CS349 - Database and Information Systems Lab

CodeQuest - A Competitive Programming Platform

Ananya Rao (22B0980)
Deeksha Dhiwakar (22B0988)
Sharvanee Sonawane (22B0943)
Sanchana Arun (22B1034)

May 2025

1 Git Repository Link

The code for our project can be found here: https://github.com/Sharvanee/DBIS-Project.git

2 Goals of Project

The primary goals of our project are as follows:

- To develop a fully functional competitive programming platform called CodeQuest (inspired from already existing platforms such as Codeforces or LeetCode).
- To provide users with an interactive and user-friendly interface to learn in-demand programming languages, practice problem solving, participate in contests, track their performance, create custom contests, write blogs, and engage with the like-minded coding community.
- To implement a robust database schema that efficiently manages users, contests, problems, and submissions.
- To ensure secure authentication, user session management, and data privacy.

3 User Perspective Implementation

The following pages have been implemented:

3.1 Login/Signup

The login/signup page provides users with a simple and intuitive interface to access the platform. Users can enter their unique handle and password to log in. A "Remember me for a month" checkbox allows session persistence across visits. Password visibility can be toggled for convenience, using an eye icon. Upon submission, credentials are securely sent to the backend using a POST request with proper headers and session cookies. If authenticated, users are redirected to their dashboard. Error messages are displayed for invalid credentials or server issues. New users can navigate to the signup page via a clearly visible link.

3.2 Dashboard

Once users log in, they land on a personalized dashboard that gives them a snapshot of everything happening on the platform. They're greeted by name and can quickly access ongoing, upcoming, or past contests, all grouped clearly for easy browsing. The most relevant contests appear upfront, and users can click through categories like "graphs" or "greedy algorithms" to find problems that match their interests. They also see recent blog posts, including short previews, to stay updated with community activity.

3.3 Profile

The profile page lets users see and reflect on their own progress and presence within the platform. Personal information like display name, email, and institution is shown, alongside performance stats such as rating and problems solved. A highlight of the page is the activity heatmap — a visual way to see daily problem-solving streaks and consistency. Users can edit their profile details and view their avatar (or initials, if no image is set), helping make the space feel more personal and informative.

3.4 Contests List

This page offers users a clean, categorized list of all contests. They can quickly spot which ones are active, what's coming up, and what's already happened. Each listing includes key details like timing and registration options. For active contests, users can track how much of it they've completed. Pagination helps them explore further, and everything is organized to help users decide what to participate in next.

3.5 Contest

When users view a specific contest, they're given all the essential info at a glance: when it runs, how much time is left, and what problems it includes. A live countdown adds urgency, and users can jump into problem-solving if they're registered. Each problem shows difficulty and community stats (like total and accepted submissions), and a leaderboard shows where users stand. It's an engaging, competitive environment designed to keep users focused and informed.

3.6 Problem Set

This page is where users browse and search the full catalog of problems. It's built to be easy and fast — users can filter by title, tag, or difficulty and sort problems however they like. Tags like "math" or "dynamic programming" help them dive into specific topics. Problems are laid out in a table that's easy to scan, and pagination helps manage large lists. It's all about helping users find the right challenge.

3.7 Problem

On a problem page, users see everything they need to solve a challenge: the problem description, examples, and a built-in code editor that supports popular languages like C++, Python, and Java. They can write code directly in the editor or upload files, and test cases can be run with a click. If the problem is part of a contest, users see a timer and may have limited access depending on the rules. Submissions are tracked in a table so users can monitor progress, and solutions become visible once contests end.

3.8 Add Contest

For users who want to create contests, this page provides a step-by-step form to set everything up. They can define timing, add problems with detailed specifications, and even include solutions or upload files. The interface guides them through building the full contest experience, including test cases and problem metadata. Once submitted, they get immediate feedback and are taken back to the contests page. It's designed for ease of use, even when building something complex.

3.9 Blogs

This page gives users access to all community blog posts, with the newest content always appearing first. Each blog shows the title, author, and a short preview so users can decide what to read. A floating button makes it easy for contributors to start writing their own blog. If there are no posts yet, users see a friendly fallback message encouraging content creation. Users can like, dislike, add comments and view other users' comments on each blog post. It's a space for learning, sharing, and building community.

4 Solution Architecture

We have formulated our solution based on the following ER diagram that represents our database schema.

