WEB701 Assessment One
Milestone Two

Xander Crocker

Table of Contents

1. Front-End Frameworks

GitHub Project link: https://github.com/users/Xander-Crocker/projects/3

1.1 Angular

1.1.1 Purpose:

Angular is an open-source JavaScript framework that is used in front-end web development. The Angular framework is built on TypeScript and uses a component-based system to incorporate dependency injection. This is achieved by using a collection of integrated libraries that offer a variety of features such as client-to-server communication, front-end routing, form management and more (Angular, n.d.). The Angular format provides developers with a consistent structure that lets them reliably format their front-end components. This minimises the need to constantly rebuild code from scratch ultimately saving development time. Angular achieves this by being a model-view-controller (MVC) framework, providing bi-directional dataflow while maintaining a fundamental and structured document object model (DOM) (Deshpande, 2024).

1.1.2 Features:

Components:

Angular offers developers enterprise web app templates and reusable components that cut down on the amount of code needing to be written which allows for a cleaner codebase. Multiple pages can be created separately and then combined as components to produce a final product. An example of this would be combining a navigation bar as one component at the top of the web page and the home page underneath. One component does not affect the other when edited (Vishal-Siddhapra, 2024).

Testing:

Unit testing works well in Angular using the Jasmin testing framework that is built in and has a plethora of testing tools at the developer's disposal. The creators of Angular built the framework with testability in mind allowing developers to test the whole web solution. A greatly beneficial feature of Angular's many testing tools allows developers to inject mock data into controllers to see the behaviour and results without having to affect any existing data.

Dependency Injection:

Dependency injection allows developers to maintain the data between the model view. This keeps the component classes separated and efficient by not fetching data from a server to validate things such as user inputs, instead using services to execute specific tasks (Deshpande, 2024).

1.2 React

1.2.1 Purpose:

React is a JavaScript-based framework used to create interactive user interfaces (UI's) that efficiently update and render as components are edited or data changes. React uses JavaScript syntax extension (JSX) that allows developers to write both JavaScript and HTML code in the same file minimalizing the about of files in the codebase. React uses encapsulated components to create complex user interfaces (UI's) that manage their own logic. Component logic allows data to pass easily through the whole web solution to maintain its state outside the document object model (DOM) by using logic written in JavaScript instead of creating dedicated templates (React – a JavaScript Library for Building User Interfaces, n.d). React makes it simple and easy to create dynamic solutions with less coding required compared to using JavaScript alone which can get overwhelmingly complex extremely quickly. React achieves this by reusing components as blocks that are puzzled together. Though a single application often contains multiple components the components and their logic can be reused several times across the whole application (Deshpande, 2024a). For this reason, it is easier to create larger web application user interfaces with less clutter in the codebase and is natively responsive when developing for use on mobile devices.

1.2.2 Features:

User Interfaces and JSX:

React can create highly complex and interactive user interfaces (UI's) with ease while maintaining performance. Because the React framework uses JavaScript syntax extension (JSX) the concerns are separated and loosely coupled keeping the markup code and logic together in the same file. This is what makes up a React component and ultimately makes the codebase easier to manage and maintain by enhancing the readability of the code (Deshpande, 2024a).

One-way Data Binding:

In React data flows in one direction, one-way data binding is the process of connecting a view, or user interface (UI) with data that is displayed to the user. This allows a component to be rendered for the user to see in the user interface (UI) but the component's logic, contained in the same file, that contains the data will be shown to the

user through the user interface (UI). This connection is called data binding when React. Data can only flow down from parent to child, however, though the child cannot send data back to the parent it can communicate with its parent to modify the parent's logic depending on what needs to be displayed to the user (GeeksforGeeks, 2024).

Component-Based Structure:

React is made up of the logic of components. Components allow developers to separate the user interface (UI) into multiple reusable sections that are independent of each other. They act like puzzle pieces that are able to fit together seamlessly. Even when a solution becomes complex, any component configuration can be assembled together. This creates a streamlined process for a developer to adjust the user interface to any configuration as necessary ultimately making the codebase easier to read, more maintainable, and more reusable (Fulness, 2023).

2. Emerging Web Technologies

2.1 AI Code Generators

For assessment three I will be using MERN (MongoDB, Express, React, and Node) to create a full-stack website. To incorporate this emerging technology, I will be using Bolt. Bolt is a sandbox AI code generator created by Stack Blitz where code can be reviewed as the AI generates it. The AI can then be prompted to make changes as the user sees fit. Stack Blitz also includes an option to export the code once generated to their online integrated development environment (IDE) for further customization. Another brilliant feature of Bolt is that you can deploy your newly generated website with a click of a button.

After some research, I have decided I will be using Bolt AI to generate a full-stack boilerplate based on the requirements given by the assessment guidelines. I will prompt Bolt AI with the assessment requirements, things such as, must use the MERN stack, must have a login and registration, and so on. From what Bolt generates me I will download the result and import it into Visual Studio Code (IDE) and make any changes if necessary to complete the website.

Description:

AI code generators use machine learning to generate code for a user based on inputted prompts given by the user. There are many levels of AI code generation. On the one hand, in the development stage AI code generators can be used to optimize any existing code. On the other, AI code generators can be used to create an advanced boilerplate for developers to do as they wish. Though AI-generated code is often not a perfect way of programming a complete solution, it is however an efficient tool for creating a great starting point. Ultimately with the use of AI developers save time using code suggestions instead of searching for a solution on the internet or generating a boilerplate for a new solution (GitHub, 2024).

Explanation:

Seeing the emerging influence that AI code generators have had over the last couple of years I wanted to see what was possible with the use of modern AI code generators. My interest peaked in class discussing what was now possible with the use of emerging AI code generation tools. From there I started researching what the best AI tools were for programming. Though there are now many to choose from I discovered there is no be all end all solution. Some AI code generation tools are better than others based on what you want to achieve. Some create the codebase and give it to you in a downloadable zip file and others just provide you with the code for each file and it must be copied from the AI generator to your IDE after creating each file individually yourself. I also found that if you want the full functionality of most AI code generators you will need to pay a subscription fee. Though there are some free AI code generators out there, they often either don’t give you the full functionality of what they offer, or they only give you a limited number of prompts before you will have to pay the subscription for more. I

chose to use AI-generating tools as my emerging technology to see not only how they may impact the information technology industry in the future, but also how fast they are progressing. They seem to be at a point where they are extremely useful but not sufficient enough to create an industry-standard website. I can see that in the future they will be a normal part of the industry, my question is how reliant the industry will become because of it.

Potential Impact:

At this point, AI still has its limitations but still has a prominent stance that is and will continue to impact the information technology industry. One of the most asked questions is “Will AI replace programmers”. This is a difficult-to-answer question. In my opinion, no, I don’t think AI will replace programmers, at least not in the near future. I believe that it will have an astonishing impact on programmers. Programmer’s tasks will become easier to manage and processes in the development and maintenance stages will become streamlined leading to solutions being produced faster and faster as the AI technology becomes better over time.

Though there are several benefits that AI technology brings to the table there are also several concerns. One of the most influential concerns is in education. Programmers will consistently be using AI to help them when learning how to program. Is this a good thing or a bad thing? On the one hand, it is a good thing because it will allow programmers to learn faster and more efficiently. However, on the other hand, the use of AI could lead to bad practices and students not having enough knowledge to create a solution without the use of AI.

For the industry as a whole AI tools are an optimistic topic for programmers that will continue to evolve as the technology improves and be used by programmers in numerous ways for numerous reasons. No matter what side of the fence you are on when it comes to AI, AI tools are now a prominent part of the industry and will not be going anywhere anytime soon.

