

Introduction

This report will contain an overview of the web application that was created by Group 5 that consists of Erik, Konrad, Mishael and Phong. The main purpose of [the application](#) is to let users create their own profile, create their own portfolio that can be seen by other users. A user would also be able to edit as well as delete their own posts. The application also provides a search bar that lets anyone search for a post by the title of it, and in addition the search feature also provides posts from [Unsplash](#).

A list of use cases

1. **Login/logout** - As a registered user, I would like to login securely, so that my information can only be accessed by me.
Acceptance criteria:
 - The user should be able to enter their username.
 - The user should be able to enter their password.
 - The user should be able to submit their username and password.
 - A message should be displayed if the login was successful or not.
 - The user should be able to logout
2. **Sign-up** - As a new user, I want to register a username and password so that the system can remember my data.
Acceptance criteria:
 - The user should be able to enter a username.
 - The user should be able to enter a password.
 - The user should be able to enter an email.
 - The user should be able to submit the username, password and email.
 - A message should be displayed if the registration was successful or not.
3. **Sort by** - As a user, I would like to be able to sort the displayed images so that I can better find an image that I'm looking for.
Acceptance criteria:
 - The user should be able to sort the images on the homepage by three different categories: recency, alphabetical order and reverse alphabetical order.
4. **Create post** - As an authorized user, I would like to create a post so that it can be viewed by other users.
Acceptance criteria:
 - The created post should contain an image, title, description and creator.
 - A message should be displayed if creating the post was successful or not.
 - A user should be able to find the created post by searching for it in the search bar.
 - The post can only be created by an authorized user.
5. **Get all posts** - As a user, I want to see what kind of images the website has to offer, so that I can find an image of my interest.
Acceptance criteria:
 - The created posts from creators should be visible on the homepage.

- Additional details of a post should show up if a user were to click on a post containing a title, description and creator.
6. **Delete post** - As an authorized user I would like to delete a post, so that it can't be viewed anymore.
- Acceptance criteria:
- A created post should be able to be deleted.
 - Only an authorized user should be able to delete their own posts.
 - A message should be displayed if the post was deleted or not.
7. **Update post** - As an authorized user I would like to update a post, so that I can make changes to an already previously made post.
- Acceptance criteria:
- An authorized user should be able to make changes to their previously created posts by changing the title or description.
 - Only an authorized user should be able to update their own posts.
 - A message should be displayed if the post was updated or not.
8. **Search posts** - As a user, I would like to be able to search for images, so that I can find an image that is of my interest.
- Acceptance criteria:
- There should be a search bar where a user can enter an input and submit it.
 - Posts should be displayed after the title when it's searched for by a user.
9. **Unsplash API** - As a user, I want the website to show images from another source if the image I'm searching for can't be found, so that I don't need to search for it somewhere else.
- Acceptance criteria:
- When a user searches for an image, results from the Unsplash API should also be displayed.
10. **Update profile picture/description** - As an authorized user, I would like to update my profile so that I can make changes to it.
- Acceptance criteria:
- Only an authorized user should be able to make changes to their profile.
 - The authorized user should be able to change the profile picture and description.

