UNIVERSITY OF TARTU
FACULTY OF SOCIAL SCIENCES
NARVA COLLEGE
INFORMATION TECHNOLOGY SYSTEMS DEVELOPMENT

Dmitri Petšurov

DEVELOPING A WEB APPLICATION FOR CREATING AND PUBLISHING A PERSONAL PORTFOLIO

Bachelor's thesis

Supervisor: Andre Sääsk, M.Sc.

Narva 2022

Tööd tehti iseseisvalt. Töö koostamisel kogu kasutatud materjal, dokumentatsioon, aga ka teiste autorite tööd on viidatud

.....

Töö autori allkiri ja kuupäev

Non-exclusive license to reproduce thesis

I, Dmitri Petšurov (date of birth: 21.01.1997),

1. Here with grant the University of Tartu a free permit (non-exclusive license) to reproduce, for preservation purposes, including for adding to DSpace archives before the copyright expires.

“Developing a web application for creating and publishing a personal portfolio” supervised by Andre Sääsk M.Sc.

2. I am aware that the author reserves the right under paragraph 1.

3. This is to certify that granting the non-exclusive license does not infringe the intellectual property rights or rights arising from the Personal Data Act.

Narva, 06.04.2022

TABLE OF CONTENTS

TERMS AND ABBREVIATIONS

TAG – A tag is some piece of information that describes some data or content to which it is assigned.

PAGINATION – This is the process of dividing a document into separate pages.

OS – An operating system (OS) is system software that manages computer hardware, software resources, and provides common services for computer programs (Wikipedia. *Operating system*. 2022).

VS Code – Visual Studio Code is a code editor (Wikipedia. *Visual Studio Code*. 2022).

JSX – JSX stands for JavaScript XML. JSX allows us to write HTML in React. JSX makes it easier to write and add HTML in React. (w3schools. *What is JSX?* 2022)

CSS – Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language such as HTML. CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript. (Wikipedia. *CSS.* 2022)

SCSS – SCSS is a preprocessor which lets you use features that aren't a part of the wider CSS standard yet, and provides better workflows for maintaining your stylesheets (dailysmarty. *What is scss.* 2022).

DOM – The Document Object Model (DOM) is a cross-platform and language-independent interface that treats an XML or HTML document as a tree structure wherein each node is an object representing a part of the document (Wikipedia. *Document Object Model*. 2022).

JSON – JavaScript Object Notation (JSON) is an open standard file format and data interchange format that uses human-readable text to store and transmit data objects consisting of attribute–value pairs and arrays (Wikipedia. *JSON*. 2022).

UI – In the industrial design field of human–computer interaction, a user interface (UI) is the space where interactions between humans and machines occur (Wikipedia. *User interface.* 2022).

HTTP – The Hypertext Transfer Protocol (HTTP) is an application layer protocol in the Internet protocol suite model for distributed, collaborative, hypermedia information systems (Wikipedia. *Hypertext Transfer Protocol.* 2022).

MERN – The technology stack is a set of frameworks and tools used to develop a software product. The MERN stack consists of the following technologies: MongoDB, Express.js, React.js and Node.js. (educative. *What is MERN Stack?* 2022).

API – An application programming interface (API) is a connection between computers or between computer programs (Wikipedia. *API.* 2022).

USENIX – USENIX is an American 501 nonprofit membership organization based in Berkeley, California and founded in 1975 that supports advanced computing systems and operating system research (Wikipedia. *USENIX.* 2022).

INTRODUCTION

The problem

Today, the IT industry is developing very rapidly and becoming more and more popular. There are a lot of candidates for any position and you need to somehow differ from them. A possible difference may be the presence of a portfolio in your resume. After analyzing the market, the author identified competitors that already provide an opportunity to create a portfolio – one of the competitors is LinkedIn. Although there are already portfolio tools on the market, but all these tools are very difficult to use and have a lot of unnecessary additional features, which is the main problem that the author aims to solve.

The solution

Given the above problem, the author is interested in developing a web platform that will help all its users to create their portfolio and publish it on the same platform. The created portfolio will be published on the main page of the platform, and each user of the platform will be able to view any published portfolio using a quick search by title or by tags. Also, the user can share the link with friends or a potential employer. The author is interested in simplifying the process of creating and publishing a portfolio for platform users as much as possible. Having this type of application will have value regardless of the number of users.

The goal

The author's goal is to develop a web platform where you can create and publish a portfolio easily and without any problems. Using this type of platform, the user's portfolio can be seen by potential employers, and the user has the option to add a link to this portfolio to their resume to give the employer a full understanding of the candidate's skills, abilities, and projects.

The motivation

The author wanted to add a portfolio to his resume, but could not find a simple and easy-to-use platform. Therefore, the author is motivated to create this type of platform that will be useful both for the author himself and for other users of the platform.

Tasks

To achieve the goal of this thesis, the author must perform the following tasks:

- Market research to find out competitors and determine the niche of application development

- Development of technical specifications
- Select appropriate tools and technologies for web platform development
- Creation of a site design layout
- Web application development
- Test and debug the application

Outline

The thesis consists of three chapters.

First of all, the author begins with an introduction. In the introduction, the author describes the existing problem, then describes his solution to this problem. After that, the author describes the goal that he must achieve, also describes his motivation for the goal, and finally sets the tasks that he must complete in order to achieve the goal.

The first chapter consists of a description of the selected technologies and tools for developing a web application, the chapter also describes existing solutions that are already on the market and the main differences between the author's application and existing solutions.

Second chapter describes the entire development process, namely includes:

- Functional and not functional requirements
- Use cases and dataflow diagrams
- Application mockup

In the third part, the author describes the functionality of the application, the database architecture, as well as future work on the application.

1 USED TECHNOLOGIES AND EXISTING SOLUTIONS TO THE PROBLEM

1.1 Existing solutions

The author has done market research to find existing tools that solve a similar problem. As a result, the author identified 5 main competitors that partially solve the problem described earlier.

1.1.1 LinkedIn

LinkedIn is a social network focused on professional communication and career development, also supports the creation of your personal page by filling out forms with questions.

The author app has the same idea as LinkedIn, but the main difference is the narrow focus of the author app, which focuses on creating a personal portfolio with the ability to choose the design and publishing it on the platform.

1.1.2 Portfoliobox

Portfoliobox is a website for creating an online portfolio. This platform allows you to create an online portfolio using pre-made templates, fonts and images.

The author's application is very similar to the idea of Portfoliobox, however, the author tries to simplify and speed up the process of creating a portfolio as much as possible, while the process of creating a portfolio on the Portfoliobox platform takes a long time and involves a large number of steps. Portfoliobox also does not have a free limited version for creating a basic portfolio, all services of this platform are paid.

1.1.3 Readymag

Readymag platform allows you to create not only a portfolio, but also websites of other directions. The process of creating a portfolio is also quite dreary and takes a certain amount of time.

This platform allows you to create a portfolio absolutely free using all the functions of the application, however, as soon as the user decides to share this portfolio with someone or add it to his resume, he will have to use the paid domain connection service from Readymag. Also, to create and publish more than one portfolio, the user must use a paid subscription.

The difference between the author’s platform and Readymag is again the simplicity and speed of creating a portfolio.

1.1.4 Wix

Wix provides the ability to create websites using customizable website templates and a website builder.

It's a beautifully designed website building platform with tons of extras, but the goals of the Wix platform and the author's platform are slightly different. The Wix platform is a constructor for creating websites, which means that as a result, the user receives a ready-made website that will no longer be related to the Wix platform. The main difference between the author platform and the Wix platform is that the author platform serves as a kind of container for creating, storing and displaying unique portfolios, which in turn can be evaluated by other users.

1.1.5 Squarespace

Squarespace is a website building platform and a direct competitor to the Wix platform, so the functionality of the platform is similar, except for the difference in design. The difference between the author platform and the Squarespace platform is the same as the Wix platform.

1.2 Technologies

After already existing applications have been described that theoretically solve the problem described in the thesis, it is possible to describe the topic regarding the technologies used to develop the author's platform.

1.2.1 Visual Studio Code

The author uses this editor because he considers it the most convenient in terms of design, as well as in terms of adding and using various extensions, which in turn significantly increase the author's productivity in the development process. Also, this editor has good compatibility with a programming language such as TypeScript, which will be used in the further development of the author's platform.

1.2.2 JavaScript

The author uses this programming language, as it is the most popular programming language for creating web applications, and JavaScript is also suitable for both frontend and backend

development. Also, there are practically no other alternatives in the form of programming languages for working with React.js and Node.js.

1.2.3 Prettier

Prettier is an opinionated code formatter with support for: JavaScript, JSX, Angular, Vue, Flow, TypeScript, CSS, Less, and SCSS, HTML, Ember/Handlebars, JSON, GraphQL, Markdown (including GFM and MDX), YAML.

It removes all original styling* and ensures that all outputted code conforms to a consistent style. (Prettier. *What is Prettier?* 2022)

One of the most popular and easy to use code formatters, the author uses it to maintain the same style of code throughout the project.

1.2.4 React.js

React is one of the most popular front-end frameworks, and the MERN technical stack was chosen to develop the application, which involves the use of React, so the author chose this framework. The author chose this framework because it greatly simplifies the process of developing the user interface of a web application by working with UI components.

1.2.5 React-router-dom

React Router DOM is an npm package that enables you to implement dynamic routing in a web app. It allows you to display pages and allow users to navigate them. It is a fully-featured client and server-side routing library for React. (Geeksforgeeks. *What is react-router-dom?* 2022)

This npm package was chosen by the author for DOM manipulations, as it has direct compatibility with the selected React framework.

1.2.6 Redux

Redux is an open-source JavaScript library for managing and centralizing application state. It is most commonly used with libraries such as React or Angular for building user interfaces. (Wikipedia. *Redux.* 2022)

The author chose this library for more convenient development and scaling of the application for any tasks, the library also improves the performance of the application itself.

1.2.7 Redux-thunk

Redux Thunk is middleware that allows you to return functions, rather than just actions, within Redux. This allows for delayed actions, including working with promises. (Freecodecamp. *Redux Thunk Explained with Examples.* 2022)

The author used the Middleware Thunk to increase the performance of the application by offloading components by storing the main logic in actions.

1.2.8 JWT authentication

JSON Web Tokens are an open and standard (RFC 7519) way for you to represent your user's identity securely during a two-party interaction. That is to say, when two systems exchange data, you can use a JSON Web Token to identify your user without having to send private credentials on every request. (LogRocket. *What is a JWT?* 2022)

The author uses JWT to verify user authentication. Using JWT allows the author to work comfortably with establishing access to certain application features for different types of users.

