CSCC01 Lab 4 — MERN & Chatter

In this lab, you will be building a full-stack social media web application similar to ~~Twitter~~ X. Our app will be called Chatter. You will expand your knowledge of React, Express, and Sequelize.

In this lab, you will:

- Learn how to perform user authentication and create sessions in the backend.
- Get practice with working in React and passing data to components.
- Work with the different React *hooks*.
- Work with a SQL-based ORM (Sequelize) and a sqlite database.
- Learn about CORS.

Note: There is a README.md in the GitHub repository that is much longer filled with useful code snippets and helpful information and insights. Please make sure to read that if you get stuck.

Logistics

- This lab is worth 9 points and 2.5% of your final grade.
- The lab will be supervised by your TA during the tutorial session of Week 5.
- If you encounter any problem while doing the steps listed in the next section, ask the TA for help.
- Attendance will be taken during the tutorial session, and a **10%** penalty will be applied for being absent with no valid excuse.
- The lab should be done individually.
- The due date is **February 8th, 2025 at 11:59PM**

Lab Setup

Firstly, clone the GitHub repository onto your local machine through GitHub classroom.

At the root of the repository, you will notice two folders:

- backend: The backend application code (**Express**)
- frontend: The frontend website code (**React**)

Before proceeding, cd into both of these directories and install all the required Node packages by running

```
npm install
```

Backend

The backend makes use of the [Sequelize (https://sequelize.org/docs/v6/getting-started/)](https://sequelize.org/docs/v6/getting-started/) ORM in conjunction with a sqlite database to store our data.

The API takes in inputs and returns data in **JSON** format.

File Structure

The backend API makes use of an MVC architecture and is organized in the following format:

- controllers/: The controllers of the application.

- models/: Database models using Sequelize.
- routes/: The routes for the API.
- database.js: The code that initializes our database connection.
- server.js: The entry-point to the application.

Starting the Backend

1. cd backend
2. npm install
3. Start the application with npm start.

NOTE: In all subsequent tasks, ensure that the correct HTTP status codes (https://developer.mozilla.org/en-US/docs/Web/HTTP/Status) are being returned by the API. You can follow the guide in that link, but if you're unsure which one should be returned in any case, feel free to ask a TA during your tutorial.

If any request is ever missing any body parameters, the API should return 400 with the following response:

```
{
  error: "Missing parameters in the request body"
}
```

Task 1: User Registration

We want users to be able to register to our application using a username and password. Of course, we don't want to store the password in clear in our database, so we will make use of bcrypt (https://www.npmjs.com/package/bcrypt) in order to generate *salted hashes* to store instead.

Task: Implement the register function in usersController.js to properly register a new user to the application, storing their information in the sqlite database given username and password in the request body.

If another user already has the same username in the database, the API should return 400 and the response should be:

```
{
  error: "The username is already taken."
}
```

Upon successful registration, the API should return 201 and the response should be:

```
{
  message: "The user has been successfully registered"
}
```

Task 2: User Authentication

We want users to be able to login to our application. After they have logged in, we want our backend to remember this through a session. A session is a server-side storage of user data that persists across multiple requests, allowing the server to remember the user between interactions given a session ID stored in a cookie.

We can use a middleware called express-session (https://www.npmjs.com/package/express-session) which implements this type of stateful authentication for us.

Note that the default behaviour of express-session is to store the user's information in memory so the sessions are lost when the express application is restarted (this is ok for the lab).

We can enable the express-session middleware by placing this in our server.js file:

```
app.use(
  session({
    secret: process.env.SESSION_SECRET_KEY,
    resave: false,
    saveUninitialized: true,
  })
);
```

Don't forget to create a .env file in backend/ and define SESSION_SECRET_KEY in it for this to work. The TA will grade your lab using their own key so there's no need to commit the .env file.

Once we've enabled this middleware, we can read and write to the session like so:

```
req.session.userId = 1;
const userId = req.session.userId; // 1
```

express-session will generate a cookie and send it back to the client using a Set-Cookie header once you have modified the session automatically. In all subsequent requests, if **credentials** are enabled (more on that later on), the client will also automatically send this cookie to the server for you.

Task: Implement the login function in usersController.js to log the user into the application given username and password in the request body. The API should store the user's user ID in req.session.userId and username req.session.username in the session upon succesful login.