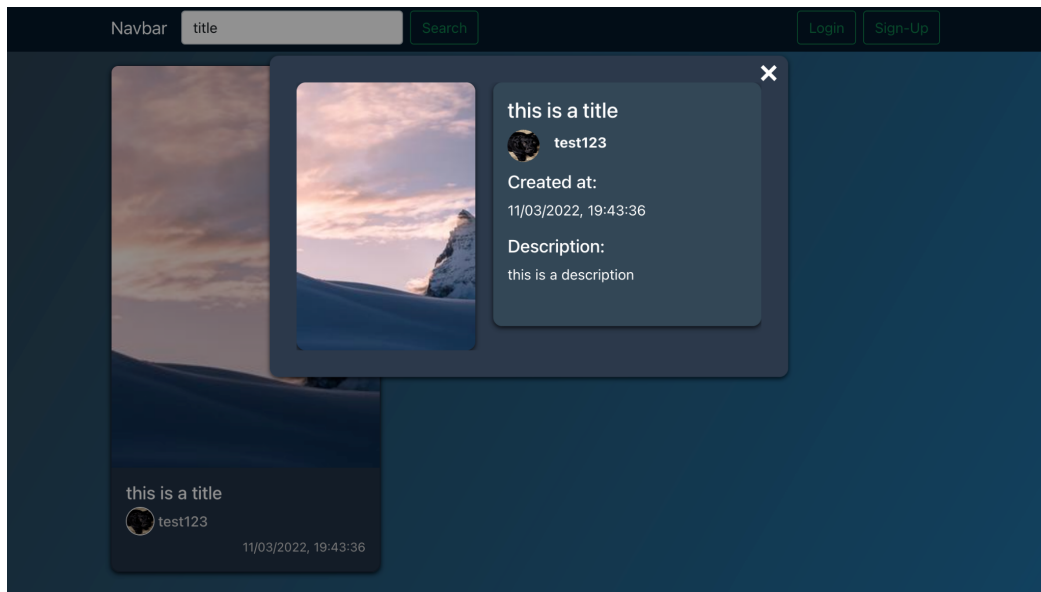
User manual

1. In the header, the user can search for images by entering input in the search bar.

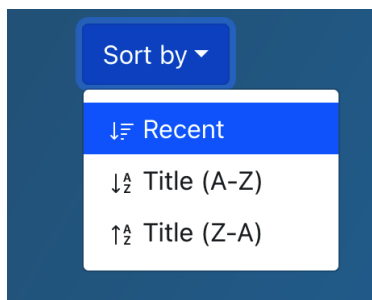


The image shows a dark blue header bar. On the left, there is a 'Navbar' label. In the center, there is a white search input field with the placeholder text 'Search'. To the right of the input field is a green 'Search' button. Further to the right are two green buttons labeled 'Login' and 'Sign-Up'.

This will display images that might be of interest to the user. A user can also press on the image to see more details about the post.



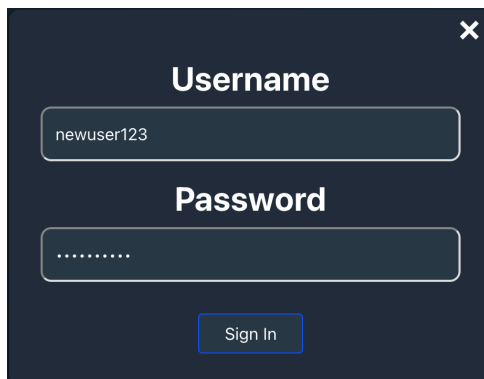
2. The displayed images can be sorted to find an image more easily. This can be done by clicking on the drop down menu with the title sort by. This will allow the user to sort the images in three different ways: Recent, Title (A-Z) and Title (Z-A).



3. The user can also create an account with the Sign-Up button located in the header. This will allow the user to enter a username, password and email. An account will be created if the input of the fields have been entered correctly and the register button is pressed.

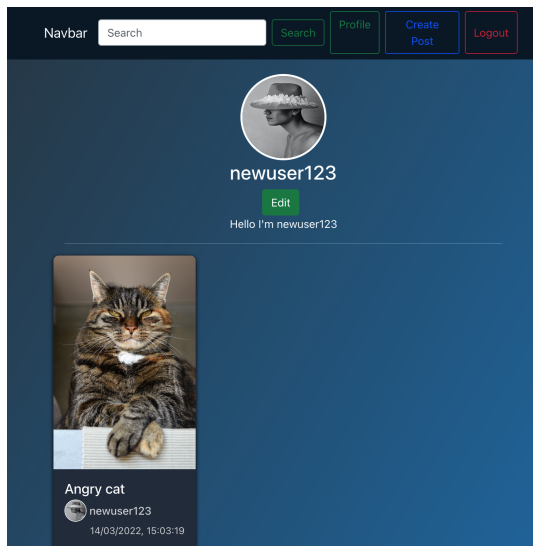
A screenshot of a user registration form. The form is titled 'Username' and has a close button (X) in the top right corner. It contains three input fields: 'Username' with the text 'newuser123', 'Password' with masked characters '.....', and 'E-mail' with the text 'newuser123@gmail.com'. Below the input fields are two buttons: 'Go back' and 'Register'.

4. After creating an account, the user can also login with their created account to enter their profile by submitting the correct input and clicking on the sign in button.

