect defined as 'postgres' is called by the Config.DATABASE_DIALECT.

Pg-hstore

Pg-hstore implements the hstore data type for storing sets of key/value pairs within a single PostgreSQL value. It basically lets us serialize and deserialize JSON data to hstore format. [24]

```
const connection = new Sequelize(Config.DATABASE_NAME, Config.DATABASE_USERNAME, Config.DATABASE_PASSWORD, {
    host: Config.DATABASE_HOST,
    dialect: Config.DATABASE_DIALECT,
    ssl: true,
    dialectOptions: {
        ssl: true
    },
    pool: {
        max: 5,
        min: 0,
        idle: 10000
    },
    logging: false
});
```

Database Connection

Security

Jsonwebtoken

JWT is used for authorizing users.

Jwt.sign() is the function used to create a token, including the payload and the secretOrPrivateKey and optional values such as the expiration date. [25]

Payload: an object, buffer or string containing the actual data

secretOrPrivateKey: secret value created by the developers or by an algorithm (in our case it is a random string coming from the config file)

The application has 3 use cases of JWT:

Login system: There is a single, common login platform for both artists and general users. JWT is used to check whether the person logging in is an artist or not. If they are an artist, the site redirects them to the artist dashboard, while as a general user, it loads the main products page.

Signup system: When signing up, the website requires email validation from all users. With the help of SendGrid, users can validate themselves through their email. In that email, the jwt is sent for authorizing them.

Cookie - In our case, we are using the cookie without a session to send an encrypted token which contains some non-sensitive values that we need to pass to the frontend to validate what kind of user just accessed ArtDec website. The reason not to use the session comes from the token stored in it. The jwt contains encrypted values and it allows us to send dynamic values to the frontend while with the session we need to define the payload in our app.js without the possibility of sending any newly generated or dynamic values to the frontend. (More on the use of cookies and jwt are in the "Security" chapter.)

```
// Sign the token
let token = jwt.sign(
    {
        id: user.id,
        isArtist: user.isArtist,
    },
    Config.JWT_SECRET,
    { expiresIn: Config.TOKEN_EXPIRE_TIME });
```

Creating a jwt

Bcrypt

Bcrypt is a hashing algorithm for protecting the users' passwords. The point of a hashing algorithm is that it transforms the string into a random value, but it is nearly impossible to "translate" it back to the original value. As an extra security layer, bcrypt uses salt as well. [26]

Only these hashed and salted passwords are stored in the database.

Validator

Validator is a library of string validators and sanitizers that can be used both the server-side and the client-side. [27] An example of where the validator is implemented can be found in the signup system where isEmail() is checking if the entered value is a valid email.

Cors

We are using cors when we talk about cross-origin HTTP request - when a client asks for a resource of a different domain, protocol or port. CORS adds a new HTTP header which allows the server to specify the domain which is authorized to request some data. [28]-[29]-[31] It is also possible to permit any kind of domain by using Access-Control-Allow-Origin: *

The main difference between a simple declaration of "Access-Control-allow-origin:*" and the use of CORS stands in the extra controls about managing two different types of request: Simple Request and Preflight Request.

The application/JSON content-type is not always a simple request.

CORS intercept the browser request and only when the server response is positive the post request will be submitted.

When a request doesn't satisfy this simple request it gets defined as Preflight Request and needs to get controlled by additional controls. A Preflight Request checks if the CORS protocol is understood and a server is aware using specific methods and headers. [30]

```
else {
    const user = {
        email: req.body.email,
        username: req.body.username,
        password: bcrypt.hashSync(req.body.password, Config.SALT_ROUNDS),
        isArtist: false,
        isActive: false,
        isVerified: false,
        isAdmin: false,
    }
```

Bcrypt - creating a user - hashing the password

```
if (!bcrypt.compareSync(req.body.oldPassword || '', artist.password)) {
    //  Wrong password.
```

Bcrypt - Comparing passwords


```
// Start slowing requests after 5 failed attempts to do something for the same user
const userBruteforce = new ExpressBrute(store, {
    freeRetries: 5,
    minWait: 5 * 60 * 1000, // 5 minutes
    maxWait: 60 * 60 * 1000, // 1 hour,
    failCallback: (req, res) => {
        console.log('FAIL! Too many attempt. Try again in few minutes')
        res.json({ status: 429, message: 'FAIL! Too many attempt. Try again in few minutes' });
        return
    },
    handleStoreError: function (req, res) {
        return res.json({ status: 428, message: 'Something went wrong' });
    }
})
```

Implementation of Express-brute

```
return res.status(200).json({
    status: 200,
    message: 'product successfully updated!',
    product: req.sanitize(product)
})
```

Implementation of Express-sanitizer

```
// Validate if EMAIL is valid
if (!validator.isEmail(req.body.email)) {
    return res.status(403).json({
        status: 403,
        message: 'Email is invalid'
    })
}
```

Implementation of Validator

Express-brute

Express brute is a middleware for Express applications. It limits the incoming requests, preventing the possibility of a brute force attack and increasing the delays between each request. [32] Our implementation limits the failed logins to 5 attempts.

Express-sanitizer

Express sanitizer is a middleware for Express applications, which can mitigate the risks of XSS attacks by sanitization. [33] It is used every time new data is saved in the database namely: new artists, new users and new products.

Additional

Cookie-parser

Cookies are data sent and stored to the client with a server request. They help to keep track of the user's activity while they are interacting with the website. [34]

In order to use the cookies with Express, the cookie-parser middleware is required. The cookie-parser parses cookies attached to the client request object. It populates the cookie header and it lives below the req.cookies with an object keyed by the cookie names. [35]

Express handlebars

Express handlebars is a Handlebars view engine for Express applications with the advantage of supporting layouts and partials. It has available helper functions, which can be called within a template to transform output or iterate over data. [36] As an example, an iteration helper function was used for showing each of the products in a page corresponding to search filters or category selection.

Multer

Multer is a middleware that handles multipart/form-data primarily used for uploading files to the website. DiskStorage gives us full control on storing files to disk, defining the destination, the filename and the maximum size of a file.