If either the username or password is incorrect, return 401 and the following response:

```
{
  error: "The username/password is incorrect"
}
```

Upon successful login, the API should return 201 (to indicate the creation of a session) and the following response:

```
{
  message: "Logged in successfully"
}
```

Task 2.1: User Information

Task: Implement the me function in controllers/usersController.js to return the username and user ID of the user making the request. We can get this information from req.session.

If no session for the user exists, the API should return 401 and the response should be:

```
{
  error: "User not authenticated"
}
```

Otherwise, if the session exists, the API should return 200 and the response should be:

```
{
  id: 1 // The user's ID
  username: "<The user's username>"
}
```

Task 3: Creating Posts

Now we get into the actual posts on our website. We're only going to implement post creation for this simple lab.

A post object has the following properties, which you can see in the models folder:

```
const Post = sequelize.define('post', {
  content: {
    type: DataTypes.STRING,
    allowNull: false,
  },
  authorId: {
    type: DataTypes.INTEGER,
    allowNull: false,
  },
  createdAt: {
    type: DataTypes.DATE,
    allowNull: false,
  },
  likes: {
    type: DataTypes.INTEGER,
    allowNull: false,
    defaultValue: 0,
  },
  dislikes: {
    type: DataTypes.INTEGER,
    allowNull: false,
    defaultValue: 0,
  },
});
```

Task: Implement the createPost function in controllers/postsController.js to create a post and insert it into the database given the post content in the body.

Only requests with a valid session may create posts. If a session doesn't exist, then the API should return 401 and the response should be:

```
{
  error: "User not authenticated"
}
```

New posts should have 0 likes, 0 dislikes, createdAt should be the current date and time, and authorId should be set to the current user's ID stored in the session.

Frontend

Starting the Frontend

1. cd frontend
2. npm install
3. Start the application with npm start.

Task 4: Registration Form

The registration form component has been nearly completed for you in src/components/register.js. All that's left for you to do is to implement the functionality for calling the API given the user's data.

Task: Implement the register function in src/components/register.js to call the backend API with the user's username and password in the request body. Provide adequate visual feedback to the user using the statusMessage state variable.

Task 4.1: CORS

You'll notice that the requests you make to the backend server are failing because of CORS. Our frontend hosted at http://localhost:3000/ is considered a different origin than our backend hosted at http://localhost:8080/, which is causing the issues. You can read more on what CORS is and how it protects you here (https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS).

To fix this problem, we have to make use of the CORS middleware package (https://expressjs.com/en/resources/middleware/cors.html) to add the relevant CORS headers to all endpoints.

Task: Configure CORS correctly in the Express backend in server.js so that the browser does not block our requests to the server.

Your configuration should look similar to this (with "???" filled out correctly):

```
const corsConfig = {
  origin: '???',
  credentials: true
};
app.use(cors(corsConfig));
```

Task 5: Login Form

Task: Implement the login function in src/components/register.js to call the backend API with the user's username and password in the request body. Provide adequate visual feedback to the user using the statusMessage state variable.

Ensure that the user is redirected to the home page upon successful login.

About Credentials

Important: When making your request to the backend using the fetch API, make sure to put credentials: 'include' in your request options for express-session to work.

```
const response = await fetch(`${apiURL}/users/login`, {
  method: 'POST',
  ...
  credentials: 'include',
  ...
});
```

You must do this for any endpoint that requires the use of express-session; otherwise, cookies will not be sent or set in the browser.

Task 6: Fetching User Information

Task: Implement the me function in src/App.js to call the backend API to check if the user is logged in or not; set the loggedIn state variable accordingly. This function should also set the user state variable with some important user information (username and/or id).

HINT: Use the endpoint implemented in **Task 2.1**.

Task 7: Creating Posts

Task: Implement the createPost function in src/components/createPost.js to call the backend API with the user's post content in the request body to create a new post. Once a user successfully creates a post, the list of posts should be refreshed.

Submission

To submit the lab, ensure the following:

- All of your commits are pushed to your assignment repository **before** the submission deadline.
- The URL to your repository is submitted under the Lab 4 assignment on Quercus.

Grading

Backend

- (1 pts) Implement user registration.
- (1.5 pts) Implement user login and express-session middleware.
- (0.5 pts) Implement /me endpoint.
- (1 pt) Implement secured create posts endpoint.
- (1 pt) Configuring CORS

Frontend

- (1 pts) Implement user registration.
- (1 pt) Implement user login.
- (1 pts) Implement me function to check user authentication.
- (1 pt) Implement creating posts.

Total: 9 pts