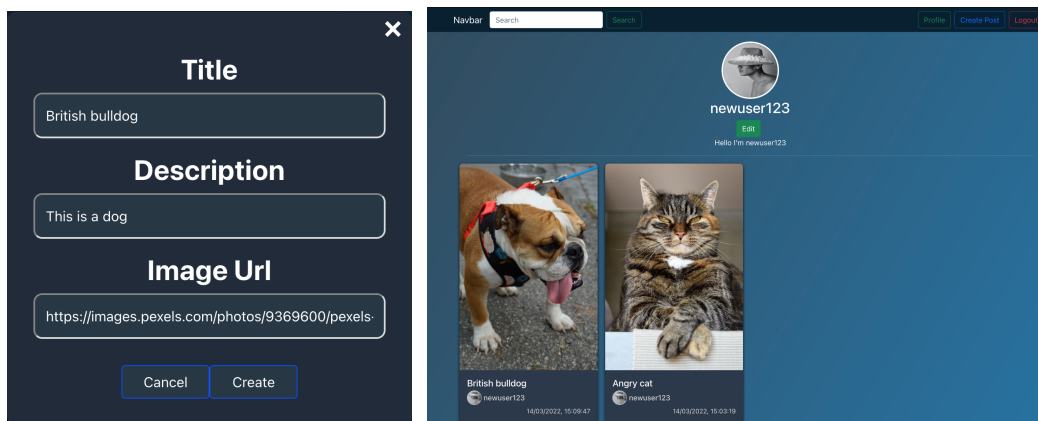


A dark-themed login modal with a close button (X) in the top right corner. It contains two input fields: 'Username' with the text 'newuser123' and 'Password' with masked characters '.....'. Below the fields is a 'Sign In' button.

5. Here users can upload a profile picture and write a description about themselves.



6. Posts can be created when a user is logged in by clicking on the create post button located in the header. This will allow the user to upload an image, enter a title and a description. The post will be submitted when the fields have been entered correctly and the submit post button is pressed.



Design

1. Security

We have outsourced the HTML escaping/ XSS injection protection to React, which means that some inputs such as Post title are not escaped and are saved as-is. And instead of saving users' passwords as plaintext, we hash them with BCrypt. Our session system is based on cookies, so we have a limited CORS policy and a strict cookie access policy to protect our users from cross-origin requests attacks.

We have validation both on the client and the server to ensure that the submitted value is valid and for user experience. For example, email validation and unique username validation.