3. Evaluation & Comparison

Part A

3.1 Angular Screenshots

Login:

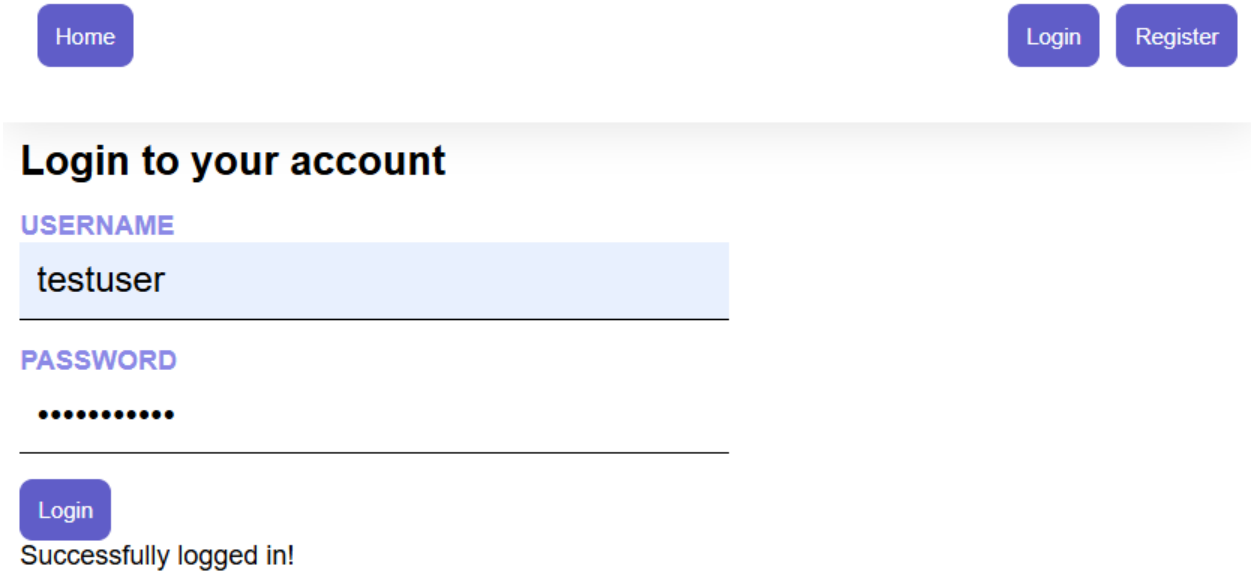
Login Component code (Imports, Component Template)

```
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';
import { FormControl, FormGroup, ReactiveFormsModule } from '@angular/forms';
import { RouterModule } from '@angular/router';
import { Router } from '@angular/router';
import axios from 'axios';
import { jwtDecode } from 'jwt-decode';

const API_URL = "http://localhost:8081/api/auth/";

You, 2 minutes ago | 1 author (You)
@Component({
  selector: 'app-login',
  standalone: true,
  imports: [CommonModule, ReactiveFormsModule, RouterModule],
  template: `
    <article>
      <section class="listing-apply">
        <h2 class="section-heading">Login to your account</h2>
        <form [formGroup]="loginForm" (submit)="submitApplication()">
          <label id="username">Username</label>
          <input id="username" type="username" formControlName="username">
          <label id="password">Password</label>
          <input id="password" type="password" formControlName="password">
          <button type="submit" class="primary">Login</button>
        </form>
        <div *ngIf="loginSuccess" class="success-message">Successfully logged in!</div>
        <div *ngIf="loginError" class="error-message">{{ loginError }}</div>
      </section>
    </article>
  `,
  styleUrls: ['./login.component.css']
})
```

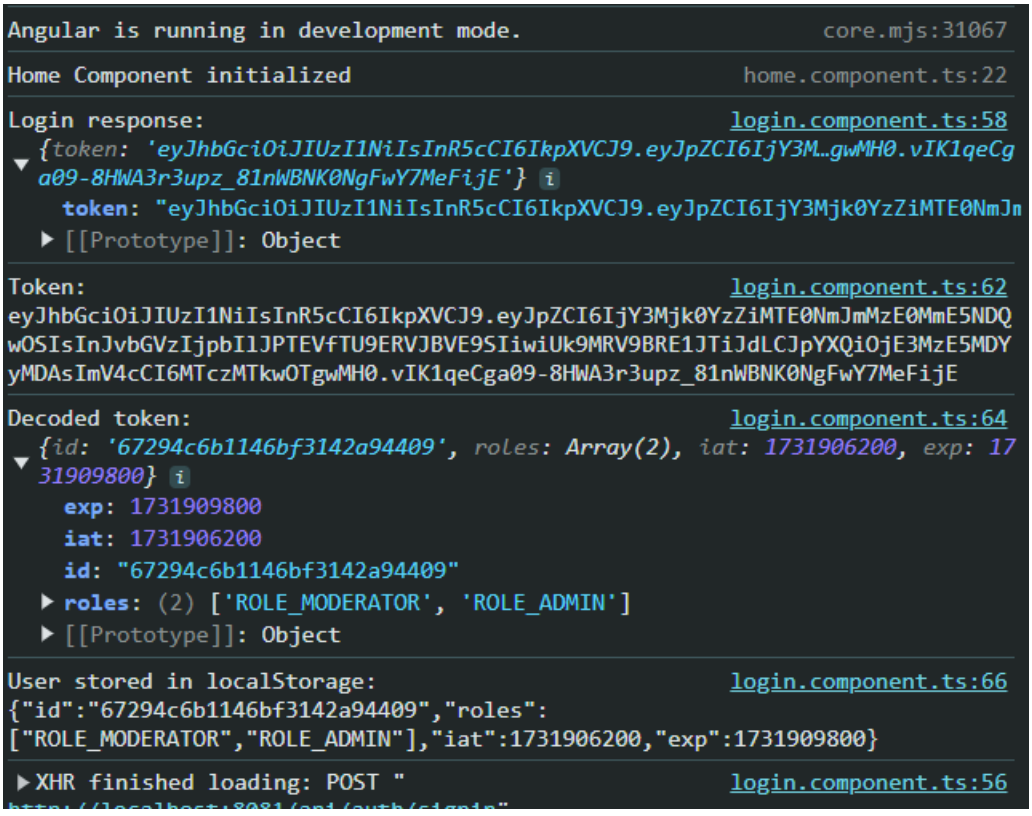
Continued Login Component code (Component logic, Login submit and post request)

```
export class LoginComponent {
  // Define the login form with username and password fields
  loginForm = new FormGroup({
    username: new FormControl(''),
    password: new FormControl('')
  });

  // Variables to store the login success status and error message
  loginSuccess: boolean = false;
  loginError: string | null = null;

  // Inject the Router service for navigation
  constructor(private router: Router) {}

  // Method to handle form submission
  submitApplication() {
    // Extract form data
    const loginData = {
      username: this.loginForm.value.username ?? '',
      password: this.loginForm.value.password ?? ''
    };

    // Send a POST request to the login API
    axios.post(API_URL + 'signin', loginData)
      .then(response => {
        console.log('Login response:', response.data);

        // If a token is received, decode it and store the user information in localStorage
        if (response.data.token) {
          console.log('Token:', response.data.token);
          const decodedToken = jwtDecode(response.data.token);
          console.log('Decoded token:', decodedToken);
          localStorage.setItem('user', JSON.stringify(decodedToken));
          console.log('User stored in localStorage:', localStorage.getItem('user'));
          this.loginSuccess = true;
          this.loginError = null;
          // Navigate to home or another component if needed
          // this.router.navigate(['/home']);
        }
      })
      .catch(error => {
        // Handle errors and display an error message
        this.loginSuccess = false;
        this.loginError = error.response?.data?.message || 'Login failed';
        console.error('There was an error!', error);
      });
```

Login Component Screen



Login Component console response when the user logged in successfully.