It also has security options like file size limits. Furthermore, provides the possibility to create additional functions for filtering specific file types. [37]

After deployment, multer is used with another library which allowed us to store images on an online cloud called Cloudinary.


```
<!-- Start Single Product -->
{{#each products}}
<div class="col-md-4 col-lg-4 col-sm-6 col-xs-12">
    <div class="category">
        <div class="ht__cat__thumb">
            <a href="/products/product-details/{{this.id}}">
                <img src="{{this.image}}" alt="product images">
            </a>
        </div>
```

Ezpress Handlebars - iteration

```
const storageProduct = new multer.diskStorage({
    //dest: __dirname + './../public/images/avatar',
    destination: directoryProduct,
    limits: { fileSize: 1000000 },
    filename: function (req, file, cb) {
        return cb(null, Date.now() + path.extname(file.originalname));

    }
});
```

Multer - defining the disk storage

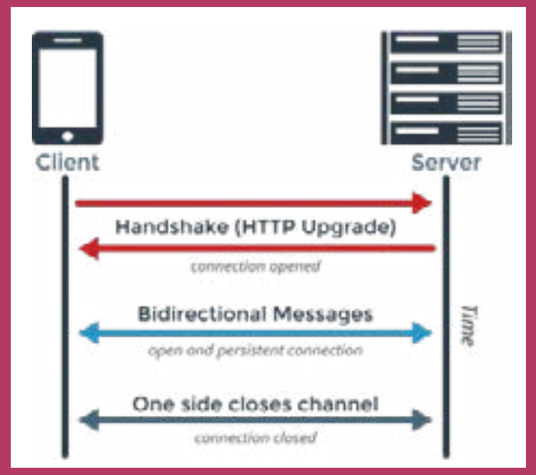
```
const fileFilter = (req, file, cb) => {
    if (!file.originalname.match(/\.(jpg|jpeg|png|)$/)) {
        return cb(new Error('File must be a jpg or a jpeg or a PNG '))
    } else {
        //sanitizeFile(file, cb)
        console.log('CORRECT FILETYPE');
        cb(undefined, true)
    }
}
```

Filtering a file

```
let upload = multer({ storage: storage, fileFilter: fileFilter })
let uploadProduct = multer({ storage: storageProduct, fileFilter: fileFilter })
```

Upload file with Multer and Filter

Live Chat



WebSocket Protocol

WebSocket is a communication protocol which allows full-duplex, bidirectional communication between client and server. As opposed to Http, with WebSocket protocol both the client and the server can initiate communication, sending or requesting data from both sides without having to reload the webpage. [17], [38]

To set up the WebSocket protocol, first, a request has to be sent to the server to initiate the WebSocket connection. This can be done from any number of clients. After the connection is set up, we can open and maintain WebSocket connections between multiple clients and a single server, until the connection is closed on one side.

Socket.io

Socket.io is a library which enables full-duplex communication between server and client, using the WebSocket protocol. It was designed to make real-time applications simple to code, providing an interface for the WebSocket. Socket.io is event-driven and lives in both the browser (client-side) and in Node.js (server-side). [39]
The main functionality of Socket.io is that it can push messages from the server. Meaning, whenever there is a new message from either side, the server receives it and pushes it out to other connected clients.

Examples of using Socket.io events: [40]

Sending a message - socket.emit() | sending to the client | can send a message containing any data that is convertible to a JSON object

Receiving a message - socket.on() | connection |is listening and waiting for a message

Push message to all connections - io.emit() sends a message to all the connected client

Broadcasting - socket.broadcast.emit() sends a message to all the connected clients, except the active client.

Implementation

In this early prototype phase of the product, the implementation of a real-time chat operates as a customer support chat for all users. It is a basic version used to show the potential of having a customer's live chat support. However, in later development, it is considered to implement more functionalities to the chat including rooms for artists and rooms between the artists and users as well. This additional feature of the website will further strengthen our main goal, to create a strong local community in Copenhagen for those who are interested in digital art.

Socket.io needs to be installed and required. On the client-side is enough to include the socket.io script as

<script src="/socket.io/socket.io.js"></script>

and create a constant called **socket = io();** which will allow the client to communicate immediately with the server. On the server-side, the implementation of the WebSocket is happening in the bin/www where socket.io is set up to listen to the same port of the server.

All the Socket.io code can be found in a distinct folder called "helper" which contains the socket API request and response within a file called "socketApi.js". The idea of keeping it separate from the rest of the code simplifies the development process and keeps the code cleaner. In this file, we can easily go through all the request and response of Socket.io and this architecture makes the debugger process easier.

```
var socket_io = require('socket.io');
var io = socket_io();
var socketApi = {};

socketApi.io = io;

const users = {}
io.on('connection', function (socket) {
    console.log('A user connected');
```

Socket.io connection

```
/**
 * Create HTTP server.
 */

//var server = http.Server(app);
var server = http.createServer(app);


/**
 * Listen on provided port, on all network interfaces.
 */

server.listen(port);
server.on('error', onError);
server.on('listening', onListening);
```

Socket.io listening to server port

Code Structure

Model-View-Controller

In order to keep the structure more readable and clean, the whole process of passing the data is divided on the server-side in different folders based on the responsibilities. This is achieved by using some principles of Object-Oriented Design Thinking focusing on making reusable code to improve the maintainability and to speed up the development process. [41] Therefore, if a piece of code is changed and restructured somewhere, it will affect all the results connected to that.

An MVC model separates the code into three main parts: [42]

- the Model - encapsulating the data and publish changes to view,
- the Controller - providing functionalities, process user input and update models,
- the View - display UI and subscribe to Model.

Implementation

Implementing the previously described Object-Oriented Design Thinking together with the MVC model It results in an MVC, where the Controller is further divided into separate folders for Services, Controllers and Routes.

In the Model folder, the models are defined specifying the attributes and their data types. The database queries can be found in a specific folder called Services. This folder contains all the separate javascript files named after the model which is being queried. The queries are called by the third main folder, the Controllers where all the actions and their logic can be found. Each controller function calls a service or multiple services and either send back the result as a JSON object or render it directly on the page.
These actions are called in the last main folder Routes, where all the endpoint are defined.
Inspired by Object-Oriented Design Thinking, the view is rendered with a powerful templating engine called Handlebars. Handlebars allows the use of partials and templates, resulting in cleaner and reusable code.

At the beginning of the development process, the team has created a rough outline for the code structure. This outline is a sheet listing all the different pages (views) and in connection to them listing all the services, controllers and routes that the page is going to use. This serves as an overview of what needs to be done, how the querying will look like and how to group the controllers and routes. Throughout the process there were some minor changes in the code, the sheet helped us planning and having an overview but it is not a strict guideline.