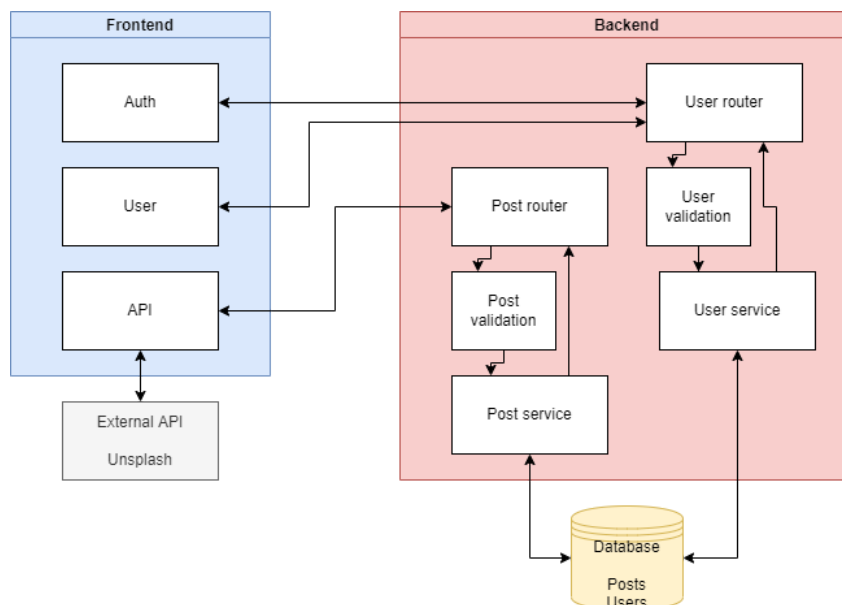
2. API specification

See appendix A

3. Library, framework and package specification

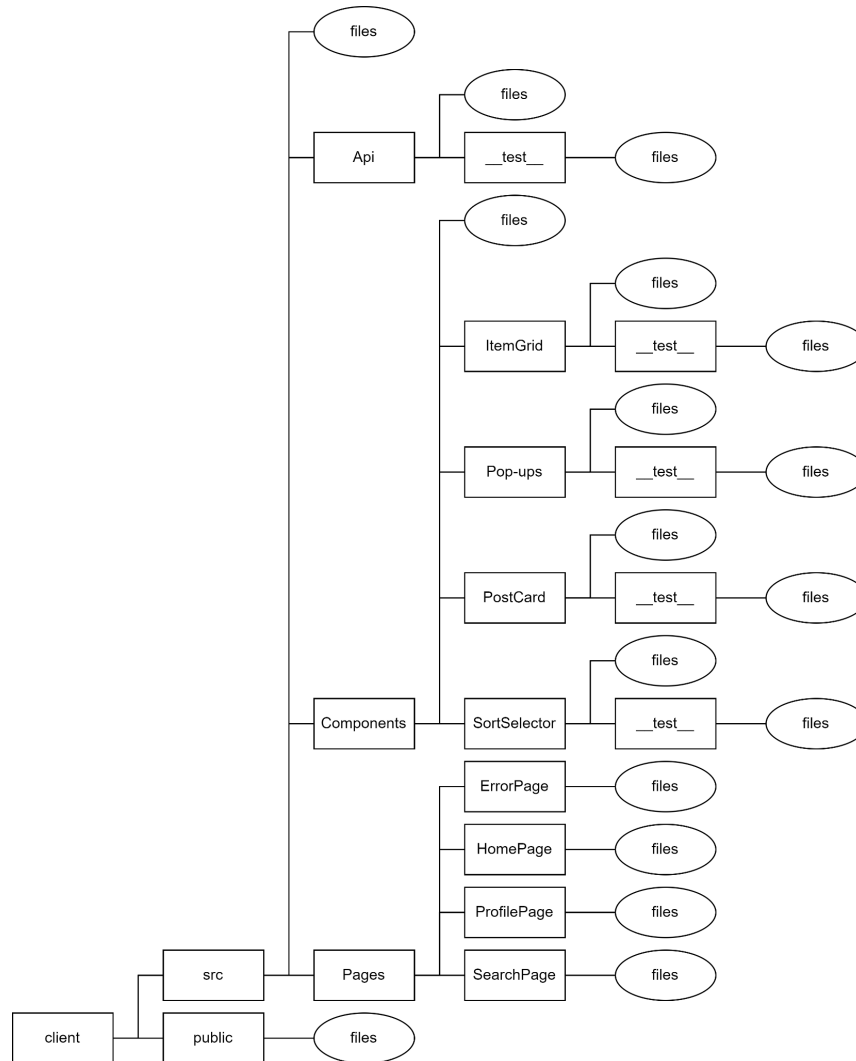
See appendix B

4. System design



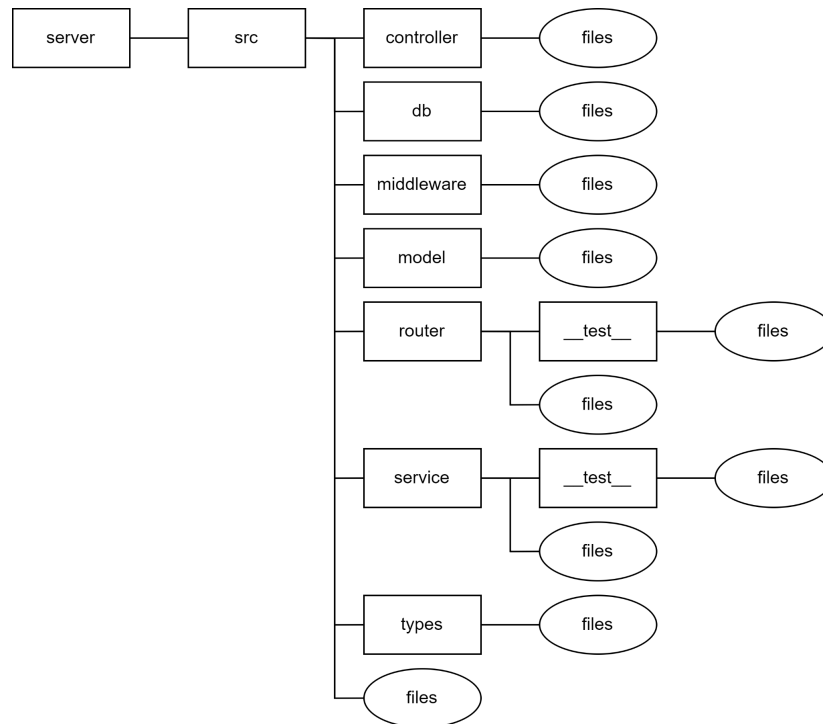
5. Folder structure

5.1. Frontend



The frontend folder structure is separated into 4 distinct categories, the public folder containing static files such as favicon, manifest file etc. The components folder which contains react components separated into subfolders based on the component(s) they contain. This includes both the .tsx files and any css file needed for the component. A similar structure is used for the pages folder where each page is in its own subfolder containing the necessary files. And lastly, the api folder contains functions used for calling the backend API. The components and api folder have subfolders called “__test__” which contain test files for the component in the parent directory.

5.2. Backend



The backend folder structure is divided into several subfolders containing related files. The controllers folder contains functions for both posts and users for performing validation. The db folder contains database model mappings and a database connection object. The middleware folder contains authentication related middleware. Model folder contains in memory format of the database models. The router folder contains the routes used in the api. The service folder contains user and post services for performing different functions on these objects. Types folder contains a type definition for the session used in the backend. Outside of the main folders but still in the src folder there are files containing the entry point to the application.

Responsibilities

1. Backend

Initially the model interfaces were a joint input from everybody and then the group divided the application where Erik was responsible for post router, Konrad for user router, Mishael for user service as well as the session feature and Phong for post service. Furthermore when the group added a database to the application Konrad setup the connection, Mishael implemented user service connected to the database and Phong implemented post service connected to the database. The group also made tests for their respective parts of the code.

2. **Frontend**