Register:

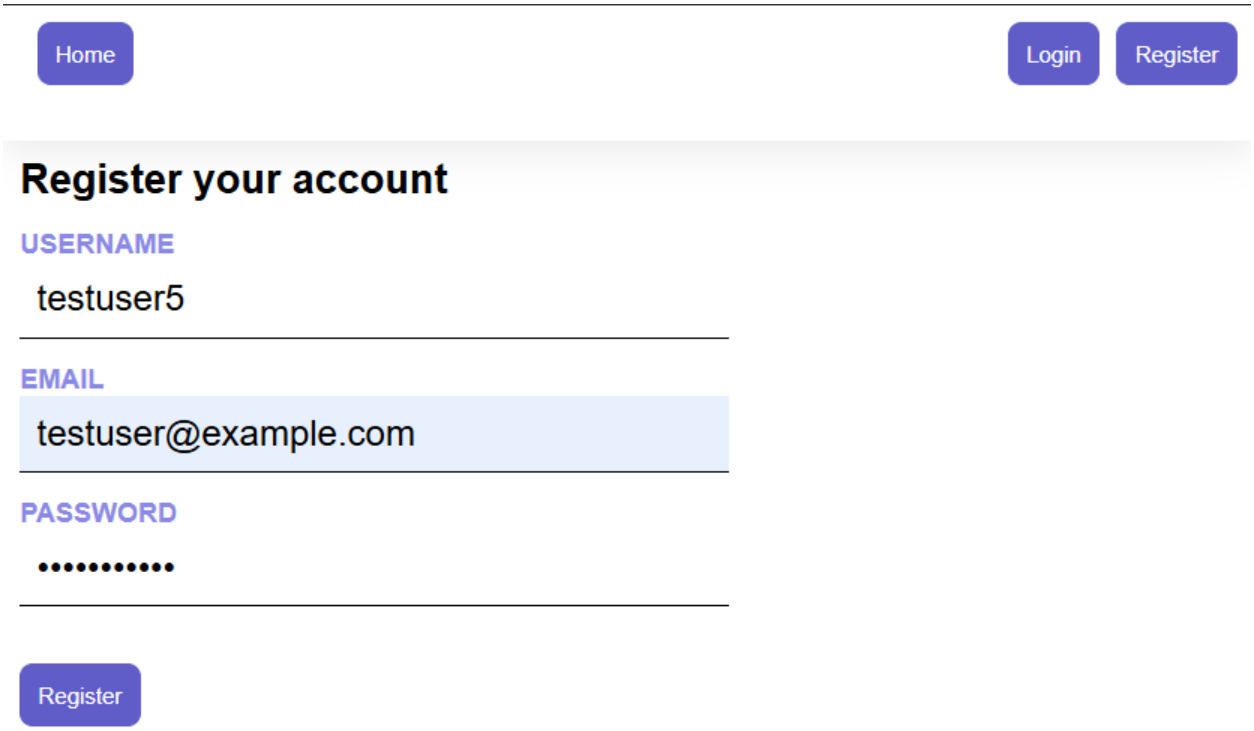
Register Component code (Imports, Component Template)

```
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';
import { FormControl, FormGroup, ReactiveFormsModule } from '@angular/forms';
import { Router } from '@angular/router';
import axios from 'axios';

const API_URL = "http://localhost:8081/api/auth/";

You, 4 seconds ago | 1 author (You)
@Component({
  selector: 'app-register',
  standalone: true,
  imports: [CommonModule, ReactiveFormsModule],
  template: `
    <article>
      <section class="listing-apply">
        <h2 class="section-heading">Register your account</h2>
        <form [formGroup]="applyForm" (submit)="submitApplication()">
          <label id="username">Username</label>
          <input id="username" type="username" formControlName="username">

          <label id="email">Email</label>
          <input id="email" type="email" formControlName="email">

          <label id="password">Password</label>
          <input id="password" type="password" formControlName="password">

          <br>
          <button type="submit" class="primary">Register</button>
        </form>
        <div *ngIf="successMessage" class="success-message">{{ successMessage }}</div>
        <div *ngIf="errorMessage" class="error-message">{{ errorMessage }}</div>
      </section>
    </article>
  `,
  styleUrls: ['./register.component.css']
})
```

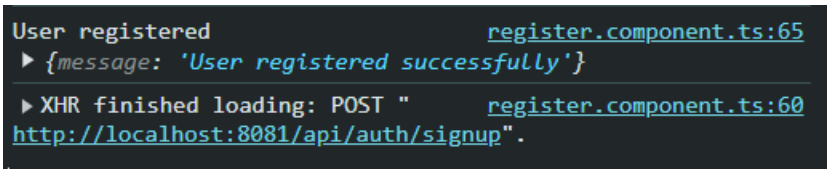
Continued Register Component code (Component logic, Registration submit and post request)

```
export class RegisterComponent {
  // Define the registration form with username, email, password, and isAdmin field
  applyForm = new FormGroup({
    username: new FormControl(''),
    email: new FormControl(''),
    password: new FormControl(''),
    isAdmin: new FormControl(false),
  });

  // Variables to store the success and error messages
  successMessage: string | null = null;
  errorMessage: string | null = null;

  // Inject the Router service for navigation
  constructor(private router: Router) {}

  // Method to handle form submission
  submitApplication() {
    // Extract form data
    const formData = this.applyForm.value;

    // Send a POST request to the registration API
    axios.post(API_URL + 'signup', formData)
      // Handle successful registration
      .then(response => {
        this.successMessage = 'User registered successfully';
        this.errorMessage = null;
        console.log('User registered', response.data);
        // Navigate to the login page
        this.router.navigate(['/login']);
      })
      .catch(error => {
        // Handle errors and display an error message
        this.errorMessage = error.response?.data?.message || 'Registration error';
        this.successMessage = null;
        console.error('Error:', error);
      });
  }
}
```

Register Component Screen

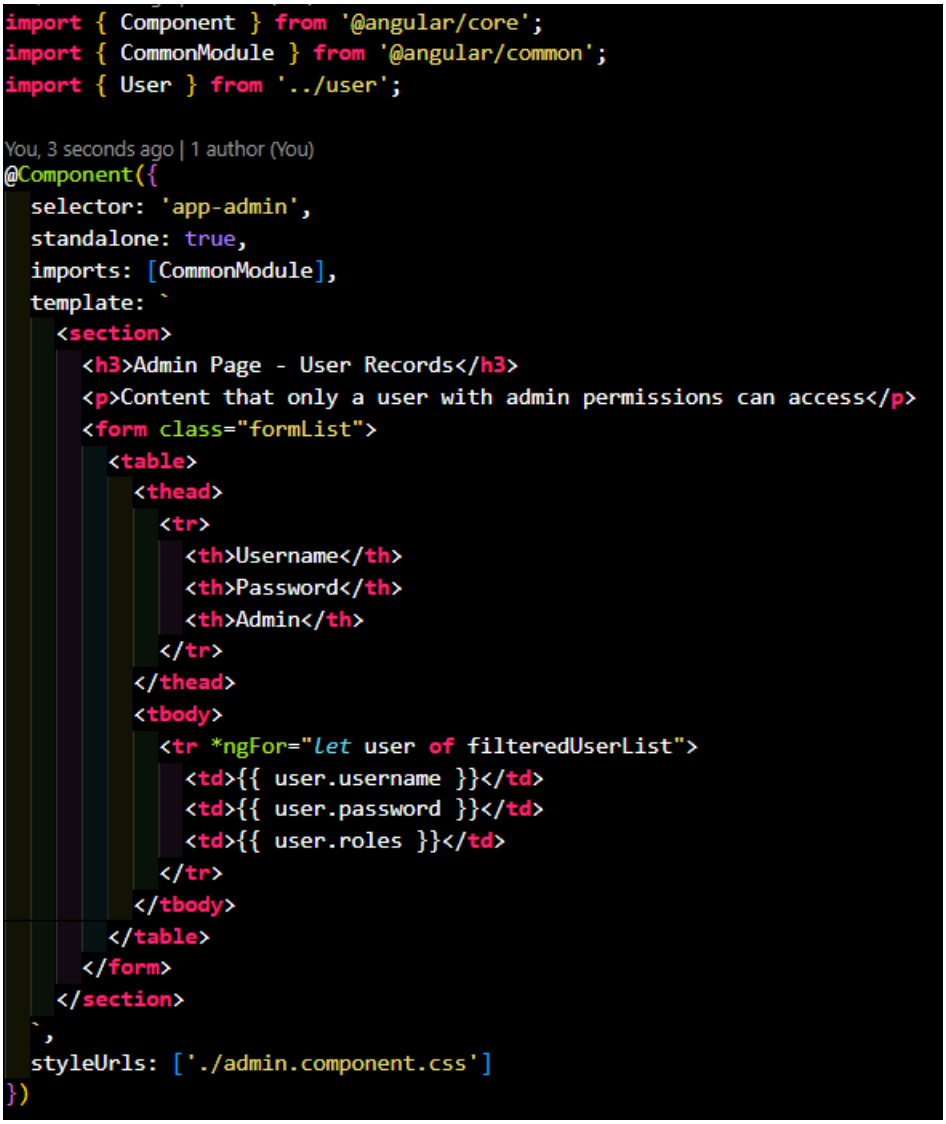


Register Component console response when user logged in successfully.



