jamk.fi

Weave Design

Eetu Manninen

Bachelor's thesis
September 2017
Technology, communication and transport
Degree Programme in Software Engineering

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Description

Author(s)	Type of publication	Date
Manninen, Eetu	Bachelor's thesis	September 2017
	Number of pages 54	Language of publication: English
		Permission for web publication: x

Title of publication

Weave design

Degree programme

Software Engineering

Supervisor(s)

Rantala, Ari

Assigned by

Description

The main goal of the thesis was to implement a web application where artisans would be able to make necessary drawings easier than when using traditional pen and paper. These drawings are used when weaving with a loom. It was important that with the application it would be possible to draw the picture starting from either the basic picture or from the main canvas. It was necessary to be able to construct the main canvas from the sides and vice versa.

One web application that would not have to be installed, was developed using web application development techniques. The MEAN stack, where Angularjs framework was replaced with Angular-framework, was used as a base for the web application development. The application was hosted on Heroku, where a REST API was set up, through which users could save their work, using Node.js backend technology with added functionalities from Express.js library. For authenticating the users and saving their passwords, Auth0-service was used. Auth0 also provided the log in component. The application interface was programmed with Angular framework, and the appearances were modified with Bootstrap library. For saving the users' creations, MongoDb was used with added functionalities from Mongoose library. The MongoDb was hosted on MongoLab server.

All of the functionalities specified in the requirement specification were met, except for managing user passwords from the user interface. The application was taken into use by the writer's sister, who uses the application to draw and design weaving patterns more easily than before.

Keywords (subjects)

Angular, JavaScript, Node.js, Express.js, MongoDb, Weaving

Miscellanous

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Manninen, Eetu	Opinnäytetyö, AMK	Syyskuu 2017
	Sivumäärä 54	Julkaisun kieli Englanti
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Weave design

Tutkinto-ohjelma
Ohjelmistotekniikan koulutusohjelma

Työn ohjaaja(t)
Rantala, Ari

Toimeksiantaja(t)

Tiivistelmä

Opinnäytetyön tavoitteena oli toteuttaa web-sovellus, jolla artesaanit kykenisivät tekemään kankaan kudonnassa tarvittavia piirustuksia helpommin kuin traditionaalisesti paperille tehtäessä. Sovelluksessa oli tärkeää, että kankaan kuvan voisi piirtää joko kankaan peruskuvasta alkaen tai sitten suoraan keskikanvaasiin piirtämällä. Reunoista täytyi kyetä rakentamaan keskikanvaasi ja keskikanvaasista myös reunat.

Työssä toteutettiin web-ohjelmointiteknologioita käyttäen web-sovellus, jota ei tarvitsisi asentaa käyttäjän koneelle. Web-sovellus ohjelmoitiin käyttäen MEAN-sovelluskokoelmaa, jossa Angularjs-sovelluskehys korvattiin Angular-sovelluskehyksellä. Palvelimena työssä toimi Heroku, jonne pystytettiin Node.js-palvelin. Palvelin ohjelmoitiin avaamaan REST-rajapinta, jonka kautta käyttäjät kykenevät tallentamaan tuotoksiaan. Node.js-palvelinta tehostamaan otettiin Express.js-kirjasto. Käyttäjien kirjautuminen toimii Auth0-palvelun kautta, joka autentikoi ja tallentaa käyttäjiä. Auth0 tarjosi myös kirjautumiskomponentin. Sovelluskäyttöliittymä ohjelmoitiin Angular-sovelluskehyksellä ja ulkonäön muokkaamiseen käytettiin Bootstrap-kirjastoa. Käyttäjien piirustusten tallentamiseen käytetään MongoDb-tietokantaa, joka hyödyntää Mongoose-kirjastoa helpompaa tiedon käsittelyä varten. MongoDb sijaitsee MongoLab-palvelimella.

Sovellukseen toteutettiin kaikki sovelluksen vaatimuksiin määritellyt toiminnot, paitsi käyttäjän salasanan vaihtamisen mahdollistaminen suoraan käyttöliittymästä. Sovelluksen otti käyttöön opinnäytetyön laatijan sisko, joka kykenee ohjelmaa hyväksikäyttäen tekemään kankaan kuvia huomattavasti helpommin kuin ennen.

Avainsanat (asiasanat)
Angular, JavaScript, Node.js, Express.js, MongoDb, Kankaankutominen

Muut tiedot

Table of Contents

1 Introduction

1.1 Overview

This paper works as a supplementary document for the program created as the writer's Bachelor project. This document will go into detail of the used technologies and, in some cases also, explain why specific technologies were chosen. The project itself is a web application for designing weave patterns that can be used for weaving. There are not many programs for this kind of work in the first place and at the time of starting this project, there was only one provided as a web application. The aim is to develop a web application that can be used to draw and store weaving patterns and expand on the functionalities offered by similar web applications available at the time of writing.

1.2 Client and partner

The idea for this application came from the writer's sister who works as an artisan for a living and acted as a client and partner for this project. She expressed a need for this kind of application as satisfying solutions were not available in the format that would have been comfortable. The requirements were mostly planned together, and she provided invaluable help when learning about weaving and thinking about the logic behind it.

1.3 Goals of the thesis

1.3.1 Concrete goal

The concrete goal of the bachelor's thesis was to plan, develop and deploy a web application for weave design that could be used anywhere, without the need to install an application. The user should be able to design weaving patterns, save them

on to their computer as a picture or in the cloud. The users should also be able to log in to the service and manage their designs anywhere.

1.3.2 Personal goal

The personal goal of the project was to learn about making a web application in its whole lifecycle. From planning to designing to deploying and trying to learn best practices in all the different steps. It was also important to learn about different technologies and compare them to see which ones make the most sense in the context of the application. Other smaller goals were to learn about security, continuous integration chain and the writing of simple tests. In other words, the goal was to try to make as 'perfect' an application as possible by spending time in comparing different choices that were available.

2 Weaving

2.1 What is weaving

In order to understand how the application works, it is first necessary to have a basic understanding of weaving. In simple terms, weaving involves two strands of thread that are interlaced. One thread is always horizontal and the other is always vertical. The vertical thread is called warp and the horizontal thread is called weft. There are many different weaves but for simplicity's sake, only one of them is looked at, called the plain weave. It is a simple pattern of over-under-over-under, and so on. The program would draw this pattern in the fashion of Figure 1, and in Figure 2 it is possible to see the real-world application of this. In this case, black represents the vertical thread (warp) and white represents the horizontal thread (weft) (Muscato n.d.).

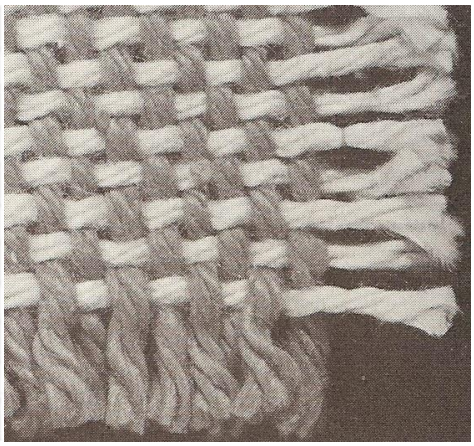

Figure 1 Simple weave in program Figure 2 Simple weave (Example of a basic weave)

2.2 What is weave design

Normally the black and white map (Figure 1) is drawn by hand, which can be an exhausting and time-consuming process. To expand on Figure 1, Figure 3 demonstrates how the design process works and some new columns have been added. When doing the design by hand, the weaver usually starts from the weave and works their way to the binding. This is because, if done the other way around, the result would appear only after considerable time had already been put into working on the weave. The warp, weft and binding are then used as a guide when weaving, producing the weave that was drawn on the paper before.

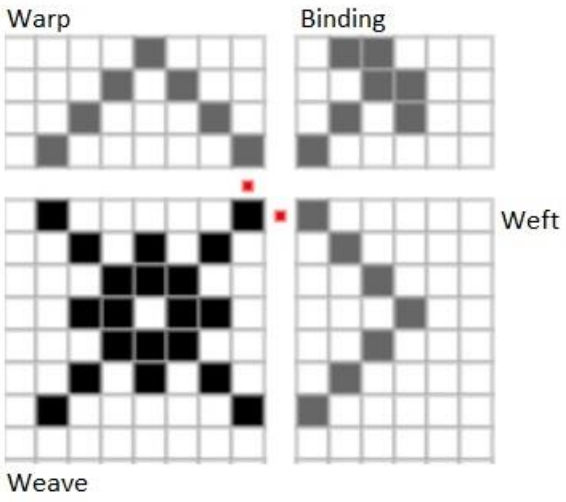

Figure 3 How the weave is made

In warp, every thread that interlaces differently is marked on its own row in the weave. If they are the same, they are marked on the same row. The same rule is then applied to the weft, only vertically.

2.3 How does it work?

When building the weave from the warp, weft and binding, in this case, the topmost row is inspected first for any grays boxes. Then the binding is looked at to see if there are any grey boxes on the same vertical line as the weft's box was. If there are, then the horizontal row is also inspected for a gray box on the same line. When this last grey box has also been located, a black box is drawn in the weave on the intercrossing of these two boxes on the weft and warp. If the binding had more than one grey box on the same vertical line, this process is repeated; however, this time one goes horizontally starting from the second box and draws the black box in their intercrossing in the weave. This process is then repeated until all the grey boxes have been checked. It is worth noting that multiple grey boxes on the same horizontal line in the weft will not be considered. This process can be done by starting from the warp as well. It can also be done from the weave to weft, warp and binding.

3 Requirement specification

3.1 In general

Writing a requirement specification is an important step in designing an application. It might be quite a heavy time investment in the beginning, however, it often ends up saving a great deal more time in the end. A well-written requirement specification has many benefits including (Kazankow 2016):

- It forces a rigorous assessment of requirements before design and programming can begin and thus, minimizes later redesign.

- It also offers a chance for the developer and the client to come to an agreement about the projected outcome of the software, reducing a number of misunderstandings.