1.2.9 Material-UI

Using the Material-UI library enables the author to develop the user interface in the client side of the application faster and with better quality. Using this library, the author can devote more time and effort to the main functionality of the application, rather than developing the basic elements for the user interface.

1.2.10 Axios

Axios is a simple promise-based HTTP client for the browser and node.js. Axios provides a simple to use library in a small package with a very extensible interface. (Axios. *Promise based HTTP client for the browser and node.js.* 2022)

Using the Axios library allows the author to avoid writing large amounts of boilerplate code and make the code cleaner and more understandable.

1.2.11 Node.js

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser. Node.js lets developers use JavaScript to write command line tools and for server-side scripting – running scripts server-side to produce dynamic web page content before the page is sent to the user's

web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm, unifying web-application development around a single programming language, rather than different languages for server-side and client-side scripts. (Wikipedia. *Node.js.* 2022)

The author uses the Node.js platform to write the server side using the JavaScript programming language. The author chose this platform because the MERN technology stack is used in development and this stack implies the use of Node.js.

1.2.12 Express.js

Express.js, or simply Express, is a back-end web application framework for Node.js, released as free and open-source software under the MIT License. It is designed for building web applications and APIs. It has been called the de facto standard server framework for Node.js. (Wikipedia. *Express.js.* 2022)

Using the express.js framework simplifies working with node.js, since this technology allows you to develop the server side with maximum performance, namely, thanks to the functions of this technology, the author does not need to repeat the same code over and over again.

1.2.13 bCrypt

Using the bCrypt function allows the author to encrypt user passwords in order to secure all data being used.

1.2.14 MongoDB

When choosing a database for the application, the author was guided by the future goals and scope of the web platform. The author had a choice between MySQL and MongoDB databases. The choice fell on the MongoDB database, as in the future the application will contain a large amount of embedded data for each JSON document used by the application. It was this reason that served as a key decision towards MongoDB, because the MySQL database does not provide such flexibility and convenience when embedding data. Also, as a result of embedding data in MySQL tables, tables can grow in size, which can make the application more demanding on performance. Using a MongoDB database simplifies all of these processes and doesn't impact the application's performance requirements as much. At the moment the MongoDB database contains the data of existing portfolios and registered users.

1.2.15 Mongoose

Mongoose is a JavaScript object-oriented programming library that creates a connection between MongoDB and the Express web application framework. (Wikipedia. *Mongoose (MongoDB).* 2022)

Using mongoose makes it possible to provide communication between the MongoDB database and the backend of the application.

1.2.16 Yarn

When choosing a package manager, the author had a choice between npm and yarn. The author uses yarn as a package manager to manage all dependencies, since for a number of technical reasons, yarn works better on the author's computer and without various errors..

1.2.17 Git

Git is software for tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development. Its goals include speed, data integrity, and support for distributed, non-linear workflows (thousands of parallel branches running on different systems). (Wikipedia. *Git.* 2022)

Using git allows the author to manage project versions by safely testing and introducing new features to the project.

2 DEVELOPMENT

2.1 Requirements

When developing any type of application, defining functional and non-functional requirements is a common practice, so these requirements will be described below.

2.1.1 Functional requirements

Functional requirements explain what needs to be done. They identify tasks or activities to be performed:

1. The user can register by entering the first name, last name and email along with a password.
2. The user can log in to the system using the specified email and password, or log in using a Google account.
3. The app should display existing portfolios on the homepage whether the user is logged in or not.
4. If the user is not authorized, a notification should be displayed on the main page stating that the user must be authorized in order to create, publish and edit a portfolio.
5. An unauthorized user can use the search and pagination of pages.
6. An unauthorized user cannot "Like" a published portfolio.
7. An unauthorized user can go to the details page of each portfolio for a detailed inspection.
8. Authorized user can create a new portfolio.
9. An authorized user can only edit and delete their own portfolio.
10. The brief version of the portfolio displayed on the main page should contain: the name of the creator, date of publication, title, tags and introduction.
11. The button for deleting and editing a post is available only to the owner of the published portfolio in the brief portfolio form, on the main page.
12. The user can put only 1 "Like" on the selected portfolio.

2.1.2 Non-functional requirements

Non-functional requirements are requirements that define exactly how an application should look or the constraints it should obey that are not related to the behavior of the system. These requirements are as follows:

1. The application must be accessible using any Internet browser.
2. The web application must be adapted to all types of computer screens.
3. The appearance of the application should be as simple and understandable as possible for the user.
4. If the platform is not in maintenance mode, all functions should work properly.

2.1.3 Use Case diagram

Use Case diagram is a scenario for the interaction of participants with the system.

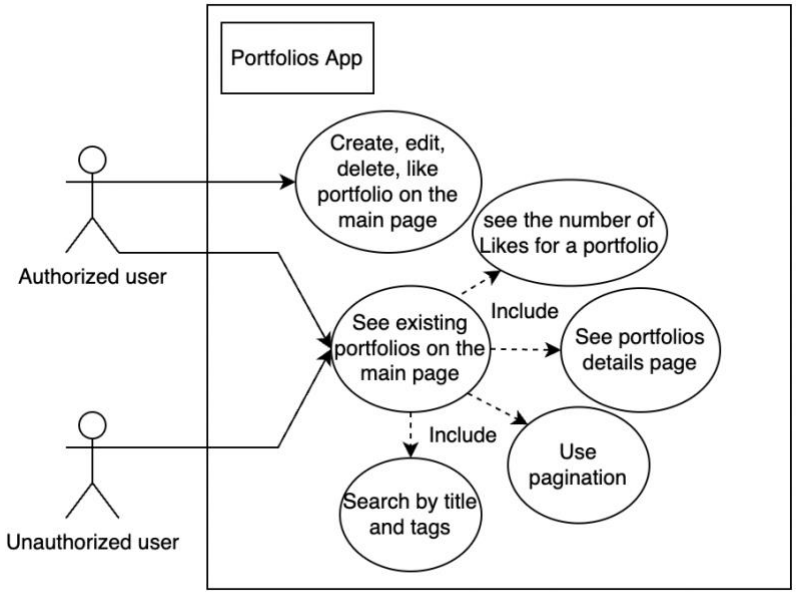

Figure 1 *Use case diagram (Source: author)*

In this Use case diagram, the author describes the interaction of two types of users with the application itself and its functions. This diagram shows two types of users: authorized user and unauthorized user. As the diagram shows, both users have access to view the main page, which displays existing portfolios. Also, on the main page there is a search box and a pagination function between pages, two types of users also can use these two functions. When clicking on any post with a portfolio – both types of users will be redirected to the details page of the portfolio itself, which displays full information about the selected portfolio. Once the user goes through the registration/authorization process, he can create, edit, delete and like existing posts. Diagram is represented in **Figure 1** Use case diagram.

2.1.4 Data Flow Diagram (DFD)

The data flow diagram shows the step-by-step operation of the system. In the diagram, you can see shapes such as rectangle, rounded rectangle, rhombus, and arrows that create a connection between each element in the system. Rectangles represent an action that an application performs, rounded rectangles represent elements in an application or user actions in a system. Rhombus represents an "if" statement and indicates the next step in the application. For a better understanding of the application, the author has built diagrams for each component, namely for: authentication page, homepage, navigation bar, details page. The diagram is represented in **Figure 2** – **Figure 5**.

