Module: CMP-7038B – Developing Secure Software  
Assignment: R002 – Secure Development Project and  
Presentation (Individual)  
Set by: Debbie Taylor Debbie.taylor@uea.ac.uk  
Date set: 14 July 2023  
Value: 70%  
Date due: 08 August 2023 15:00  
Returned by: 05 September 2023  
Submission: Blackboard  
Checked by: Eleanor Leist E.leist@uea.ac.uk

**Learning outcomes**  
• Understand the importance of designing software with the security needs of  
an end user in mind  
• Develop a secure and usable website that meets the needs of the user  
• Analyse the effectiveness of a range of security methods and tools  
• Analyse the evolving threats associated with the internet

**Specification** **Overview**  
The aim of this R002 assignment is for you to code a secure usable and accessible web-based movie blog system that mitigates, at a minimum, the five most common security vulnerabilities of account enumeration, session hijacking, SQL injection, cross-site scripting and cross-site request forgery.  
You will work individually to code and secure the web-based blog using JavaScript and Node.js, with a MySQL database.  
At minimum, the movie blog system will require registration and login authentication (via 2FA of username/password and email One-Time-Passwords (OTP)), search functionality, and the ability to add, edit and delete posts. You can use pre-built security libraries, but you must clearly  
and concisely explain how they work and how they improve security for your movie blog system.  
To evidence your system’s security mitigations working, you need to create a maximum 15-minute MP4 (max 720P) video demonstration, showing both the front-end (user website view) and back-end (code and database) elements of your system and try attacking the system yourself to evidence you have protected your system from a threat actor attacking each vulnerability/element.

**Description**  
You are required to individually develop a small, secure, usable and accessible, web-based move blog site that mitigates various security vulnerabilities.

**Development coding of web-based movie blog:**  
At a minimum, your code should defend against the five most common vulnerabilities of:  
• Account enumeration  
• Session hijacking  
• SQL injection  
• Cross-site scripting  
• Cross-site request forgery

You need to concentrate on coding the security, usability and accessibility aspects of the web-based movie blog and not on web development, as you only need to produce a basic usable and accessible  
front-end. This will be used to evidence your security processes and mitigations during a 15-minute MP4 video demonstration. Functionality of the front-end should be prioritised over the aesthetics, but you still need to consider usability and accessibility.  
You must code your website using JavaScript and Node.js, with a MySQL database. Any Node framework, such as Express, is acceptable but you cannot use any other types of SQL databases, as you are restricted to using MySQL.  
To secure your movie blog you must include hashing and/or salting, encryption and a 2FA authentication of username/password and email One Time Password (OTP).  
The movie blog should not fully sacrifice security or usability and accessibility, and there will be some trade-offs needed. You must discuss and justify any trade-offs you have chosen, during your video  
demonstration. You can use any pre-built security libraries you believe will be useful, but you must be clearly and concisely explain how they work, what they secure against and exactly how they provide security protection specifically for this movie blog. If you cannot or do not fully explain your library use, you will not attain any marks for that mitigation. You should also consider coding some of your own processes, as extra marks are available for self-coded mitigations.  
Each mitigation must be valid across the whole web-blog site, e.g., you cannot mitigate SQL injection and then break it later when mitigating another vulnerability.  
The code you produce must be fully tested, using think-aloud user testing and system unit testing. Evidence of these tests (and results) should be recorded via separate test plans and must be shown during the MP4 video demonstration

Below is an outline of the key components and steps you'll need to consider when developing the web-based movie blog system:

1. **Setup and Configuration:**
   * Set up a Node.js project with a package.json file to manage dependencies.
   * Install and configure the necessary packages, including Express for the server and MySQL for the database.

**sudo mysql\_secure\_installation**

* + Create a MySQL database and set up the necessary tables for users, posts, etc.

1. **Authentication and User Management:**
   * Implement user registration, login, and 2FA (Two-Factor Authentication) using username/password and email OTP.
   * Hash and salt user passwords before storing them in the database.
   * Generate and verify email OTPs for 2FA authentication.
   * Create user sessions and implement secure session management.
2. **Input Validation and Sanitization:**
   * Validate and sanitize all user inputs to prevent SQL injection and XSS attacks.
   * Use parameterized queries to mitigate SQL injection risks.
3. **Cross-Site Request Forgery (CSRF) Protection:**
   * Implement CSRF tokens to prevent CSRF attacks on sensitive actions.
4. **Secure Communication:**
   * Enforce HTTPS to secure communication between the client and server.
5. **Post Management:**
   * Allow users to add, edit, and delete posts.
   * Implement proper authorization checks to ensure users can only modify their own posts.
6. **Search Functionality:**
   * Implement search functionality to search for movie blog posts.
7. **Front-end Development:**
   * Develop a basic front-end interface using HTML, CSS, and JavaScript.
   * Ensure the front-end is accessible and usable for all users.
8. **Security Libraries and Self-Coded Mitigations:**
   * Clearly and concisely explain how any pre-built security libraries used improve security for the movie blog system.
   * Consider adding some self-coded security mitigations for additional marks.
9. **Testing and Demonstration:**