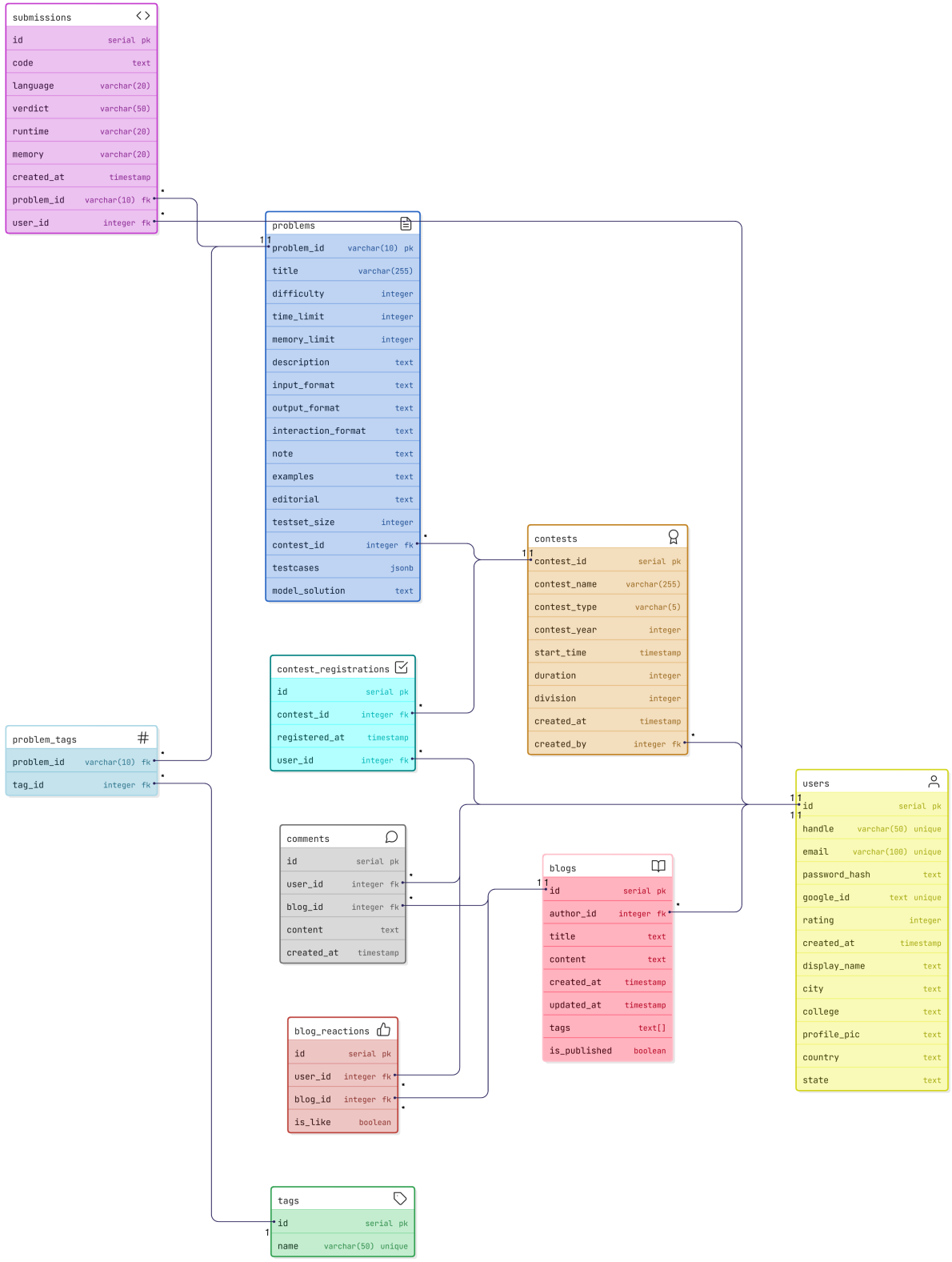

Figure 1: ER Diagram for Database

5 Functionalities Implemented

5.1 Frontend

Our frontend was built using **React.js** to enable a dynamic application structure. We used **custom CSS** for styling and component design.

- **Authentication Flow:** The login and signup pages include input validation, password visibility toggles, and a "Remember Me" option.
- **Dashboard and Profile:** The dashboard dynamically fetches user-specific data such as upcoming/ongoing contests and blog previews. The profile page includes editable user info and an activity heatmap using a React charting library to visualize daily problem-solving.
- **Code Editor Integration:** We integrated the open-source `@monaco-editor/react` library (based on VSCode's editor) to allow users to write, edit, and run code directly in the browser. It supports syntax highlighting, autocomplete, and multiple language modes (C++, Java, Python).
- **Problem Page and Submissions:** Each problem has a description, examples, and a Monaco-powered editor. Users can run test cases or submit final solutions. A submission history table displays verdicts and language.
- **Contests and Leaderboards:** Users can register for contests, view timers and participate live. Leaderboards are automatically updated based on submission results.
- **Blogs and Community Features:** The blog interface supports Markdown input and previews, allowing users to post articles.
- **Routing and Navigation:** React Router is used for client-side routing.