Admin:

Admin Component code (Imports, Component Template)



```
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';
import { User } from '../user';

You, 3 seconds ago | 1 author (You)
@Component({
  selector: 'app-admin',
  standalone: true,
  imports: [CommonModule],
  template: `
    <section>
      <h3>Admin Page - User Records</h3>
      <p>Content that only a user with admin permissions can access</p>
      <form class="formList">
        <table>
          <thead>
            <tr>
              <th>Username</th>
              <th>Password</th>
              <th>Admin</th>
            </tr>
          </thead>
          <tbody>
            <tr *ngFor="let user of filteredUserList">
              <td>{{ user.username }}</td>
              <td>{{ user.password }}</td>
              <td>{{ user.roles }}</td>
            </tr>
          </tbody>
        </table>
      </form>
    </section>
  `,
  styleUrls: ['./admin.component.css']
})
```

Continued Admin Component code (Fetch request)

```
export class AdminComponent {
  // List of all users
  userList: User[] = [];
  // List of users to be displayed, can be filtered
  filteredUserList: User[] = [];

  // Lifecycle hook that is called after data-bound properties are initialized
  ngOnInit() {
    this.fetchUsers();
  }

  // Method to fetch users from the server
  async fetchUsers() {
    try {
      const response = await fetch('http://localhost:5050/api/user/all/');
      if (!response.ok) {
        throw new Error('Network response was not ok');
      }
      const userList: User[] = await response.json();
      this.userList = userList;
      this.filteredUserList = userList;
    } catch (error) {
      console.error('There was a problem with the fetch operation:', error);
    }
  }
}
```

Routes:

Front-end Page Routes

```
import { Routes } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { RegisterComponent } from './register/register.component';
import { LoginComponent } from './login/login.component';
import { AdminComponent } from './admin/admin.component';

// Define the route configuration for the application
const routeConfig: Routes = [
    {
        path: '',
        component: HomeComponent,
        title: 'Home Page'
    },
    {
        path: 'register',
        component: RegisterComponent,
        title: 'Registration Page'
    },
    {
        path: 'login',
        component: LoginComponent,
        title: 'Login Page'
    },
    {
        path: 'admin',
        component: AdminComponent,
        title: 'Admin Page'
    }
];

// Export the route configuration
export default routeConfig;
```

User Schema:

User Schema for MongoDB

```
import mongoose, { Document } from 'mongoose';

// Define the User interface for TypeScript type checking
You, 3 days ago | 1 author (You)
export interface User {
    id: number;
    username: string;
    email: string;
    password: string;
    roles: string[];
    token: string;
}

// Define the IUser interface extending mongoose Document for MongoDB schema
You, 2 months ago | 1 author (You)
interface IUser extends Document {
    id: number;
    username: string;
    password: string;
    admin: boolean;
}

// Define the User schema for MongoDB
const UserSchema = new mongoose.Schema({
    username: { type: String, required: true, unique: true }, // Username field, required and unique
    email: { type: String, required: true, unique: true }, // Email field, required and unique
    password: { type: String, required: true }, // Password field, required
    roles: [{ type: String }], // Roles field, array of strings
    token: { type: String, default: '' } // Token field, default value is an empty string
});

// Create the User model from the schema
const User = mongoose.model<IUser>('User', UserSchema);

// Export the User model
export default User;
```

User Service:

User Service (Fetch methods and Application submission)

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

You, 1 second ago | 1 author (You)
@Injectable({
  providedIn: 'root'
})
export class UserService {
  // Base URL for the API
  url = 'http://localhost:8081/api/auth/';

  constructor() { }

  // Method to fetch all users from the server
  async getAllUsers(): Promise<User[]> {
    const data = await fetch(this.url);
    return await data.json() ?? [];
  }

  // Method to fetch a user by their ID
  async getUserById(id: number): Promise<User | undefined> {
    const data = await fetch(`${this.url}/${id}`);
    return await data.json() ?? {};
  }

  // Method to handle form submission (currently logs the username and password)
  submitApplication(username: string, password: string) {
    console.log(username, password);
  }
}
```

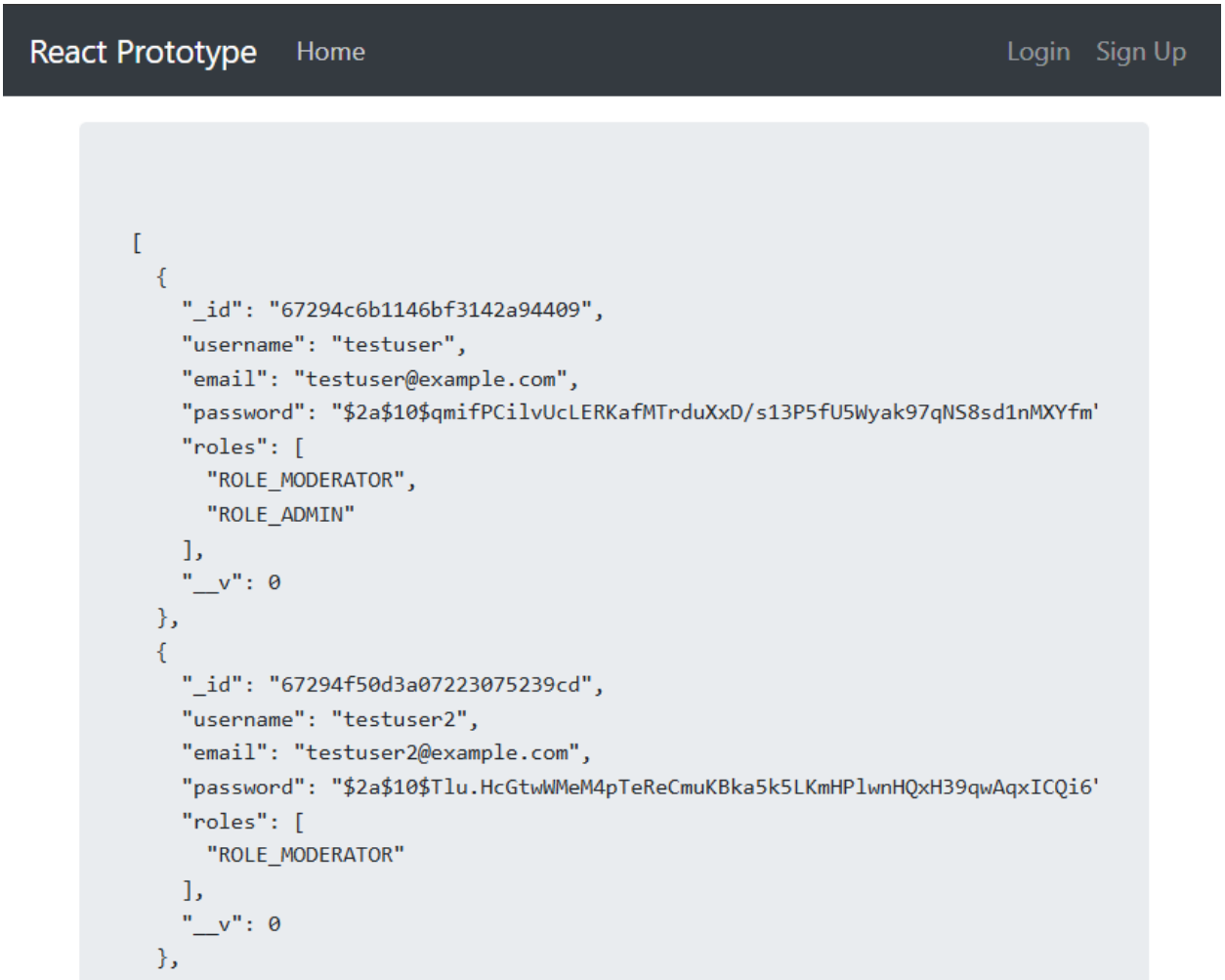
3.2 React Screenshots

Home:

Home Component code (Imports, Component, JSX)

