Sprint 2 Artifacts & Requirements

Overview

Sprint 2 focuses on **user authentication, profile management, and daily coding problem functionality**. The backend will be implemented in Flask with SQLite, and the frontend will use React (Vite) with Tailwind CSS for styling. Security, modularity, and maintainability remain priorities.

Functional Requirements (Sprint 2)

2.1 Implement JWT-based authentication system for signup/login

- **Owner:** Daniel Neugent
- **User Story:** As a user, I want to securely sign up and log in so that my data and progress are protected.
- **Functional Requirements:**
    - FR2.1.1: Implement Flask endpoints for signup and login.
    - FR2.1.2: Use JWT tokens to manage session authentication.
    - FR2.1.3: Ensure tokens are securely signed and verified for each request.

2.2 Add bcrypt password hashing for user security

- **Owner:** Brett Balquist
- **User Story:** As a user, I want my password to be securely hashed to prevent exposure of sensitive information.
- **Functional Requirements:**
    - FR2.2.1: Hash passwords using bcrypt before saving to SQLite.
    - FR2.2.2: Validate hashed passwords on login.

    - FR2.2.3: Ensure salts are unique per user.

2.3 Build user profile dashboard with streaks and basic stats

- **Owner:** Tej Gumaste
- **User Story:** As a user, I want to see my coding streaks and statistics on my profile so I can track my progress.
- **Functional Requirements:**
    - FR2.3.1: Display current streak, total problems solved, and success rate.
    - FR2.3.2: Fetch user data securely from the backend using authenticated API calls.
    - FR2.3.3: Update dashboard dynamically when new problems are solved.

2.4 Develop backend logic for daily coding problem release and lock

- **Owner:** Jay Patel
- **User Story:** As a user, I want a new coding problem released daily, with previous days locked, to maintain challenge integrity.
- **Functional Requirements:**
    - FR2.4.1: Schedule daily problem availability using Flask logic.
    - FR2.4.2: Lock previous problems to prevent edits/submissions.
    - FR2.4.3: Store problem release timestamps and track user submissions in SQLite.

2.5 Create frontend interface for daily challenge with timer

- **Owner:** Arnav Jain
- **User Story:** As a user, I want a clear interface with a timer for daily challenges to manage my coding sessions.

- **Functional Requirements:**
    - FR2.5.1: Display daily challenge description, input area, and timer.
    - FR2.5.2: Prevent submission after the timer ends.
    - FR2.5.3: Provide feedback for correct/incorrect submissions.

2.6 Implement backend endpoints to evaluate code submissions

- **Owner:** Daniel Neugent
- **User Story:** As a user, I want my code submissions automatically evaluated so I can receive instant feedback.
- **Functional Requirements:**
    - FR2.6.1: Create Flask endpoints to submit and evaluate code securely.
    - FR2.6.2: Execute code in an isolated environment (sandboxing).
    - FR2.6.3: Store submission results in SQLite and return feedback to frontend.

2.7 Ensure secure handling of authentication and submissions

- **Owner:** Brett Balquist
- **User Story:** As a user, I want my credentials and code submissions handled securely to prevent unauthorized access.
- **Functional Requirements:**
    - FR2.7.1: Validate JWT tokens for all sensitive endpoints.
    - FR2.7.2: Sanitize code inputs to prevent injection attacks.
    - FR2.7.3: Log suspicious activity for monitoring.

2.8 Implement reusable frontend components for challenge display and stats

- **Owner:** Tej Gumaste

- **User Story:** As a user, I want a consistent UI for challenges and stats to improve usability and maintainability.
- **Functional Requirements:**
    - FR2.8.1: Create React components for challenge card, timer, and stats display.
    - FR2.8.2: Use Tailwind CSS for consistent styling.
    - FR2.8.3: Ensure components are modular and reusable for other features.

2.9 Implement unit and integration tests for authentication and problem logic

- **Owner:** Jay Patel
- **User Story:** As a developer, I want tests for authentication and daily problem logic to ensure reliability and prevent regressions.
- **Functional Requirements:**
    - FR2.9.1: Write Flask unit tests for signup, login, and JWT verification.
    - FR2.9.2: Write tests for daily problem release and submission evaluation logic.
    - FR2.9.3: Integrate test execution with GitHub Actions for CI/CD.