5.2 Backend

The backend of our platform is implemented using **Node.js** with the **Express** framework. It acts as an API server that communicates with a MySQL database, enabling core functionalities such as authentication, contest management, and submission handling.

Key backend features include:

- **Authentication APIs:** Implements login and signup routes using secure password hashing (bcrypt). Upon successful login, user sessions are tracked using HTTP-only cookies.
- **User Session Management:** The backend supports persistent login via session cookies. Middleware ensures route protection, allowing access to restricted pages only for authenticated users.
- **Problem and Contest Retrieval:** Allows retrieval of:
  - All problems (with filtering by tag, difficulty, title)
  - All contests (upcoming, ongoing, past)
  - Contest-specific problems and metadata
- **Submissions Handling:** Handles code submission by users. Stores verdicts, code, language, and problem ID. Submissions are processed (simulated) and results returned. Post-contest, users can view solution verdicts and codes.
- **Add Contest API:** Allows contest creators to define new contests, set time windows, and upload problems with metadata, test cases, and expected outputs. Includes form validation and proper relational integrity checks with the database.
- **User Profile and Activity:** Provides endpoints to fetch and update user profiles. Includes a route to compute and return activity data for the heatmap on the profile page.
- **Rating and Leaderboards:** Backend calculates rating updates based on contest performance and stores them per user. Contest-specific leaderboards are served as ranked lists with usernames, scores, and submission stats.

- **Blog APIs:** Users can post blogs, retrieve all blog previews, and read full posts. Blogs are stored with Markdown support and timestamps.

All routes return JSON responses. CORS is configured to allow secure cross-origin requests between frontend and backend during development and deployment.

6 Implementation Tools

6.1 Dataset

In this project, for problems and contests we use the publicly available dataset from this link. It contains 10024 rows of data, where each row has the details of a CodeForces problem such as the problem ID, contest ID, difficulty, tags, description, testcases etc.

To populate the `users` table, we generate fake user data (including email ID, username, location, joining date etc) using a Python script.

For submission data, we use the publicly available dataset from this link. It has been generated based on actual user submissions till the end of 2024, using aliases for all the users to protect their anonymity. It contains several details of the submissions such as the code, verdict, runtime, programming language etc.

6.2 Code

- Schema design and SQL, Psycopg queries - Implemented based on material learnt in CS317/349.
- Node JS and React code - Implemented based on material learnt in CS317/349, along with references from official documentation Node JS, React etc.
- Certain custom features and styling - GenAI (ChatGPT).

7 Conclusions and Future Work

We have successfully implemented CodeQuest, a competitive programming site inspired from existing sites like CodeForces and LeetCode, replicated all the key features, and added custom features which improve upon the existing sites. Through comprehensive testing we have verified that our implementation works correctly and is able to handle various corner cases elegantly.

This project can be further expanded upon in the following ways:

- **Google Sign In:** We tried implementing a sign in using google as an alternative provided to users for login. We used OAuth 2.0, created a OAuth Client ID for a Web application and set Authorised redirect URIs to http://localhost:3000. This completes the frontend integration; however, the backend requires implementing user authentication using Passport.js along with Express sessions to handle authorization, which we could not finalize. Enabling Google Sign-In would allow users to log in without manually entering a handle and password.
- **Mobile App Support:** While the current platform is web-based, developing a dedicated mobile application would improve accessibility and usability for users on the go, especially for tracking contests or submitting solutions from mobile devices.
- **Social Features:** Adding features like user following, messaging, or collaborative problem-solving rooms could foster stronger community engagement.
- **Contest Reminders and Notifications:** Implementing an in-app and/or email notification system for upcoming contests, submission results, or blog activity would help keep users informed and engaged.
- **Gamification Elements:** Adding badges, achievements, or streak rewards would increase motivation and user retention.

- **Version control in IDE:** Users prefer to code on local IDEs as they provide a version control feature but the existing platforms do not. So to encourage users to use the IDE CodeQuest provides, we aspire to add a Version control feature that allows users to access previous iterations of their code (inspired from VS Code timeline feature)