The full sheet is available in the Appendix, or click here.

	products.hbs	single-product.hbs	artist.hbs [reached by regular user]	artist-profile.hbs [reached by artist]
SERVICE	**FILTER:** getAllProductsBySubcategory() **FILTER:** getAllProductsByCategory() **FILTER:** getByTags() **FILTER:** getByPrice()	getProduct() addToCart() **FILTER:** getAllProductsByArtist()	**FILTER:** getAllByArtistId() (products+artist info) **FILTER:** getAllProductsByArtistAndSubcategory() **FILTER:** getAllProductsByArtistAndCategory() **FILTER:** getByArtistAndTags() **FILTER:** getByArtistAndPrice()	getAllByArtistId() (products+artist info) addProduct() updateProduct() deleteProduct() updateArtist()
CONTROLLER	**FN** findBySubcategory **FN** findByCategory **FN** findByTags **FN** findByPrice	**FN** showProduct **FN** insertIntoCart **FN** showAllArtistProducts	**FN** findAllByArtistId **FN** findByArtistAndSubcategory **FN** findByArtistAndCategory **FN** findByArtistAndTags **FN** findByArtistAndPrice	**FN** findAllByArtistId **FN** addNewProduct **FN** editProduct **FN** removeProduct **FN** editArtist
ROUTER	**products**/findAllBySubcategory **products**/findAllByCategory **products**/findAllByTags **products**/findAllByPrice	**products**/getProduct **products**/addProductToCart	**artists**/getAllByArtistId **artists**/findBySubcategory **artists**/findByCategory **artists**/findByTags **artists**/findByPrice	**artists**/getAllByArtistId **products**/addProduct **products**/updateProduct **products**/deleteProduct **artists**/updateArtist

Planning the code structure

Database

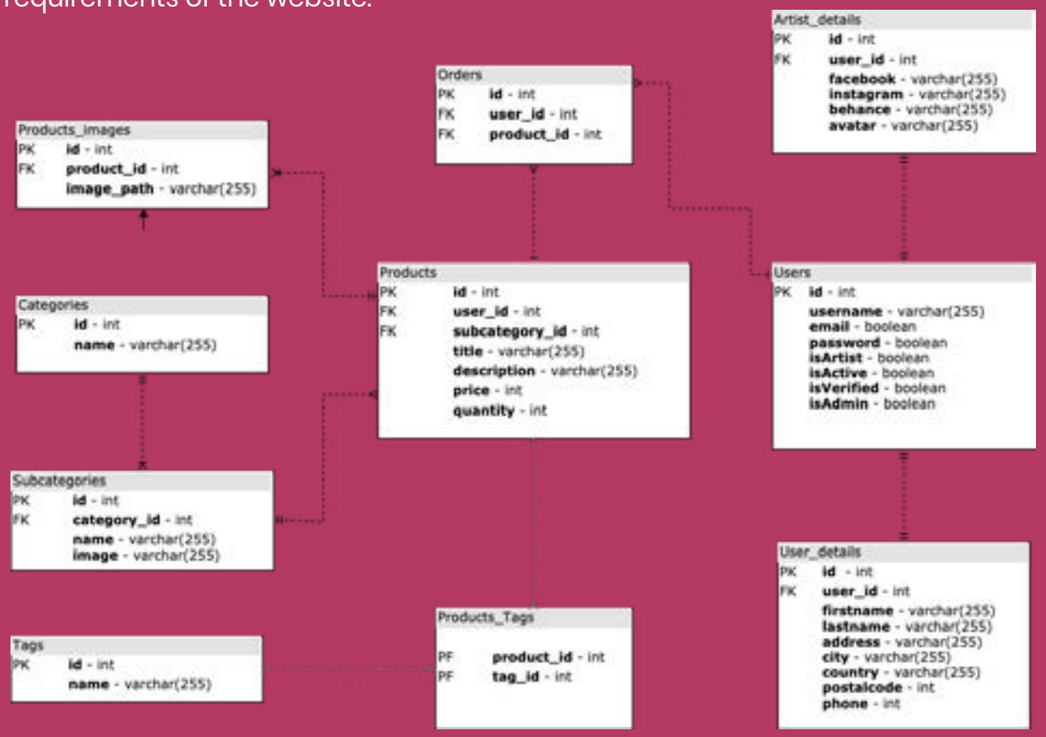
PostgreSQL was chosen during the analytical hierarchy process, winning the comparison over MySQL and MongoDB thanks to its many benefits. Postgresql includes the best parts of an SQL database with some possible advantages of a NoSQL database. It works with transactions and joins to prevent data integrity and with an additional library called Sequelize, it can be queried as a NoSQL database, which comes very handy when writing Javascript code on the server-side.

Another great benefit of using Sequelize is that it doesn't require too much planning on how to query the model and simplifies the querying process by using the keyword 'alias'. Whenever an alias is defined as a foreign key, the query can retrieve the entire model or some specified attributes related to that alias. This makes any data easily available to extract. [44] As of data safety, PostgreSQL's ACID-compliant transactions make sure that all the commits are completely atomic, consistent, isolated and durable.

Conceptual design and Normalization

The major goal of a database is to store and retrieve data accurately when it is needed. This goal becomes very hard to reach with a bad design, such as duplicated data or insertion and deletion anomalies. In order to free the database from these issues, we have to go through normalization, making sure that the entities, attributes and relations all have a logical structure. [43]

After the first sketches of the database design, the Entity-Relationship (ER) diagram was made to finalize the table structure with their relations and the data types. It is useful for clarifying the logical structure and focusing on what are the needed requirements of the website.



ER diagram

Users

The 'users' table is split into two other tables to properly store the data. It is split into two because general users and artists users require different types of data to be stored. For example, an artist has to add their avatar and social media links as well, while a general user needs to give their shipping details. In order to differentiate whether it is a general user or an artist, we use 'isArtist' boolean data. This separate signup system is also split into two on the frontend side as well.

Many-to-Many relations

There are two many-to-many relationship tables in the database: Orders (users and products) and Products' Tags. The process of setting up these tables starts with decomposing them, copying their primary keys into a third one which becomes junction table. It is important to give this junction table a suitable name that represents the intersection of the data. In some cases, it's simply the combination of the two original table names (see as Products_Tags) while other times it makes more sense to give it a new, self-explanatory name, such as Orders (instead of Users_Products).

Logically thinking, in the Products_Tags junction table a product can have many tags and a tag can belong to many products. However, a particular tag can be assigned to a product only once. This means, that both the product_ids and the tag_ids can be repeated, but the combination of those together can appear only once in the table. To achieve this, a compounded key is required, meaning that the two foreign keys (product_id and tag_id) combined will create a unique primary key. Each component of the key is related back to the original table in the pair of one-to-many relationships. [46]

On the picture below it is shown how to use compound keys to relate to a table in Sequelize. In the model of the junction table each column that makes up the compound key has to contain unique: "linked_product", where "linked_product" is the name of the key which has to be identical in all fields. [22]

Storing Images

Another important part of the database design which worths to highlight is the way images are stored for the products. For each image in the database, only the path is stored while the actual images are stored in folders. This is a procedure often used against overloading the database, considering that the website is expected to handle a large number of images in the future.