3.2 Requirements in the context of the application

The application will be developed under no license, which basically means that no 3rd party has the permission to use, modify or share the code of the application (No license n.d.). However, the application will be usable freely through the website. This is to ensure that the product is not tampered with in malicious ways that could affect the end-user in a negative way. The application will have a user guide available through the website.

3.3 Web application

3.3.1 Architectural requirements

The web application should be completely usable through all major internet browsers, such as Chrome, Firefox and Edge. It should also be noted that only the latest versions of the browsers are going to be supported, although, in most cases older versions will offer similar performance. The application is going to be developed desktop first and due to the inherent difficulty of using this kind of application with a phone, phones and tablets are not going to be tested to be working.

3.3.2 Functional requirements

Users should be able to use the application without logging in. Users should also be able to log in using a personal username and password. It should also be possible to explicitly log out of the application. Users should be able to change their password. Users must be able to draw weave patterns. Users should be able to save the patterns to the server if they are logged in and also edit them afterward. Users should also be able to remove these designs from the server when no longer needed. Users should be able to print out or export the pattern in png. Users should be able

to change the colors of the lines. Users should be able change the size of the squares. Users should also be able to change the size of the pattern. The application should also be fast to use.

4 Technology choices

4.1 General info

In software development, it is important to do the groundwork and study different options to be able to make a good choice. It is not always as simple as choosing the technology stack that one is the most comfortable with, rather, choosing it based on the application and what makes the most sense. However, when the developers working on the project are not completely familiar with best business and coding practices, it is more important to choose something and start learning.

4.2 Technology stack

4.2.1 What is a technology stack?

“A tech stack is a combination of software products and programming languages used to create a web or mobile application. Applications have two software components: client-side and server-side, also known as front-end and back-end.” (Choosing the right software stack) The most common stacks in use today include LAMP (Linux/Apache/MySQL/PHP), MEAN (MongoDb/Express.js/AngularJS/Node.js), Django (Python/Django/Apache/MySQL) and Ruby (Ruby/Ruby on Rails/RVM/MySQL/Apache/PHP). There are also many variations of these popular stacks where usually one or two technologies are swapped with another one, for example, MEEN stack that changes Angular 2 to Ember.js (Edelman 2017) (Wodehouse 2015).

4.2.2 Front-end

Front-end typically means the “visible” part of the application. This means everything that the user can see on their screen when they browse a web page. This includes everything that can be seen in the browser’s developer options. Therefore, frameworks such as Angular are not backend even if it might seem like the user never sees the things written. Angular is compiled into JavaScript and provided to the users’ browser freely, and thus, with enough time and energy, everything written with TypeScript can be changed from within the browser.

Front-end is technically the easiest choice, as practically ~~a~~ll websites are going to use HTML, CSS, and JavaScript. The choice then is which, if any, framework is going to be used. There are so many JavaScript frameworks to choose from that it boils down to a personal taste, as all of them are a superset of JavaScript, meaning that all the same basic functionalities can be found in all of them, however, each of them incorporates some extra ones. The most popular ones at the time of writing are jQuery, Angular, and React.js (Elliot 2016).

There are also many other choices for developing a web application, such as ASP.Net and PHP with their various frameworks. These were abandoned as user experience had to be as fast as possible and these frameworks do not provide such functionality easily. Granted ASP.Net does have UpdatePanels that can update data on the website without a need to reload, however, they come with their own set of problems, the biggest of which is the fact that the element being updated is completely created anew. This means that all the JavaScript function subscriptions are lost and need to be reapplied on an update.

As such, the top choices for this project were Angular, React, jQuery and Vanilla JavaScript. This is because the app was required to feel “fast” or “snappy”, and these libraries/frameworks are capable of supporting SPAs. SPA is an acronym for single page application, which means that the application loads everything it needs when the user navigates to the website, unless the specified path is lazy loaded, which means loading the page’s code only when the route is activated. The application is then running on a single instance, even though the URL is changed in the browser.

This allows web apps to feel incredibly fast since everything is already loaded, which might result in longer loading times for the first visit, however, the perceived speed increases when actually running the application trumps the negatives in this case. After doing some testing with each of them, it was decided that Angular would be the framework to be used in this project. The reasons were mainly due to its popularity, previous experiences and because the Angular way of doing things seemed good. One of the major factors was the fact that Angular provides ahead of time compilation (AOT) easily out of the box, which means that the typescript code is compiled to JavaScript before it is uploaded to the server. This means that all the rendering and processing is done on the user's browser, thus causing less load on the server computer, which in this case is not very powerful. AOT also decreases the load times between page changes, as compared to JIT compilation. Just in time (JIT) compilation means that the code is compiled and provided to the user as they request parts of the website. It was primarily used before AOT compilation became popular. One more added benefit of angular was the fact that, when compared to jQuery and Vanilla JavaScript, the SPA is already configured. The main problem with React was the fact that had it been chosen, Redux would also need to be learned, increasing things to learn, also, the fact that it is maintained by Facebook, which was a personal problem. jQuery's biggest flaw was its age, even though it is still very popular and robust, although, jQuery had to also be included in the project thanks to Bootstrap, which requires jQuery to work properly. Vanilla JavaScript was abandoned mainly because the frameworks and libraries extending it offer many valuable components and functions that would have to be made separately, increasing the workload (Mittag & Ore4444 2017) (Skólski 2016).

4.2.3 Backend

Backend refers to the part of the web application that will never be seen by the user, not even from their developer tools. Backend is completely separated from the user and these two interfaces are only able to communicate through the API layer. This is to ensure that nothing unwanted happens in the critical server processes, such as dropping database etc. (Sonmez 2016).

On the back-end side of things, things quickly become more complicated as there are several choices with different languages and their benefits. Common back-end languages are Ruby, Python, JavaScript, PHP and C# with their respective frameworks.

JavaScript was chosen as the backend language using the node.js framework with added functionalities from Express.js. This choice was made mainly due to the simplicity of node.js. A server file could be created and data from server started serving in matter of hours instead of weeks. Heroku also offered immediate, hassle free, configuration of a MEAN stack (N being node.js) application. The backend choice is not very critical in a project of this scale as there are almost no API requests or other needs for users to communicate with the database and thus the choice was just about how easy it was to use and how fast it was to setup.

4.2.4 Database

There are currently two primary database types, SQL and NoSQL. Both have their benefits, however, ultimately both do the same thing, store data. SQL stores related data in tables and provides very powerful tools for querying specific information, for example, table JOINs that allows retrieval of data from multiple tables in a single command. NoSQL, on the other hand, stores data in JSON-like name-value documents and provides excellent performance and scalability (Buckler 2015).

For the project, MongoDB was chosen because it is a NoSQL database. The canvas data in the weave design is saved in JSON format, which is a great deal easier to save in a NoSQL database than in SQL database, because, for example, Mongo saves everything as JSON objects. Also, there are not many benefits from having strong relations in the project as almost all the data saved on the database is going to be only canvas data. In addition to the personal MongoDB, where canvases are saved, auth0 has its own separate database.

4.2.5 MEAN stack

As can be seen from above, MEAN stack ended up being the technology stack used as it satisfies all the needs of the program and is one of the most popular tech stacks,

which in turn means more support in development. Its biggest benefit as a bachelor's thesis project stack is Heroku, which allows users to host MEAN projects freely in their cloud if their needs are not too high.

4.3 Package management

4.3.1 What is a package manager?

In short, a package manager is software that manages different software installed on the computer and allows the user to easily install new software, upgrade installed software, or delete installed software. For example, the very popular bootstrap can be installed through a package manager very effortlessly and its dependency to the application is then easily tracked. Packages also track dependencies between each other. Some of these dependencies might be critical, which means that, the package will not work without another package. In most cases these dependencies are only recommendations, and can be safely ignored (What is a package manager? n.d.).

4.3.2 Node package manager

Node package manager, or NPM, is one of the most popular package managers in the web development scene at the moment. It is included with node.js, which a very popular server, and this partially explains its popularity. NPM makes it very easy for JavaScript developers to share and reuse code, and it also makes it easy to update the code that one is sharing. NPM has a very large library of free to use code shared by other users. These shared codes are called packages, and NPM takes care of them in a package.json file. The general idea of a package is to provide a solution to a problem, and with many packages available, these can then be used as building blocks to create better applications (What is npm? 2017).

4.4 Web hosting or cloud database service

4.4.1 General info

These days there are many different kinds of hosting solutions and all of them serve a different need. Some of the notable ones are shared hosting, VPS and the cloud computing stack: SaaS, PaaS and IaaS. Figure 4 illustrates this computing stack. Shared hosting and VPS have been around for a long time, which makes them more robust and time tested. However, the need to deploy builds constantly and make programs faster than before has increased and so have the requirements. Usually with shared hosting and VPS, a great deal of configuration and maintenance on the server are required (Gil 2017).

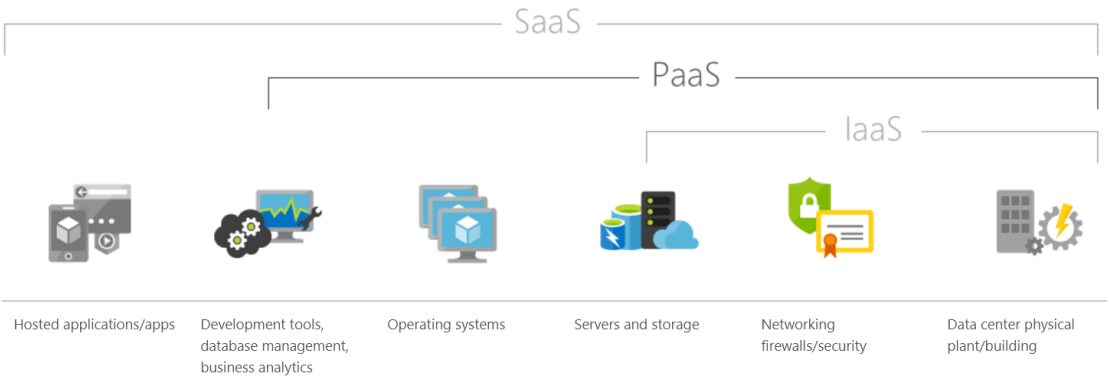

Figure 4 Rough representation of SaaS, PaaS and IaaS layers (Overview of IaaS, SaaS and PaaS)

4.4.2 SaaS

SaaS (Software as a Service) means that a company rents or borrows a software that can be ran in the cloud instead of locally on one's computer. SaaS offers many benefits when compared to local software. The most notable one is the security aspect. If a vulnerability is found in the software, fixing it on the version that is up in the cloud fixes it for everyone who uses it, whereas in the case of local software, it gets fixed whenever the user decides to install the update. Using SaaS also reduces the cost for everyone. The user does not have to pay a large upfront fee to use the program, which makes SaaS more attractive. On the downside, one does not own the product, and in the long run it might end up costing more than having a local one. The company providing SaaS also does not have to spend plenty of time supporting

users with their technological problems that usually stem from different hardware. If something is wrong, it can simply be fixed in the live app and thus it is fixed for everyone (Gil 2017).