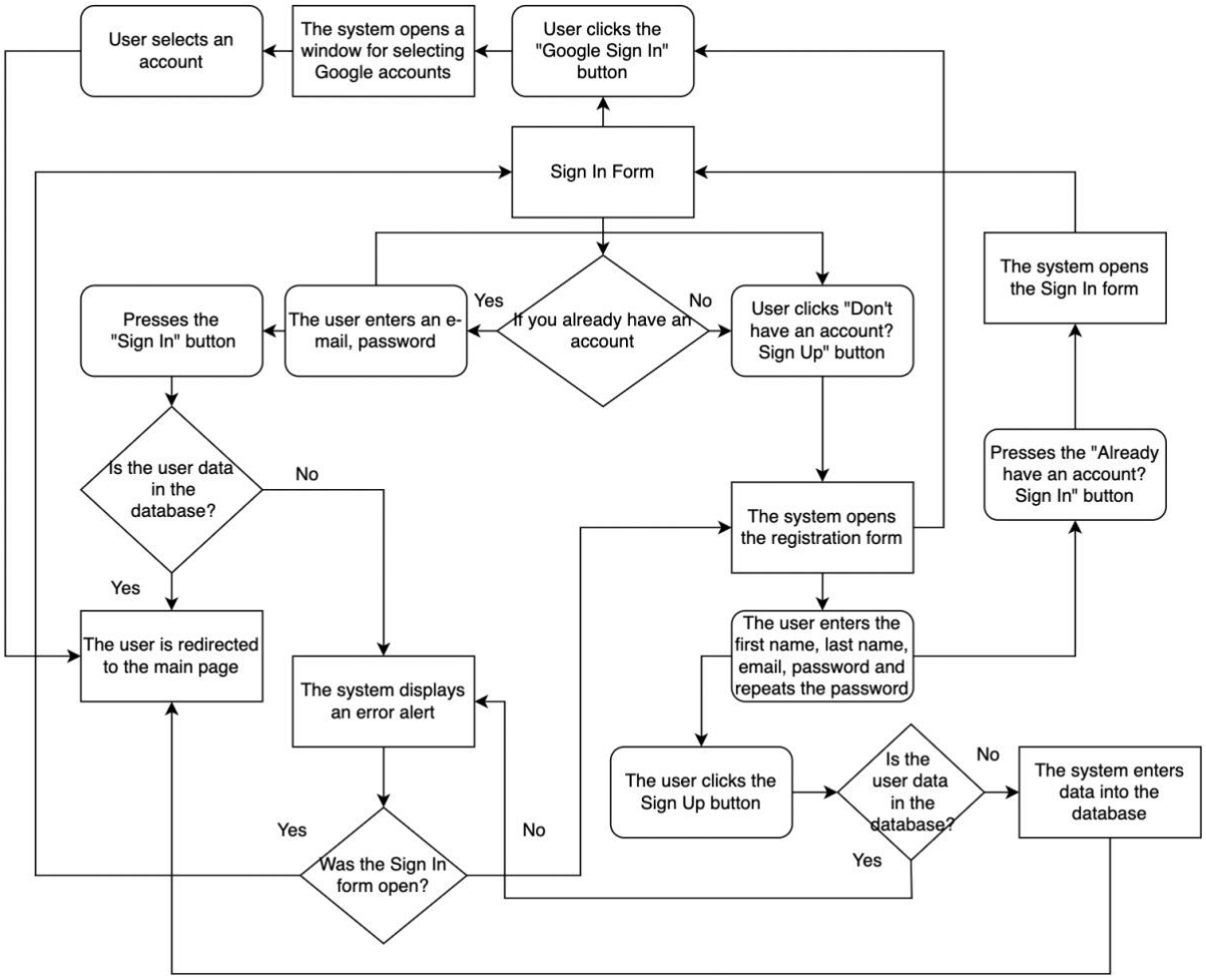

Figure 2 *Data Flow Diagram of the authentication page (Source: author).*

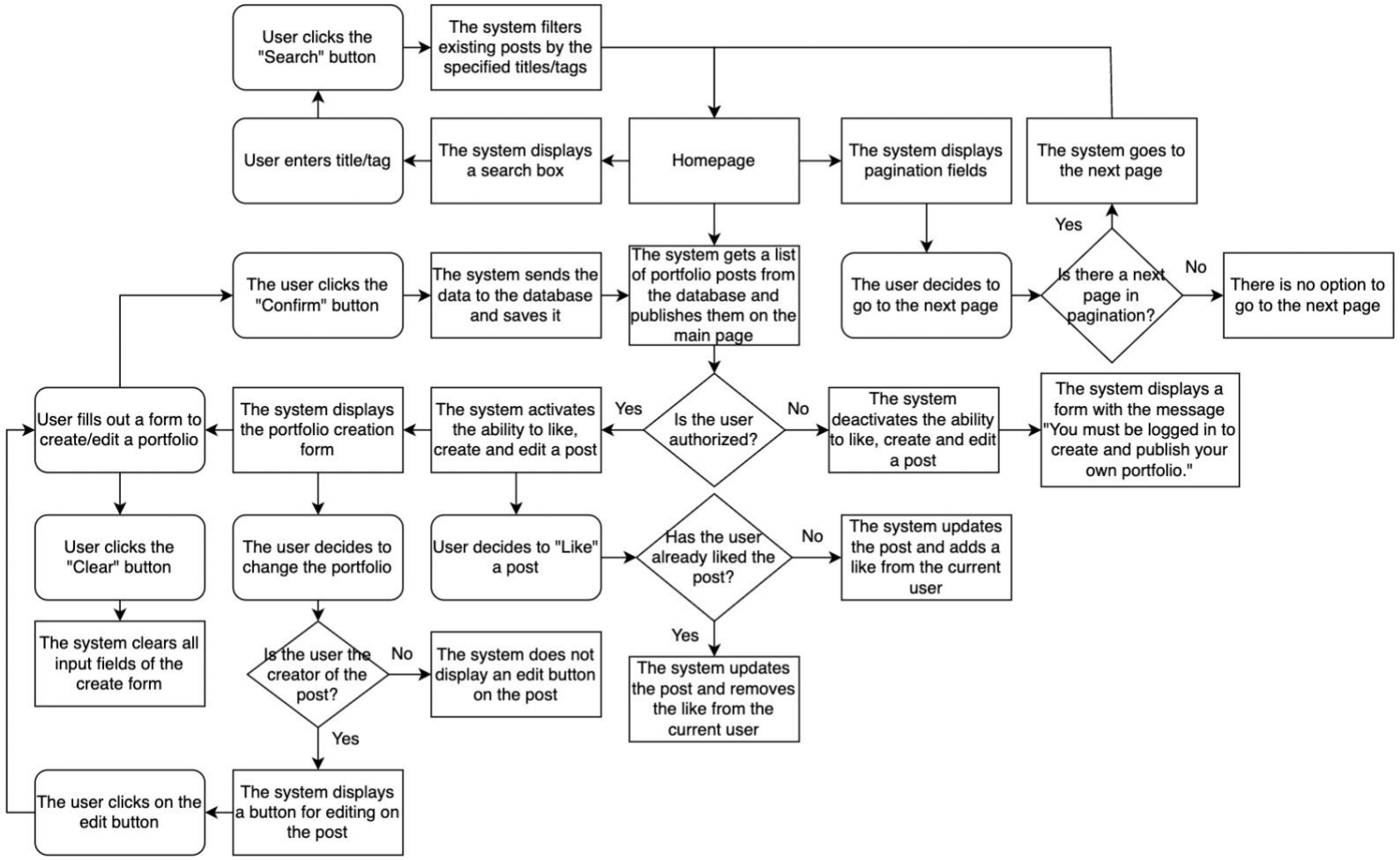

Figure 3 *Data Flow Diagram of the homepage (Source: author).*

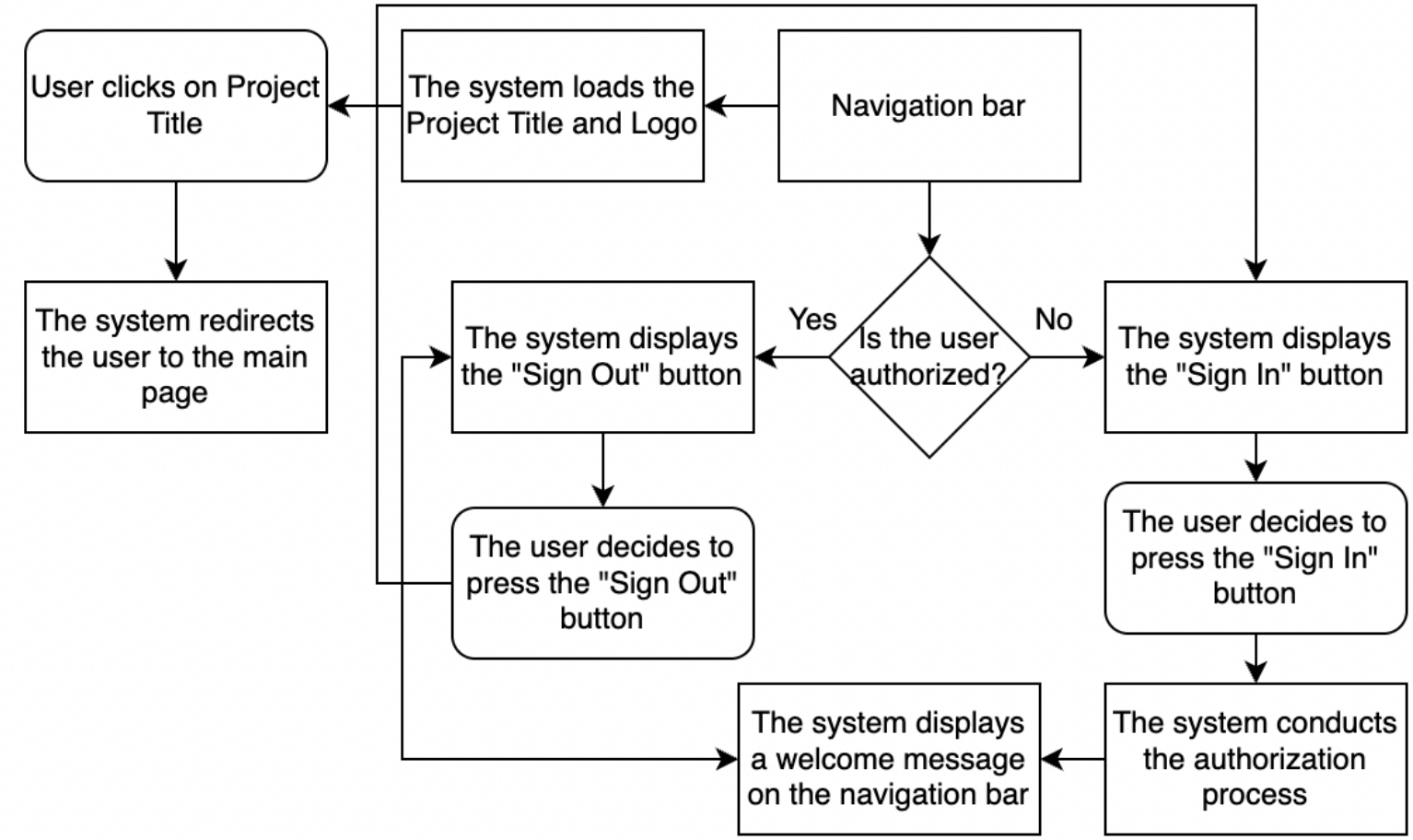

Figure 4 *Data Flow Diagram of the navigation bar (Source: author).*

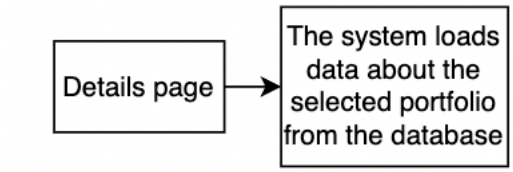

Figure 5 *Data Flow Diagram of the Details page (Source: author).*

2.1.5 Database design

Database design shows how a database is created and how data is stored in tables. This is necessary to fully understand where the data will be stored and how the data will be placed in the system.

There are two main tables in the database: there is the table with existing portfolios, labeled as portfolios, which contains information about all the posts with the portfolio, and the second table is the table with users, labeled as users, which has all the information about registered users, it needed to verify existing users as well as register new users. Portfolio posts table is also needed for checking existing portfolio posts and for newly created portfolio posts.

More information about the database used is contained in **Figure 6**.

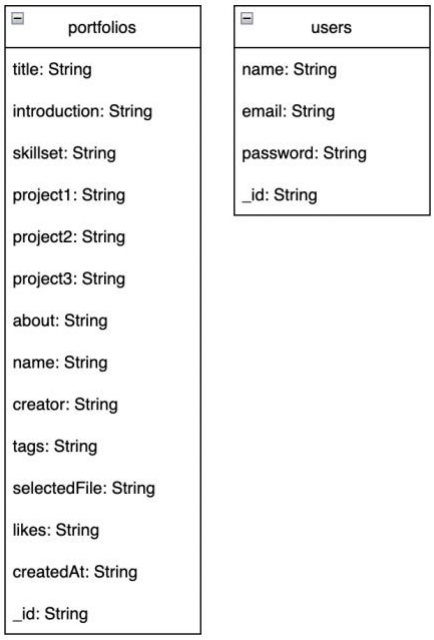

Figure 6 *Application database design (Source: author).*

2.1.6 Application mockup

Now, after a detailed description of the operation of the entire application system, the author can start working on the mockup, following all the above requirements.

The main purpose of an application mockup is to demonstrate exactly how the application should look and show how it works in different situations.