In the products table, a single image path is stored to serve as the main picture/thumbnail of the product. It is possible to upload more pictures of a product, in that case, every additional picture's path is going to be saved in the Products_images table.

```
module.exports = (sequlize, type) => {
    return sequlize.define('products_tags', {
        //FK - PF
        product_id: {
            type: type.INTEGER,
            allowNull: false,
            //create a composite/compounded key
            unique: "linked_product"
        },
        //FK - PK
        tag_id: {
            type: type.INTEGER,
            allowNull: false,
            //create a composite/compounded key
            unique: "linked_product"
        }
    }, {
        timestamps: true
    });
};
```

Setting up a many-to-many table

Implementation

Connection

To connect the database with the code first a new Sequelize instance has to be created passing all the required parameters for the connection. These parameters are securely stored in the config file and are imported in the document.

When setting up the connection, Sequelize needs the database name, the username and password, host and it requires a database dialect. Sequelize supports MySQL, MariaDB, Postgres and MSSQL dialects. [22]

After the new Sequelize instance is created, it will automatically generate a connection pool on initialization. Every time the website needs to retrieve data from the database, it will generate database connections which will be saved and reused again by Sequelize. The connection pool is the collection of these connections.

The connection pool is configured to never have more than 5 open connections. There are no minimum connections required, furthermore, idle option tells the program to remove the connection from the pool after it has not been used for 10 seconds. [47]

```
const connection = new Sequelize(Config.DATABASE_NAME, Config.DATABASE_USERNAME, Config.DATABASE_PASSWORD, {
    host: Config.DATABASE_HOST,
    dialect: Config.DATABASE_DIALECT,
    ssl: true,
    dialectOptions: {
        ssl: true
      },
    pool: {
        max: 5,
        min: 0,
        idle: 10000
    },
    logging: false
});
```

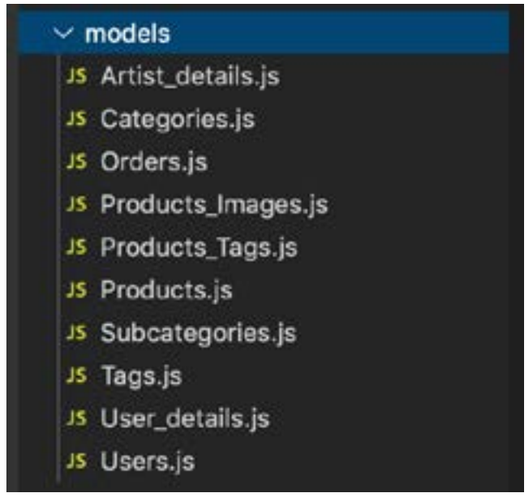
Database Connection

Models and Relations

In the Models folder, all the previously planned entities are defined. The picture shows an example of a model written for Sequelize, using the method define() followed by the name of the table. Each of the columns must have a defined data type. All the primary keys are set as an id that is an auto-incrementing integer and can not be null. [22], [45]

Sequelize requires the use of a timestamp so by default it adds two additional attributes createdAt and updatedAt to each table. There are many validations and options besides data types that can be set such as unique, required, not null etc. [22]

Lastly, the picture below shows the setup of a many-to-many relation between tables. It's defined in the database file where a very important keyword to highlight is the "as". This word serves as an alias and it is assigned to a foreign key (FK) to retrieve the entire model containing/related to that alias. [22]



Models Folder

```
module.exports = (sequilize, type) => {
    return sequilize.define('subcategories', {
        id: {
            allowNull: false,
            autoIncrement: true,
            primaryKey: true,
            type: type.INTEGER
        },
        name: {
            type: type.STRING,
        },
        image: {
            type: type.STRING,
            required: true,
        },
        //FK
        category_id: {
            type: type.INTEGER,
            allowNull: false
        }
    }, {
        timestamps: true
    });
};
```

Model example - Subcategories

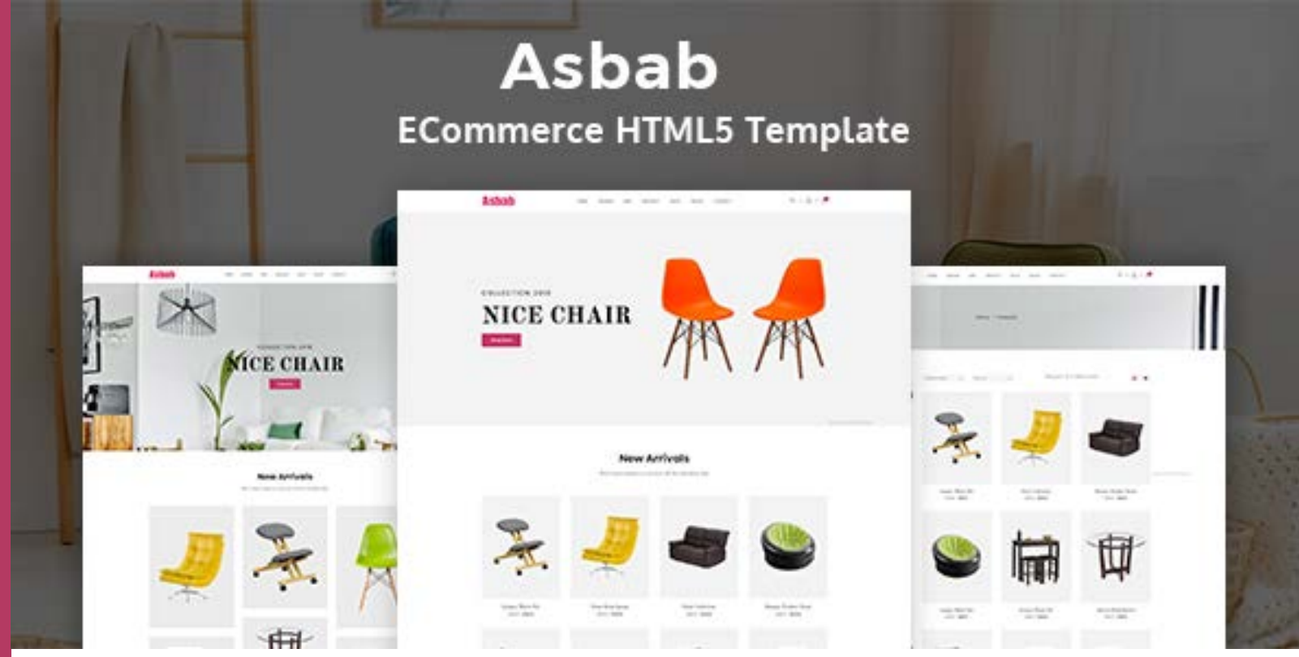
```
// RELATIONS

//PRODUCTS & TAGS - many to many
DatabaseManager.Products.belongsToMany(DatabaseManager.Tags, { through: 'products_tags', foreignKey: 'product_id', as: 'tags' })
DatabaseManager.Tags.belongsToMany(DatabaseManager.Products, { through: 'products_tags', foreignKey: 'tag_id', as: 'tagged_products' })
DatabaseManager.Products_Tags.belongsTo(DatabaseManager.Products, { foreignKey: 'product_id', as: 'tags' })
DatabaseManager.Products_Tags.belongsTo(DatabaseManager.Tags, { foreignKey: 'tag_id', as: 'tagged_products' })
```