The downside of SaaS is the risk of using it. The service provider 'owns' everything in the cloud and can use this fact for their own purposes, which is why a high level of trust and security is needed between the provider and the user. The software being in the cloud also means that if, for some reason, the server goes down, the users can do nothing but patiently wait for it to go back up again (Gil 2017).

Some products offered as SaaS include Microsoft office and Adobe creative cloud.

4.4.3 IaaS

IaaS (Infrastructure as a Service) means that the hardware is provided by an external provider and managed for users. All cloud computing services provide access to computing resources in a virtualized environment. In the case of IaaS, however, the client is given access to these virtualized components in order for them to build their own platform. This requires a lot of experience and expertise from the customer, and thus is not very feasible on its own for smaller products or companies. Few examples of IaaS would be Microsoft's Azure and AWS (amazon web services). Both of these are cloud computing platforms that can be configured a lot and can be used for multitude of purposes. From hosting websites, having build servers to simply hosting codebases (Sharma 2017) (What is IaaS? n.d).

4.4.4 PaaS

PaaS (Platform as a Service) is a complete development and deployment environment in the cloud with resources enabling the user to deliver everything. The resources are purchased on pay-as-you-go basis, which simplifies scalability. PaaS allows the user to avoid the expense and complexity of buying and managing software licenses, the underlying application infrastructure and middleware. The developer has to only to worry about the applications and services being developed. Unlike IaaS, PaaS is designed to support the whole web application lifecycle from building, testing and deploying, to managing and updating. Commonly PaaS is used

as a development framework, which lets the developers use built-in components and cloud features, which significantly reduces the amount of coding developers have to do, which is also one of the biggest advantages of PaaS (What is PaaS? n.d.).

4.4.5 Heroku

Heroku is a Platform as a service that allows developers to quickly and easily setup their server, database and deployment. Heroku was chosen as the hosting platform as it provides free hosting in the scope of the app, domain linking and full MEAN stack support. The application then will be SaaS running on top of a PaaS, whereas Heroku runs on top of an IaaS.

4.5 REST API

API (Application Programming Interface) means the layer that separates different functionalities to a deeper layer. RESTful API calls are are stateless, meaning that they can run asynchronously. This functionality is very useful because stateless components can be redeployed if something fails and are easily controlled by changing how the URL is decoded. The basic presumption is that nothing can be retained between service executions, allowing every call to be treated as a new instance. API in itself is a black box that the program calls with specific parameters and gets a response from the server. Authors of the API can thus create controlled interfaces from where other parts of the same application, or even outside applications, can request for a specific information from the API server. RESTful APIs are quickly becoming the de facto standard of web development (Rouse 2016).

4.6 Summary of technology choices

The technology choice is the popular MEAN stack that is preferred by many developers for the added benefit of everything being JavaScript in the end. This allows the developer to focus on what is important and not on the semantics of different coding languages. Node also comes with its own package manager NPM, which is a very popular package manager with the largest amount of third party free

to use libraries. Figure 5 illustrates a very rough chart of the flow of the applications and some of the relations between different parts.

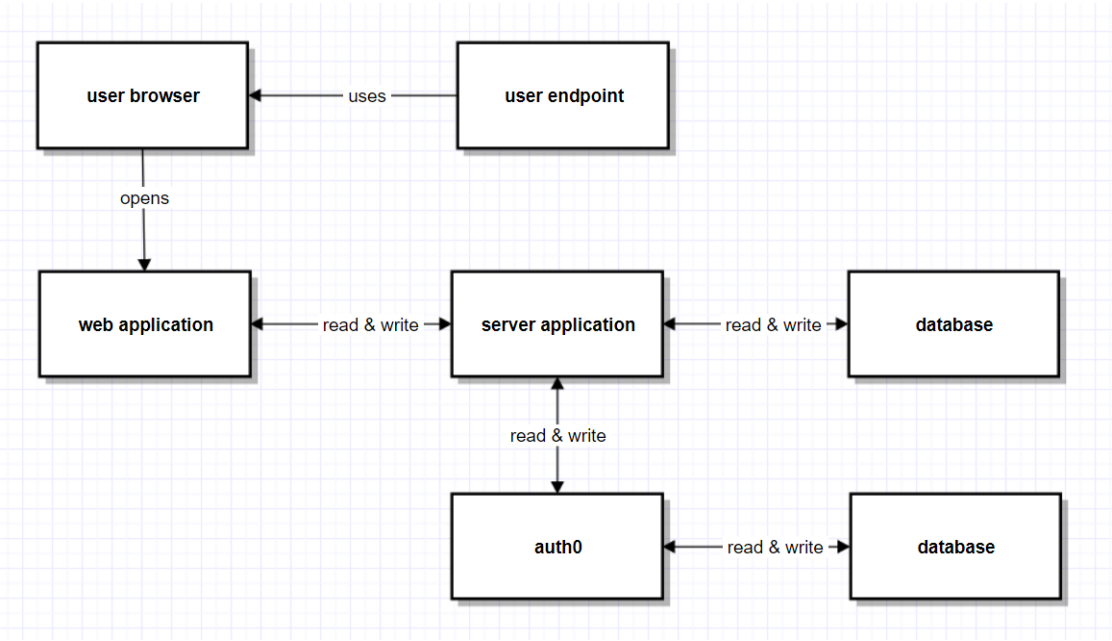

Figure 5 Rough representation of the application flow and dependencies

5 Security

5.1 In general

When developing applications deployed on the internet accessible for anyone, matters quickly become very complex, especially if the service handles user data and passwords. Even if this is not the case, there are various other ways that a malicious user could bring harm to the users, for example, by denying the use of the service. This is evident in the fact that web application attacks are the most frequent pattern in security breaches. There is an ongoing effort to increase the security of the web and many companies have spawned from this effort. One of the most known is the non-profit organization OWASP. (OWASP Top 10 Vulnerabilities n.d.)

5.2 OWASP Top 10

What OWASP is the most known for is their top 10 vulnerabilities list. It is a compilation of the most common vulnerability types where applications fail. For

example, the most common vulnerability across the years is injection. The website called W3Schools, defines injection as follows (SQL Injection n.d.):

- SQL injection is a code injection technique that might destroy your database.
- SQL injection is one of the most common web hacking techniques.
- SQL injection is the placement of malicious code in SQL statements, via web page input.

SQL injection most often occurs when user input is asked, and the query is not properly parameterized, e.g. injections based on batched SQL statements. One such query could be “SELECT * FROM users; DROP TABLE suppliers”. As an example, an input field where the user is asked for username is illustrated below.

Username input field

This is a very typical scenario and quite commonplace when browsing the web. What if something was written after the username? This is illustrated below.

Malicious SQL command after username

If the query is properly parametrized, there is no problem, however, if the query is written something like this: “userId = getUserId(‘userId’)” and then executed like so: “sql=’SELECT * FROM users WHERE userId = ’ + userId”, this results in an SQL statement like this: “SELECT * FROM users WHERE userId = ‘ImAUser’; DROP TABLE Suppliers”. The suppliers table would be gone unless there were backups. (SQL Injection) OWASP’s top 10 list is something to be evaluated when designing a secure modern web application.

5.3 Auth0

‘Auth0 is a service that abstracts how users authenticate to applications.’ (Auth0 Overview n.d.). Auth0 provides quite effortless user and password management to

any language or stack. It provides multiple ways of signing in and up and even their own login widget. Most importantly, the management of users is outsourced to a bigger company that sells security. Auth0 is also widely configurable and the developer can specify different kinds of rules and hooks to provide additional functionality. For example, enforcing a user to verify their email before being able to sign in and making API call to a backend on first login. In small applications, robust user security is quite hard to achieve. Auth0 provides a free plan for developers; up to 7,000 active users for free, and after that it asks for payment. For a student project, 7,000 active users should be plenty (Flexible pricing, for developers and companies n.d.).

6 Implementation

6.1 In general

Developing a complex application as a one-person team can be quite challenging. It offers plenty of flexibility and freedom but also possibly requires the person to do more research as there is no one to ask for help. Different responsibilities of the application should be defined well as to prevent duplicate code, and some of the functionalities have to be developed concurrently with each other to be able to test them.

6.2 Planning

Planning a software beforehand can be time consuming, however, usually it pays itself off in the long run. The requirement specification gives a good basis for what the application needs to be able to do, and thus it is later easier to plan these different requirements and develop them. In the thesis project, the requirement specification is very “bare bones” and does not limit the used technologies or anything else. It simply provides a basis for planning. In the context of a thesis, planning an application gives the developer a chance to think about what he is going to do and allows for experimentation and research.

6.2.1 Visual planning

Often it is not enough just to plan an application on the paper. This might leave the developer in a situation where they do not know what they want from the website. In situations like these, mocking is very powerful. Spending time mocking an application might feel like a waste of time, since the same could be done in the application and not in a mock. This ignores the fact that usually people do not know what they want, and mocking is the best way to showcase what was understood. There is no sense in starting to develop something that will be scrapped as soon as it is done. Figure 6 was created in a web application called NinjaMock. It is a free mocking tool allowing users to mock their layout without complicating things with styles and colors.