When the application is launched, the main page is loaded, where the existing portfolio posts are displayed, however, the author decided to start with a description of the application from the authorization page. It is worth clarifying that if the user has already passed the authorization process, then he will be in the system for only 1 hour, after which the user account will automatically log out of the system. As soon as the user enters the authorization page, a login window will be displayed in front of him, where 2 input fields will be available: email and password. Also, 3 buttons will be displayed in the login window: "Sign In", "Google Sign In", and the button "Don't have an account? Sign Up". If the user is already registered and has a personal account, then to enter he will need to enter his email, password and click the "Sign In" button. To log in through a google account, the user just needs to click the "Google Sign In" button, after which the user will be redirected to a page with a google account selection for signing in. If the user wants to register a new account, then he should click the button “Don't have an account? Sign Up”, after which the user will be redirected to the registration page. On the registration page, a form will be displayed with required input fields, namely: first name, last name, email, password and a field with a password repetition. After that, the user should click the "Register" button. If all the fields have been filled in and there is no account in the database with the same data, then the user will be registered and redirected to the main page, otherwise an error pop-up will be displayed. The registration page also has a google sign in button to sign in with a google account, and a button called "Already have an account? Sign In", if the user clicks it – the user will be redirected to the login page. The user may not go through the authorization process, but in this case, he will have limited rights to use the platform. Representation of mockup can be found on **Figure 7**.

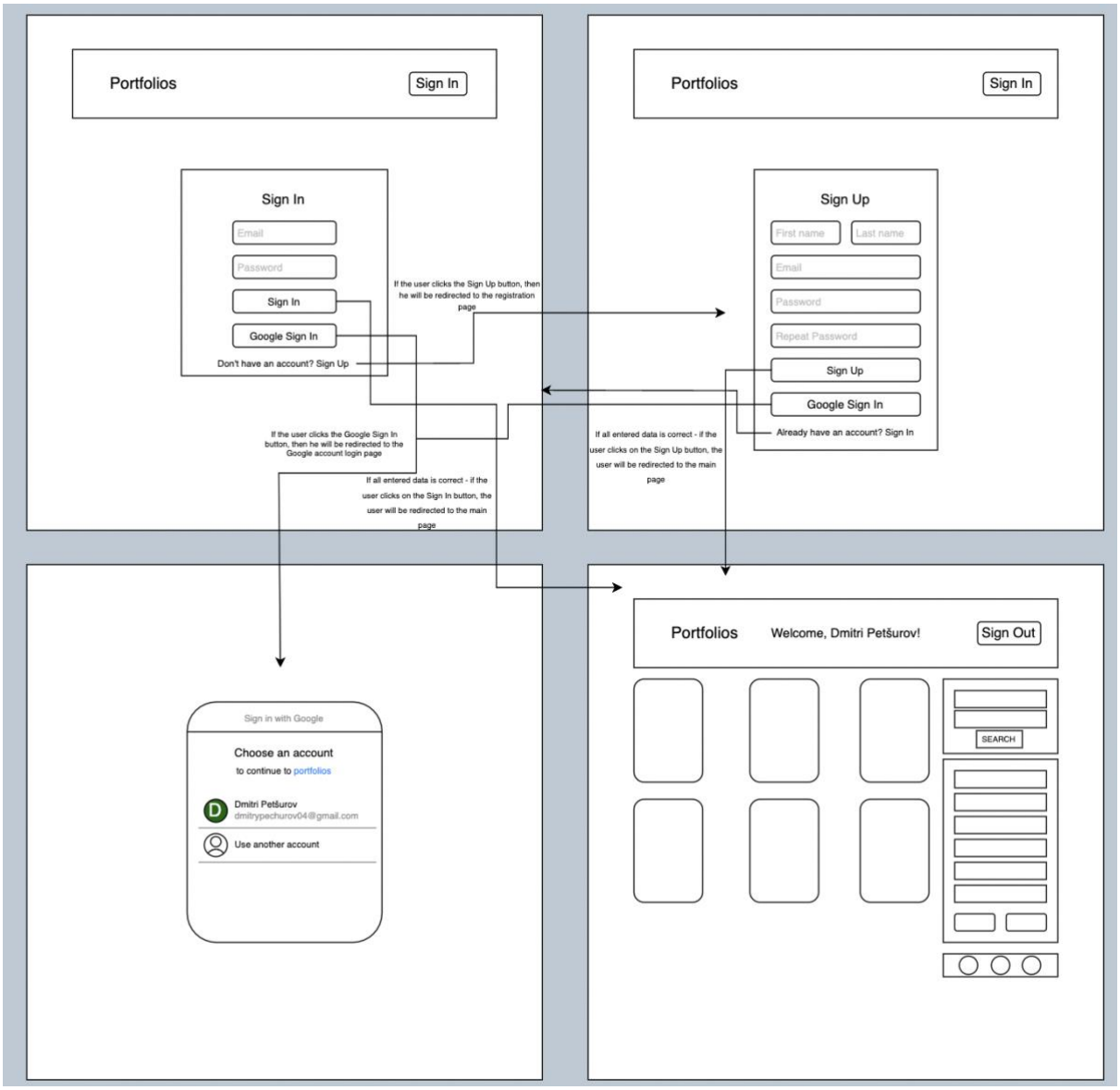

Figure 7 *Mockup of authentication page (Source: author).*

There are two types of users who can use the application, namely: an authorized user and an unauthorized user. What follows is a description of the functionality of all the components used and their interaction with each type of user.

There are several components on the main page at once, namely: navigation bar, search box, posts, creation form and pagination.

The navigation bar component contains a “Sign In” button, when it is pressed, an unauthorized user will be redirected to the authorization page. For an authorized user, a welcome message will be displayed on the navigation bar, as well as a "Sign Out" button.

By clicking the Sign Out button, the user will be automatically logged out and redirected to the authentication page.

The search box component allows both types of users to use the search for portfolio posts by titles and tags, once the user enters a title or tags in the appropriate fields, the user must click the "Search" button, after which the system will automatically display all portfolio posts that contain those specified in search box component data.

The portfolios component contains separately implemented portfolio components displayed in view of the portfolio post. Both user types have access to view existing portfolio posts. In addition to the fact that existing posts are simply displayed, an authorized user can like by clicking the "Like" button. The number of likes for each user is limited – 1 like for 1 post, pressing the "Like" button again will remove the like. Also, an authorized user can delete and edit their portfolio. To do this, buttons will be displayed on each portfolio of the author: the "…" button for editing the portfolio and the "Delete" button for deleting the portfolio. When you click the button for editing, all information about the existing portfolio will be displayed in the creation form component, an authorized user can make their changes and click on the "Submit" button, all the data of the portfolio post will be updated. When clicking on the portfolio post itself, both types of users will be redirected to the details page of the selected portfolio post, which is the details component, where all the portfolio information will be displayed. The details component will be described after the description of all the components that the main page contains.

The creation form component has already been partially touched on earlier. The main task of this component is to create new portfolio posts and edit existing portfolios. This component will be displayed only for an authorized user. Once the user has passed the authorization process, he can create his portfolio by filling in all the fields in the creation form component. After filling in, the user can use the two buttons that the current component contains, namely: the "Submit" button to confirm the entered data and the "Clear" button to clear all filled fields. As already described, if the user clicks the submit button, the system accepts the entered data and creates a new portfolio post, which will be displayed on the main page in the portfolios component. If the user clicks the Clear button, all filled fields will be cleared and the data will not be saved.

The last component on the main page is the pagination component. This component is available to both user types and serves for pagination between pages. Each page contains 6

portfolio posts, the rest of the posts go to the next page, that is, as soon as the next post is added and the number of posts on the page becomes more than 6, then the added post is automatically transferred to the next page. The pagination component also updates and starts rendering the new page. Representation of mockup can be found on **Figure 8**.

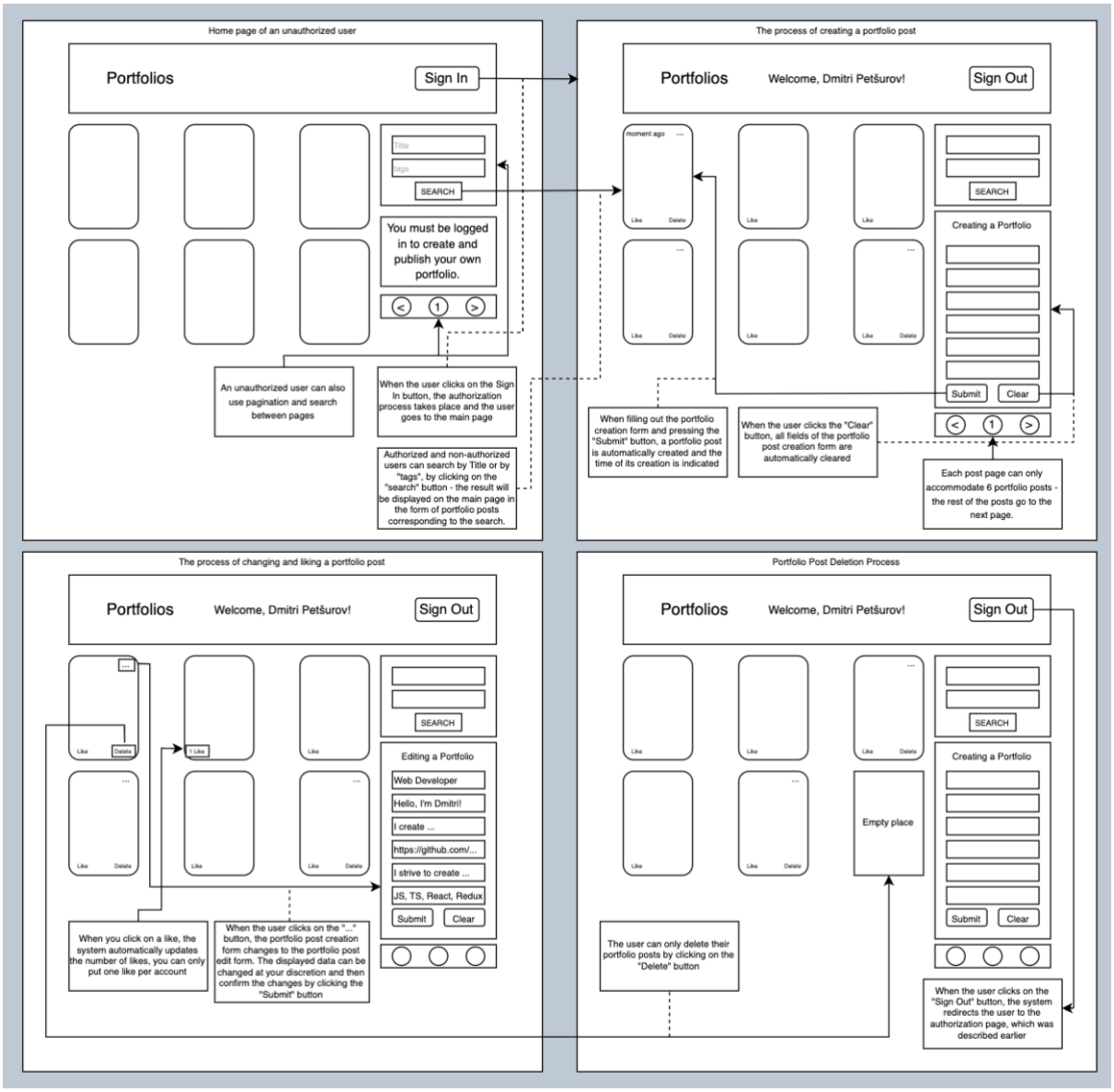

Figure 8 *Mockup of main page (Source: author).*

The details page is a details component. When clicking on any existing portfolio post that is on the main page, the user will automatically be redirected to the details page, which displays the full information of the selected portfolio post. The details page is a uniquely styled page that can also be accessed using a unique path in the address bar. Representation of mockup can be found on **Figure 9.**