* Perform think-aloud user testing to ensure usability and accessibility.
* Conduct system unit testing to verify the functionality and security of the application.
* Record evidence of the tests and results through separate test plans.
* Create a 15-minute MP4 video demonstration showcasing both the front-end and back-end elements of your system, including how you protect against each vulnerability/element.

Remember that security is an ongoing process, and it's essential to stay updated with the latest best practices and security considerations. Additionally, ensure that you thoroughly document your code, explaining the purpose and security features of each component.

This outline should provide you with a starting point for developing the web-based movie blog system. As you proceed with the implementation, make sure to refer to official documentation, security resources, and best practices for each security mitigation you're implementing. Good luck with your project!

**Project-Files**

- app.js (Main application file)

controllers/

- authController.js (User authentication related functions)

- postController.js (Post management related functions)

middleware/

- authenticationMiddleware.js (Handles user authentication checks)

- sessionMiddleware.js (Handles session management)

utils/

- otpUtils.js (Utility functions for generating and verifying OTPs)

- emailUtils.js (Utility functions for sending emails)

routes/

- authRoutes.js (User authentication routes)

- postRoutes.js (Post management routes)

- authMiddleware.js

models/

- OTPverification.js

- userModel.js (Database model for users)

- postModel.js (Database model for posts)

config/

- database.js (Database configuration)

views

-**index.html**: The main landing page of the movie blog site.

**-auth.html**: Combined login and sign-up page for user authentication.

**-create-edit-del-post.html**: Combined page for creating new posts, editing and deteling existing ones.

**-search-reviews.html**: Page for displaying search results and movie reviews.

**-account-settings.html**: Page for managing user account settings, including 2FA configuration and verification options.

**-about.html**: Page with general information about the movie blog site.

**-contact.html**: Page with contact details and a form for users to get in touch.

styles

css

- index.css

-auth.css

-craete-edit-del-post.css

-search-reviews.css

-account-settings.css

-contact.css

movie\_blog.sql

package\_lock.json

package.json

node\_modules

**Server-side mods:**

Your app.js implementation looks good overall. You have configured the necessary middleware, set up routes for user authentication and post management, and handled email sending using the emailUtils module.

Here are a few suggestions and improvements for your app.js implementation:

CSRF Protection: You have included the CSRF protection middleware for all POST requests, which is a good security measure. However, it's essential to consider using more fine-grained CSRF protection, as some routes may not require CSRF protection. For example, routes like /register, /login, and /logout are usually exempted from CSRF protection.

Session Configuration: You've set the secure option for the session cookie to true. This is appropriate if your application is using HTTPS. However, if your application is not served over HTTPS during development, you should set secure: false or omit the secure option to avoid session issues during testing.

Error Handling: Implement proper error handling middleware to catch and handle errors in a more structured way. This will make it easier to identify and respond to errors gracefully.

Validation and Sanitization: Ensure that you validate and sanitize user inputs to prevent security vulnerabilities like SQL injection and XSS attacks. Use a validation library or write custom validation middleware to validate user inputs.

Password Hashing: Implement password hashing and salting during user registration to securely store passwords in the database. Use a robust password hashing library like bcrypt to ensure strong password security.

Authentication Middleware: Consider implementing authentication middleware to check if the user is logged in before allowing access to certain routes. This can be achieved by checking if the user has an active session or valid authentication token.

Authorization: Implement authorization checks to ensure that users can only perform actions they are allowed to. For example, only allow users who created a post to edit or delete it.

Database Integration: Ensure that the user authentication and post management routes interact with the MySQL database securely, using prepared statements or an ORM (Object-Relational Mapping) library to prevent SQL injection.

Error Pages: Create custom error pages to handle various HTTP error status codes (e.g., 404 Not Found, 500 Internal Server Error) to provide a better user experience.

Logging: Implement proper logging to record important events and errors in your application. Logging can be helpful for troubleshooting and identifying potential security issues.

Remember to keep your dependencies up-to-date, especially security-related dependencies, and follow best practices for secure web development.

Overall, your app.js provides a good starting point for building a secure web-based movie blog site. With the suggested improvements and considerations, you can enhance the security and functionality of your application. Keep up the good work!