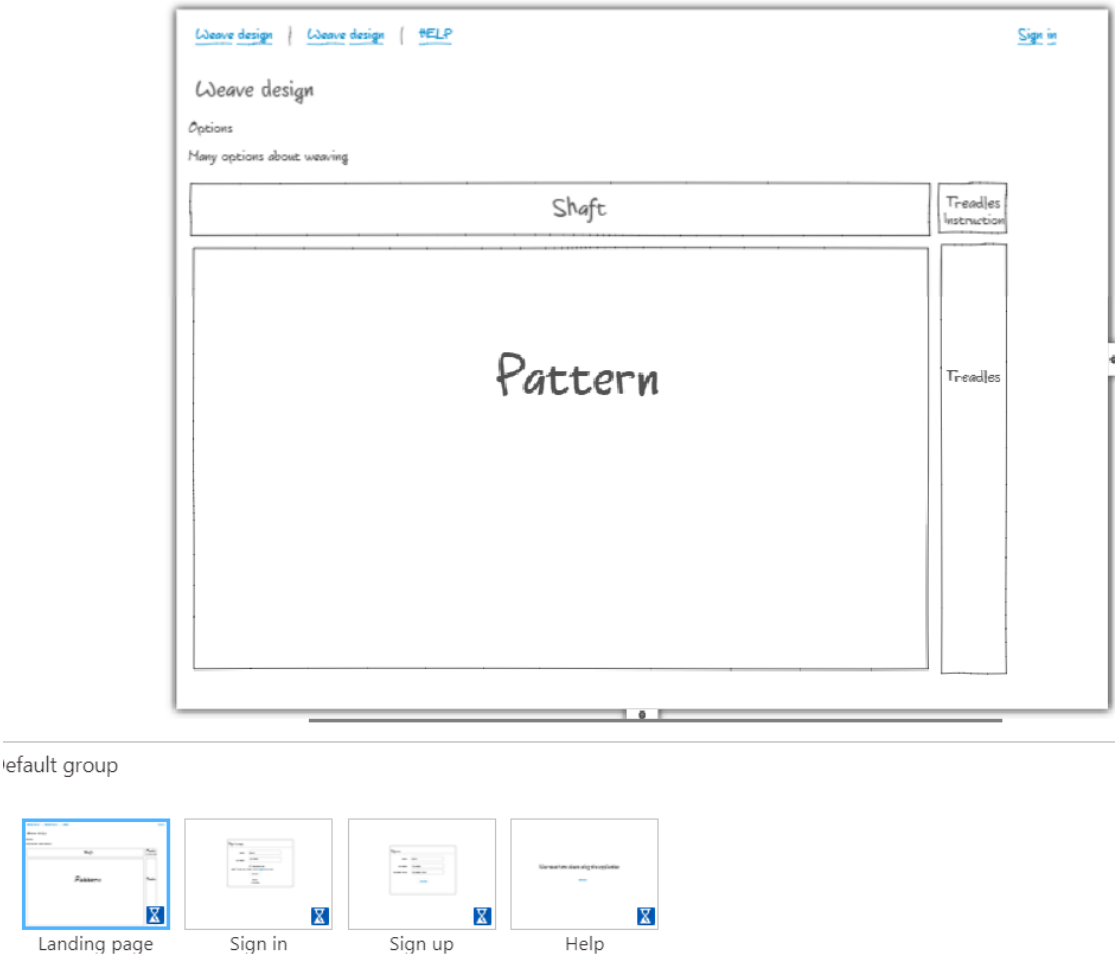

Figure 6 Rough mock of the weaving app user interface

6.2.2 Weave logic

The hardest part of the application was the weave logic and how to implement it in a way that would make sense programmatically and still honor the wishes of the end-user. For example, when making the design section of the page, it was found out that it is not always a good idea to provide instant feedback to the user. At first, it was made so that when the user clicked on the main canvas, the sides would update instantly, which made the process very unpleasant as the canvas kept moving around after clicks. What made this especially hard was the fact that it was necessary to study weaving and understand the underlying principles in that, which basically meant extra work for something not strictly related to programming.

6.2.3 API design

Designing an API is difficult since it might not be clear what functionalities are needed from the API in the end, and thus the API design was more on a need basis and not pre-planned. The API works through the express server which exposes some routes for the frontend to call. These paths were mainly used for getting and deleting canvas data the user had saved. One API route was also made for an Auth0 rule as it was required to save the user's id to the mongo database for canvas management. Auth0 sends a base64 encoded the request to the specified path where the request is checked and validated. If it is valid, it returns a success code to auth0, which then changes a Boolean to true to signify that the user has also been added to the MongoDB. If it fails, the process is then retried upon further logins until it completes.

6.3 Development software and services

To help to develop a software, developers usually use many kinds of applications and services. Many of these choices are usually made out of preference, and not necessarily present the best tool for the job.

6.3.1 Development environment

Technically, only a text editor is needed for programming. Even the Windows default Notepad gets the job done. Although, in most cases, this is not recommended and usually not even feasible since the complexity of the program and the number of used libraries increases alongside the amount of code. To help with this, there are many kinds of IDEs (Integrated Development Environment) offering various helping functionalities such as automatic fill. Basically, this means that the IDE suggests for the developers as to what they were going to type next which the developer can then decide to autofill. This is especially helpful when one is not familiar with different functionalities available, for example, to an object. Also, a lifesaver is error checking. The IDE continuously monitors the code being written and tries to find syntax errors that would lead to fatal errors when building. Of course, this does not mean that the programmer is making perfect code; it just reduces the amount of errors made. This project was started with Atom, a free customizable IDE that offers many plugins and extensions. Later Atom was replaced by Visual Studio Code, mainly because Atom did not perform very well with the increasing size of the project.

6.3.2 Source control

Developing a program is usually an incremental process, where small steps make the application. Previously, source control was a concept of manually saving one's files in different version folders and then restoring them from those as needed. This quickly becomes very arduous as the size of the project increases and there is no easy way to keep those files with one, no matter where one is. Luckily, these days there are many options for local or cloud based source control. For example, mercurial, Git, TFS and SVN. The most popular of these is probably Git, as it is included in Ubuntu by default and offers powerful integrations to cloud services, such as Github or Gitlab. This project was also made with the help of Git, and although there are many graphical Git clients, most of these opt for a more streamlined use of Git and thus, do not utilize all the functionalities available to Git that is being ran from the command line.

6.4 DB

The database used in this application is very simple and has only two models. The models are basically the same as tables, meaning they store data as key-value pairs. The user can have many canvases but the canvas can only have one user. The contents and their relations can be seen in the figure below (Figure 7).

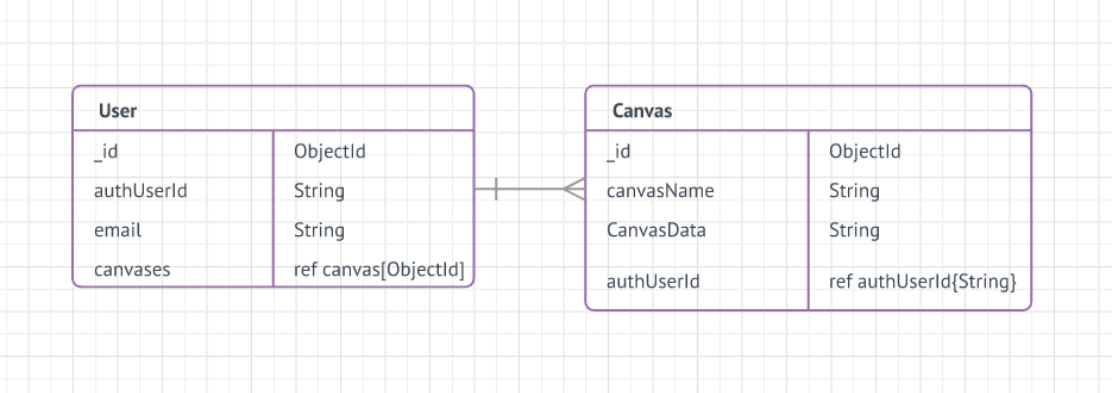

Figure 7 Relations of the models and their contents

The database only saves the canvas data that the user has saved on the server and the user info. The user info is mainly required for easy searching of canvases that belong to a specific user. The user model contains four fields, _id, authUserId, email and canvases. The _id signifies the random unique ID that MongoDb automatically assigns to every new object created inside the database. AuthUserId is the same as the ID auth0 provides to the user in their internal database upon signing up. Email just contains the email address of the specific user. Canvases object contains all the references (ids) to the canvas objects that have been saved on this specific user. The canvas model also has four fields, _id, canvasName, canvasData and authUserId. _id is, again, the unique id mongo provides. CanvasName signifies the name the user has provided for their saved canvas. CanvasData contains all the data required to reconstruct the main canvas. AuthUserId contains the reference to the user owning this specific canvas.

6.4.1 Database optimization

The database was hosted in MongoLab, which is a PaaS service specializing in MongoDb data storage. MongoLab was chosen thanks to Heroku providing a direct plugin allowing connections from the server to the MongoLab server. MongoLab's

main fault was the fact that free users are only provided 500MBs of free storage, which is not much. This means that the data saved to the server per user had to be limited to 10 and the data being saved had to be compressed. As the main canvas array contains the most amount of data, it was completely left out. This was not a problem since it can be reconstructed from the data contained in the sides. Since the vertical and horizontal arrays can only contain one click per line, it was possible to reduce them from two-dimensional array to a one-dimensional array. In practice, this means that a single array can contain all the same data thanks to the fact that the length of the array can act as indication of the location of the first value in a two-dimensional array, and the contents of that element then contains the info for where in the two-dimensional array the click was. As the result canvas (top-right) is already a compressed version of the main canvas, it cannot be compressed further; however, luckily it was possible to simplify the data of the color array. Instead of saving every row as the color string, which takes much space, it is possible to save all the unique colors in a single array, and then save only an index of the color that was contained in the specific memory slot in the color array. Before optimizations, a canvas of size 1000 x 1000 took closer to 50KBs of space per canvas. After these optimizations, saving the same canvas took 15KB. Effectively this means that 6700~ users would be able to save five canvases of the size 1000 x 1000 each. In practice, this optimization works on the assumption that the main canvas can be reduced to be smaller than the result canvas, and that the user does not give a unique color to every line in the whole canvas. If this were the case, saving the canvas in this manner would actually cost more.

