Leetle Requirements Artifacts (Sprint 1)

1. System overview

The project will establish a secure and scalable full-stack web application using Python, Flask, SQLite, and React with Tailwind CSS. It will include CI/CD integration, reusable components, and a structured setup to ensure reliability and maintainability.

2. Functional requirements

2.1 Set up Python backend with Flask
Name: Daniel Neugent
User Story: As a user, I want the system to have a backend built with Flask Python so that the application can handle server-side logic and API requests efficiently.
Functional Requirements:

- **FR2.1.1:** The system shall initialize a Flask project with Python installed.
- **FR2.1.2:** The backend shall define a basic server that listens on a configurable port.
- **FR2.1.3:** The routing structure shall be modularized to support multiple endpoints.

2.2 Set up React frontend with Tailwind CSS and responsive navigation
Name: Brett Balquist
User Story: As a user, I want a responsive frontend built with React and styled using Tailwind CSS so that I can access the application seamlessly across devices.
Functional Requirements:

- **FR2.2.1:** The system shall initialize a React project using a modern build tool (e.g., Vite or CRA).
- **FR2.2.2:** Tailwind CSS shall be configured and integrated into the React project.
- **FR2.2.3:** The system shall include a responsive navigation bar that adapts to mobile and desktop views.

2.3 Connect backend to SQLite and verify local connection
Name: Tej Gumaste
User Story: As a user, I want the backend to connect to SQLite so that data can be stored and retrieved reliably.
Functional Requirements:

- **FR2.3.1:** The system shall install and configure an SQLite database for data storage.
- **FR2.3.2:** The backend shall establish a successful connection to a local SQLite instance.

- **FR2.3.3:** The connection status shall be logged in the console for verification.

2.4 Establish folder structure for scalability and modularity
Name: Jay Patel
User Story: As a developer, I want a well-defined folder structure so that the application remains organized, scalable, and easy to maintain.
Functional Requirements:

- **FR2.4.1:** The project shall include separate directories for routes, controllers, models, and utilities.
- **FR2.4.2:** Common configuration files shall be grouped in a `config` directory.

2.5 Initialize GitHub repository and add branch protection rules
Name: Jay Patel
User Story: As a developer, I want a GitHub repository with proper protections so that code integrity is maintained and collaboration is streamlined.
Functional Requirements:

- **FR2.5.1:** The project shall be pushed to a GitHub repository.
- **FR2.5.2:** Branch protection rules shall be configured to require pull request reviews before merging.
- **FR2.5.3:** The repository shall include a `.gitignore` file and a `README.md`.

2.6 Configure GitHub Actions for CI/CD builds
Name: Arnav Jain
User Story: As a developer, I want automated CI/CD pipelines set up so that code changes are tested and deployed consistently.
Functional Requirements:

- **FR2.6.1:** A **GitHub Actions workflow** file shall be added to run Flask unit tests automatically.
- **FR2.6.2:** The workflow shall trigger on pull requests and pushes to main branches.

2.7 Create .env configuration for sensitive keys
Name: Arnav Jain
User Story: As a developer, I want sensitive keys stored in a `.env` file so that configuration data remains secure and separated from source code.
Functional Requirements:

- **FR2.7.1:** Sensitive data such as database URIs and Flask secret keys shall be stored in `.env`.
- **FR2.7.2:** The `.env` file shall be excluded from version control via `.gitignore`.

2.8 Implement API base routes for health check and status

Name: Health Check Endpoints
User Story:
As a user, I want to verify that the backend API is running and healthy so that I can ensure system availability.

Functional Requirements:

- **FR2.8.1:** The backend shall expose a `/health` or `/status` route.
- **FR2.8.2:** The route shall return a JSON object indicating system uptime and operational status.
- **FR2.8.3:** The route shall return a 200 HTTP status for a healthy server.

2.9 Develop initial reusable Jinja2 components

Name: UI Component Setup
User Story:
As a user, I want consistent and reusable UI components so that the interface remains cohesive and efficient to maintain.

Functional Requirements:

- **FR2.9.1:** The frontend shall include reusable components such as buttons, forms, and alerts using Jinja2 templates.
- **FR2.9.2:** Components shall follow consistent styling using Bootstrap classes.

2.10 Ensure secure environment setup for backend

Name: Backend Security Configuration
User Story:
As a user, I want the backend to be securely configured so that the system protects sensitive data and prevents unauthorized access.

Functional Requirements:

- **FR2.10.1:** The backend shall include middleware for input validation and sanitization.
- **FR2.10.2:** The server shall use HTTPS in production environments.

- **FR2.10.3:** Security-related headers shall be configured using Flask extensions such as `Flask-Talisman`.