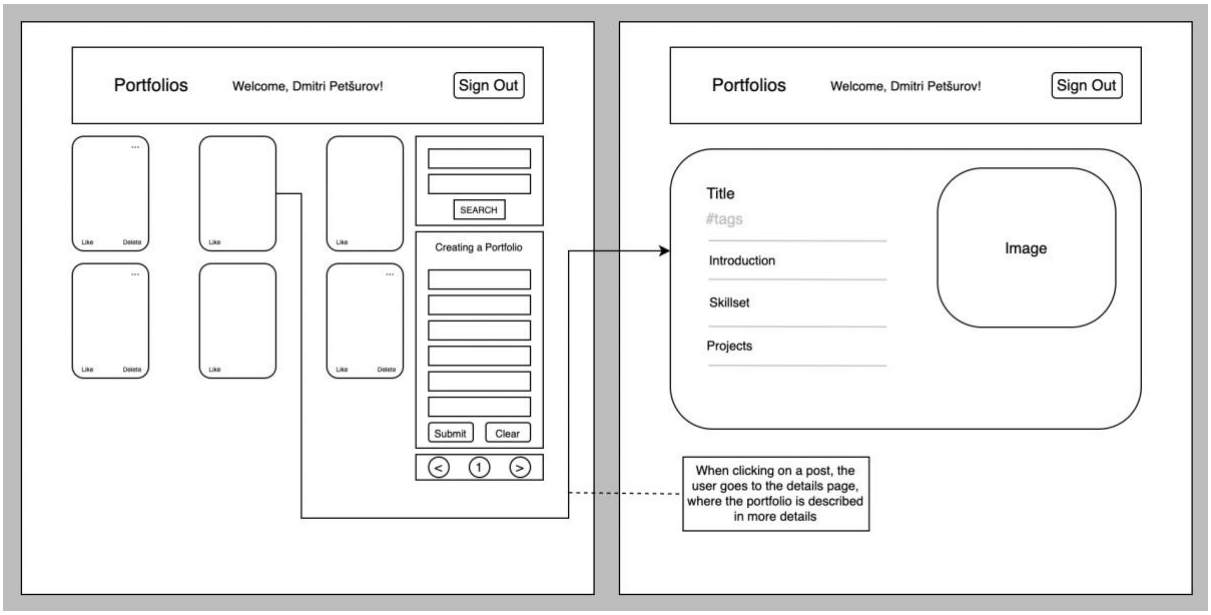

Figure 9 *Mockup of details page (Source: author).*

3 APPLICATION FUNCTIONALITY

Since the work of the author is a web application, the project consists of two parts: the client part and the server part. In this section, the author will describe in detail how the application works and all the existing functions that give the application the appropriate functionality.

First of all, the author will describe the work of the client part, namely the description of all existing components and what they are responsible for. Then the client side with all related functions will be described.

3.1 Client Side

Before starting to describe all the functionality of the client part of the application, it is necessary to describe the available components and what they are responsible for. Using components, the author creates the graphical interface of the application, in other words, this is what the user sees on the pages of the application. The entire client side uses 8 components, namely: auth, details, form, home, navigation, pagination, portfolios and portfolio. In addition to the used components, the client-side also contains other files that help to give the existing components some functionality and help to combine all the components and display them in the application itself.

3.1.1 Navigation component

As soon as the user enters the author's web application, he finds himself on the main page, which contains all the previously described components. In order to go through the authorization process, the user must click the "Sign In" button, which is in the navigation component, and we will talk about it.

All the functionality of the navigation component is in the client/src/components/navigation folder, by following this path you can find 2 files: navigation.component.jsx and navigation.styles.js. These two files form the navigation component. The navigation.component.jsx file contains the main logic of the component, while the navigation.styles.js file sets the styles for all the elements used in this component. The navigation component can change depending on whether the user is logged in or not. In both cases, the navigation component displays the logo, for which the img element is responsible, which, in turn, accepts an image in .png format, which is displayed on the navigation bar in the application itself. If the user clicks on this logo, it will be automatically redirected to the

main page, this function allows the user to return to the main page at any time. This function is implemented in the link element returned by the navigation component. As mentioned earlier, depending on whether the user is logged in or not, the content of the navigation component changes. If the user is authorized, then the welcome message is displayed on the component, for which the typography element is responsible.

This element contains the following logic: If the user is authorized, then a welcome message should be displayed, otherwise nothing should be displayed. Also, depending on the status of the user, the content of the authorization button changes. If the user is authorized – then the button displays the message "Sign Out" and when it is pressed, the user will automatically log out and be redirected to the authorization page, this logic is implemented by the "logout" function, which contains the navigation component. If the user is not authorized, then the button displays the message “Sign In” and when it is pressed, the user will also be redirected to the authorization page, this logic contains the Link element. Representation of screen can be found on **Figure 10** and **Figure 11**.

Figure 10 *Appearance of the navigation component if the user is logged in (Source: author).*

Figure 10 *Appearance of the navigation component if the user is logged out (Source: author).*

3.1.2 Home component

Once the user enters the main page of the application, everything displayed below the navigation component that was previously described is the home component. This component is in the client/src/components/home folder. The home component is a kind of container that contains several other components at once. In this case, the home component contains components such as portfolios, search box, form and pagination. Representation of screen can be found on **Figure 12.**

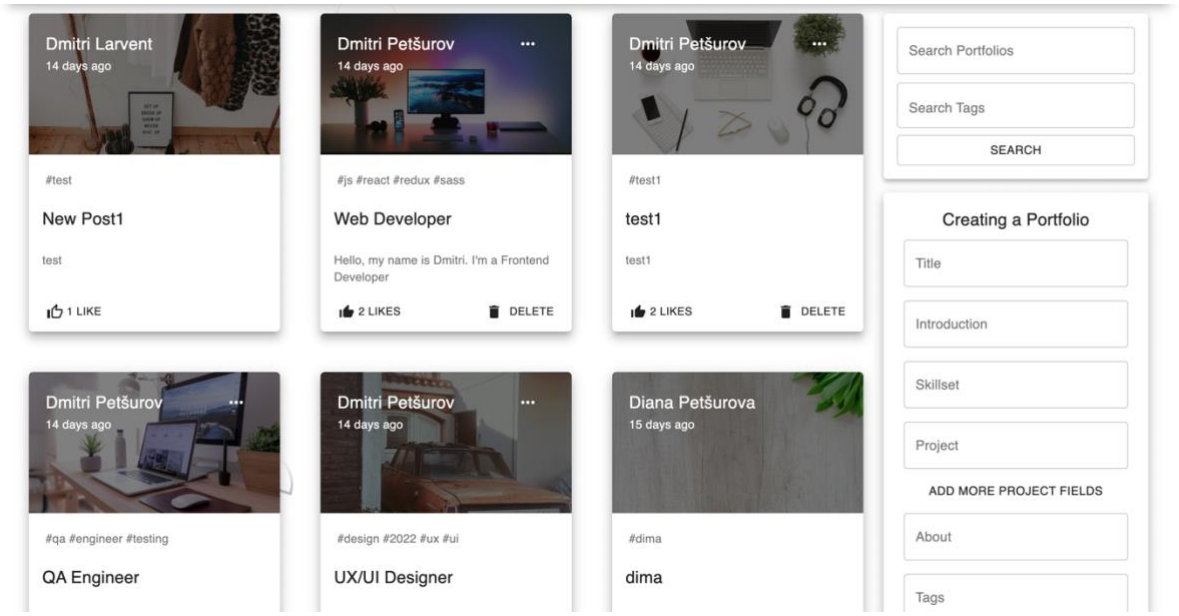

Figure 12 *Component home containing the rest of the components (Source: author).*

3.1.3 Portfolios component

It's worth starting with the most noticeable component, which is in the home component – this is the portfolios component. This component is in a separate client/src/components/portfolios folder. The portfolios component is also a kind of container for the portfolio component, however, portfolios also have their own role. The main purpose of the portfolios component is to group portfolio components in a certain order, which can be very large, but this will be described later. The main logic of the portfolios component begins with the fact that each time the main page is loaded, the component checks the length of the portfolios array, which in turn receives through the useSelector hook from the application store.

