Uniconn: My Implementation of a Simple Web Forum Application

Kavish Sathia
kavishwer@u.nus.edu

12th December 2024

1 Features

These are the features that I have included in Uniconn. To try out these features, go to: Uniconn Web (You can click on "Lurk" to skip the authentication process):

- Account-based authentication with JWT stored in a HTTP Cookie.
- Write threads using markdown and include a thumbnail picture for more engagement.
- Real-time notifications when the user's thread or comment is liked or commented on.
- Nested comments (up to 5 levels).
- Mobile responsiveness
- Progressive web app (allowing users to install the web app onto their devices).
- Searching for forums based on title, description, tags and author username.
- A point system (Aura) that will increase when a user's thread or comment gets liked. This will incentivize participation.

These are features that I plan to include or is currently in progress:

- Content moderation using AI and community feedback.
- AI-generated descriptions for threads
- Communities feature with each community having its own theme and color scheme
- I'm also planning to write unit tests for the backend and end-to-end tests for the frontend using cypress.io

2 Tech Stack

The frameworks and languages used follows the assignment guidelines, using React and Typescript for the frontend and Golang with the Gin web framework for the backend. I will list additional libraries and the reason I used them below:

- React Router: For frontend navigation and handling different routes.
- TailwindCSS: For inline styling.
- MUI: Used by CVWO, so I tried it out.
- MDXEditor: For the markdown editor.
- Lucide-React: For iconography.
- Gin: For the web framework.

- GORM: Used as the ORM to query the database.
- Redis-Go: To maintain PubSub connections to the Redis server.

I used two databases for this assignment:

- PostgreSQL: To store application data. Threads, comments and user data is stored here. (I will include an ERD diagram in Section 4).
- Redis: As the PubSub server to publish and listen to real-time notifications from user activity. (I will explain this further in Section 5).

3 User Interface

There are 4 interfaces for larger screens and 5 interfaces for smaller screens. These consist of:

- Login interface
- Sign up interface
- Thread interface (this includes the comments)
- Thread editor interface
- Thread list interface (this exists as a sidebar in larger screens, but exists as an interface of its own on mobile screens)