```jsx
import React, { useState, useEffect } from "react";

import UserService from "../services/user.service";

// Home component
const Home = () => {
  const [content, setContent] = useState(""); // State variable for content

  // useEffect hook to fetch public content when the component mounts
  useEffect(() => {
    UserService.getPublicContent().then(
      (response) => {
        const data = response.data;
        // Set content to the response data, formatted as a string if necessary
        setContent(typeof data === 'string' ? data : JSON.stringify(data, null, 2));
      },
      (error) => {
        // Handle errors and set content to the error message
        const _content =
          (error.response && error.response.data) ||
          error.message ||
          error.toString();

        setContent(_content);
      }
    );
  }, []); // Empty dependency array means this runs once when the component mounts

  return (
    <div className="container">
      <header className="jumbotron">
        <pre>{content}</pre> {/* Display the content */}
      </header>
    </div>
  );
};

export default Home;
```

Home Component Screen



Login:

Login Component code (Imports, Component)

```
import React, { useState, useRef } from "react";
import { useNavigate } from 'react-router-dom';
import Form from "react-validation/build/form";
import Input from "react-validation/build/input";
import CheckButton from "react-validation/build/button";
import AuthService from "../services/auth.service";

// Validation function to check if a field is filled
const required = (value) => {
  if (!value) {
    return (
      <div className="alert alert-danger" role="alert">
        This field is required!
      </div>
    );
  }
};

// Login component
const Login = () => {
  let navigate = useNavigate(); // Hook to navigate programmatically

  const form = useRef(); // Reference to the form
  const checkBtn = useRef(); // Reference to the check button

  // State variables for form fields and status messages
  const [username, setUsername] = useState("");
  const [password, setPassword] = useState("");
  const [loading, setLoading] = useState(false);
  const [message, setMessage] = useState("");

  // Handler for username input change
  const onChangeUsername = (e) => {
    const username = e.target.value;
    setUsername(username);
  };

  // Handler for password input change
  const onChangePassword = (e) => {
    const password = e.target.value;
    setPassword(password);
  };
```

Continued Login Component code (Component, JSX)

```
// Handler for form submission
const handleLogin = (e) => {
  e.preventDefault();

  setMessage("");
  setLoading(true);

  // Validate all fields
  form.current.validateAll();

  if (checkBtn.current.context._errors.length === 0) {
    AuthService.login(username, password).then(
      () => {
        navigate("/profile");
        window.location.reload();
      },
      (error) => {
        const resMessage =
          (error.response &&
            error.response.data &&
            error.response.data.message) ||
          error.message ||
          error.toString();

        setLoading(false);
        setMessage(resMessage);
      }
    );
  } else {
    setLoading(false);
  }
};
```

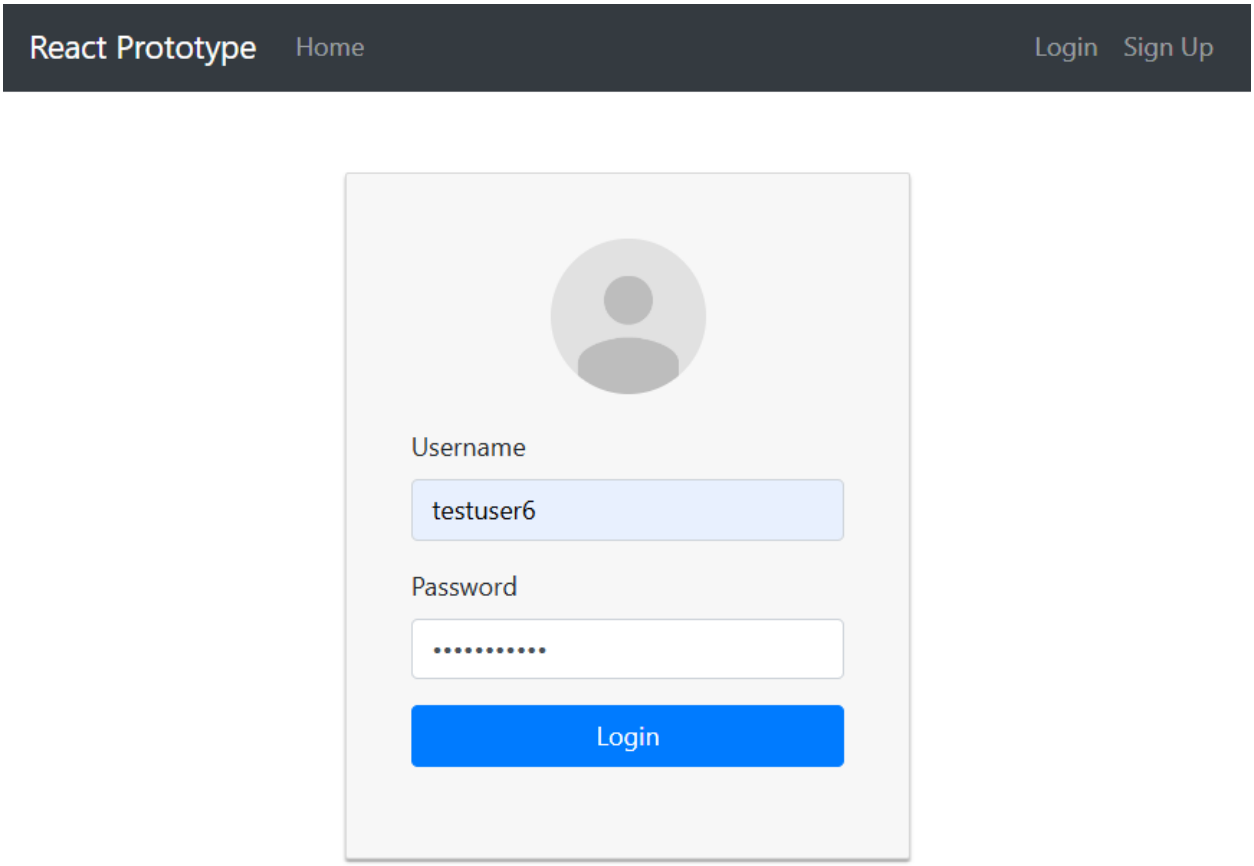
Login Component code (JSX)

```jsx
return (
  <div className="col-md-12">
    <div className="card card-container">
      <img
        src="//ssl.gstatic.com/accounts/ui/avatar_2x.png"
        alt="profile-img"
        className="profile-img-card"
      />

      <Form onSubmit={handleLogin} ref={form}>
        <div className="form-group">
          <label htmlFor="username">Username</label>
          <Input
            type="text"
            className="form-control"
            name="username"
            value={username}
            onChange={onChangeUsername}
            validations={[required]}
          />
        </div>

        <div className="form-group">
          <label htmlFor="password">Password</label>
          <Input
            type="password"
            className="form-control"
            name="password"
            value={password}
            onChange={onChangePassword}
            validations={[required]}
          />
        </div>

        <div className="form-group">
          <button className="btn btn-primary btn-block" disabled={loading}>
            {loading && (
              <span className="spinner-border spinner-border-sm"></span>
            )}
            <span>Login</span>
          </button>
        </div>
```

Continued Login Component code (JSX)