Relations example - Products_Tags (many-to-many)

4. Quality Assurance

Quality assurance is a way to prevent mistakes before launching the website. It ensures that the quality of the code respects a certain standard. We quality assure focusing on consistency and reliability, furthermore unit tests ensuring that main functionalities are working correctly.



Asbab Bootstrap template

Consistency

Keeping the consistency in mind as a primary focus when it comes to design, a Bootstrap template was implemented. Using a style template can improve the user experience, providing flexibility, consistent design and good usability. The used template is called Asbab eCommerce. [48] It has valid HTML5 and CSS3 code, with more than 12 available pages it can really speed up the development process. It has a responsive layout and it is compatible with most of the major browsers. Additional Bootstrap code was added in some cases such as alerts, modals and confirmations.

Furthermore, Handlebars as templating engine for Javascript is used to add templates to our HTML files that will be parsed with the data that is passed to the page. Express-handlebars is the library selected for the development of ArtDec. It allows us to use partials and layouts and it makes it easy to manage the HTML content.

Reliability

In the future, the ArtDec website might have to handle a lot of traffic, resulting in possible overloads and crashes of the database. To secure the data and its integrity during a potential fail, transactions, as well as the constant use of try-catch blocks, are implemented. These will ensure that even if the website fails it will fail in a secure way. [49]

Results of the unit tests

```
PASS  __test__/user.test.js
  ✓ should signup a new User (151ms)
  ✓ should signup a new Artist (80ms)
  ✓ should check if user/artist email is invalid (4ms)
  ✓ Should check if user/artist authentication fails (28ms)
  ✓ Should check if user/artist is unauthenticated (25ms)
  ✓ Should check if user success in login (28ms)
  GET/dashboard
    ✓ access dashboard for artists (83ms)
  Should not login without being authorized
    ✓ Should check if user/artist is unauthenticated (25ms)

Test Suites: 1 passed, 1 total
Tests:       8 passed, 8 total
Snapshots:   0 total
Time:        2.829s, estimated 8s
Ran all test suites.
```

Testing

Unit tests in the ArtDect project are used to test the main functionalities of login and signup. They are quick and specific tests to perform. Unit tests ensure that errors down the line don't stem from the code that has been unit tested. So if we unit test a function which does not result in an error, then we can exclude the possibility of having errors in that part of the code. This process makes debugging much easier and more structured.
Jest and Supertest, two Javascript testing frameworks were used during the testing process. For some of the tests using Agent was also required.

The agent is used to send a login post request to access the private dashboard route. The agent is setting some parameters like cookies in order to pass the authentication middleware and being able to reach the desired route.

The tests:

Sign up as a regular user
Sign up as an artist
Log in as a not authenticated user
Login with wrong credentials
Login as an artist and allow them to access their dashboard route with the use of the agent sending the cookies
Login a regular user and ensure the security of not being able to access any dashboard route.

The unit test can be performed in ArtDec/server folder by running npm test.

5. Security

This section shows how ArtDec is secured from the most possible vulnerabilities that were discussed in the first chapter. The main goal was to prevent from SQL injection, cross-site scripting and avoid sensitive data exposure. Additionally, other security issues that were lower on the risk analysis are also covered, such as broken authentication with brute force attacks and broken access control. [50]

jwt and cookies

```
// Sign the token
let token = jwt.sign(
    {
        id: user.id,
        isArtist: user.isArtist,
    },
    Config.JWT_SECRET,
    { expiresIn: Config.TOKEN_EXPIRE_TIME });

res.cookie('auth', token, { httpOnly: true, signed: false, expires: new Date(Date.now() + 2 * 60 * 60 * 1000) })
```

```
app.use(cookieParser());
```

Jsonwebtoken and cookies

ArtDec makes double use of a jsonwebtoken. [25] The first time the user interacts with a JWT is after signing up. For security reasons we must ensure that the email used to signup is an existing email owned by a real person. To verify the user, a confirmation email is sent with a link containing the encoded token. The user has to click that link to be able to access ArtDec website. This token has the functionality of toggling the value of the isValid attribute inside the database and setting it as true.

The second token appears during the login and is used to check the authorization of the user deciding whether they are allowed to access the desired route or not. For this kind of authorization, usually, the token is sent in the header as bearer token using “Authorization” key. However, this project can not reach that point since our application doesn’t have any frontend frameworks to help us in handling the jsonWebToken correctly in the header.

Rather than setting the token in the header, the choice fell on the use of a server-side generated cookie.

This solution is still secure as much as storing the jwt in the header. The server-side generated cookie with the HttpOnly flag is not accessible from the front-end side with the use of any javascript function and so it cannot be retrieved by simply calling document.cookie.

This solution makes it harder for a hacker to execute such an attack where they steal the cookie information, guaranteeing a good level of security of the ArtDec website.

Sanitizing and Validating

One of the most common attacks is related to input fields. It is important to keep in mind that if an input field is not sanitized any attacker can inject malicious code to the website and it could steal any kind of information, manipulate the display of pages to cheat a user, or even reach the point of destroying the whole database. ArtDec has a good level of security when it comes to sanitizing and validating, covering frontend, backend and database.

In the frontend with the use of bootstrapValidator all the forms can be easily validated with the possibility of communicating immediately between the database and the client. The database is sending a JSON file containing a true or false boolean which can be used for real-time responses of the validation, for example, to check the availability of a username or email during the registration. To reinforce the frontend validation the server-side perform a validation again, and only if it passes this second validation, then it is going to be stored in the database, sending back a successful response to the user.

