Team 10
Team Members: Daniel Neugent, Brett Balquist, Tej Gumaste, Jay Patel, Arnav Jain
Project Name: Leetle
Project Synopsis: We are creating a webapp where people can do daily Leetcode questions similar to Wordle.

Leetle Requirements Artifacts (Sprint 3)

1. System Overview

Sprint 3 focuses on advanced UI features, code editor integration, leaderboard, difficulty tagging, and admin dashboard. The backend continues with Flask + SQLite, while the frontend uses React (Vite) with Tailwind CSS for styling. Emphasis is on modular, reusable components, smooth user experience, and secure handling of user data.

2. Functional Requirements

3.1 Integrate Monaco Editor for in-browser coding with syntax highlighting
Owner: Daniel Neugent
User Story: As a user, I want an in-browser code editor with syntax highlighting to solve coding challenges efficiently.
Functional Requirements:

- **FR3.1.1:** Integrate Monaco Editor into React frontend.
- **FR3.1.2:** Ensure editor supports multiple languages (e.g., Python, JavaScript).
- **FR3.1.3:** Enable resizing and responsive layout for different screen sizes.
- **FR3.1.4:** Implement autosave functionality to preserve user code.
- **FR3.1.5:** Provide configurable themes (light and dark mode) for better user experience.

3.2 Add leaderboard with dynamic ranking based on streaks and accuracy
Owner: Brett Balquist
User Story: As a user, I want to see a leaderboard ranking users by streaks and accuracy to encourage friendly competition.
Functional Requirements:

- **FR3.2.1:** Create Flask endpoint to fetch leaderboard data.
- **FR3.2.2:** Dynamically calculate rankings based on streaks and success rate.
- **FR3.2.3:** Display leaderboard in a React component with responsive design.
- **FR3.2.4:** Include filters for time period (daily, weekly, all-time).

- **FR3.2.5:** Highlight the logged-in user’s position on the leaderboard.

3.3 Implement difficulty tagging (Easy, Medium, Hard) for challenges
Owner: Tej Gumaste
User Story: As a user, I want challenges labeled by difficulty so I can choose problems appropriate for my skill level.
Functional Requirements:

- **FR3.3.1:** Add difficulty field to SQLite challenge table.
- **FR3.3.2:** Provide backend filtering endpoints by difficulty.
- **FR3.3.3:** Display difficulty tags clearly in React components.
- **FR3.3.4:** Allow admin to assign difficulty levels when creating challenges.
- **FR3.3.5:** Enable frontend sorting and filtering by difficulty level.

3.4 Build admin dashboard for CRUD on questions and user progress
Owner: Jay Patel
User Story: As an admin, I want to manage challenges and track user progress to maintain the platform effectively.
Functional Requirements:

- **FR3.4.1:** Implement Flask endpoints for creating, updating, and deleting challenges.
- **FR3.4.2:** Track user progress and streaks in SQLite.
- **FR3.4.3:** Create React admin dashboard with modular, reusable components.
- **FR3.4.4:** Ensure admin routes are protected with authentication and authorization.
- **FR3.4.5:** Provide admin filters to view active and inactive users.

3.5 Add streak tracker logic on backend and frontend
Owner: Arnav Jain
User Story: As a user, I want a streak tracker to see how many consecutive days I’ve solved problems.
Functional Requirements:

- **FR3.5.1:** Implement backend logic to calculate streaks based on submission timestamps.
- **FR3.5.2:** Display streak count dynamically on frontend components.
- **FR3.5.3:** Update streak logic in real-time after successful submissions.
- **FR3.5.4:** Include longest streak tracking in user stats.
- **FR3.5.5:** Provide a visual indicator (flame icon or progress bar) for active streaks.

3.6 Conduct testing of Monaco Editor integration and debugging
Owner: Daniel Neugent
User Story: As a developer, I want to test the Monaco Editor integration to ensure it works

reliably across browsers and devices.
Functional Requirements:

- **FR3.6.1:** Write unit and integration tests for editor functionality.
- **FR3.6.2:** Verify syntax highlighting, code execution, and resizing features.
- **FR3.6.3:** Log errors and debug issues in development and staging environments.
- **FR3.6.4:** Test autosave and theme-switching functionalities.
- **FR3.6.5:** Ensure consistent performance across browsers and screen resolutions.