```
          {message && (
            <div className="form-group">
              <div className="alert alert-danger" role="alert">
                {message}
              </div>
            </div>
          )}
          <CheckButton style={{ display: "none" }} ref={checkBtn} />
        </Form>
      </div>
    </div>
  );
};

export default Login;
```

Login Component Screen



Login Console Response

Register:

Register Component code (Imports, Validation)

```
import React, { useState, useRef } from "react";
import Form from "react-validation/build/form";
import Input from "react-validation/build/input";
import CheckButton from "react-validation/build/button";
import { isEmail } from "validator";
import AuthService from "../services/auth.service";

// Validation function to check if a field is filled
const required = (value) => {
  if (!value) {
    return (
      <div className="alert alert-danger" role="alert">
        This field is required!
      </div>
    );
  }
};

// Validation function to check if the email is valid
const validEmail = (value) => {
  if (!isEmail(value)) {
    return (
      <div className="alert alert-danger" role="alert">
        This is not a valid email.
      </div>
    );
  }
};

// Validation function to check if the username length is between 3 and 20 characters
const vusername = (value) => {
  if (value.length < 3 || value.length > 20) {
    return (
      <div className="alert alert-danger" role="alert">
        The username must be between 3 and 20 characters.
      </div>
    );
  }
};
```

Continued Register Component code (Component)

```
const vpassword = (value) => {
  if (value.length < 6 || value.length > 40) {
    return (
      <div className="alert alert-danger" role="alert">
        The password must be between 6 and 40 characters.
      </div>
    );
  }
};

// Register component
const Register = () => {
  const form = useRef(); // Reference to the form
  const checkBtn = useRef(); // Reference to the check button

  // State variables for form fields and status messages
  const [username, setUsername] = useState("");
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [successful, setSuccessful] = useState(false);
  const [message, setMessage] = useState("");

  // Handler for username input change
  const onChangeUsername = (e) => {
    const username = e.target.value;
    setUsername(username);
  };

  // Handler for email input change
  const onChangeEmail = (e) => {
    const email = e.target.value;
    setEmail(email);
  };

  // Handler for password input change
  const onChangePassword = (e) => {
    const password = e.target.value;
    setPassword(password);
  };
```

Continued Register Component code (Component)

```
// Handler for form submission
const handleRegister = (e) => {
  e.preventDefault();

  setMessage("");
  setSuccessful(false);

  form.current.validateAll(); // Validate all fields

  // Check if there are no validation errors
  if (checkBtn.current.context._errors.length === 0) {
    AuthService.register(username, email, password).then(
      (response) => {
        setMessage(response.data.message);
        setSuccessful(true);
      },
      (error) => {
        const resMessage =
          (error.response &&
            error.response.data &&
            error.response.data.message) ||
          error.message ||
          error.toString();

        setMessage(resMessage);
        setSuccessful(false);
      }
    );
  }
};
```

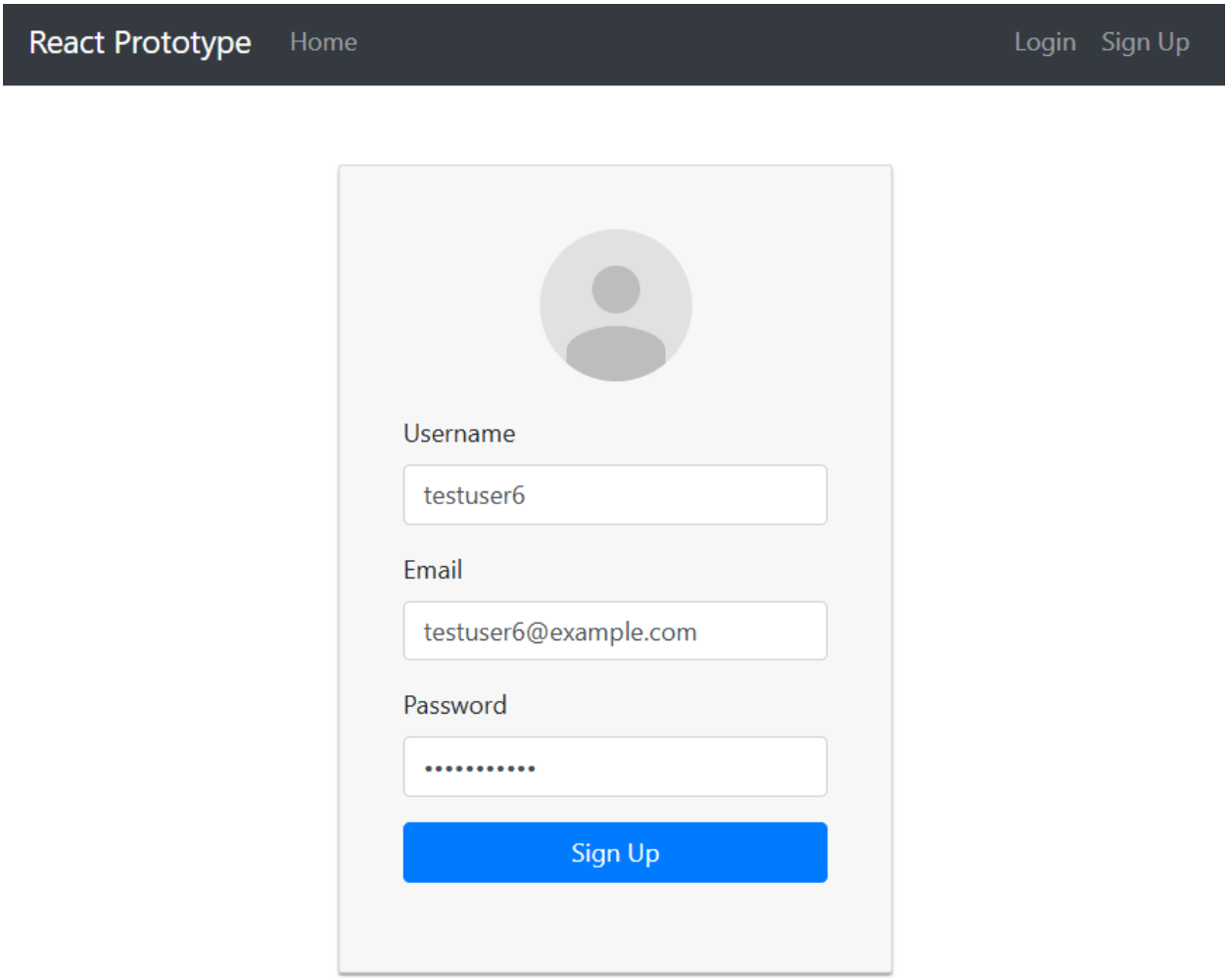
Continued Register Component code (JSX)

```jsx
return (
  <div className="col-md-12">
    <div className="card card-container">
      <img
        src="//ssl.gstatic.com/accounts/ui/avatar_2x.png"
        alt="profile-img"
        className="profile-img-card"
      />

      <Form onSubmit={handleRegister} ref={form}>
        {!successful && (
          <div>
            <div className="form-group">
              <label htmlFor="username">Username</label>
              <Input
                type="text"
                className="form-control"
                name="username"
                value={username}
                onChange={onChangeUsername}
                validations={[required, vusername]}
              />
            </div>

            <div className="form-group">
              <label htmlFor="email">Email</label>
              <Input
                type="text"
                className="form-control"
                name="email"
                value={email}
                onChange={onChangeEmail}
                validations={[required, validEmail]}
              />
            </div>
```

Continued Register Component code (JSX)

```
            <div className="form-group">
              <label htmlFor="password">Password</label>
              <Input
                type="password"
                className="form-control"
                name="password"
                value={password}
                onChange={onChangePassword}
                validations={[required, vpassword]}
              />
            </div>

            <div className="form-group">
              <button className="btn btn-primary btn-block">Sign Up</button>
            </div>
          </div>
        )}