The group divided the frontend where Erik implemented the navigation bar and the create post popup. Konrad implemented the post card component, item grid, home page, search page and bar together with the unsplash feature. Mishael setup the login, logout, signup logic and the profile page. Phong implemented the remaining of the pop up components in the pop up folder.

A. API Specification

Posts

Method	Endpoint
GET	/post/

Query

Param	Description	Type	Values
order	Post order. (Optional, default: "recent-descending")	string	"recent-descending", "title-ascending", "title-descending"

Response

HTTP Code	Body
200 OK	[{ "id": 1646572706328, "title": "Title", "description": "Description", "imageUrl": " http://test.test/imageUrl ", "creator": 1, "_id": "6224b4a297335037ab69c182", "createdAt": "2022-03-06T13:18:26.333Z", "modifiedAt": "2022-03-06T13:18:26.333Z", "__v": 0 }]
500 Internal Server Error	{ "status": "Server error", "reason": "<Error message>" }

Method	Endpoint
POST	/post/

Body

Param	Description	Type
title	Post title.	string
description	Post description.	string
imageUrl	Url to post image.	string
creator	Creator ID.	number

Response

HTTP Code	Body
200 OK	<pre>{ "id": 1646572706328, "title": "Title", "description": "Description", "imageUrl": "http://test.test/imageUrl", "creator": 1, "_id": "6224b4a297335037ab69c182", "createdAt": "2022-03-06T13:18:26.333Z", "modifiedAt": "2022-03-06T13:18:26.333Z", "__v": 0 }</pre>
400 Bad Request	<pre>{ "status": "Could not create post", "reason": "<Error message>" }</pre>
500 Internal Server Error	<pre>{ "status": "Server error", "reason": "<Error message>" }</pre>

Method	Endpoint
GET	/post/search

Query

Param	Description	Type
search	String to search.	string

Response

HTTP Code	Body
200 OK	[{ "id": 1646572706328, "title": "Title", "description": "Description", "imageUrl": " http://test.test/imageUrl ", "creator": 1, "_id": "6224b4a297335037ab69c182", "createdAt": "2022-03-06T13:18:26.333Z", "modifiedAt": "2022-03-06T13:18:26.333Z", "__v": 0 }]
400 Bad Request	{ "status": "Error searching for posts", "reason": "<Error message>" }
500 Internal Server Error	{ "status": "Server error", "reason": "<Error message>" }

Method	Endpoint
PUT	/post/:id

Url param

Param	Description	Type
id	Post id.	number

Body

Param	Description	Type
title	Post title. (Optional)	string
description	Post description. (Optional)	string
imageUrl	Url to post image. (Optional)	string

