B3 - Production Server with Real-Time Web

Name: Marteinn Hjaltason
Student id: mh226eh

1 Security Measure

To secure both the codebase and the application server, I followed the OWASP Top 10 checklist as a general guideline. Some issues like injection and SSRF were not directly relevant to the scope of this app, but I still implemented several key measures:

In code:

- **HTTP Headers:** The Express server uses the helmet middleware to set secure HTTP headers that help protect against common attacks like cross-site scripting (XSS), clickjacking, and content sniffing, with minimal configuration needed.
- **CSRF Protection**: In the GitLab OAuth flow, a state parameter is used to prevent CSRF attacks. This ensures that the callback is only accepted if it matches the state set during the initial request. This is explicitly recommended by gitlabs documentation on the OAuth usage and implementation.
- **Input Validation**: Since comments are the only place where user input is accepted I paid special attention to sanitizing and validating the input before processing. This meant checking that the comment is in fact a string, that it is at least 1 character and less than 1000:

```
const { comment } = req.body
    if (typeof comment !== 'string' || comment.length === 0 || comment.length > 1000) {
    return res.status(400).send('Invalid comment')
    }
```

- **Secure Cookies**: Session cookies are set with the secure flag enabled, meaning they are only transmitted over HTTPS. They are set as one of the parameters of the session middleware in express.

On the Server:

- **HTTPS Setup**: TLS certificates were obtained using Let's Encrypt and Certbot, ensuring encrypted traffic.
- **Reverse Proxy**: NGINX is used as a secure reverse proxy to separate the app from direct external access. It handles HTTPS termination and forwards traffic to the Node.js process, improving security and performance.

2 Key components and their Purpose

The main setup of the production server follows the steps provided in the lecture videos on the coursepress page, they include the following configuration of components:

Reverse Proxy

NGINX sits in front of my Node.js app and forwards incoming requests to it. This adds a layer of separation between the internet and the app server. It handles HTTPS (port 443) and HTTP (port 80), while my app listens only on a local port (specifcally I set it to 5050). This setup keeps only safe ports open to the public and hides the internal app server. It also improves performance and security by managing routing and serving static files.

Process Manager

PM2 runs the Node.js app in the background and makes sure it stays online. If the app crashes or the server restarts, PM2 automatically restarts it. It also helps manage logs, making it easier to monitor the app in production.

TLS Certificates

TLS certificates encrypt all data between the user's browser and the server to keep it secure. I used Let's Encrypt with Certbot to set up HTTPS for free. This ensures that sensitive information (like login tokens) is protected from eavesdropping.

Environment Variables

Environment variables are used to keep sensitive data and configuration separate from the source code. I store values like WEBHOOK_SECRET, GITLAB_TOKEN, PROJECT_ID, PORT, GITLAB_CLIENT_SECRET, GITLAB_CLIENT_ID, and GITLAB_REDIRECT_ID in environment variables to avoid hardcoding secrets and to make the app easier to configure across environments.

3 Development vs. Production

The main difference between development and production lies in how the application is started and configured (mostly how I structured my package.json).

- **In development**: I use the npm start:dev script, which runs the server with nodemon and sets NODE_ENV=development. This allows for automatic reloads on code changes and easier debugging.

- **In production**: I use the start script with NODE_ENV=production. This disables debug features and enables optimizations intended for performance and stability. In simple terms it reduces the logging outputs, so that they are only relevant for the runtime operations, such as logging status codes and any errors if they do occur.

4 External Modules

Outside of what has been presented in the course I did not use many third party libraries or modules that differ from what has been exemplified in the teaching materials but here are the essential modules I used:

- **helmet**
  Automatically sets secure HTTP headers (mentioned in the first section of this report).
- **dotenv**
  Motivation: Loads environment variables from a .env file so secrets aren't hardcoded.
  Security: .env is excluded from version control using .gitignore.
- **express-session**
  Motivation: Used to store the GitLab token during the OAuth login flow so it can be reused across requests.
  Security: The session cookie is marked as secure and httpOnly, meaning it's only sent over HTTPS and cannot be accessed via JavaScript in the browser.
- **ws**
  Motivation: Enables real-time updates using WebSockets.
  Security: Only handles specific, limited message types; doesn't expose sensitive data.
- **morgan**
  Motivation: Logs requests in development for debugging.
  Security: Disabled in production to avoid leaking sensitive data.
- **cross-env**
  Motivation: Ensures NODE_ENV can be set consistently across OSes. I needed this because I am developing on a windows machine and wanted to be able to test and run locally throughout development.
  Security: No risk; used only for script execution.
- **nodemon** (development only)
  Motivation: Auto-reloads the server during development.
  Security: Not used in production.