        {message && (
          <div className="form-group">
            <div
              className={
                successful ? "alert alert-success" : "alert alert-danger"
              }
              role="alert"
            >
              {message}
            </div>
          </div>
        )}
        <CheckButton style={{ display: "none" }} ref={checkBtn} />
      </Form>
    </div>
  </div>
);
};

export default Register;
```

Register Component Screen



Register Console Response

App:

App.js code

```
import React, { useState, useEffect } from "react";
import { Routes, Route, Link } from "react-router-dom";
import "bootstrap/dist/css/bootstrap.min.css";
import "./App.css";
import AuthService from "./services/auth.service";
import EventBus from "./common/EventBus";

import Login from "./components/Login";
import Register from "./components/Register";
import Home from "./components/Home";
import Profile from "./components/Profile";
import BoardUser from "./components/BoardUser";
import BoardModerator from "./components/BoardModerator";
import BoardAdmin from "./components/BoardAdmin";

const App = () => {
  const [showModeratorBoard, setShowModeratorBoard] = useState(false); // State to show/hide moderator board
  const [showAdminBoard, setShowAdminBoard] = useState(false); // State to show/hide admin board
  const [currentUser, setCurrentUser] = useState(undefined); // State for the current user

  useEffect(() => {
    const user = AuthService.getCurrentUser(); // Get the current user

    if (user) {
      setCurrentUser(user); // Set the current user
      setShowModeratorBoard(user.roles.includes("ROLE_MODERATOR")); // Show moderator board if user has moderator
      setShowAdminBoard(user.roles.includes("ROLE_ADMIN")); // Show admin board if user has admin role
    }

    // Listen for logout event
    EventBus.on("logout", () => {
      logOut();
    });

    // Cleanup the event listener on component unmount
    return () => {
      EventBus.remove("logout");
    };
  }, []);
```

App.js code (Navigation bar, JSX)

```
// Function to log out the user
const logOut = () => {
  AuthService.logout();
  setShowModeratorBoard(false);
  setShowAdminBoard(false);
  setCurrentUser(undefined);
};

// Navigation bar
return (
  <div>
    <nav className="navbar navbar-expand navbar-dark bg-dark">
      <Link to={"/"} className="navbar-brand">
        React Prototype
      </Link>
      <div className="navbar-nav mr-auto">
        <li className="nav-item">
          <Link to={"/home"} className="nav-link">
            Home
          </Link>
        </li>

        {showModeratorBoard && (
          <li className="nav-item">
            <Link to={"/mod"} className="nav-link">
              Moderator Board
            </Link>
          </li>
        )}

        {showAdminBoard && (
          <li className="nav-item">
            <Link to={"/admin"} className="nav-link">
              Admin Board
            </Link>
          </li>
        )}
```

App.js code (Navigation bar, JSX)

```
        {currentUser && (
          <li className="nav-item">
            <Link to={"/user"} className="nav-link">
              User
            </Link>
          </li>
        )}
      </div>

      {currentUser ? (
        <div className="navbar-nav ml-auto">
          <li className="nav-item">
            <Link to={"/profile"} className="nav-link">
              {currentUser.username}
            </Link>
          </li>
          <li className="nav-item">
            <a href="/login" className="nav-link" onClick={logOut}>
              LogOut
            </a>
          </li>
        </div>
      ) : (
        <div className="navbar-nav ml-auto">
          <li className="nav-item">
            <Link to={"/login"} className="nav-link">
              Login
            </Link>
          </li>

          <li className="nav-item">
            <Link to={"/register"} className="nav-link">
              Sign Up
            </Link>
          </li>
        </div>
      )}
    </nav>
```

App.js code (Route Paths, JSX)

```
      {/* Route paths */}
      <div className="container mt-3">
        <Routes>
          <Route path="/" element={<Home/>} />
          <Route path="/home" element={<Home/>} />
          <Route path="/login" element={<Login/>} />
          <Route path="/register" element={<Register/>} />
          <Route path="/profile" element={<Profile/>} />
          <Route path="/user" element={<BoardUser/>} />
          <Route path="/mod" element={<BoardModerator/>} />
          <Route path="/admin" element={<BoardAdmin/>} />
        </Routes>
      </div>

    </div>
  );
};

export default App;
```

Profile:

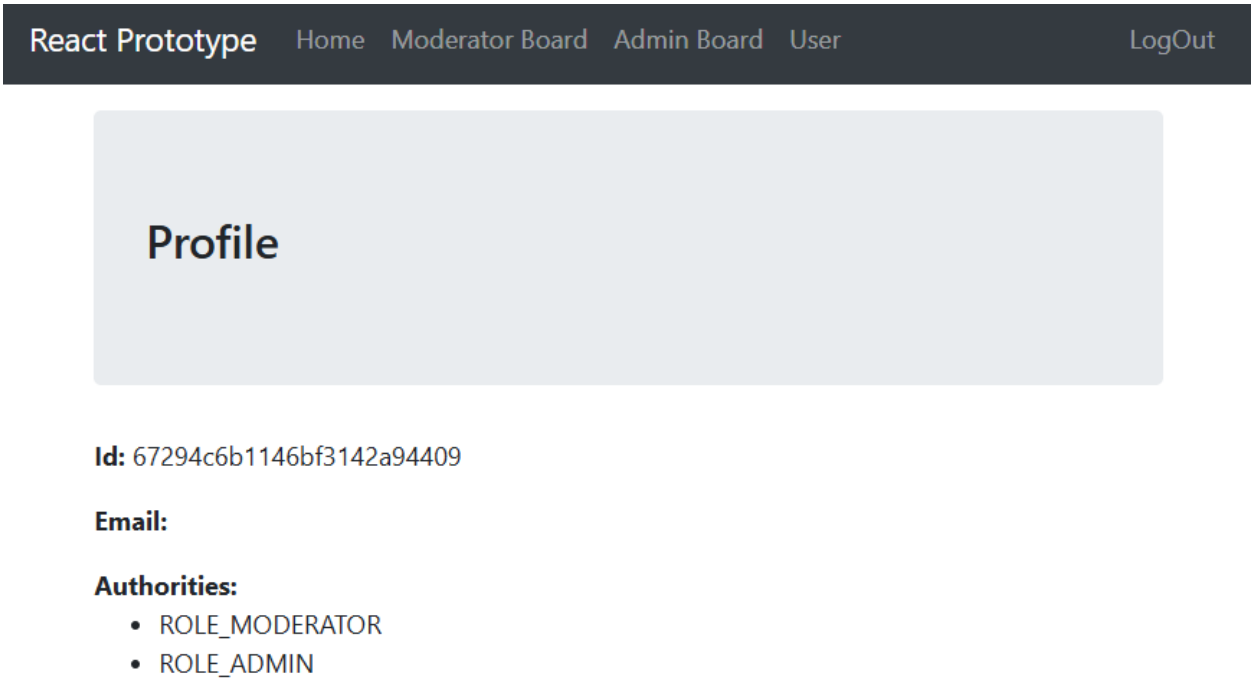
Profile Component code (Imports, Message)

```
import React from "react";
import AuthService from "../services/auth.service";

// Profile component
const Profile = () => {
  // Get the current user from AuthService
  const currentUser = AuthService.getCurrentUser();

  // If no user is logged in, display a message
  if (!currentUser) {
    return (
      <div className="container">
        <header className="jumbotron">
          <h3>No user is logged in</h3>
        </header>
      </div>
    );
  }
```

Profile Component code (JSX)

```jsx
// If a user is logged in, display their profile information
return (
  <div className="container">
    <header className="jumbotron">
      <h3>
        <strong>{currentUser.username}</strong> Profile
      </h3>
    </header>
    <p>
      {currentUser.token && (
        <>
          <strong>Token:</strong> {currentUser.token.substring(0, 20)} ...{" "}
          {currentUser.token.substring(currentUser.token.length - 20)}
        </>
      )}
    </p>
    <p>
      <strong>Id:</strong> {currentUser.id}
    </p>
    <p>
      <strong>Email:</strong> {currentUser.email}
    </p>
    <strong>Authorities:</strong>
    <ul>
      {currentUser.roles &&
        currentUser.roles.map((role, index) => <li key={index}>{role}</li>)}
    </ul>
  </div>
);
};

export default Profile;
```

Profile Component Screen



User Service:

User Service code (Imports, GET Requests)