3.7 Implement reusable React components for leaderboard, admin dashboard, and challenge display

Owner: Brett Balquist
User Story: As a developer, I want consistent, reusable components to simplify UI updates and improve maintainability.
Functional Requirements:

- **FR3.7.1:** Modularize components for leaderboard, challenges, and admin tools.
- **FR3.7.2:** Use Tailwind CSS for styling consistency.
- **FR3.7.3:** Ensure accessibility and responsive design across devices.
- **FR3.7.4:** Document component usage for developer reference.
- **FR3.7.5:** Optimize components for performance and reusability.

3.8 Ensure secure backend handling of admin endpoints and user data

Owner: Jay Patel
User Story: As a user or admin, I want sensitive actions protected to prevent unauthorized access.
Functional Requirements:

- **FR3.8.1:** Require JWT authentication for admin routes.
- **FR3.8.2:** Validate input and sanitize data before database operations.
- **FR3.8.3:** Log and monitor suspicious activity in backend.
- **FR3.8.4:** Implement role-based access control for admin and standard users.
- **FR3.8.5:** Use environment-based configuration for database and API keys.

3.9 Add challenge analytics and insights dashboard

Owner: Tej Gumaste
User Story: As an admin, I want to view statistics and insights about challenge engagement to monitor platform performance.
Functional Requirements:

- **FR3.9.1:** Collect data on challenge attempts, success rates, and time spent.

- **FR3.9.2:** Visualize analytics in charts using React.
- **FR3.9.3:** Allow filtering analytics by date range and difficulty level.
- **FR3.9.4:** Store analytics data securely in SQLite.
- **FR3.9.5:** Export reports as CSV or JSON.

3.10 Enhance frontend UI/UX with animations and accessibility improvements
Owner: Arnav Jain
User Story: As a user, I want a smooth and accessible interface that improves usability and engagement.
Functional Requirements:

- **FR3.10.1:** Add animations and transitions using Framer Motion for better interactivity.
- **FR3.10.2:** Ensure all interactive elements meet accessibility (ARIA) standards.
- **FR3.10.3:** Optimize layout and navigation for smaller devices.
- **FR3.10.4:** Include visual feedback for loading states and form submissions.
- **FR3.10.5:** Conduct user testing to validate overall UI/UX improvements.

3.11 Implement achievements and badges system
Owner: Daniel Neugent
User Story: As a user, I want to earn achievements and badges for completing challenges and maintaining streaks to stay motivated.
Functional Requirements:

- **FR3.11.1:** Define achievement categories (e.g., streak milestones, total problems solved).
- **FR3.11.2:** Display earned badges on user profiles.
- **FR3.11.3:** Track badge progress dynamically in the backend.
- **FR3.11.4:** Notify users upon earning new achievements.
- **FR3.11.5:** Allow admins to add or modify badge criteria.

3.12 Implement social sharing features
Owner: Brett Balquist
User Story: As a user, I want to share my daily challenge results or achievements on social media to engage with others.
Functional Requirements:

- **FR3.12.1:** Enable sharing of challenge results to platforms like Twitter and LinkedIn.
- **FR3.12.2:** Generate shareable images or links containing user stats.
- **FR3.12.3:** Provide privacy settings to control sharing options.
- **FR3.12.4:** Log shared activity for analytics.
- **FR3.12.5:** Implement Open Graph tags for enhanced link previews.

3.13 Add backend caching and performance optimization
Owner: Jay Patel
User Story: As a developer, I want to optimize backend performance to reduce load times and improve scalability.
Functional Requirements:

- **FR3.13.1:** Implement caching for frequently accessed endpoints.
- **FR3.13.2:** Optimize SQL queries and database indexing.
- **FR3.13.3:** Add rate limiting for high-traffic API routes.
- **FR3.13.4:** Monitor API performance and latency using logging tools.
- **FR3.13.5:** Set up background jobs for non-critical computations.