5 Additional Features (higher grade)

To aim for a higher grade, I implemented several features that extend the functionality and improve user experience. For each, I also considered how to make it secure for production use. Most additional features are handled through routes, since I decided to implement the server side code in the MVC pattern to keep things structured and well organized and manageable.

Richer User Interface

Motivation: Improves usability by making the app more visually clear.
Security: Minimal frontend logic, its mainly css implementations that make the user interface richer. The app is still a single page application, only the DOM is updated and user has no input availability except commenting, which was explained in the 1 section on security.

Close Issue Button

Motivation: Gives users control over issue by closing them directly from the app.
Security: The server uses a token limited to one specific project. Only known issues can be closed, and the action goes through a POST request to reduce misuse. Since the repo is public and the token is scoped, the risk is low, but further restrictions could be added if needed.
Implementation: The implementation of this extra feature was more or less no different from implementing the issues features. I simply fetch the issue by issue ID and then using a PUT http request I set the state_event to 'close'. This process is initiated when the user presses a “close” button that redirects to router post method with the route being ‘/issues/:id/close’.

Commenting on Issues

Motivation: Enables more complete issue management from within the app.
Security: Input is strictly validated on the server (type, length), and requests are sent securely through GitLab's API.
Implementation: For this feature, again it starts by the user pressing a send button that redirects to a post route defined in the server code. The route calls a controller method that handles the comment logic, where the comment content is contained in the request body, validated and then the comment is updated on gitlab by using fetch with a post method, containing a body of the JSON string with the comment itself.

Commit Integration

Motivation: Provides context on recent activity, making the app more informative.
Security: Commit data comes from GitLab webhooks; no user input involved, and only data from the configured repo is shown (unless the user is authenticated via GitLab Oauth).
Implementation: Commits were simple to implement because just like like the issues they only

need to be rendered to the view using the same webhook, websocket strategy. I made a route /commits that calls a getter method that simply captures the token and project ID in order to fetch the commit with the projectID parameter and bearer token in the fetch header. The websocket on the client side is also listening for broadcasts of commits.

GitLab OAuth

Motivation: Offers a more seamless and secure login experience compared to basic auth. Allows access to private resources.
Security: OAuth flow includes CSRF protection using a state parameter. Tokens are stored securely in memory/session and never exposed to the client. All communication is encrypted over HTTPS.
Implementation: When a user clicks "Log in with GitLab," they are redirected to GitLab's authorization page via the /auth/gitlab route. A randomly generated state value is attached to the request and stored in the session to prevent CSRF attacks. After the user authorizes the app, GitLab redirects them back to /oauth/callback with an authorization code and the same state. The app checks that the state matches, then exchanges the code for an access token from GitLab's /oauth/token endpoint. The access token is stored in the session (never sent to the client), and the app fetches the user's GitLab profile to display their name and avatar. Authenticated users can then list their projects (/projects) and select one to add a webhook to (/projects/:id/webhook). The session also tracks which project the user chose. On the frontend, the login state is checked via /me, and the UI updates to either show a "Log in" link or a welcome message with "Log out."

6 Evaluation

Overall I am very satisfied with my app as I spent a lot of time on it and after awhile I realised that it is easy to always want more features and more refinements. I think this project made me realise and feel that there is no single point where I realise that the app is "complete" but rather a point where I have to stop and say that it is "good enough". Given more time I would have enjoyed implementing way more user centric features, such as tags, release, seeing branches, ability to see and filter closed issues and reopen them, etc. I am especially satisfied with my working implementation of the authentication strategy using gitlab's OAuth. I think it makes the app a lot more interesting to use and to see the individual projects loaded onto the app.