```
import axios from "axios";
import authHeader from "./auth-header";

const API_URL = "http://localhost:8081/api/test/";

// Function to get public content
const getPublicContent = () => {
  return axios.get(API_URL + "all");
};

// Function to get user board content
const getUserBoard = () => {
  return axios.get(API_URL + "user", { headers: authHeader() });
};

// Function to get moderator board content
const getModeratorBoard = () => {
  return axios.get(API_URL + "mod", { headers: authHeader() });
};

// Function to get admin board content
const getAdminBoard = () => {
  return axios.get(API_URL + "admin", { headers: authHeader() });
};

// Exporting the UserService object with the defined functions
const UserService = {
  getPublicContent,
  getUserBoard,
  getModeratorBoard,
  getAdminBoard,
};

export default UserService;
```

Auth Service:

Auth Service code (Imports, Login POST Request, Register POST Request, Logout Function)

```
import axios from "axios";
import { jwtDecode } from 'jwt-decode'

const API_URL = "http://localhost:8081/api/auth/";

// Function to register a new user
const register = (username, email, password) => {
  return axios.post(API_URL + "signup", {
    username,
    email,
    password,
  });
};

// Function to login a user
const login = (username, password) => {
  return axios
    .post(API_URL + "signin", {
      username,
      password,
    })
    .then((response) => {
      console.log("Login response:", response.data);

      if (response.data.token) {
        console.log("Token:", response.data.token);
        const decodedToken = jwtDecode(response.data.token);
        console.log("Decoded token:", decodedToken);
        localStorage.setItem("user", JSON.stringify(decodedToken));
        console.log("User stored in localStorage:", localStorage.getItem("user"));
      }

      return response.data;
    });
};

// Function to logout a user
const logout = () => {
  localStorage.removeItem("user");
};
```

Continued Auth Service code (GET Request)

```
// Function to get the current logged-in user
const getCurrentUser = () => {
  const currentUser = JSON.parse(localStorage.getItem("user"));
  return currentUser;
};

const AuthService = {
  register,
  login,
  logout,
  getCurrentUser,
};

export default AuthService;
```

Auth Header:

Auth Header code (Establish JWT Token)

```
export default function authHeader() {
  const user = JSON.parse(localStorage.getItem('user'));

  if (user && user.token) {
    return { Authorization: 'Bearer ' + user.token }; // for Spring Boot back-end
    // return { 'x-access-token': user.token };       // for Node.js Express back-end
  } else {
    return {};
  }
}
```

Part B

3.3 Angular

I. Solution Description:

In the Angular prototype, users who are charity members are considered ‘users’ and beneficiaries are considered ‘moderators’. Lastly, the final user is considered an ‘admin’. Charity members (users) are able to create accounts and browse the website. Beneficiaries have the same functionality as charity members but in the final version of the website will be able to create events and will have a discount when booking events. Users with administrative privileges will be able to edit their details such as their role.

The Angular prototype uses the React repositories back-end to communicate to the server allowing data to flow from the database to the front-end client side. The server in the React repository needs to be running for the Angular prototype to work as intended.

This prototype consists of both a registration page and a login page for users to create accounts and log in using their registered credentials. Once a user with admin permissions is logged in, they should be able to access the admin page. The admin page will allow users to edit details of their account such as the role/permissions they have. When a user logs in, depending on their role/permissions they will get a JWT token that will determine what pages the user will be able to access.

II. Comparison:

Though creating a front-end in Angular was easier than creating one in React, there is a noticeable difference between the two. The larger difference for me was that the Angular components required three files by default. One was the code for the component, which was the TypeScript file, another was the spec TypeScript file which was in the unit test for that component, and lastly, each component has a CSS file. Though having a unit test file built-in when a component was created is awesome to have, the codebase feels bloated and messy compared to the tidy React component system.

I also thought that the setup for the Angular CLI was a nightmare, this could just be my experience, but it was difficult to get running and from what I could find on the Angular website there was little to no information for someone who had little to no experience with Angular. I had to go searching the internet and eventually found a solution to start the CLI. This gives Angular a large gap in favor of React as React can be started via Node commands.

I can see that the Angular framework is better suited to smaller applications as the component system is more bloated and confusing.

III. Helps Development:

Angular is a great front-end framework at the end of the day and is very suitable for this type of small web application. However, I can also see that as the web application gets larger and larger over time it will become a nightmare to maintain.

3.4 React

I. Solution Description:

The React prototype has the same roles/permissions as the Angular prototype as the Angular front-end was built off the back-end contained within the React repository. This means that charity members are considered ‘users’, beneficiaries are considered ‘moderators’, and some users can have administrative privileges ‘admin’.

Users are able to create accounts using the registration page and log in using their registered credentials on the login page. Once a user is logged in, depending on their role/permissions they will be able to access the admin page, moderator page, and user/profile pages respectively. When the user logs into their account they are issued with a unique token that will enforce the role/permission system. Only allowing the users with the correct roles/permissions to access their respective pages.

II. Comparison:

I much prefer using React components compared to Angular components because of their complex simplicity. Though this is a personal opinion, I really like the implementation of JSX in React. It just makes sense. Keep all the front-end JavaScript and HTML code in the same file. Doing this is less cumbersome overall and makes the front-end code easier to read and maintain. This is definitely the biggest positive that React has over Angular in my opinion, though Angular has its own way of achieving the same result, I just prefer React component syntax and do not miss the file bloat that comes with Angular.

There is something I really adore about using React hooks. Hooks allows you to add state, side effects, and context to components without the need for additional components. Before I started using them, I didn’t know what I was missing and now after using them, I don’t want to give them up.

III. Helps Development:

There are two amazing things that I found when using React. Creating user interfaces (UI’s) is so easy when you understand how to manipulate the component using JSX syntax. This makes the user interfaces dynamic and allows developers to create complex user experiences with ease. Another big positive is the large amount of community support React has. Whether this is due to being around for a good amount of time or developers just enjoy using React. I was

blown away by the amount of help and support I had from forums on websites such as Stack Overflow or the resources available on the React development website.

4. Conclusion

I feel that I am biased in my recommendation for which framework to use for the creation of this website. Over the course of creating these two prototypes, I have found a love for one and a dislike for the other. I believe this was due to the simplicity and complexity of my recommendation.

For multiple reasons, I will have to recommend React as the framework that will be used to create the final website. I want to be able to use React hooks and the JSX syntax on the front end. I realise that Angular has mostly the same functionality as the React hooks and the JSX syntax, but I find the whole framework cumbersome to use and maintain. Compared to React, Angular just adds way too much to the codebase, which is not necessarily a bad thing as Angular has some amazing features.

Personally, I believe that React is the better framework overall and will be using it to create the final version of the website.

5. References

Angular. (n.d.). https://v17.angular.io/guide/what-is-angular

Deshpande, C. (2024b, August 30). *What is Angular? A guide to the Angular framework*. Simplilearn.com. https://www.simplilearn.com/tutorials/angular-tutorial/what-is-angular

Vishal-Siddhpara. (2024, September 1). *React vs Angular: Which One is Best for Your Next Front-end Project?* Radixweb. https://radixweb.com/blog/react-vs-angular

React – A JavaScript library for building user interfaces. (n.d.). React. https://legacy.reactjs.org/

Deshpande, C. (2024a, July 25). *The best guide to know what is react*. Simplilearn.com. https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs

GeeksforGeeks. (2024, October 10). *ReactJS Data Binding*. GeeksforGeeks. https://www.geeksforgeeks.org/reactjs-data-binding/

Fulness, O. (2023, October 4). Main features of React that developers must know | Medium. *Medium*. https://medium.com/@ojebiyifulness/5-main-features-of-react-js-that-developers-must-know-759e222d3699

GitHub. (2024). *What is AI code generation?* GitHub. https://github.com/resources/articles/ai/what-is-ai-code-generation