If the portfolios array contains nothing, then the message “No portfolios” is displayed, if the portfolios array is not empty, then using the “.map()” method, the system accesses each individual portfolio and passes this post to the portfolio component for further processing. This populates the portfolios component with the individual portfolio components and passes a well-grouped list of components to the main Home component for further processing. Representation of screen can be found on **Figure 13.**

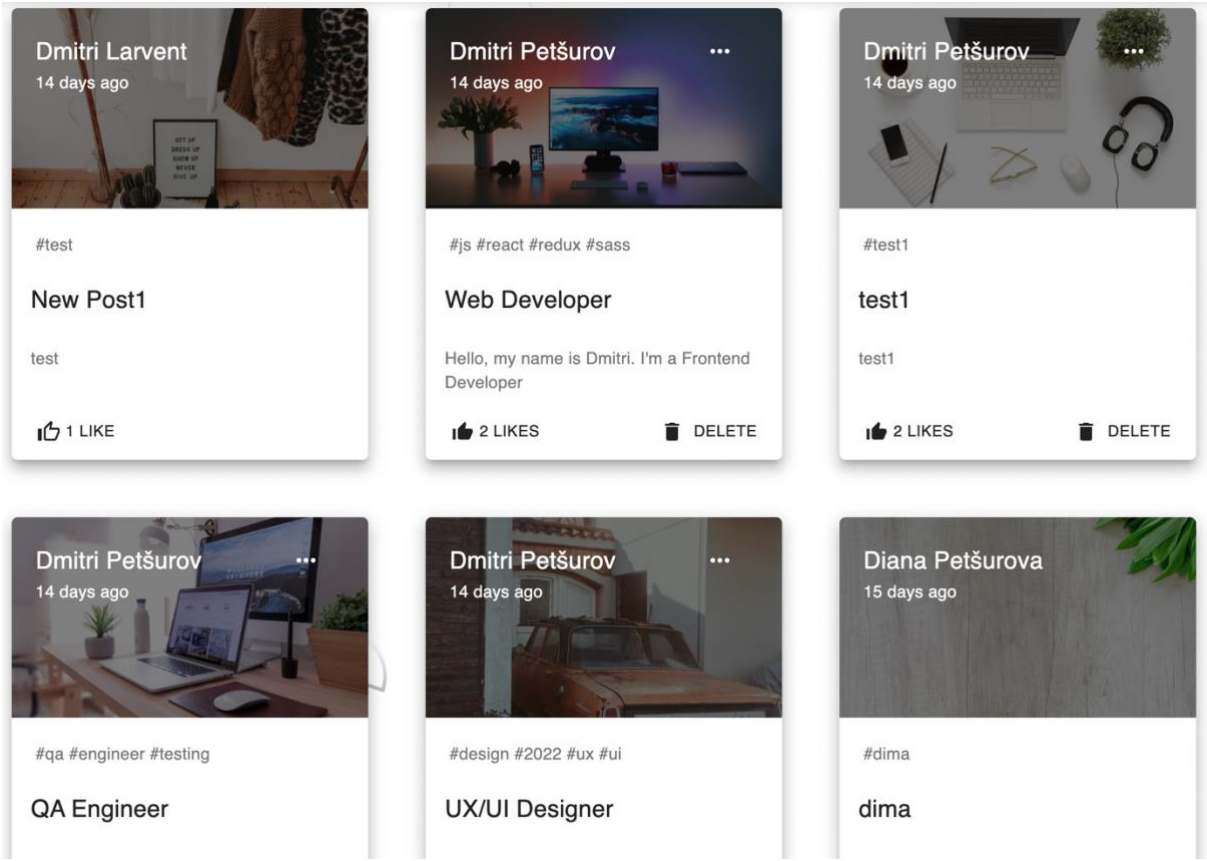

Figure 13 *Component portfolios, which contains individual portfolio components (Source: author).*

3.1.4 Portfolio component

Since we were talking about the portfolios component and the portfolio component was partially affected there, this component should be described in more detail. The portfolio component is in the client/src/components/portfolios/portfolio folder. As mentioned earlier, the portfolio component gets information about each individual portfolio post from its parent portfolios component. After the portfolio component has received the information, it processes all the received data and displays them using special elements in a beautifully designed separate post. The following elements are used to display data:

- Card element for the formation of a Card, where all the information received will be displayed.
- The ButtonBase element, which allows you to make part of the area of the Card element clickable and add some function to this click.
- A CardMedia element that displays a Title and a user-selected image.

- Typography element to display all text on the Card element
- The Button element to display the button.
- The CardActions element for combining buttons with actions set to them.
- ThumbAltIcon for Like button.
- DeleteIcon for the delete button

All listed elements and icons were imported from the Material-UI library.

When clicking on the portfolio component itself, the openDetails function is activated, which redirects the user to the details page of the selected post.

Further, it is worth mentioning that if the user is authorized and he is the owner of the selected portfolio post, then 2 buttons become available for this user: the post editing button "…" and the post deletion button "Delete". If you click on the "…" button, then the id value of the current post is set and passed to the Form component for subsequent editing of the data of the selected post. The Like button can only be used by an authorized user, it works in this way – when pressed, the likePortfolio function is activated, which updates the Like value of the selected post. If the user is not authorized, then he cannot like the selected post, he can only see the number of Likes for the post. If the user is the owner of the selected post – then the delete button "Delete" is available to him, if it is pressed – then the deletePortfolio function is activated and deletes the post selected by the user, deleting the post from the list of the total number of portfolio posts. Representation of screen can be found on **Figure 14.**

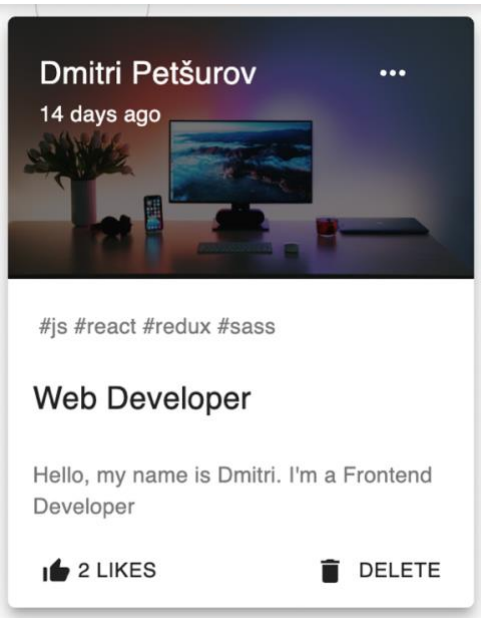

Figure 14 *portfolio component styled using all described elements (Source: author).*

3.1.5 Search box component

The next component to be described is the search box component, which has all of its main logic in the home component. The path to this component is the following: client/src/components/home, in fact, the search box component is not a full-fledged component, because it is assembled from the existing elements of the Material-UI library directly in the return() method, however, it will be more convenient to describe it further call it a component. The search box component is 2 text input fields along with a search button, all combined into one container. In order to get such a component, the author used the following elements: AppBar, TextField, ChipInput, Button. The AppBar element serves as just the same container for combining other elements. The TextField element is the input field that the author used to search for portfolios by titles. The ChipInput element, as the name implies, is a smart input field that was used to search for a portfolio by individual tags. Button is an element for sending our search request, when you click on the search button, the "searchPortfolio" function is triggered, which in turn calls the "getPortfoliosBySearch" function and it already sends a request with data to the server side of the application and then receives a response back. The home.styles.js file is responsible for styling the elements, which is in the same folder as the home component. Representation of screen can be found on **Figure 15.**

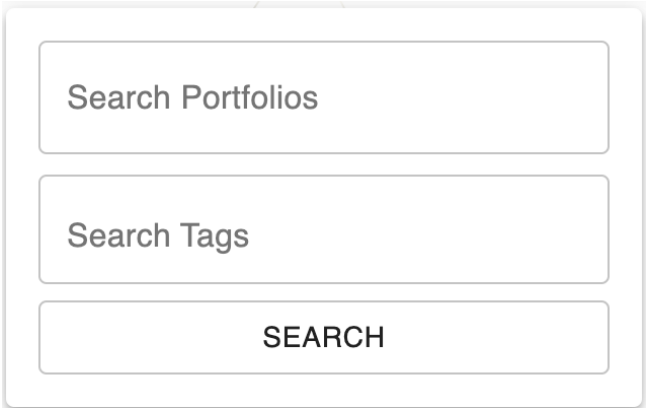

Figure 15 *search box component (Source: author).*

3.1.6 Form component

The following will describe one of the most important components – this is the form component. The form component is located at the following address in the

client/src/components/Form folder. The Form.component.jsx file is responsible for building the entire component, while the Form.styles.js file is responsible for styling all elements of this component. The form component is a form for creating and editing a portfolio. Before the component is loaded on the main page, it checks whether the user is authorized or not. If the user is not authorized, then instead of displaying the form for creating and editing a portfolio, the Paper element is displayed with a text message: "You must be logged in to create and publish your own portfolio.", This message is contained in the Typography element, all elements used are provided by the Material-UI library. If the user is authorized, then the component displays the following elements: the form element to group all other elements into one container, Typography to display the form header, TextField to display input fields, Button to add additional input and confirmation fields with clearing input fields, FileBase64 to add pictures.

When filling out the form to create a portfolio, the user can add a link to his project, but if the user wants to add several projects, he can click the "Add more project fields" button, which, when clicked, calls the addMoreFields function, this function changes the boolean value to the opposite the moreFields variable, and depending on the value of the variable, the form shows more input fields or hides them.

As soon as the user has filled in all the input fields and decided to save all the data in a new portfolio, he must click on the Submit button.

When the submit button is pressed, the handleSubmit function is called and then a check is made: if the value of the currentId variable is not 0, then the system will not create a new portfolio, but edit the selected portfolio using the updatePortfolio function, which in turn sends a .patch() request to the application server side to change the selected portfolio. If the value of currentId is 0, then the system will create a new portfolio using the createPortfolio function, which in turn will send a .post() request to the server side of the application and create a new portfolio.

When the clear button is pressed, the clear function is called, which sets the value of the currentId variable to 0 and clears all input fields using the setPortfolioData variable. Representation of screen can be found on **Figure 16 and Figure 17.**

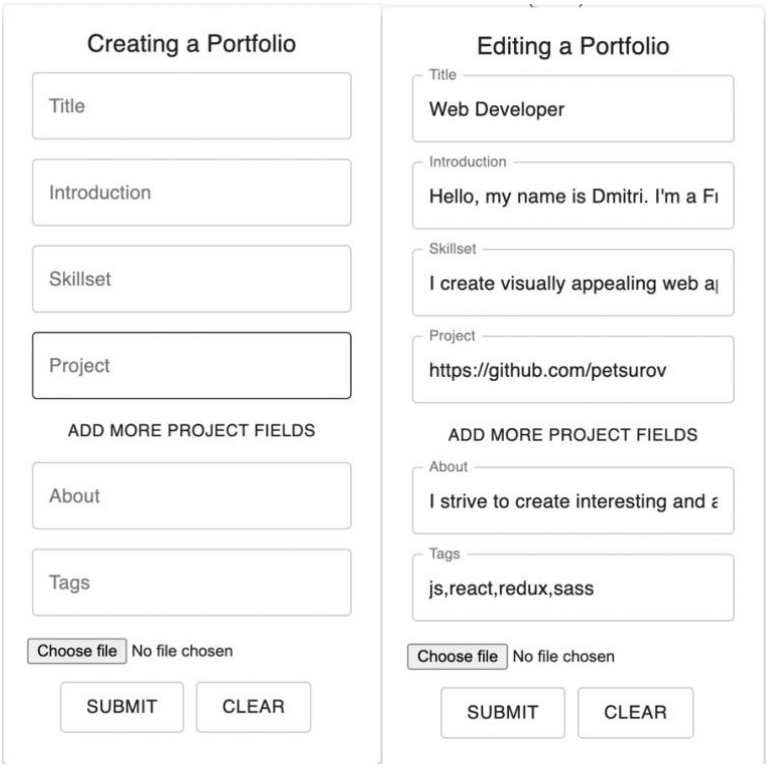

Figure 16 *Portfolio creation and editing form component for an authorized user (Source: author).*

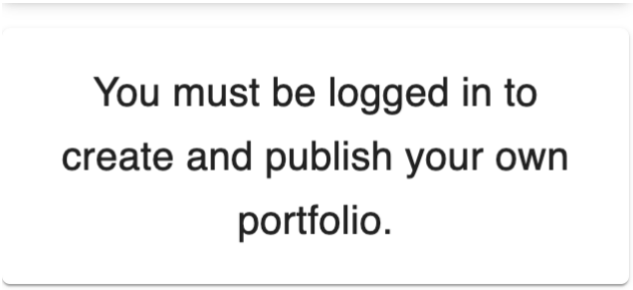

Figure 17 *View of the form component for an unauthorized user (Source: author).*

3.1.7 Pagination component

The last component on the main page is the pagination component. This component is in the client/src/components/pagination folder. The pagination.component.jsx file contains the main logic and elements of the pagination component, while the pagination.styles.js file is responsible for styling all the elements of the pagination component. Pagination is a navigation bar for existing pages with portfolio posts. This component consists of the

following elements: a pagination to combine and style all the elements it contains, and a PaginationItem element that displays the element itself inside the pagination panel.