6.5 Backend and API

As the application was not very heavy on handling critical user data, the backend and API ended up being quite simple in practice.

6.5.1 Node.js server with express.js

The server file consists of four main parts: making the database connection, forcing https, exposing paths for API calls, and starting the server. Making a database

connection from the server is very simple when using mongoose. It needs to be imported and then writing “mongoose.connect(url)” and providing an URL as parameter is all that is required. Similarly, express.js makes the rest of the configuration also incredibly simple.

The code seen in Figure 8 enforces the usage of https instead of http. It listens to all the requests made to the server, and if it comes from an http link, the http request is then redirected to https. When exposing routes to be used, it is only required to import the API configuration and then tell the server to use it with “app.use()”.

```
const path = require('path');

// If an incoming request uses
// a protocol other than HTTPS,
// redirect that request to the
// same url but with HTTPS
const forceSSL = function() {
  return function (req, res, next) {
    if (req.headers['x-forwarded-proto'] !== 'https') {
      return res.redirect(
       ['https://', req.get('Host'), req.url].join('')
      );
    }
    next();
  }
}
// Instruct the app
// to use the forceSSL
// middleware
app.use(forceSSL());
```

Figure 8 Forcing https on user

In the code (Figure 9) the folder from where the files should be served from to the user is expressed, which basically means that the server usually stores the files that it serves to the user in a separate folder, thus the location needs to be told to the server. Then the server is told which port it should be listening to. Unless a port is specified to node on startup, it will default to 8080, which is the default port Heroku

uses. Lastly, as the application is a single page application, the server needs to be told that all GET requests send the user back to index.html. This needs to be done because as mentioned before, SPAs work only on one page, even if the browser's URL looks different to the user when navigating the website.

```
// Run the app by serving the static files
// in the dist directory
app.use(express.static(__dirname + '/dist'));

// Start the app by listening on the default
// Heroku port
app.listen(process.env.PORT || 8080);
//app.listen(4200)

// For all GET requests, send back index.html
// so that PathLocationStrategy can be used
app.get('/*', function(req, res) {
  res.sendFile(path.join(__dirname + '/dist/index.html'));
});
```

Figure 9 Configuring node server to run on Heroku

6.5.2 API

Since there were not many requests that the user could make, there are only five API paths and currently only four are used. The first path is for getting all the canvases the user has and returning them. The second path is for saving canvas data. The third is for deleting a specific canvas from the database. The fourth is meant only for auth0 to add the user on signup to the database. The fifth one, which is not currently used, is for updating a canvas's data. Only the second and the fourth path will be discussed here in detail.

Before allowing the user to save the canvas (Figure 10), it will first be checked that the user does not already have 10 canvases saved. Here Mongoose's functionalities are used to help with MongoDb queries. Canvas.count gets the canvas model and runs a query for all canvases that have a reference to the authUserId. If there were

any errors while doing the query, an http response 500 is returned and if the user has more than 9 canvases linked to their id, an http response 418 is returned with description of the problem. If neither of these are true, one goes to save the canvas data. Figure 11 illustrates this.

```
function checkCount (user_id, req, res){
    Canvas.count({authUserId: user_id}, function(err, count){
        if(err){
            return res.status(500).json({
                title: 'An error occured',
                error: err
            });
        }
        if(count > 9){
            return res.status(418).json({
                title: "I'm NOT a coffee pot >:(",
                error: 'Not enough storage space'
            });
        }
        else
        {
            saveCanvasData(user_id, req, res);
        }
    });
}
```

Figure 10 How to prevent the user from saving too many canvases

```
function saveCanvasData(user_id, req, res){
    User.findOne({authUserId: user_id}, function(err, user){
        if(err){
        return res.status(500).json({
                title: 'An error occured',
                error: err
            });
        }
        var canvas = new Canvas({
            canvasName: req.body.canvasName,
            canvasData: req.body.canvasData,
            authUserId: user_id
        });
        canvas.save(function(err, result){
            if(err){
                return res.status(500).json({
                    title: 'An error occured',
                    error: err
                });
            }
            user.canvases.push(result);
            user.save();
            res.status(201).json({
                message: 'Saved canvas data',
                obj: result
            });
        });
    });
}
```

Figure 11 Example of how to save a canvas in the database

Here (Figure 11) Mongoose's findOne query function is used which gets the user that has a matching id with the authUserId provided. Again, if an error occurs 500 response is returned; otherwise, a new canvas model is created with the data provided by the API call. As can be seen, the _id is not provided, as it is assigned automatically. Then Mongoose's save function is called on the canvas. Again, the error is 500. Then the result, or in this case the canvas, is pushed to the canvases array that the user model has so that the reference is updated. The user is then saved to persist the change. Lastly, http 201 is returned for success.

In this API path (Figure 12), it is expected for Auth0 to send a request. Auth0 sends a base64 encoded request to the server. This is then decoded and stringified JSON object is parsed back into a JSON object. It is compared that it contains a secret that has been provided in the auth0 rule. If not, the server returns http 403 for unauthorized. It should be noted that this is in no way secure. It only prevents very simple attacks and should not be used when handling sensitive data as base64 encoding can be decoded by anyone. If the secret matches, a new user model is created and saved to the database. Notice that nothing is provided for the canvases field, as it is going to be empty at first.

```
router.get('/', function(req, res){
    var b64string = req.query.data;
    var buf = new Buffer(b64string, 'base64');
    var call = JSON.parse(buf.toString());
    if(call.properties.secret === "****************"){
        var user = new User({
            authUserId: call.properties.distinct_id,
            email: call.properties.distinct_email
        });
        user.save(function(err, result){
            if(err){
            return res.status(500).json({
                title: 'An error occured',
                error: err
            });
            };
            res.status(201).json({
            message: 'Saved user data',
            obj: result
            });
        });
    }else{
        res.status(403).json({
            title: 'UNAUTHORIZED',
            error: err
        });
    }
});
```

Figure 12 API Path for the Auth0 call

6.6 Auth0

Using auth0 to authenticate users is fairly simple. The auth0 “lock” widget is downloaded from their server and is used like in the code below (Figure 13).

```
private options = {
  loginAfterSignUp: false,
  auth: {
    redirect: false
  },
  languageDictionary: {
    success: {
      signUp: 'Thanks for signing up. Please confirm your email before signing in.'
    }
  }
};

lock = new Auth0Lock(
  '**********************',
  '**********************',
  this.options
)

constructor(private router: Router) {

}

login() {
  this.lock.show();
  this.lock.on("authenticated", authResult => {
    this.lock.getUserInfo(authResult.accessToken, function (error, profile){
      if(error){
        return;
      }
      localStorage.setItem('id_token', authResult.idToken);
      localStorage.setItem('accessToken', authResult.accessToken);
      localStorage.setItem('profile', JSON.stringify(profile));
      localStorage.setItem('email', profile.email);
    });
  });
}
```

Figure 13 How to use Auth0 lock widget

First some options are provided for the lock API to use. LoginAfterSignUp is set false since the user is required to confirm their email before they can sign in. It is also not necessary to redirect the user anywhere after successful login, as this might cause users to lose data if they are logging in after they have made changes to a canvas. Then a custom message is provided for signing up so that the user knows what they should do next. After this, we initialize the canvas with the options and the admin’s auth0 profile’s API key. When a user clicks the log in button, the lock widget is

shown. If the user authenticates successfully, their data is set into local storage. This data is used to authenticate the user on requests, to check if user is logged in, and to log the user out.

In Figure 14, is the API call from auth0 to the weave design server. This is an auth0 rule, which is called every time a user tries to log in. If the user has verified their email, but a custom variable isInDb is not true, we set the API URL to point to the server's backend. The request data will be base64 encoded in preparation for the transmission. Then an asynchronous request is made to the API and when it returns, the auth0 rule checks the http status code. If the status code is 203, which means success, the custom variable isInDb is changed to true, and this function will not be activated again for the user.

```
if(user.email_verified){
  if(!user.app_metadata.isInDb){
    var api_url = "https://weave-design.herokuapp.com/api/adduser";
    var myEvent = {
      "event": "Sign up",
      "properties": {
        "distinct_id": user.user_id,
        "distinct_email": user.email,
        "secret": "**************"
      }
    };
    var base64Event = new Buffer(JSON.stringify(myEvent)).toString('base64');
    request.get({
      url: api_url,
      qs: {
        data: base64Event
      }
    }, function(e, r, b){
      if(r.statusCode === 201){
        user.app_metadata.isInDb = true;
        auth0.users.updateAppMetadata(user.user_id, user.app_metadata)
          .then(function(){
            callback(null, user, context);
          })
          .catch(function(err){
            callback(err);
          });
      }else{
        return callback(new UnauthorizedError('There was an error logging in, please try
      }
      return callback(null, user, context);
    });
  }else {
    return callback(null, user, context);
  }
}
return callback(null, user, context);
```

Figure 14 Auth0 rule configuration

6.7 Frontend

The largest part of this application was the frontend and it took the most time. Angular applications are built around modules that orchestrate the whole application. Usually an Angular app has one main module and many sub modules. These modules are there to introduce the dependencies of the applications components to each other.

In Figure 15, the application's main module is presented. It is a collection of components, services, dependencies and etc. The code is first imported and then it will be placed under the correct "title". Other modules are imported, components are declared and services are provided. This is very important to Angular and placing any of these in the wrong place will cause an error.

```
import { AuthGuard } from './services/auth.guard';
import { B64ToBlobService } from './services/b64-to-blob.service';
import { SharedDataService } from './services/shareddata.service';
import { CompressService } from './services/compress.service';

@NgModule({
  imports: [
    BrowserModule,
    FormsModule,
    AppRoutingModule,
    HttpModule,
    CanvasDrawingModule
  ],
  declarations: [
    AppComponent,
    NotFoundComponent,
    NavbarComponent,
    HomeComponent,
    DesignComponent,
    HelpComponent,
    PatternMakerComponent,
    CallbackComponent,
    UserComponent,
    CanvasListComponent
  ],
  providers: [
    MyGlobalsService,
    AuthService,
    CanvasService,
```

Figure 15 Main module of the application

The main part of the application also has its own component, and possibly, html and css templates. If the app has routing, which it probably does, a very important part is located in the html section (Figure 16). Router outlet specifies where the angular router module will render the component that the user has selected through the URL.

