Secure programming course project documentation

Login-app

Repository: https://github.com/Tumpbe/Login-app

Sami Mononen H267178

Tuomas Knaapi H274483

General description and structure of the program

The program is a simple login application which demonstrates widely used web authentication practices and security measures. The application features consist of registering a new user, logging in with an existing user, changing the password of an existing user, deleting an existing user and logging out the currently logged in user.

The main technologies used in the application are as follows:

Database: MongoDB https://www.mongodb.com/docs/

Frontend: React https://reactjs.org/

Backend: Express https://expressjs.com/

User data is stored in the database. The backend communicates with the database by adding new users, deleting users and validating authentication on login. The backend structure consists of a controller, a database model, a router and middleware for authentication token validation. Requests from the frontend go to the router which calls the appropriate controller and middleware functions. The controller communicates with the database, but before any changes are made the database model checks and formats the saved data. Lastly, authentication middleware is called on requests that require authentication.

On the frontend side there is a simple user interface mainly consisting of button and form input fields. On the main page there are buttons for registering and logging in. Once the user has logged in successfully, a profile button appears. The profile interface gives the user the ability to logout, change password and delete the user. Also, some basic user-friendly checks for password/email validation are done before the various forms are submitted.

Communication between the frontend and the backend is done via axios HTTP requests. A request is sent from the frontend to which the backend responds to. Responses and requests hold data e.g., user data.

For more details on the components of the application, see components section below.

Secure programming features

Since the program is done for a secure programming course, it has many security features. Most of the security features are on the backend side of the program, but the frontend has some too.

Secure frontend

Before any form data is submitted, basic validation checks are performed. For the password field it is made sure that the password is at least 10 characters long and has special

characters, upper- and lowercase characters and numbers. Email is validated via email regex which basically makes sure its format is *something@anotherthing.foo*. At the login page frontend sends a verification code for user's email and login is only successful after correct code is given by the user. Also, frontend offers to the user possibility to change his password.

Secure backend

In the backend database model for user data passwords are hashed. Every time a password is saved to the database the hashing functions is called. Bcrypt packages hashSync-function is used for the hashing with 10 salt rounds. This means that *all* passwords in the database are always hashed. Bcrypt is also used to verify (compare) passwords when logging in or changing the password. This way the passwords stay hashed during the operation.

Reference and more information on bcrypt: https://www.npmjs.com/package/bcrypt

The database model also has a functionality which removes passwords from user objects sent as response payloads, so the frontend never gets access to them.

JSON web tokens are used for authentication. On a valid login, a token is generated and saved as a cookie for the current session. With the token the user can use features which require authentication such as deleting the user. Basically, the authentication middleware checks the token before any controller function is called. The token gets deleted on logout or user deletion. Cookies are used as http only to help mitigate the risk of client-side scripts accessing the protected cookie.

Reference and more information on JSON web token:
https://www.npmjs.com/package/jsonwebtoken

The backend is using cross-origin recourse sharing (CORS). CORS is a mechanism that prevents the browser from loading recourses from any other origin than the one indicated. In this case any other origin than *http://localhost:3000*.

Widely used web package helmet is also utilized to add a variety of security features.

Helmet features used:

-**Content security policy:** Helps mitigate e.g., cross-site scripting attacks.

-**Cross-origin-embedded-policy:** Helps mitigate cross-origin attacks done via embedded recourses.

-**Cross-origin-opener-policy:** Ensures that a top-level document doesn't share a browsing context group with cross-origin documents.

-**Cross-origin-resource-policy:** Blocks cross-origin requests.

-**Expect-CT:** Helps mitigate misissued SSL certificates.

-**Referrer-policy:** Mitigates security issues caused by the referrer header.

-**Strict-transport-security:** Use https instead of http.

-**Nosniff:** Mitigates MIME type sniffing by setting X-Content-Type-Options to nosniff.

-**Origin-agent-cluster:** Allows a web application to isolate its origins.

-**DNS prefetch control:** Controls DNS prefetching to improve privacy at the expense of performance.

-**IE no open:** Forces potentially unsafe downloads to be saved instead of execution. Only for Internet Explorer 8.

-**Frameguard:** Helps mitigate clickjacking attacks where the user clicks something different from what is perceived and ends up leaking information.

-**Permitted cross domain policies:** Tells some clients, mainly Adobe products, your domain's policy for loading cross-domain content.

-**Hide powered by:** Offers limited secure benefits by removing X-Powere-By header. Mostly used to save bandwidth.

-**XSS filter:** Removes browsers' buggy cross-site scripting filter.

Reference and more information on helmet: https://www.npmjs.com/package/helmet

Known vulnerabilities

When the password is changed no check for password length is performed by the database model. The password change relies on frontend to do the validity check which might be a security risk.

Suggestions for improvement

One improvement that comes to mind is adding a Joi validation functionality to the backend. Joi validation would allow email/password validations on backend side without having to rely on the database model. The database model is useful in many cases but lacks some features such as validation on value change.

Considering that both group members are still learning web programming, there might be improvements that we are not even seeing as of now. We think that revisiting the project after some time will be beneficial for learning and rehearsing the subjects learned during the project.

Components

App

Application’s parent-component. All other components are accessible through Routing-logic.

Also includes ‘loginCallBack’ -function implementation that saves the function parameter ‘payload’ (which is user-object) for the frontend and navigates to that user’s profile-page when used. This user-object is passed for the Home-component, and the function is passed for the Login-component.

Home

The home component displays buttons for navigating to different sections of the app. These sections are register, login and profile-page. The button for navigating to the profile-page is only visible after user has logged in.

Path: /

Props: ‘user’, user-object that has logged in

Login

Displays the login-form and handles its submit event by making a POST-request to the backend with given form. The login component is also responsible for sending and checking verification code for the user.

Path: /login

Props: ‘loginCallBack’, function passed through the App-component

Register

Displays the register-form and handles its submit by making a POST-request to the backend with given form, so the backend can add the user to the database. The register component performs password quality checks and email validation checks.

Path: /register

Profile

The profile component provides various features for the logged in user.

- Password changing: PUT-request is made to the backend with the new password as an argument. Also does same password quality checks for the new password as login component.
- User deletion: DELETE-request is made to the backend and backend handles the deletion from the database.
- User logout: GET-request is made to the backend and backend handles cookie reseting.

Path: /profile/:id