Sanitizing the input fields on the frontend is implemented a function to sanitize the input, but since the frontend is the most vulnerable part of a website and to reinforce this protection, the server makes use of express-sanitizer middleware library which ensure that any retrieved value sent from the database to the frontend will be sanitized again. To let it works on the server-side is enough to require it and using it as app.use(expressSanitizer()); [33]

Last but not least, Sequelize's communication with the database include by default protection against sql injection vulnerability. Sequelize escapes replacements, which avoids the problem at the heart of SQL injection attacks: unescaped strings. It also supports binding parameters when using SQLite or PostgreSQL, which alleviates the risk further by sending the parameters to the database separately to the query. [22]

Signup and Login System

To strongly protect the website different layer of security are added for the signup process. The flow is made up keeping in mind also the user flow for a secure but smooth signup process. The security focus is upon XSS, CSRF and SQL Injection trying to mitigate those attack with the use of a strong hashed password, verification email, sanitization of inputs and with the use of httpOnly flag cookie. [51]

We want to allow the user to use only strong passwords consisting of at least 8 characters, including uppercase, lowercase characters and numbers. The password is hashed with bcrypt. This and the fact that we are using Express-bruteforce will mitigate hackers to brute force the login because it would take too long to verify the validity of the values inserted by the user.

Furthermore, all the input field are sanitized in both client and server side. This will protect more accurately our website against XSS (cross-site scripting) and SQL injections.

To certify the validity of the email after the signup form is submitted the user receives an email with a link which include an encoded token sent from the server. This link relies on an endpoint called confirmation which will redirect the user to the login page while toggling the verified boolean stored in the user table to true. Only after the email link is clicked, the user will be able to access our website as verified user inserting the required credentials.

Multiple levels of security for the login system is also applied to warranty a good security level to our website and make the users trust us.

As mentioned previously, to avoid brute forcing there is a limited number of times a user can try to login with invalid credentials. With the express middleware called express-brute we can easily succeed in restricting the brute forcing attempts. By default it stores all the attempts in its memory-store which will lock out the user after five failing attempt for about five minutes. To prevent CSRF the jwt generated on the server is checking if the token that is sent to the frontend inside the HttpOnly cookie is the same as the one stored in the backend.

To check the privilege in our token two different values are stored. One is the id and the other one is a boolean which defines the user role and privileges to access some routes. The authentication of the server is reinforced on the client side with sessions where a random session id is generated and stored with two different key values based on the role of the user. If a regular user tries to access a protected route without the authorization is going to be redirected to the login page and logged out from the system.

Further Considerations & Development Issues

→

Every project has limitations that force the development team to compromise. Due to the scope of the school project, our main goal was to meet those deadlines and limitations set by the educational institute. This results in a prototype that contains important topics, however it can be further developed into a more complex and polished project. In this section we will list some of the ideas and issues that should be considered to further develop artDec in the future.

Superadmin

At the beginning of the planning phase, we were considering to create a super admin user with the most privileges. This is why we have an attribute in the Users entity called “isAdmin” but it is never called in the project. (It is called in the project only to set it to false by default whenever a user signs up, which is a Least of Privilege practice.)

Superadmin could be able to manage all users and products through the super admin dashboard. However, this would just require us to create another CRUD for another type of user, implementing the authorization and validation middlewares. These are already demonstrated, so for showing purposes, we decided not to focus on this topic. In reality, there should be a dashboard for admins to have control over the website.

Live Chat

The live chat was one of the main benefits of using Node.js as our backend solution. As it is now, the live chat is available as a customer service chat. However, in the long run, it should be developed into a fully functioning social chat with the ability to create rooms and groups, to provide the users with the possibility to reach out to the artists. With this initiative, we achieve one of our main desires: to strengthen and shape the art community.

To complete this extra functionality, we will use the already implemented Socket.io which supports the possibility of chat rooms, broadcasting and private messaging.

Extend functionalities

As a first prototype, the website provides some functionalities to create an eCommerce environment, however, it is not questionable that it needs more service to create a full and better experience for both the artists and general users.

Payment

First of all, to complete the whole customer journey, there must be some kind of secure payment implemented. Once the payment system set up, since the project is heavily focused on creating a local community and environment, it might be a good idea to implement MobilePay, as it is one of the easiest and most popular payment methods used locally in Denmark. MobilePay has available APIs such as MobilePay Connect, [52] which is a single sign-on application that allows users to share data with other apps or websites through their MobilePay account.

This API will allow users to select MobilePAy as they desired payment method.

These considerations need to be further researched in order to properly make a decision upon what and how to build up a secure payment system.

Users and Artists

To extend the website, there should be more beneficial functionalities available for both the artists and general users.

Artists should be able to have an overview of their selling. A dashboard with a monthly sales overview can serve as a unique selling point, providing them to track their performance and gain insights into how their products and designs are performing.

Even though this project focuses on the artists as the primary target group, general users’ needs should be satisfied as well. Filtering the products and the tagging system needs to be included to make the search more specific and browsing more enjoyable. To make the whole customer journey personal and customizable, the users - after logging in - should be able to save products into their wishlists, save and follow their favourite artists.

With all these extra functions the website becomes a much more social and friendly platform with more than just eCommerce purposes.

Security

In 2018 May, the new GDPR regulations enforce the protection of data. This affects web developers to be conscious about what data to collect, how to collect it and who to share it with. When launching the website, GDPR should be considered, to protect the privacy rights of people. [55]

The website is secured as much as our personal and project's limitations allowed us. However, security is one of those topics that can never be perfected enough. We could easily talk about any possible threats, and most probably there are many that our team is too small to solve. If this project enters the real market, there should be serious steps taken in the direction of security, especially when payment is implemented as well.

However, based on what we can work with, the team agreed that we need to perform pentesting and the next big step should definitely be to protect against one of the OWASP top 10, Insufficient Logging & Monitoring.

This threat is basically meaning that whenever an attack happened, you are not able to track down attackers and gain back control over your system as early as possible. Meaning that this will not necessarily prevent attacks, but gives more insights about attacks or attempts of attacks. With proper monitoring, it is also possible to notice, detect and catch suspicious behaviours, for example, brute force attempts. The express-brute middleware is already restricting the brute force attempts, however, attempts should be logged in the session and the database or monitored in some way. [53], [54]

Logging might sound like an easy task, just collecting data over the activities, but in reality, it is quite a complex topic and it's hard to get it right. There should be a nice balance in how much we monitor and collect the data passed through our website.