The pagination element accepts parameters such as classes for styling, count for displaying the total number of pages, page for displaying the current page, and the renderItem parameter, which takes the PaginationItem element to specify the path for redirection user. When clicking on the PaginationItem element, the user will be redirected to the page specified in the element, the “to” parameter is responsible for this. Representation of screen can be found on **Figure 18.**

Figure 18 *View of the pagination component (Source: author).*

To complete the description of the client side, 2 more components remained undescribed – this is the auth component, which is responsible for the authorization page, as well as the details component, which displays the details page of each portfolio, which stores complete information about each selected portfolio.

3.1.8 Auth component

The auth component will be described first. This component is in the client/src/components/auth folder. The specified folder contains several files, each of which has its own task. The auth.component.jsx file contains all the logic of the auth component, and the auth.styles.js file sets the styles for all elements of this component. The input.component.jsx file contains a separate input component that is used in the main auth component, and the input.styles.js file sets the styles for the input component.

As described earlier, the auth component is used to display the login and registration form. To get to the authorization page, the user must click the "Sign In" or "Sign Out" button, which is located on the navigation bar, depending on whether the user is authorized or not.

As soon as the user has pressed the button, he will immediately be redirected to the authorization page, where the auth component will be automatically located. Initially, the authorization page displays the Sign In form, but it can be changed to the Sign Up form, for

this the user must click the “Don’t have an account? Sign Up”, but if the user wants to return to the Sign In form, there is a button “Already have an account? Sign In", which is located on the Sign Up form. If the user enters the authorization page, the Sign In form will always be displayed in front of him, since the boolean variable isSignUp is responsible for this, which initially has a value of false. The logic for changing the form is as follows: if the value of the isSignUp variable is false, then the Sign In form will be displayed, if it is true, then the SignUp form will be displayed. The value of the isSignUp variable changes when you click on the buttons: “Don’t have an account? Sign Up” or “Already have an account? Sign In", which in turn call the switchMode function, where all the main logic is located. Next, it is worth describing all the elements used to build the entire auth component, namely: Container to combine all elements together, Paper has a kind of paper view to display other elements on it, the Avatar element displays a LockOutlinedIcon, Typography to display text, form to confirm input data, a Grid for grouping elements, an Input element for displaying a text input field, a Button for displaying a button, and a GoogleLogin element for building the google account login logic.

If the user already has an account, then by filling in all the input fields in the Sign In form, he can log in, after which he will be redirected to the main page. The login process with an existing account looks like this: the user enters an email and password, and then clicks the Sign In button. When the Sign In button is pressed, the handleSubmit function is activated, this function immediately checks the value of the isSignUp variable and if it is false, then the signin function is called, which in turn sends a .post() request to the server side of the application and checks whether this user exists or no. If the user exists, then the authorization was successful, and the user is redirected to the main page, otherwise an error is generated in the system.

If the user wants to register a new account, then he must click the “Don’t have an account? Sign Up" and the Sign Up form will open. In this form, the user will have to enter First name, Last name, email, password and repeat password. Once the user has entered all the data, he must click the Sign Up button. When you click Sign Up, the handleSubmit function will be called, where the value of the isSignUp variable will be checked immediately. If isSignUp is true, then the signup method will be called, which sends a .post() request to the server side of the application, where it checks if there is a user with the same data in the database, and if there is no such user, then the user will be automatically registered and redirected to the main page, otherwise an error will be displayed in the system.

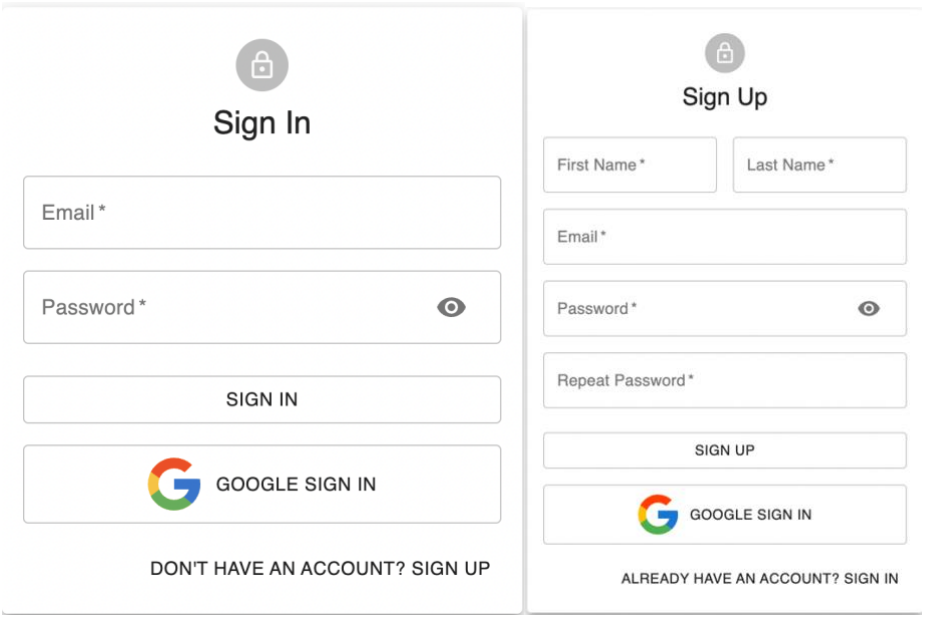

Figure 19 *View of the auth component (Source: author).*

The last feature left in the auth component is the google account login feature. As soon as the user clicks the “Google Sign In” button, an error check is performed, if no errors have occurred, the onSuccess parameter and the googleSuccess function are activated, if an error occurs, then the onFailure parameter and the googleFailure function are activated. The googleSuccess function sends the received data to the server side and the authorization process takes place, after which the user is redirected to the main page. Representation of screen can be found on **Figure 19.**

3.1.9 Details component

The last component on the client side is the details component. The details component is in the client/src/components/details folder. This component is responsible for the detail page of each portfolio. The details page displays all the information that the user entered when creating the portfolio. There are 2 files in the component folder itself: detailsPage.component.jsx, which has all the main logic of the component, as well as the detailsPage.styles.js file, it is responsible for styling the details component.

The details component itself consists of the following elements: The Paper element to display the paper, where all the other components will be located, it also uses several div components to divide the elements into sections. The Typography element is used to display the text, and

the Devider element is used for the visual effect of the separator between the text. To display the text of links to projects and the ability to navigate through them to other sites, the <a> and CardActions elements respond. The last element responsible for displaying the image is the img element.

This element also contains some functionality, for example, as soon as the user opens the details page, the system should immediately start loading information from the portfolio selected by the user. The getPortfolio function is responsible for this functionality, to which the id parameter is passed, passing the id parameter to the function is necessary so that the server side of the application, when receiving a request, can understand exactly which portfolio information it needs to send to the client side of the application for further display. While the system is still waiting for a response from the server side of the application, the download icon will be displayed, it will disappear as soon as the information is received. The isLoading variable is used to display the loading icon, which, when set to true, renders a CircularProgress element inside the Paper element. As soon as the client side of the application has received information about the selected portfolio, the elements used in the component can freely use this information to display it on the screen. Representation of screen can be found on **Figure 20.**

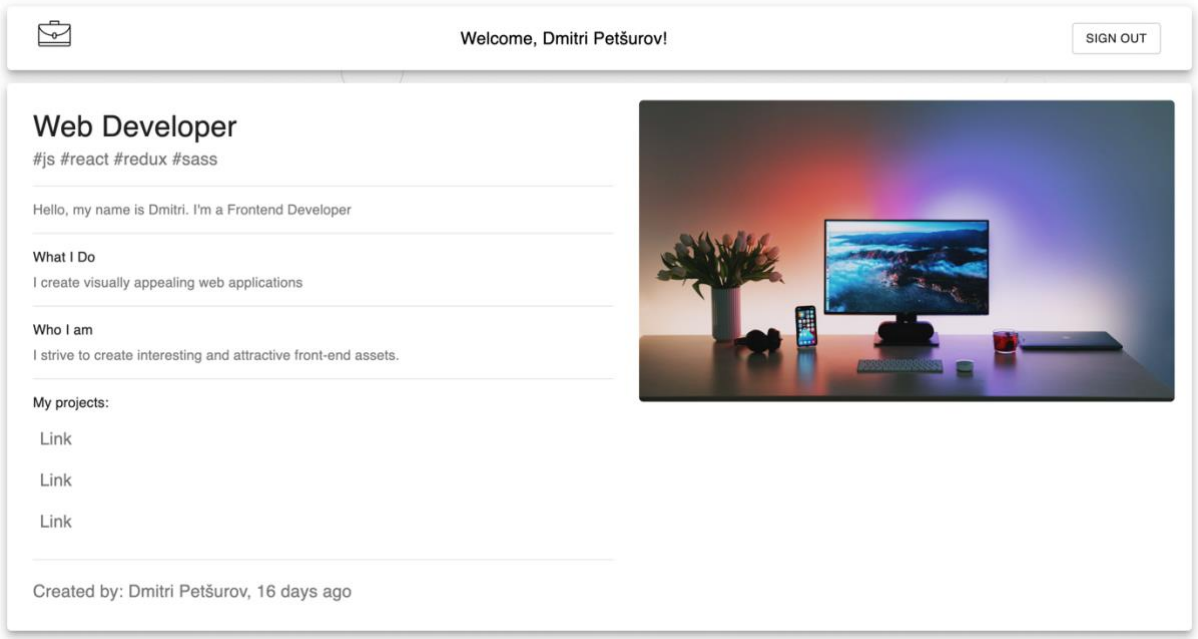

Figure 20 *View of the details component (Source: author).*

3.2 Server side

All the main functionality of the client side of the application has been described, the server side of the application has not been described. The server side of the application contains all the main application logic, all paths and all database connections. In simple terms – without the server side of the application – the author's application simply would not work, since there would be nowhere to save all the data and most of the application's functionality would not work. For a more understandable description of this side of the application, it is worth dividing it into several parts. To form the server side, 4 main folders with files are used, namely: controllers, middleware, models and routes.