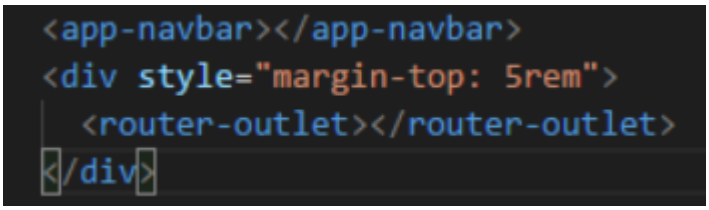

Figure 16 Main html

In Figure 17, is the main routing module of the application. Here all of the paths available to the user are declared. The paths represent a part of the URL, for example, path: 'help' tells Angular to look out for an URL containing help after the first slash, like this, example.com/help. If a path match found, angular renders the component declared after the path inside the router outlet seen in Figure 16. Angular also provides numerous other functions here, such as; canDeactivate and canActivate. Depending on the conditions set on them, users will not be able to leave the specific URL if it has canDeactivate, or will not be able to access the specific URL if it has canActivate. These can be very handy when designing a seamless user experience.

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';

import { NotFoundComponent } from './not-found/not-found.component';
import { HelpComponent } from './help/help.component';
import { HomeComponent } from './home/home.component';
import { DesignComponent } from './design/design.component';
import { PatternMakerComponent } from './pattern-maker/pattern-maker.component';
import { UserComponent } from './user/user.component';
import { AuthGuard } from './services/auth.guard';

const appRoutes: Routes = [
  { path: '', component: HomeComponent},
  { path: 'help', component: HelpComponent},
  { path: 'design', component: DesignComponent, canDeactivate: [AuthGuard]},
  { path: 'maker', component: PatternMakerComponent, canDeactivate: [AuthGuard]},
  { path: 'user', component: UserComponent, canActivate: [AuthGuard] },
  { path: '**', component: NotFoundComponent }
];

@NgModule({
  imports: [
    RouterModule.forRoot(
      appRoutes
    )
  ],
  exports: [
    RouterModule
  ]
})
export class AppRoutingModule { }
```

Figure 17 Main routing module

An Angular application is nothing without components, which ought to be easily reusable through the application. One example of a component is the user-list component (Figure 18). This component belongs to the user component, where it will be rendered. Dependencies are imported at the top of the code and after that, a decorator is used to determine the type of the following code. Selector tells Angular that everywhere where this specific selector is provided and found in html, it should render the contents of the html found from the URL in templateURL at the place of the selector. After this, the class that is going to be exported is defined. This is a typical place for the application logic to be written in. Everything declared public here, can be used from the html file.

```
import { SharedDataService } from '../services/shareddata.service';
import { CompressService } from '../services/compress.service';

@Component({
  selector: 'app-canvas-list',
  templateUrl: './canvas-list.component.html',
})
export class CanvasListComponent implements OnInit {

  public canvases: CanvasModel[] = [];
  public canvas: CanvasModel;
  public noData: boolean = false;

  constructor(
    private canvasService: CanvasService,
    private shareddata: SharedDataService,
    private compressService: CompressService
  ) {}

  public remove(canvas: CanvasModel)
  {
    if(confirm("Are you sure you want to delete this canvas?")){

      this.canvasService.removeCanvas(canvas)
        .subscribe(
            result => alert("Canvas deleted."),
            error => alert("Something went wrong!")
        );
    }
  }
```

Figure 18 User list component

```
<app-canvas-list></app-canvas-list>
```

User html where everything from Figure 18 will be rendered

Figure 19 is what is going to be renderer at where the selector was found. Everything that has *ng written before it inside the html, is an Angular function. For example, if *ngIf is given a false value, it will not render its child components at all. They will not be rendered and thus, cannot be accessed by the user, or even the code itself. It will not exist. This specific ngIf tests against a variable called noData, which can also be seen declared public in Figure 18. Thanks to the fact that noData is declared public, Angular can easily check to see if it is truthy every time a change happens. Functions

that have been declared can also be used from the html as is the case in all of the click events. When a user clicks on these elements, the function inside the parentheses will be called and its function is then executed.

Angular also provides something called a directive. Directives are used to manipulate html elements without directly accessing them. Angular provides many different ways of accessing an element and editing it without doing it directly.

```
<div class="ml-3">
  <h3>Canvas list</h3>
  <ul *ngIf="!noData">
    <li class="mt-2" *ngFor="let canvas of canvases; let i = index">
      <span>{{i}}.  {{canvas.canvasName}}</span>
      <span>Edit in: </span>
      <a class="btn btn-secondary btn-sm" (click)="takeCanvasToDesign(canvas, true)" [routerLink]=" ['/design']" rout
        Weave design
      </a>
      <a class="btn btn-secondary btn-sm ml-2" (click)="takeCanvasToDesign(canvas, false)" [routerLink]=" ['/maker']"
        Pattern maker
      </a>
      <button class="btn btn-danger btn-sm ml-3" (click)=remove(canvas)>remove</button>
    </li>
  </ul>
  <h4 *ngIf="noData">No canvas data found. What are you waiting for? Go and make something! :)</h4>
</div>
```

Figure 19 Canvas list html

Figure 20 illustrates a typical composition of a directive. Everything else is same as in a component, but the decorator says @Directive instead of @Component. Directives also do not have a template, as they are only used to enhance a component. This particular directive uses the renderer2 service provided by Angular, which can access an html element and add attributes to it. In the application, we want only certain canvases to be clickable depending on which side the user is on. Angular also provides various lifecycle hooks that can be utilized on a page load. Here ngAfterViewInit is used to prevent the directive from activating before all of the elements have actually been rendered and are accessible to the renderer service.

```
import { OnInit, Directive, Input, ElementRef, Renderer2 } from '@angular/core';

import { CanvasDrawingComponent } from './canvas-drawing.component';

@Directive({
    selector: '[canvasDrawing]'
})
export class CanvasDrawingDirective implements OnInit {
  @Input() isDesign: boolean;
  @Input() eventListener: any;

  constructor(private renderer2: Renderer2, private el: ElementRef){}

  ngOnInit() {}

  ngAfterViewInit(){
    if (this.isDesign){
      this.renderer2.listen(this.el.nativeElement, 'click', this.eventListener)
    }
  }
}
```

Figure 20 Example listener that adds a click event

In Figure 21 one can observe that the selector in Figure 20 is written inside an html element. The directive is then attached to this particular element and can manipulate its properties. The [eventListener] tag passes the value inside the parentheses and, because it is declared as an Input in the directive, the directive is provided with the function that it should bind to the click event. Now, whenever a user clicks on the canvas, a click event will be fired.

```
width="0" height="0" canvasDrawing [isDesign]="!isDesign" [eventListener]="horCanvasListener.bind(this)">
```

Figure 21 Using the directive

Last Angular building block that will be explained is a service. Services are used in Angular to do simple, highly reusable actions that can be injected anywhere in the application. The point is to avoid duplicate code by separating the functions into small pieces and also to separate the business logic from the components. If the application had many places where the canvas should be saved to png, it would be possible to write a service that does exactly that.

Figure 22 shows the typical structure of a service. The most important thing to notice here is that there is no selector or, in fact, anything before the decorator and the class. The decorator is @Injectable, which means that this piece of code can be injected and used from anywhere in the application, as long as it is provided in the

module, imported in the component and constructed there. Then it can be used simply like in Figure 23.

```
import { Injectable} from '@angular/core';

import { B64ToBlobService } from './b64-to-blob.service';

@Injectable()
export class SaveToPngService {
  constructor(private b64ToBlob: B64ToBlobService) {}

  public saveToPng(canvasArray: HTMLCanvasElement[], tempCanvas: HTMLCanvasElement, size: number, padding: number,
    heddles: number, lines: number, verMax: number, horMax: number){

    let ctx = tempCanvas.getContext("2d");
    ctx.canvas.width = size * heddles + padding * 2 + size * verMax + size;
    ctx.canvas.height = size * lines + padding * 2 + size * horMax + size;

    ctx.drawImage(canvasArray[0], 0, 0)
    ctx.drawImage(canvasArray[1], 0, size + padding)
    ctx.drawImage(canvasArray[2], heddles * size + padding, size + padding)
    ctx.drawImage(canvasArray[3], 0, size * horMax + size + padding * 2)
    ctx.drawImage(canvasArray[4], heddles * size + padding, size * horMax + size + padding * 2)
    ctx.drawImage(canvasArray[5], heddles * size + padding * 2 + size * verMax, size * horMax + size + padding * 2)
    var base64 = tempCanvas.toDataURL();
    let blob = this.b64ToBlob.ToBlob(base64);
    return URL.createObjectURL(blob);
  }

}
```

Figure 22 Saving canvas to png service

```
public saveToPng(){
  let tempCanvas = this.renderer2.createElement('canvas');
  let href = this.saveToPngService.saveToPng(this.canvasArray, tempCanvas, this.rectSize,
  let link = <HTMLLinkElement>document.getElementById('btn-download');
  link.href = href;
}
```

Figure 23 Using a service inside a component

The saveToCanvasService (Figure 23) is declared inside the constructor of the component and can be accessed here. First, a temporary canvas, from which the png is exported from, will be created. This canvas won't be visible in any way to the user. Then this canvas is passed to the saveToPng function alongside with other required parameters. This function returns a string containing a blob of the canvas, which can then be downloaded from the link.

Central component

The central component to the whole application, the canvas-drawing component is too large to be showcased properly here, however, some functionalities will be looked at.

Figure 24 illustrates the make design functionality. This function takes the main canvas and constructs the sides from its data. First, the state of the main canvas

array will be checked and if it is empty, an error is returned. Otherwise, the process is started by creating unique arrays of both sides.