Additionally, you'll need to implement the server-side logic (app.js) to handle the form submissions from this page and perform the corresponding actions, such as updating the user profile and enabling/disabling 2FA.

**Search Functionality**

**Yes, the code provided in the previous response will work on the client side in a production environment, assuming that the server is properly set up to handle the /search endpoint and return the appropriate JSON data with search results and movie reviews.**

Here's why it should work:

1. **Client-side JavaScript Execution**: The JavaScript code in the <script> tag will be executed on the client side when the search-reviews.html page is loaded in the browser. This means that the Fetch API call will be made from the user's browser, and the received JSON data will be processed on the client side.
2. **Fetching Data from Server**: The Fetch API is a standard way to make HTTP requests from JavaScript. The code fetches data from the /search endpoint on the server using a GET request with the appropriate search term as a query parameter.
3. **Handling JSON Data**: The server should be designed to handle the /search endpoint and respond with JSON data that includes search results and movie reviews. The JavaScript code expects to receive this JSON data in the response.
4. **Dynamically Updating HTML**: Once the JSON data is received, the JavaScript code dynamically updates the HTML content on the client side. It creates HTML elements for each search result and movie review and inserts them into the respective sections of the page.
5. **Error Handling**: The code includes basic error handling using .catch to handle any errors that might occur during the Fetch API call or JSON data processing.

However, there are a few things to consider to ensure it works smoothly in production:

1. **Server-Side Implementation**: Make sure the server is correctly implemented to handle the /search endpoint, perform the database queries securely, and return the expected JSON data.
2. **Data Validation and Sanitization**: Ensure that the data received from the server is properly validated and sanitized on the client side before inserting it into the HTML. This prevents potential security vulnerabilities like Cross-Site Scripting (XSS).
3. **Error Handling Improvement**: The provided code includes basic error handling, but in a production environment, you may want to enhance it to handle various error scenarios more gracefully and provide better feedback to the user.
4. **Security Considerations**: For any sensitive or private data, ensure that appropriate security measures are in place, and access to such data is properly restricted based on user authentication and authorization.
5. **Performance Considerations**: Depending on the amount of data being fetched and processed, consider optimizing the code for better performance. For example, you can implement pagination or lazy loading to manage large datasets efficiently.

By taking these considerations into account, the provided code should work well on the client side in a production environment to display search results and movie reviews dynamically.

views folder:

Inside the view folder, you'll have different views (HTML files) that will be rendered to the users based on the routes and actions. For simplicity, let's assume you have the following views:

a. login.html - This view will display the login form.

b. dashboard.html - This view will display the user's dashboard after successful login.

c. create-post.html - This view will display the form to create a new post.

d. edit-post.html - This view will display the form to edit an existing post.

e. search-results.html - This view will display the search results for posts.

Should have a dashboard with a black background. The dashboard should have an icon on the left to identify the page as a movie blog, then towards the right side, it should have the Home button, About button, Movie Reviews button, Guest Reviews button, Trailers button, blog button and ContactUs Button. At the far right, there should be a login and sign up buttons, when you click the login button, it proceeds to give a login page and a signup option below, just like in gmail. Above the dashboard there should be the name – Secure Movie Blog – coloured in Blue with a description below it – A secure, usable and accessible web-based movie blog site that mitigates various security vulnerabilities. The search functionality should be on the right and have a placeholder of search. Below the dashboard there should be banners of pictures of various movies as links to the particular movies then add relevant footer notes below the page

So, in this system I have already created the backend functionalities. In the frontend I have thought of mayeb having the following files, index.html, create-post.html, edit-post.html, login.html, search-results.html, verify-2fa.html, about.html, contact.html,sign-up.html, movie-reviews.html and blog.html. Do you think theser ea the rigth files to have or some of them can be combined to reduce the number of files and provide a godo user epeerience?

**index.html**: The main landing page of the movie blog site.

**auth.html**: Combined login and sign-up page for user authentication.

**create-edit-del-post.html**: Combined page for creating new posts, editing and deteling existing ones.

**search-reviews.html**: Page for displaying search results and movie reviews.

**account-settings.html**: Page for managing user account settings, including 2FA configuration and verification options.

**about.html**: Page with general information about the movie blog site.

**contact.html**: Page with contact details and a form for users to get in touch.

With the e blow app.js code, is the server connected to the frontend for seamless operation? Does this system handle this - I want to make it such that if no one is not loged in, he/she cant access the create, edit and delete post functionalities. I want to set the correct routes for the page rendering and harden the site. Make it such that if you’ve never created an account, you can’t log in and once your account is created, you can use your details stored in the mysql database to log in after being authenticated by the security measures in place. What files should be modified for these to happen?