3.2.1 Controllers

This folder contains all the functions for the stable operation of the application. The path to this folder is: server/controllers. There are 2 files in the controllers folder, namely: portfolios.js and user.js. The portfolios.js file contains all the basic functions for manipulating existing portfolio posts. These functions are divided into several types: getting information, creating, updating information and deleting.

The functions for getting information include functions such as getPortfolios, getPortfoliosBySearch, getPortfolio. The getPortfolios function is implemented so that the author can call it at any time and get a list of all available portfolios. The getPortfoliosBySearch function is implemented for the search component, which has already been described. As soon as the user enters a title or tag into the search and presses the search button, the getPortfoliosBySearch function is activated and returns the result of the query. The getPortfolio function is used to get information about only one portfolio post, it is used to display portfolio post information on the details page.

There is only one function for creating a post – this is createPortfolio, it is used when creating a new portfolio post in the form component. All specified data from the form component is collected and sent to the database.

Two functions can be immediately attributed to the update functions – these are updatePortfolio and likePortfolio. The updatePost function is used to edit an existing portfolio post, once new data has been entered, when the submit button on the form component is clicked, a query is sent to the database with the new data and the update process takes place. The likePortfolio function has a similar principle of updating information.

In the portfolio component, when the Like button is pressed, the likePortfolio function is activated and the server sends a request to the database to update the number of likes.

The last remaining function is the deletePortfolio function. It is activated when the delete button is pressed in the portfolio component. The function takes the id parameter from the selected post and deletes it from the database.

The user.js file contains 2 main functions for authorization: signin and signup. The signin function is activated when the user clicks on the Sign In button in the auth component. The function accepts email and password from the client side of the application and checks the entered email and password, whether such an account exists in the database, if it exists, then the user logs in successfully, if not, the system generates an error.

The signup function is activated if the user clicks on the Sign Up button in the auth component. As soon as the button has been pressed, the data from the client side is sent to the server side of the application and the verification process takes place. First, the system checks if the entered email is already in the database, if the email is not in the database, then the system checks whether the password and its confirmation match. If the passwords match, then the system encrypts the entered password and registers a new user, the user is added to the database.

3.2.2 Middleware

The folder is located at: server/middleware. It contains only one auth.js file. The main purpose of the logic of this file is to allow the server side to allow an authorized user to perform some action by validating the user's data. The auth.js file itself contains the auth function, where it first checks what type of account the authorized user has, a google account or a manually registered account. After that, the function returns the id from the account, or otherwise it throws an error. After the system has received an id from the used account, it understands that the account exists and is valid. With this information, you can set permissions on the application's actions in the server/routes/portfolios.js file. This will be discussed in more detail a little later.

3.2.3 Models

The folder location is server/models. The folder contains 2 files: portfolio.js and user.js. Both files create a schema for the database, meaning they set up fields that will be populated by the application and displayed in the database. To create a schema, the mongoose library is used,

namely the mongoose.Schema() function, an object with the fields and their types specified for the database is passed to this function.

3.2.4 Routes

The folder location is server/routes. The folder contains 2 files portfolio.js and users.js. These files contain all the routes that are used in the application and the functions assigned to them, such as getPortfolio, getPortfolios, updatePortfolio etc. As mentioned earlier, the portfolios.js file uses the auth function, which was described in the middleware section. All paths that contain the auth function can only be used by an authorized user, it was these access rights that were discussed in the middleware section.

3.2.5 Index.js

The file is in the root folder server. The index.js file contains standard methods for establishing a MongoDB database connection with the application, specifying a unique url to the author's database and the port on which the local server will be launched to develop the application.

3.3 Database architecture

The application consists of two parts – this is the web application itself and the database, which stores information about existing portfolio posts and registered users.

The database also consists of two parts – the first part contains data about created portfolios, namely: id, title, introduction, skillset, project1, project2, project3, about, name, creator, tags, selectedFile, likes and createdAt. All this data is subsequently displayed on the details page of each portfolio. Realization of database can be found on **Figure 21.**

Figure 21 *Database for portfolios (Source: author).*

The second part of the database contains all information about registered users, such as: id, name, email and password. The information is used to display data about the user, as well as to be able to restrict certain functions in use, namely, to make them available only to registered users. Realization of database can be found on **Figure 22.**

Figure 22 *Database for users (Source: author).*

3.4 Source code

Application files can be found by GitHub link: https://github.com/petsurov/portfol_io

3.5 Future plans for working on the platform

At the time of defense of the work, the application has all the basic functions from which the author can build on and expand the project. Thanks to the existing basic functionality, it is already possible to put more emphasis on the design of the application and start attracting first users in order to test the application and collect feedback.

In the future, the author plans to continue to develop the application. There are a lot of possibilities for developing the application. From the existing basic functionality, the author can move in different directions, such as improving the design, adding custom portfolio templates, integrating a portfolio rating system. Also, as soon as the application is ready, there are a lot of opportunities for earning, namely the integration of paid subscriptions that will place the owner's portfolio on the first pages, as well as the addition of new paid unique portfolio templates. The author also considers the option of reworking the idea with authorization, namely, to implement 2 types of accounts: an account for a regular user who wants to submit his portfolio for public evaluation, as well as an account for a user who wants to hire an employee based on his professional skills, which will be displayed in portfolio. Also in the plans for the future is the creation of a mobile version of the platform in order to expand the audience of the application.

SUMMARY

As a result of the thesis work, the author completed all the tasks. The authorization page has been implemented, it is possible to register a new account and use it to log in or log in through a google account. An authorized user has access rights to use all the functions of the application, namely creating a portfolio, editing and deleting it. There is also a "Like portfolio post" feature. An unauthorized user can only view created portfolios and their number of likes. Both types of users can use search by titles and tags, as well as pagination between pages. Also, each user can view the full version of the selected portfolio by going to the portfolio details page, which has its own unique address.

All functions work properly and are ready to go. The basic functionality of the application is ready for testing by other users to collect feedback and further work on the application.

KOKKUVÕTE

Lõputöö tulemusena täitis autor kõik ülesanded. Rakendatud on autoriseerimisleht, kus on võimalik registreerida uus konto ja kasutada seda sisselogimiseks või sisse logida kasutades Google kontot. Autoriseeritud kasutajal on juurdepääsuõigused kõigi rakenduse funktsioonide kasutamiseks, sealhulgas portfoolio loomiseks, täiendamiseks ning kustutamiseks. Samuti on saadaval funktsioon "Meeldib". Autoriseerimata kasutaja saab veebilehel olevaid portfooliosid sirvida ja näeb nende meeldimiste arvu. Mõlemat tüüpi kasutajad saavad kasutada otsingut pealkirja või märksõna järgi ning sirvida lehekülgede vahel. Samuti saab iga kasutaja vaadata valitud portfoolio täisversiooni, minnes portfoolio üksikasjade lehele, millel on oma kordumatu aadress.

Kõik funktsioonid töötavad korralikult ja on kasutamiseks valmis. Rakenduse põhifunktsioonid on valmis testimiseks teiste kasutajate poolt ja tagasiside kogumiseks rakendusega edasiseks tööks.

REFERENCES

Wikipedia. *Operating system*. Available at https://en.wikipedia.org/wiki/Operating_system, accessed April 8, 2022.

W3schools. *What is JSX?* Available at https://www.w3schools.com/react/react_jsx.asp#:~:text=JSX%20stands%20for%20JavaScript%20XML,and%20add%20HTML%20in%20React, accessed April 8, 2022.

Wikipedia. *CSS*. Available at https://en.wikipedia.org/wiki/CSS, accessed April 8, 2022.

Dailysmarty. *SCSS*. Available at https://www.dailysmarty.com/posts/what-is-scss, accessed April 8, 2022.

Wikipedia. *DOM*. Available at https://en.wikipedia.org/wiki/Document_Object_Model, accessed April 8, 2022.

Wikipedia. *JSON*. Available at https://en.wikipedia.org/wiki/JSON, accessed April 8, 2022.

Wikipedia. *User interface*. Available at https://en.wikipedia.org/wiki/User_interface, accessed April 8, 2022.

Wikipedia. *Hypertext Transfer Protocol*. Available at https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol, accessed April 8, 2022.

Educative. *What is MERN?* Available at https://www.educative.io/edpresso/what-is-mern-stack, accessed April 8, 2022.

Wikipedia. *API*. Available at https://en.wikipedia.org/wiki/API, accessed April 8, 2022.

Wikipedia. *USENIX*. Available at https://en.wikipedia.org/wiki/USENIX, accessed April 8, 2022.

Prettier. *What is Prettier?* Available at https://prettier.io/docs/en/index.html, accessed April 8, 2022.

Geeksforgeeks. *What is react-router-dom?* Available at https://www.geeksforgeeks.org/what-is-react-router-dom/#:~:text=React%20Router%20DOM%20is%20an,side%20routing%20library%20for%20React. , accessed April 8, 2022.

Wikipedia. *Redux (JavaScript library).* Available at https://en.wikipedia.org/wiki/Redux_(JavaScript_library), accessed April 8, 2022.

Freecodecamp. *Redux Thunk Explained with Examples.* Available at https://www.freecodecamp.org/news/redux-thunk-explained-with-examples/, accessed April 8, 2022.

LogRocket. *What is a JWT?* Available at https://blog.logrocket.com/how-to-secure-a-rest-api-using-jwt-7efd83e71432/, accessed April 8, 2022.

Axios. *Promise based HTTP client for the browser and node.js.* Available at https://axios-http.com/, accessed April 8, 2022.

Wikipedia. *Node.js.* Available at https://en.wikipedia.org/wiki/Node.js, accessed April 8, 2022.

Wikipedia. *Express.js.* Available at https://en.wikipedia.org/wiki/Express.js, accessed April 8, 2022.

Wikipedia. *Mongoose (MongoDB).* Available at https://en.wikipedia.org/wiki/Mongoose_(MongoDB), accessed April 8, 2022.

Wikipedia. *Git.* Available at https://en.wikipedia.org/wiki/Git, accessed April 8, 2022.