Conclusion

→

The goal of the project was to develop a well-performing, scalable and safe local online marketplace for digital artists that strengthens the digital art community in Copenhagen. The report highlights the main points of the development process and issues.

Started with research, first, we examined the primary target group, the digital designers. This led us to have an overview of the basic features to be implemented on the website, such as the artist dashboard and its functionalities. To stay within the scope of the project, this dashboard presents some features, such as managing the products and the personal profile. However, it is still missing a few of the users' needs such as the monthly overview of the sales.

Further research on security was made to decide on which design principles to use to protect our platform and the users from the most probable attacks. The team tried their best to follow and implement the principles. Logic and libraries were applied to secure the ArtDec website as much as we could against SQL injections, cross-site scripting, sensitive data exposure, broken access control and broken authentication. However, a website is not and will never be 100% secure from all the possible attacks. To further protect the ArtDec project there are several steps that should be taken, such as proper monitoring and logging system and adding a trustworthy payment handler service.

As a next step, we needed to define our technology stack and to reach the best result we made use of the Analytical Hierarchy Process decision-making method. The requirements were to be scalable, well-performing, secure and to be aligned with the experience and knowledge of the team. Moreover, the possibility to easily build a live chat was also important, for the social and community-strengthening purpose. As a result, we built the website with Node.js with Express framework, PostgreSQL as database using Sequelize to work with it from the Node.js server-side. Lastly, on the frontend simple HTML and Javascript is written with Handlebars templating engine. The most beneficial part of selecting this stack was the fact that client-side and server-side are sharing the same language, Javascript. Having Javascript on both frontend and backend plus a well-planned code structure speed up our development process.

In conclusion, the project resulted in a high fidelity prototype that covers most, but not all of the features planned for ArtDec. Further development is required to reach the desired goals and to make use of the chosen tech stack, such as a more complex live chat or extended functionalities on the artist dashboard. However, the team met all personal goals while experience a steep learning curve throughout the period of the assignment.

Reference list

Persona
[1]"Personas 101: What Are They and Why Should I Care?", Get Elastic Ecommerce Blog, 2011. [Online]. Available: https://www.getelastic.com/personas-101-what-are-they-and-why-should-i-care.

Owasp top 10
[2]The OWASP Foundation, OWASP Top 10 - The Ten Most Critical Web Application Security Risks. The OWASP Foundation, 2017.

Web Security Report 2nd Semester
[3] M. Pawlicki, M. Genchev, P. Todorov and B. Hegedus, Web Security - Forum.diy. 2019.

User flow
[4]A. Handley, "User flow is the new wireframe", Medium, 2018. [Online]. Available: https://uxdesign.cc/when-to-use-user-flows-guide-8b26ca9aa36a. [Accessed: 01- Dec- 2019].

Figma
[5] "Figma: the collaborative interface design tool.", Figma. [Online]. Available: https://www.figma.com/.

Analytic Hierarchy Process
[6]Analytic Hierarchy Process AHP - Business Performance Management. 2010. [Online]. Available: https://www.youtube.com/watch?v=18GWVtVAAzs

Backend Scalability
[7]"Node.js vs PHP: Which is better for web development?", Hackernoon.com, 2019. [Online]. Available: https://hackernoon.com/nodejs-vs-php-which-is-better-for-your-web-development-he7oa24wp. [Accessed: 20- Dec- 2019].

Backend Performance
[8] V. Rambhiya, "What is the difference between PHP & Node.js Development", websoptimization.com, 2018. [Online]. Available: https://www.websoptimization.com/blog/what-is-the-difference-between-php-and-node-js/. [Accessed: 27- Dec- 2019].

Handlebars
[9] D. Markov, "Learn Handlebars in 10 Minutes or Less", Tutorialzine, 2015. [Online]. Available: https://tutorialzine.com/2015/01/learn-handlebars-in-10-minutes. [Accessed: 06- Dec- 2019].
[10]"Built-in Helpers | Handlebars", Handlebarsjs.com, 2019. [Online]. Available: https://handlebarsjs.com/guide/builtin-helpers.html. [Accessed: 06- Dec- 2019].
[11]"A Beginner's Guide to Handlebars — SitePoint", SitePoint, 2015. [Online]. Available: https://www.sitepoint.com/a-beginners-guide-to-handlebars/. [Accessed: 12- Dec- 2019].

Database Scalability
[12]"Scalability, MongoDB, Vs PostgreSQL?", reddit, 2017. [Online]. Available: https://www.reddit.com/r/node/comments/564r3j/scalability_mongodb_vs_postgresql/. [Accessed: 15- Dec- 2019].
[13]"Comparing PostgreSQL vs MongoDB", MongoDB, 2019. [Online]. Available: https://www.mongodb.com/compare/mongodb-postgresql?lang=it-it. [Accessed: 30- Nov- 2019].
[14]A. Bui, "PostgreSQL vs. MongoDB", Blog.panoply.io, 2018. [Online]. Available: https://blog.panoply.io/postgresql-vs-mongodb. [Accessed: 21- Nov- 2019].

Node.js
[15]M. Wandschneider, Learning Node.js. London: Pearson Education, 2016.
[16]G. Ornbo, Sams teach yourself Node.js in 24 hours. Indianapolis, Ind.: Sams, 2013
[17]A. Mead and R. Percival, The Complete Node.js Developer Course (3rd Edition). 2019.
[18]P. Niedringhaus, "Node.js 12: The future of server-side JavaScript - LogRocket Blog" LogRocket Blog, 2019. [Online]. Available: https://blog.logrocket.com/node-js-12/. [Accessed: 06- Dec- 2019].
[19] "express-generator", npm, 2019. [Online]. Available: https://www.npmjs.com/package/express-generator. [Accessed: 06- Nov- 2019].
[20]"Modules | Node.js v13.5.0 Documentation", Nodejs.org, 2018. [Online]. Available: https://nodejs.org/api/modules.html#modules_core_modules. [Accessed: 06- Nov- 2019].

Sequelize
[21]"An Introduction to Sequelize.Js", Optimism, 2014. [Online]. Available: https://milinaudara.wordpress.com/2014/05/24/an-introduction-to-sequelize-js/. [Accessed: 06- Dec- 2019].
[22]"Manual | Sequelize" Sequelize.org. [Online]. Available: https://sequelize.org/master/manual/getting-started.html.

Pg
[23]"pg", npm, 2020. [Online]. Available: https://www.npmjs.com/package/pg. [Accessed: 06- Dec- 2019].