4 Data Model

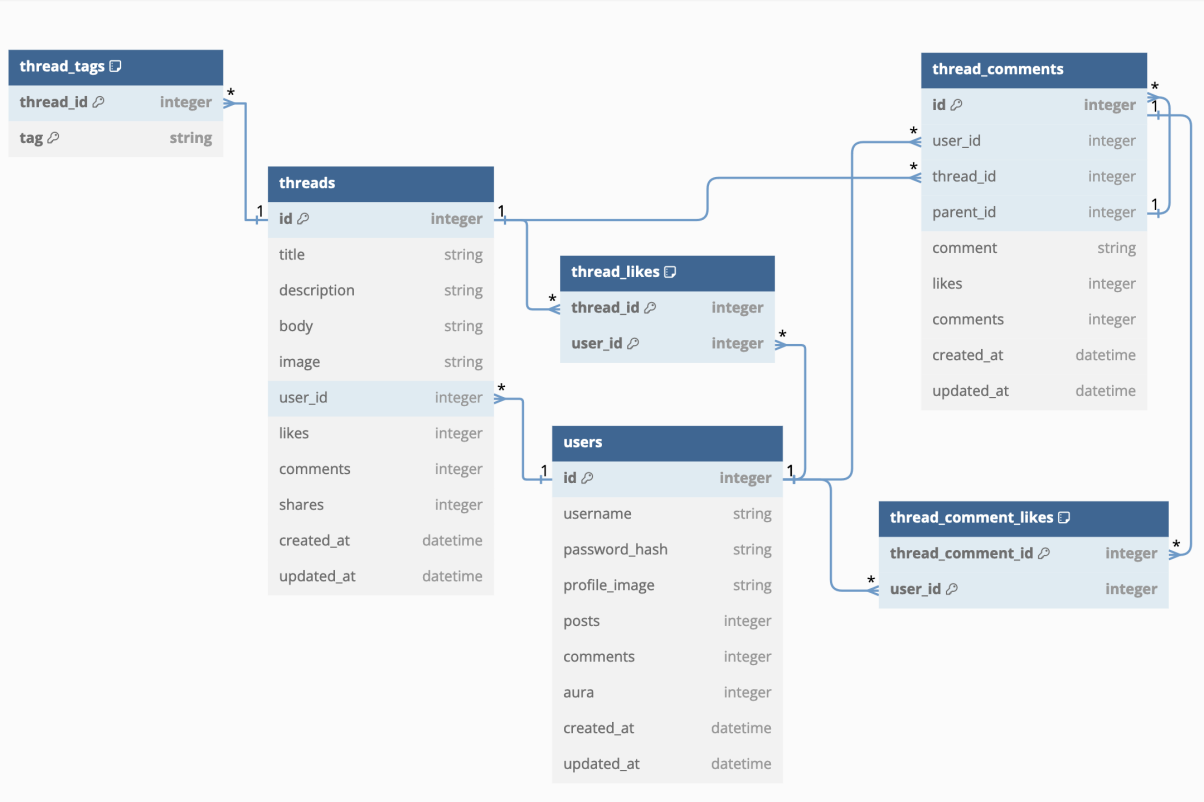

Figure 1: The ERD diagram for Uniconn's relational database

5 Architecture

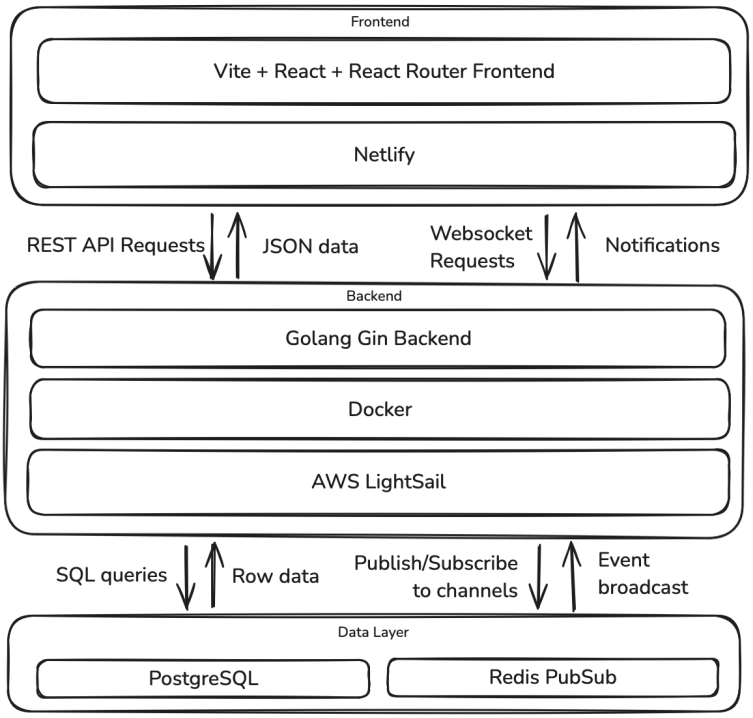

Figure 2: The tech architecture for Uniconn

The architecture consists of 3 segments: the frontend which is hosted on Netlify, the backend which is hosted on AWS Lightsail as a Docker container and the data layer which has two databases, PostgreSQL and Redis. The Redis database is not used for data storage but for its PubSub capabilities. I will now outline how the real time notification system works: For each user, there exists a channel on the Redis server whose name is the user ID of the user to which it corresponds. Every time an action has been taken on a user's post or comment, a message will be published to the channel that corresponds to the user. If the user is connected to the WebSocket endpoint on the Gin backend, the WebSocket endpoint will read the published message and send it to the user through the Websocket connection. When the frontend receives the message, it will display a notification.

6 Challenges

1. There were some minor technical challenges due to the cross origin auth cookie as the frontend and backend are operating on different domains. Different browsers seem to handle these type of cookies differently. Initially, the cookie only worked on Firefox and after some minor changes it works on Chrome as well. It still does not work on Safari.

2. I am not very proficient at designing frontends, so the UI/UX took me a while to get (sort of?) right. I have settled on a decent looking frontend for now but I might choose to change how it looks later on.

3. Not really a challenge but the code looks like it was written by a competitive programmer right now (I am not one). So, over the next two weeks, I will focus less on the features and more on the code quality by refactoring the codebase and following SOLID principles as much as possible.

7 Appreciation

Thanks to the CVWO Team for this awesome assignment. Although I had some previous experience, this assignment allowed me to fill in the gaps in my knowledge and push my boundaries.