Midterm Project: Wild West Forum

COS 498 – Fall 2025

October 29, 2025

Due date: Nov 7th, 11:59pm

Overview

Build an *extremely insecure* web forum as a stepping stone before we get to building more secure applications with databases. The application allows users to create minimal "accounts," log in via a session cookie, and post comments. Views must be implemented using Handlebars with partials (e.g., a navigation bar). The entire stack must be containerized using Docker and the project must be maintained in a Git repository.

Core Requirements

- **Accounts (intentionally weak):** Users register with a username and password. Credentials are stored server-side in a simple in-memory array (no hashing, no salting, no database).
- **Login:** After successful login, the server sets a session cookie indicating the user is logged in (no secure flags or cryptographic signing required for this assignment).
- **Comments:** Authenticated users can submit a comment via a form. The server stores both the author (username) and the comment text in memory.
- **Feed Page:** Provide a single page that lists all comments along with their authors.
- **Views:** Use Handlebars for templates and Handlebars partials for shared UI, such as a navigation bar.
- **Docker:** Provide a Dockerfile (and optional docker-compose) to run the full application in containers.
- **Git:** The project must be a Git repository with meaningful commit history.

Intended Insecurity (for pedagogy)

This is a *deliberately insecure* app. You **should not** harden it beyond the specification. We will later analyze the vulnerabilities. Examples of expected insecurity include: plaintext passwords, insecure session cookies, missing CSRF protection, XSS/templating pitfalls, and lack of input validation.

Suggested Technology Stack

- **Runtime:** Node.js
- **Web framework:** Express
- **Templating:** Handlebars (with partials)
- **State:** In-memory arrays for users and comments
- **Containerization:** Docker (Dockerfile; optional docker-compose.yml)

Data Model (in-memory)

- `users`: Array of objects like { `username: string, password: string` } stored in server memory.
- `comments`: Array of objects like { `author: string, text: string, createdAt: Date` } stored in server memory.
- `session`: Object like { `user: string, sessionId: string, expires: Date` } stored in server memory. This will be primarily handled by the express-session middleware and you don't need to do much.

Minimum Routes

- **GET Routes**
  - `GET /` – Home page with nav partial containing links to other parts.
  - `GET /register` – Render the registration form.
  - `GET /login` – Render the login form.
  - `GET /comments` – Show all comments and authors.
  - `GET /comment/new` – Render the new comment form (only useful when logged in). If the user is not logged in, show login form instead.
- **POST Routes**
  - `POST /register` – Create account by pushing {`username, password`} into the in-memory users array. If the username is already taken, show an error message.
  - `POST /login` – Authenticate and set a session cookie (e.g., `loggedIn=true; user=...`). If the username or password is incorrect, show an error message.
  - `POST /logout` – Clear the session cookie to log the user out. Also clear the session object from server memory.
  - `POST /comment` – Accept and store a new comment with author and text (requires logged-in flag and a valid session object).

Views and Partials

- Use Handlebars layouts and partials. At minimum, create a `nav` partial and a `footer` partial used across pages.
- The nav partial should contain links to the home, some indication of the logged-in user, and links to the register, login, and logout pages (depending on if the user is logged in or not).
- Pages: home, register, login, comments list, new comment form.

Session Cookie Behavior

- On login, set a session cookie that indicates authentication status and username. Also set a session object in server memory with the username, sessionId, and expires date.
- On logout, clear the session cookie and the session object from server memory.
- For this assignment, do not sign, encrypt, or secure the cookie.

Dockerization

- There should be two containers: one for the nginx that will handle routing and one for the nodejs server that will handle the server-side logic.
- Decide which container will serve the static files. If you have nginx handle the static files, you will need to modify the nginx configuration to route required traffic to the nodejs container and serve static files otherwise. If you want to have the nodejs serve the static files, you have nginx route all traffic to it and the express needs the static file middleware (this is the easier option).
- Each container should have its own `Dockerfile` that installs dependencies, copies the app, and launches the server.
- A `docker-compose.yml` to run the app, expose the web port, and link the two containers together.

Git Repository

- Initialize a PUBLIC Git repository at project start.
- Commit at meaningful milestones (initial scaffold, views, auth, comments, Docker).
- Include a `README.md` with run instructions.

Brightspace Submission

- Link to git repository (include `README.md`).
- Link to website where the app is hosted (needs to stay up, so pick a unusual port).
- A link to a video where you demonstrate the following:
  - While logged into your server through ssh, you clone your repository.
  - You edit the port that the server is running on and build the containers.

  - Navigate to the website and demonstrate the app functionality.
  - In an IDE such as VSCode, change one of the views (it can be small, like changing the background color or some text).
  - Rebuild the containers and navigate to the website again to demonstrate the changes.

Recommended Steps

- Create a repository on github first, make sure it's public. Use the Nodejs default .gitignore
- Clone the repository to the server, create the two directories that you'll need (one for the nginx and one for the nodejs). Push those changes to github to make sure the git workgflow is working.
- Work on getting a development version of the docker containers working without worrying about the content of the site yet.
  - Create a Dockerfile for the nginx container.
  - Create a Dockerfile for the nodejs container.
  - Create a basic nodejs express server.js along with some basic static files. Place the static files in the docker container that is expected to serve them. Depending on your setup, this could be the nginx container or the nodejs container.
  - Create a docker-compose.yml file that links the two containers together.
  - Build the containers and start them.
  - Test that the website is accessible and that the static files are being served.
  - Test that you can bring down the container, make a change, bring back up the server, and the change is reflected.
- Work on getting the content of the site working.
  - Create the views and partials. Test them.
  - Create the routes and test them.
  - Fill your comment object with fake information, get it to properly show on the site
  - Get your registration to work, confirm that a new user is added to the internal object.
  - Get your login to work, confirm that the session cookie is set and the session object is set in the server memory.
  - Get your logout to work, confirm that the session cookie is cleared and the session object is cleared from the server memory.
  - Get your comment posting to work, confirm that the comment is added to the internal object.
  - Get your comment listing to work, confirm that the comments are displayed on the site.
  - Get your registration error message to work, confirm that the error message is displayed on the site.
  - Get your login error message to work, confirm that the error message is displayed on the site.

  - Get your comment posting error message to work, confirm that the error message is displayed on the site.
  - Get your comment listing error message to work, confirm that the error message is displayed on the site.
- Funcionality should be done. Test it all.
- Make the site look nice with some css and images.
- Make a docker-compose.yml file that builds the containers without mounting the volumes. Test it.

Grading (out of 50 points)

- **Functionality: 30 points**
  - Login/Logout/Register: 10 points
  - Comment Posting: 10 points
  - Comment Listing: 10 points
- **Dockerization: 10 points**
  - Docker containers are used: 5pts
  - Nodejs and nginx are seperate containers: 3pts
  - docker-compose.yml file that builds the containers without mounting the volumes: 2pts
- **Git: 5 points**
  - Git repository is used and is public: 2pts
  - Git repository has a meaningful commit history: 3pts
- **Consistent Code Commenting: 5 points**
- **The site makes my eyes bleed (or no attempt was made to make it look nice): lose 5pts**

AI Use

You are allowed to use AI to help you with the html and styling.