Pg-hstore
[24]"pg-hstore", npm, 2019. [Online]. Available: https://www.npmjs.com/package/pg-hstore. [Accessed: 06- Dec- 2019].

jwt
[25]"jsonwebtoken", npm, 2019. [Online]. Available: https://www.npmjs.com/package/jsonwebtoken. [Accessed: 06- Dec- 2019].

Bcrypt
[26]"bcrypt", npm, 2019. [Online]. Available: https://www.npmjs.com/package/bcrypt. [Accessed: 06- Dec- 2019].

Validator
[27]"validator", npm, 2019. [Online]. Available: https://www.npmjs.com/package/validator. [Accessed: 06- Dec- 2019].

Cors
[28]"cors", npm, 2018. [Online]. Available: https://www.npmjs.com/package/cors. [Accessed: 06- Dec- 2019].
[29]"CORS", MDN Web Docs. [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/CORS. [Accessed: 06- Dec- 2019].
[30]"Preflight request", MDN Web Docs. [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/Preflight_request. [Accessed: 06- Dec- 2019].
[31]A. Hevia, "Using CORS in Express", Medium, 2017. [Online]. Available: https://medium.com/@alexishevia/using-cors-in-express-cac7e29b005b. [Accessed: 06- Dec- 2019].

Express brute
[32]"express-brute", npm, 2017. [Online]. Available: https://www.npmjs.com/package/express-brute. [Accessed: 06- Dec- 2019].

Express sanitizer
[33]"express-sanitizer", npm, 2019. [Online]. Available: https://www.npmjs.com/package/express-sanitizer. [Accessed: 06- Dec- 2019].

Cookie-parser
[34]"cookie-parser", npm, 2019. [Online]. Available: https://www.npmjs.com/package/cookie-parser. [Accessed: 06- Dec- 2019].
[35]"ExpressJS - Cookies - Tutorialspoint", Tutorialspoint.com. [Online]. Available: https://www.tutorialspoint.com/expressjs/expressjs_cookies.htm. [Accessed: 06- Dec- 2019].

Express-handlebars
[36]"express-handlebars", npm, 2019. [Online]. Available: https://www.npmjs.com/package/express-handlebars.

Multer
[37]"multer", npm, 2019. [Online]. Available: https://www.npmjs.com/package/multer.

Websocket protocol
[38]"WebSocket vs Socket.io | Know The Top 5 Amazing Differences", EDUCBA. [Online]. Available: https://www.educba.com/websocket-vs-socket-io/. [Accessed: 06- Jan- 2020].

Socket.io
[39]"Socket.IO — Docs", Socket.IO. [Online]. Available: https://socket.io/docs/.
[40]"Socket.IO — Chat", Socket.IO. [Online]. Available: https://socket.io/get-started/chat/#Emitting-events.

Object Oriented design thinking
[41] http://web.engr.oregonstate.edu/~budd/Books/oopintro3e/info/chap01.pdf

MVC
[42]J. Erik Gostischa-Franta, "Best Practice Software Engineering - Model View Controller", Best-practice-software-engineering.ifs.tuwien.ac.at, 2013. [Online]. Available: http://best-practice-software-engineering.ifs.tuwien.ac.at/patterns/mvc.html.

Database
[43] J. Harrington, Relational Database Design and Implementation, Fourth Edition. pp. 53-55.
[44] L. Perkins, E. Redmond and J. Wilson, Seven Databases in Seven Weeks. p. 9.
[45] H. Das, "Object-Relational Mapping in Node.js with Sequelize | Codementor", Codementor.io, 2016. [Online]. Available: https://www.codementor.io/@hari577/object-relational-mapping-in-nodejs-with-sequelize-du1088h3l.
[46]"Database Normalization (Explained in Simple English) - Essential SQL", Essential SQL. [Online]. Available: https://www.essentialsql.com/get-ready-to-learn-sql-database-normalization-explained-in-simple-english/.

Sequelize connection pool
[47]"How to use database connections pool in Sequelize.js", Stack Overflow. [Online]. Available: https://stackoverflow.com/questions/35525574/how-to-use-database-connections-pool-in-sequelize-js.

Bootstrap template
[48]"Asbab – Furniture HTML Template", ThemeForest, 2019. [Online]. Available: https://themeforest.net/item/asbab-ecommerce-html5-template/21807871.

Development Environments
[49] C. Corsetti, T. Smith, T. Gadliauskas, S. Pour Mozafari and V. Zobbe, 2019. https://docs.google.com/document/d/1WENo5FO0DcOGY2OpBa_y4CTAsA5dhN1ikA5F2VZbDpM/edit?usp=sharing

Security
[50] https://blog.sucuri.net/2019/01/owasp-top-10-security-risks-part-v.html https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf

HttpOnly flag
[51] "HttpOnly - OWASP", Owasp.org, 2017. [Online]. Available: https://www.owasp.org/index.php/HttpOnly. [Accessed: 13- Dec- 2019].

MobilePay
[52] "| MobilePay Developer", MobilePay Developer. [Online]. Available: https://developer.mobilepay.dk/products/connect.

Logging
[53]"4 Node.js Logging libraries which make sophisticated logging simpler | Log Analysis | Log Monitoring by Loggly", Log Analysis | Log Monitoring by Loggly, 2017. [Online]. Available: https://www.loggly.com/blog/node-js-libraries-make-sophisticated-logging-simpler/.

Owasp Security risks
[54]G. Ruiz, "OWASP Top 10 Security Risks", Blog.sucuri.net, 2019. [Online]. Available: https://blog.sucuri.net/2019/01/owasp-top-10-security-risks-part-v.html. [Accessed: 10- Dec- 2019].

GDPR
[55]"How GDPR Will Change The Way You Develop — Smashing Magazine", Smashing Magazine. [Online]. Available: https://www.smashingmagazine.com/2018/02/gdpr-for-web-developers/. https://www.smashingmagazine.com/2018/02/gdpr-for-web-developers/

Deserialization
G. Messina, "10 Steps to Avoid Insecure Deserialization", Infosec Resources, 2018. [Online]. Available: https://resources.infosecinstitute.com/10-steps-avoid-insecure-deserialization/#gref. [Accessed: 15- Dec- 2019].

Shopping locally
https://blog.mass.gov/blog/consumer-advice/think-local-7-reasons-why-supporting-local-business-is-good-for-your-community/

Camilla Corsetti
Berenike Hegedus

Bachelor Thesis
Supervisor: Santiago Donoso

January 2020
Web Development
Copenhagen School of Design and Technology