Response

HTTP Code	Body
200 OK	{ "status": "Post updated" }
400 Bad Request	{ "status": "Could not update post", "reason": "<Error message>" }
500 Internal Server Error	{ "status": "Server error", "reason": "<Error message>" }

Method	Endpoint
GET	/post/:id

Url param

Param	Description	Type
id	Post id.	number

Body

Param	Description	Type
verifyCreator	Post creator ID.	number

Response

HTTP Code	Body
200 OK	{ "status": "Post deleted" }
400 Bad Request	{ "status": "Could not delete post", "reason": "<Error message>" }
500 Internal Server Error	{ "status": "Server error", "reason": "<Error message>" }

Method	Endpoint
DELETE	/post/:id

Url param

Param	Description	Type
id	Post id.	number

Body

Param	Description	Type
verifyCreator	Post creator ID.	number

Response

HTTP Code	Body
200 OK	{ "status": "Post deleted" }
400 Bad Request	{ "status": "Post not found", "reason": "<Error message>" }
500 Internal Server Error	{ "status": "Server error", "reason": "<Error message>" }

Users

Method	Endpoint
POST	/user/login

Body

Param	Description	Type
username	Username.	string
password	Password.	string

Response

HTTP Code	Body
200 OK	{ "status": "Authorized" }
401 Unauthorized	{ "status": "Unauthorized", "reason": "Invalid Credentials" }
403 Forbidden	{ "status": "Forbidden", "reason": "Already logged in" }

Method	Endpoint
POST	/user/sign-up

Body

Param	Description	Type
username	Username.	string
password	Password.	string

Response

HTTP Code	Body
201 Created	{ "status": "User created" }
400 Bad Request	{ "status": "User could not be created", "reason": "<Error message>" }
403 Forbidden	{ "status": "Forbidden", "reason": "Already logged in" }

Method	Endpoint
POST	/user/logout

Response

HTTP Code	Body
200 OK	{ "status": "User logged out" }
403 Forbidden	{ "status": "Forbidden", "reason": "Not logged in" }

Method	Endpoint
GET	/user/:id

Url param

Param	Description	Type
id	User id.	number

Response

HTTP Code	Body
200 OK	<pre>{ "user": { "id": 1, "username": "Username", "profileImageUrl": "http://test.test/imageUrl", "description": "Description", "createdAt": "2022-03-06T12:40:14.765Z" }, "posts": [{ "id": 1646572706328, "title": "Title", "description": "Description", "imageUrl": "http://test.test/imageUrl", "creator": 1, "_id": "6224b4a297335037ab69c182", "createdAt": "2022-03-06T13:18:26.333Z", "modifiedAt": "2022-03-06T13:18:26.333Z", "__v": 0 }] }</pre>
404 Not Found	<pre>{ "status": "Invalid user" }</pre>

Method	Endpoint
GET	/user/update

Body

Param	Description	Type
email	Email to set. (Optional)	string
description	Description to set. (Optional)	string
profileImageUrl	Url to profile image. (Optional)	string

Response

HTTP Code	Body
200 OK	{ "status": "User updated" }
400 Bad Request	{ "status": "User could not be updated" }
403 Forbidden	{ "status": "Forbidden", "reason": "Not logged in" }
500 Internal Server Error	{ "error": "<Error message>" }

Method	Endpoint
PUT	/user/update/password

Body

Param	Description	Type
password	Password to set. (Optional)	string

Response

HTTP Code	Body
200 OK	{ "status": "Password updated" }
400 Bad Request	{ "status": "Password could not be updated" }
403 Forbidden	{ "status": "Forbidden", "reason": "Not logged in" }

Method	Endpoint
GET	/user/session

Response

HTTP Code	Body
200 OK	<pre>{ "id": 1, "username": "Username", "email": "Email@test.com", "profileImageUrl": "http://test.test/imageUri", "description": "Description", "createdAt": "2022-03-06T12:40:14.765Z" }</pre>
403 Forbidden	<pre>{ "status": "Forbidden", "reason": "Not logged in" }</pre>

