COMP.SEC.300-2021-2022-1 Secure Programming

Exercise work

Aatos Maunula
aatos.maunula@tuni.fi
245675

Description

The application is a basic full-stack web-application. It consists of React frontend, Express Backend and PostgreSQL database. Client and server are implemented with TypeScript. The application is containerized and runs with Docker.

The idea of the application is to be a generic time-reservation application. Users need to register and login to the service and after that they can make reservations for some spaces. Users can manage their own reservations (show, reserve, delete), so the application has basically the CRUD functionality related to reservations.

Secure Programming solutions

The application is created with secure programming concepts in mind.

TypeScript

The whole project has been implemented with TypeScript. I think TypeScript is a great choice for a programming language for the concept of “secure programming”.

TypeScript is related to the development side of secure programming. TypeScript is a statically typed language, so it prevents developers from creating type-related errors on code. TypeScript is generally considered a good choice for especially larger codebases.

Static code analysis

I use eslint as a linter tool on both the frontend and backend of the application. Linters are generally used for flagging programming errors, bugs, stylistic errors and suspicious constructs. I also use the tool called Prettier, which is a code formatter. Prettier enforces the same code styling for all the code and this way makes the whole codebase similar and cleaner.

Both eslint and prettier uses ruleset and the config files can be found on
`/client/` and
`/server/`

OWASP top ten related subjects:

Input field validations / sanitation

The project has three input forms: register, login and reservation additional info.
The register and login forms have checks on both client and server side. The user is asked for email, password and retype password on register form, and email and password on login form. The email is checked to have the email in correct form and the passwords need to match. The reservation additional info form has a sanitation step on the server side, which replaces `><&'"/` characters with HTML entities.

XSS

One of the reasons I selected React for the project, is that React automatically escapes any user input before rendering it. So there is a built-in XSS-protection on the client side.

CSRF

I wanted to implement CSRF-protection to create an extra step of protection on the reservation form. The frontend of the application is a Single Page App, so the CSRF-tokens are implemented as:

1. The client makes a GET api request to the server for CSRF-token when the user is logged in.
2. The token is saved client-side and sent with POST new reservation requests, in the HTTP header.
3. The backend API has a middleware on POST reservation route that handles the checking of the correct CSRF-token.

Access control

Users need to be logged in to use the reservation service. All of the API routes that require the user to be logged in, are protected by a middleware that checks that the session has the user session token. This prevents unauthorized access to the site.

The login functionality itself is implemented with a third party library called passport.

Further development ideas

There is a lot of bugs and stuff to improve related to user interface and catching server errors client side, but other than that:

Spam protection

Initially my idea was to implement one extra layer for spam protection. Spam is one problem that needs to be considered in this type of application. It is very unpleasant and time-consuming for the service provider to manually delete the reservations, if users can abuse the reservation creating.

Two different alternatives are Google ReCaptcha and request throttling. ReCaptcha is implemented on the client side and it is based on inspecting the user behavior on form. ReCaptcha prevents the usage of automated form-fillers (bots), and prevents spam this way. The request throttling is based on limiting the API request from certain IP-addresses, and blacklisting the IP-address for some time if the amount of requests go over a certain limit on a certain time range. The throttling can be implemented with some fast in-memory data store, for example Redis, by saving the key-value pairs of IP-addresses and request counts. The throttling is implemented on the server side.

Integrate linter on git commits

Linting process is manual at the moment. Linting the project could be added to the git commit -flow of the project, so that if the linting has errors, the commit cannot be done. This would prevent pushing errors to the git repository.

SSO login

Login -flow could be improved by adding Single Sign On functionality, so that the users could sign on application straight with Google or Facebook for example.