```
public makeDesign(){
  if(!this.isMainCanvasEmpty()){
    let horUniqueArray = this.fillHorUniqueArray();
    let verUniqueArray = this.fillVerUniqueArray();
    let temp = this.canvasDrawingService.makeDesignArray(horUniqueArray);
    let horArray = temp.array;
    this.horMax = temp.count;
    temp = this.canvasDrawingService.makeDesignArray(verUniqueArray);
    let verArray = temp.array;
    this.verMax = temp.count;

    this.verCanvasArray = this.canvasDrawingService.prepare2DArray(Array(), this.height, this.verMax);
    this.horCanvasArray = this.canvasDrawingService.prepare2DArray(Array(), this.horMax, this.width);
    this.resultCanvasArray = this.canvasDrawingService.prepare2DArray(Array(), this.horMax, this.verMax);

    this.verCanvasArray = this.compressService.decompressVerCanvasArray(verArray, this.verCanvasArray);
    this.horCanvasArray = this.compressService.decompressHorCanvasArray(horArray, this.horCanvasArray);
    this.resultCanvasArray = this.compressService.decompressResultCanvasArray(horArray, verArray, this.mainCanv

    this.drawHorCanvas();
    this.drawVerCanvas();
    this.drawResultCanvas();
  }
  else
  {
    alert("Cannot make design when canvas is empty.");
  }
}
```

Figure 24 Make a design function

Figure 25 shows the function that makes a horizontal unique array. What this function does, is return an array containing the numerical value of all the rows in main canvas. This numerical value is calculated with the help of binaries. Due to the fact that the main canvas array only contains zeroes and ones, it is easy to give each row its own numerical value. Every time a one is found; the iteration is taken to the power of two and added to the array. This is repeated until all of the horizontal rows are exhausted.

```
private fillHorUniqueArray(): number[] {
  let horUniqueArray = this.canvasDrawingService.prepareArray(Array(this.width), this.width);
  for(let i = 0; i < this.height; i++){
    for(let j = 0; j < this.width; j++){
      if(this.mainCanvasArray[i][j]){
        horUniqueArray[j] += (i+1) ** 2;
      }
    }
  }
  return horUniqueArray;
}
```

Figure 25 Function that fills the array with a numeric representation of horizontal array

After the unique arrays have been made, both of them are passed to the function in Figure 26 and from the unique array, the corresponding side array is formed. This function works by reducing its array's size until it is zero and if it finds a value that has already been added to the array, it is given the same value as the one added before. The resulting array will contain all the data that is needed to construct the two-dimensional array. Then some arrays are initialized and finally the two-dimensional arrays are constructed.

```
public makeDesignArray(uniqueArray: any[]): any{
  let array = Array(uniqueArray.length);
  return uniqueArray.reduce(function(values, v, i) {
    if(v !== 0){
      if(values.set[v]){
        let len = Object.keys(values.set).length
        for(let j = 0; j <= len; j++){
          if(values.set[v] === j){
            values.array[i] = values.set[v]
          }
        }
      }
      else if (!values.set[v]) {
        values.count++;
        values.array[i] = values.count;
        values.set[v] = values.count;
      }
    }else{
      values.array[i] = 0;
    }
    return values;
  }, { set: {}, count: 0, array});
}
```

Figure 26 Make a design array from unique value array

The array created in Figure 26 is passed to a decompress function (Figure 27) that parses the one-dimensional array into a two dimensional one. The reason this function is named decompress, is because it is also used when getting the compressed data from the database. This function takes the position of the canvas

array's index and uses that to determine where it should place a click horizontally. If that index contains a value other than zero, that value is then used to determine where the box will be located vertically. This will continue until the array is exhausted. Lastly, the canvas has to be refreshed and redrawn to complete the process.

```
public decompressHorCanvasArray(horArray: number[], horCanvasArray: number[], r
  let width = horArray.length;
  let horMax = 0;
  for(let i = 0; i < width; i++){
    if(horArray[i] > 0){
      horCanvasArray[horArray[i]-1][i] = 1;
      if (horArray[i] > horMax){
        horMax = horArray[i];
      }
    }
  }
  return horCanvasArray;
}
```

Figure 27 Function that makes a two-dimensional array

In Figure 28 the function first gets the context of the canvas from an object where they are stored and then the context is updated with the updated width and height of the new canvas. Lastly, the rectangles are drawn on the updated canvas.

```
private drawHorCanvas(){
  let ctx = this.ctxObject.horCanvas;
  ctx = this.setCanvasWidthAndHeight(ctx, this.width * this.rectSize, this.horMax * this.rectSize);
  this.drawRects(this.width, this.horMax, ctx, this.defaultWhite, this.horCanvasArray)
}
```

Figure 28 Function that draws the horizontal canvas

The function in Figure 29 draws the rectangles on the canvas and takes into consideration all of the rectangles that might have already been clicked. The draw functions provided by html5 canvas are powerful, but can be very difficult to use for more complex drawings. For this application, html5 canvas was the perfect match, as everything is drawn in rectangles.

```
}

private drawRects(width: number, height: number, ctx: CanvasRenderingContext2D,
  rectColor: string = this.defaultWhite, array: any[] = null,  secondaryColor: string = this.defaultGray){

  this.refreshCtx(ctx)
  for (let a = 0; a < height; a++) {
    for (let b = 0; b < width; b++) {
      if(array === null || !array[a][b]){
        this.drawRect(b * this.rectSize, a * this.rectSize, ctx, rectColor);
      }
      else if(array !== null && array[a][b]){
        this.drawRect(b * this.rectSize, a * this.rectSize, ctx, secondaryColor);
      }
    }
  }
}
```

Figure 29 Function that draws rectangles on canvases

Html 5 canvas vs canvas libraries

In this chapter, it will be explained why Html 5 canvas was chosen for this project, instead of any third-party canvas library. In preparation of the project, many different drawing libraries were tested out to see which one would do the job the best. Two of them will be highlighted here, D3, Snap.svg and Konva. D3 is a drawing library that specializes in graphs. Initially, D3 seemed like a good fit for what was needed, as their website showcased an extremely well performing animation with a lot going on. The tests were quickly concluded in the case of D3, because it was too complex for this application and it did not work well with the design philosophies of Angular, which is to avoid direct html element manipulation as much as possible. In its defense, it must be said that it did perform quite well, but not well enough to trump the negatives. Snap.svg is a drawing library for SVG elements. SVG elements are like canvas elements, they can be drawn on and they have the added benefit of being completely scalable, as the drawing is done in vectors. Snap.svg promised to be fast and small and it did perform very well compared to the other two. It was easy to use, but ultimately was not performant enough, as its performance suffered from too much functionalities that did not benefit this project. Canvases of the size of 1200 pixels times 1200 pixels already started seeing some significant slowdowns. The last library was Konva. It is an html5 canvas based drawing library that is incredibly easy to use. It offers a lot of functionalities out of the box and can be customized quite easily. For the tests, Konva was optimized the most, but even after all the optimization, it only managed to draw a canvas with a size of 1100 pixels times 1100 pixels. Its biggest downfall was the fact that for every element that it created, it also

added an event listener to it. This meant that increasing the size of the canvas increased the memory cost of the canvas too much causing it to crash.

In conclusion, all of the libraries tested added too many functionalities to warrant the greatly reduced performance when compared to plain Html5 canvas. For reference, the canvas element is easily able to handle canvases with a size of 10000 pixels times 10000 pixels. However, it should be noted that when drawing more complex shapes and doing animations for games etc., these functions would be necessary and help greatly reduce the complexity of drawing.

7 Testing

7.1 In general

Testing an application is a very important part of the development process. In its most basic form, testing can be manual testing, where a person tries the application and tries to find bugs or inconsistencies in the logic. This is very good way of testing the looks and basic functions of the app, however, in the long run this can be very inefficient. Manual tester is not going to be able to try every combination, every page, every component etc. on their own. It gets especially hard when all of these things should be tested every time the application is updated. For this purpose, there are many kinds of test automation frameworks out there. Roughly, testing can be divided into three main categories, unit testing, integration testing, and end to end testing.

7.2 Unit testing

Unit tests are usually written for very specific small pieces of the software. They could simply be testing one function that increments a value by one. Unit tests are usually written programmatically at the same time as the application is being developed. This ensures that the programmer understands what they are trying to do and forces them to think about the testability of code before writing. Usually, if a test is too hard to write, the functionality that is being tested is too convoluted and

should be refactored. Often unit tests are written even before the functions. This is called test driven development. When the programmer writes a test before they write the function, they have a very clear understanding of what needs to be done, and what they want the function to do. Often this also increases the quality of the code, as easily testable functions are well written (McFarlin 2012).

7.3 Integration testing

Integration tests are written to ensure that all the code that different people write will work well together. These tests are usually run by a build server before it accepts the changes that the programmer is trying to push into the working tree. This is called continuous integration. Continuous integration means that the code is always tested before changes are accepted and errors reported to the programmer if their change broke something else in the application that they did not notice.

7.4 End to end testing

End-to-end testing refers to tests that try to simulate how a user will see the application. Usually end-to-end tests are run on the browser and the test runners (see the framework section below) click around the webpage in a “natural manner”. These tests provide the most confidence that the application is working correctly, but they are also the most arduous to write and even slight changes to the code, might break the test. This in turn increases the workload of programmers as they have to update these tests often. Due to this fact, end-to-end tests are only used in very critical parts of the application and there are only few of them (Lakshay 2017).

7.5 Framework

There are several different test frameworks, and all of the main categories might have their own ones. For example, Ranorex is an automation framework mostly designed for end-to-end tests. Ranorex can record the actions of the programmer, and it then runs these tests to see if everything occurs in a similar manner again to the user’s actions. Other frameworks might try to do everything, such as Jasmine for

JavaScript. These tests usually also need something called a “test runner”. This is the program that runs all the tests. For example, Jasmine comes with a test runner framework called Karma.

7.6 Testing weave design app

For this app, all testing was done manually, as the complexity of the application did not warrant test automation immediately. Granted, testing is useful no matter the size of the application and should be done. In this case, the size of the thesis would have become too big. Simple tests could have been written with Jasmine, however, making redundant tests for the sake of writing tests is worth nothing, and if done properly, it would have quickly become very resource intensive and time consuming. Thus, in the scope of this thesis, test automation was dropped.

Due to time constraints, the application was only tested to be working on the latest version of Google Chrome, however, most modern browsers should offer similar performance and functionality.

8 Discussion and results

8.1 Final product

The concrete result of this thesis was a working web application that can be used to create and design weaving patterns. All of the requirements for this application were met, except for one; currently users are not able to change their password through the application, however, it is possible to change them through Auth0 control panel upon request. Beside the requirements set in the requirement specification, the following functionalities were added: Users can see the reverse side of the canvas they are designing with a click of a button, colors used are saved temporarily on the color picker menu, it is possible to fill the entire color row with one click of a button, users can change the size of the shaft on the fly without losing their progress, users can add and delete rows without losing their progress, and users can take their

canvas that they have been working on to the other part of the application without losing their progress.

When counting the lines of code, we can see that it is relatively meager.