B. Library and package

- The external API used for this project was Unsplash which provides the web browser with high quality images that are free to use.
source: <https://unsplash.com/developers>
- "@testing-library/jest-dom": "^5.16.2" - A library that provides a set of custom jest matchers which makes tests more declarative, clear to read and maintain.
source: <https://github.com/testing-library/jest-dom>
- "@testing-library/react": "^12.1.2" - A library for testing React components with utility functions on top of react-dom.
source: <https://testing-library.com/docs/react-testing-library/intro/>
- "@testing-library/user-event": "^13.5.0" - A library which aids testing by simulating user interaction on a browser.
source: <https://testing-library.com/docs/user-event/intro/>
- "axios": "^0.26.0" - A JavaScript library that makes HTTP requests for node.js and the browser.
source: <https://axios-http.com/docs/intro>
- "bootstrap": "^5.1.3" - A CSS Framework on the frontend which features numerous HTML and CSS templates.
source: https://www.w3schools.com/whatis/whatis_bootstrap.asp
- "react": "^17.0.2" - A JavaScript library used for creating user interfaces on single page applications while also handling the view layer for both web browsers and mobile applications.
source: <https://reactjs.org/>
- "react-bootstrap": "^2.1.2" - A component based library that provides native Bootstrap components as pure React components.
source: <https://react-bootstrap.github.io/>
- "react-dom": "^17.0.2" - A package that enables access and modifications to the DOM in a more efficient way as opposed to directly manipulating the DOM elements.
source: <https://www.geeksforgeeks.org/reactjs-reactdom/>
- "react-icons": "^4.3.1" - A package that adds icons to the React project.
source: <https://www.freecodecamp.org/news/how-to-use-react-icons/>
- "react-masonry-css": "^1.0.16" - An npm package that allows a Masonry layout which uses React's Virtual DOM.
source: <https://www.npmjs.com/package/react-masonry-css>
- "react-router-dom": "^6.2.1" - A fully featured client and server-side routing library for React which enables the user to implement dynamic routing in a web app. The npm package contains many useful components which makes both the user experience and performance better.
source: <https://www.geeksforgeeks.org/what-is-react-router-dom/>
- "react-scripts": "5.0.0" - This includes a set of scripts when creating a react app which handles the configuration of the project automatically by running the scripts.
source: <https://www.freecodecamp.org/news/create-react-app-npm-scripts-explained/>

- "typescript": "^4.5.5" - This enables the developer to add additional syntax to JavaScript which makes the code easier to read and debug.
source: <https://www.typescriptlang.org/>
- "web-vitals": "^2.1.4" - This is a utility included when creating a React app which allows the developer to measure and analyze the performance of the app.
source: <https://create-react-app.dev/docs/measuring-performance/>
- "jest": "^27.4.7" - Jest is a JavaScript Testing Framework for running and structuring tests.
source: <https://jestjs.io/>
- "supertest": "^6.2.2" A library built on Node.js and HTTP which helps developers test APIs.
source: <https://www.npmjs.com/package/supertest>
- "ts-jest": "^27.1.3" - This allows developers to perform Jest tests on projects written in TypeScript.
source: <https://kulshekhar.github.io/ts-jest/docs/>
- "ts-node-dev": "^1.1.8" This is a development tool which automatically restarts the node process when a required file is changed but shares Typescript compilation process between restarts.
source: <https://www.npmjs.com/package/ts-node-dev>
- "bcrypt": "^5.0.1" - A library which helps the user hash passwords.
source: <https://www.npmjs.com/package/bcrypt>
- "cors": "^2.8.5" - A node.js package that provides Connect/Express middleware which allows cross-origin resource sharing (CORS).
source: <https://www.npmjs.com/package/cors>
- "dotenv": "^16.0.0" - This is a module that loads environment variables from a .env file into a process.env object.
source: <https://www.npmjs.com/package/dotenv>
- "express": "^4.17.2" - A framework used for designing and building web applications. Express is part of the MEAN stack and handles the backend part with routing, sessions, HTTP, requests, error handling, etc.
source: <https://expressjs.com/>
- "express-session": "^1.17.2" - This is a framework for the HTTP server-side used to create and manage a session middleware.
source: <https://www.section.io/engineering-education/session-management-in-nodejs-using-expressjs-and-express-session/>
- "mongodb": "^4.4.0" - A document database part of the MEAN stack used to build highly available and scalable internet applications.
source: <https://www.mongodb.com/why-use-mongodb>
- "mongoose": "^6.2.3" - This is a Node.js based Object Data Modeling (ODM) library for MongoDB which allows the developers to model their data.
source: <https://www.mongodb.com/developer/article/mongoose-versus-nodejs-driver/>