```
C:¥coding¥heroku¥weave-design>git diff --shortstat 4b825dc642cb6eb9a060e54bf8d69288fbee4904
 91 files changed, 4054 insertions(+)
```

The amount of line breaks in the codebase

Only 4000 lines of code in total, written by the author, if this method of counting can be trusted. In part, this can be explained by the fact, that a lot of time was spent on making the code modular so that it could be reused when needed, and writing short concise functions that could also be reused. For example, the original project code written during the second year of their studies by the writer of this thesis used roughly 240 lines of code to construct the sides of the main canvas from the clicks that the user had made. The revamped code made for this thesis project achieves the same result with only 22 lines of code. The topic of this thesis was also quite unfamiliar, as the writer had never actually done any weaving and the earlier project did not provide a lot of help as it never worked correctly. Thus, a large portion of the time allocated for this thesis was spent on understanding the intricacies of weaving, which left less time for coding. Lastly, some npm packages were utilized to provide functionalities and they were not taken into consideration in the total amount of line breaks, as they were not written by the writer of this thesis.

8.2 User testing

It is very important to have relevant users test the application before it is released. The users can provide invaluable insights into the design of the program that the programmer could not have thought of, because it is very easy to be blinded by their produce.

As the target group for this application was very specific, it was hard to find enough users inside the target group for testing the application properly. Due to this, the application was only tested by one person who had any experience with weaving; the writer's sister. She was able to provide some insight to how she wanted the application to work, but it was not enough to be able to confidently say that the

application works very well. In addition, many people outside of the target group tested the application, but, as was expected, they were not able to provide valuable data in regard to the logic of the application, however, considerable amount of opinions were shared about the look and feel of the website and this helped shape the end result.

With these tests, it is possible to conclude that the application is very hard to approach without prior knowledge in weaving, however, this is not a problem as the application does not provide considerable amount of value to users outside of the target group.

8.3 Problems and future development

The lack of relevant user tests leads to the fact that plenty of possible bugs and problems have gone unseen. Even if this were the case, it is desirable that the application would be perfect, it is necessary to understand and admit that no application is perfect. Many problems were already fixed during the development as the writer tested the functionalities often by hand. There are bound to be more problems and bugs if and when more users start using the application, and these have to be addressed at that time.

When developing a program, it is important to set boundaries to what will be done now and what will be (possibly) done in the future. Otherwise the project might easily become an eternity project that will never be finished. This means that there will always be improvements that could be made. While developing and doing user tests, many new ideas and places of improvement were found. One of these came up while a user was testing the application. They expressed a need to use same colors between sessions, as at the moment of writing, when saving a canvas to the database, the temporary colors in the color picker menus are not saved. It was proposed that there could be a color picker tool that would allow the user to pick existing colors from the canvas and continue using them. The writer, when making a new canvas found another need. With the current limits in the application, one user is able to save ten canvases into the database before running into an error. When this error comes up, the user is not able to easily delete an earlier canvas, so that

they would be able to save their newest one. There could be a list from where the application asks which canvas the user wants to delete to be able to save the new one. This can, however, be circumvented by opening a new tab or window and navigating to the website. There the user simply has to go to their user menu and delete one canvas, before returning to the original window where they can now save their canvas. Other possible improvements include tests, CI-chain, making the application more secure, localization, saving the canvas data periodically to prevent loss of data, etc.

8.4 In conclusion

The whole thesis project was a very interesting and educational experience. The time spent on coding was about 2.5 months of hard work, as balancing between work, wellbeing and thesis proved to be quite difficult from time to time. The pressure from all of these three factors, however, was a great driving force, because it helped staying in focus and increased the willingness to get it done on time.

The most difficult things while developing was the constant need to learn new concepts, frameworks and coding practices, as most of the used frameworks were completely new to the writer. Spending the time to understand the logic behind weaving also took a considerable amount of time, but having a real-world application for the software was very interesting and rewarding. Everything learned during this process was very enlightening and many of the concepts were immediately applicable to the writer's work as a programmer.

Overall, there are still many functionalities that could be improved or added in the program, and probably will be. Despite this, the thesis does what it was required to do, and was immediately very well received by the writer's sister, who was already thinking of how to best use the program for weaving. Thus, we can conclude that the thesis project was successful.

References

Auth0 Overview. N.d. Auth0 docs. Accessed on 13 August 2017. Retrieved from https://auth0.com/docs/overview

Buckler, C. 2015. SQL vs NoSQL: How to choose. Article about differences between NoSQL and SQL from Sitepoint website. Accessed on 11 April 2017. Retrieved from https://www.sitepoint.com/sql-vs-nosql-choose/

Edelman, G. 2017. How to choose your tech stack. Article about tech stacks from Silicon Valley Software Group website. Accessed on 11 April 2017. Retrieved from https://svsg.co/how-to-choose-your-tech-stack/

Elliot, E. 2016. Top JavaScript Frameworks & Topics to Learn in 2017. Article about JavaScript frameworks from Medium website. Accessed on 11 April 2017. Retrieved from https://medium.com/javascript-scene/top-javascript-frameworks-topics-to-learn-in-2017-700a397b711

Figure 2. Example of a basic weave. Accessed on 24 March 2017. Retrieved from http://1.bp.blogspot.com/-TnGQqXRdOSc/VQ-K0cbYZyI/AAAAAAAAQdk/uIOAdT8WJmk/s1600/4A.jpeg

Figure 4. Overview of IaaS, SaaS and PaaS. Accessed on 13 August 2017. Retrieved from https://azure.microsoft.com/en-us/overview/what-is-paas/

Flexible pricing, for developers and companies. N.d. Documentation about auth0 pricing from Auth0 website. Accessed on 18 September 2017. Retrieved from https://auth0.com/pricing

Gil, P. 2017. What is 'SaaS' (Software as a Service)?. Accessed on 11 April 2017. Retrieved from https://www.lifewire.com/what-is-saas-software-2483600

Kazankow, V. 2016. Software Requirements Specification Helps To Protect IT Projects From Failure. Article about the importance of requirement specification from Bell IT Soft website. Accessed on 13 August 2017. Retrieved from https://belitsoft.com/php-

development-services/software-requirements-specification-helps-protect-it-projects-failure

Lakshay, S. 2017. What does End-to-End Test mean? ToolsQA. Accessed on 18 September 2017. Retrieved from http://toolsqa.com/software-testing/what-does-end-to-end-test-mean/

McFarlin, T. 2012. The Beginner's Guide to Unit Testing: What Is Unit Testing?. Code Tutsplus. Accessed on 18 September 2017. Retrieved from https://code.tutsplus.com/articles/the-beginners-guide-to-unit-testing-what-is-unit-testing--wp-25728

Mittag, J., & Ore4444. 2017. Understanding the differences: traditional interpreter, JIT compiler, JIT interpreter and AOT compiler. Forum post explaining differences between JIT and AOT from Software Engineering Stack Exchange website. Accessed on 18 September 2017. Retrieved from https://softwareengineering.stackexchange.com/questions/246094/understanding-the-differences-traditional-interpreter-jit-compiler-jit-interp

Muscato, C. N.d. What is weaving in textiles? Video lesson transcription from Study website. Accessed on 24 March 2017. Retrieved from http://study.com/academy/lesson/what-is-weaving-in-textiles.html

No license. N.d. Page explaining what no license means from Choose a License website. Accessed on 13 August 2017. Retrieved from https://choosealicense.com/no-license/

OWASP Top 10 Vulnerabilities. N.d. Veracode. Accessed on 13 August 2017. Retrieved from https://www.veracode.com/directory/owasp-top-10

Rouse, M. 2016. RESTful API. Article explaining what RESTful API is from Techtarget website. Accessed on 18 September 2017. Retrieved from http://searchcloudstorage.techtarget.com/definition/RESTful-API

Sharma, H. 2017. What is Azure? – An Introduction To Microsoft Azure Cloud. Edureka. Accessed on 18 September 2017. Retrieved from https://www.edureka.co/blog/what-is-azure/

Skólski, P. 2016. Single-page application vs. multiple-page application. Article about single-page and multiple-page applications from Medium website. Accessed on 18 September 2017. Retrieved from https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58

Sonmez, J. 2016. What Is Back-End Development? Article explaining back-end from Simple Programmer website. Accessed on 18 September 2017. Retrieved from https://simpleprogrammer.com/2016/12/12/what-is-back-end-development/

SQL Injection. N.d. W3Schools. Accessed on 13 August 2017. Retrieved from https://www.w3schools.com/sql/sql_injection.asp

What is a package manager?. N.d. Aptitude Alioth Debian. Accessed on 13 August 2017. Retrieved from https://aptitude.alioth.debian.org/doc/en/pr01s02.html

What is IaaS?. N.d. Interoute. Accessed on 13 August 2017. Retrieved from http://www.interoute.com/what-iaas

What is npm? 2017. NPM Documentation. Accessed on 13 August 2017. Retrieved from https://docs.npmjs.com/getting-started/what-is-npm

What is PaaS?. N.d. Microsoft Azure website. Accessed on 13 August 2017. Retrieved from https://azure.microsoft.com/en-us/overview/what-is-paas/

Wodehouse, C. 2015. Choosing the right software stack. Article about choosing a software stack from Upwork website. Accessed on 11 April 2017. Retrieved from https://www.upwork.com/hiring/development/choosing-the-right-software-stack-for-your-website/

Appendices

Appendix 1. Screenshots from the web application