Project Documentation: Customer Feedback System

# 1. Overview

This project is a full-stack web application that allows users to submit feedback on various products and view feedback history. It features a Flask backend for handling the business logic and data storage, and a React frontend for creating a responsive user interface. The project leverages SQLAlchemy for database management, Flask-Migrate for handling database migrations, and SQLite as the default database (though it can be switched for other databases).

# 2. Backend

The backend of the application is built using Flask, a lightweight Python web framework, and includes the following key components:

## 2.1 app.py

This file initializes the Flask app and sets up key components like the database and migration handling. It serves as the entry point for the Flask application.

- Flask App Initialization: Starts the Flask app with the necessary configurations and extensions.  
- Database Setup: Configures the database using SQLAlchemy, allowing interaction between the app and the database.  
- Routing: Imports the routes defined for handling the feedback system.

## 2.2 config.py

This file contains the app's configuration settings, such as the secret key and the database URL. It abstracts out sensitive data (like secret keys and database URIs) into environment variables, with fallback values for development.

- SQLAlchemy URI: Uses SQLite for development but can be swapped with another database (such as PostgreSQL) for production.  
- Security Configuration: Manages sensitive configurations like the secret key to ensure secure sessions.

## 2.3 models.py

Defines the database models for the system, including users, products, and feedback.

- User Model: Represents users with unique IDs, usernames, and emails.  
- Product Model: Represents products with IDs and names.  
- Feedback Model: Stores feedback data, including ratings, comments, and timestamps. Each feedback is tied to a user and a product, with foreign key relationships.

## 2.4 routes.py

Contains the Flask routes that handle interactions with the feedback system.

- Submit Feedback: Allows users to submit feedback for a product.  
- Get Feedback: Retrieves all feedback associated with a specific product.  
- Average Rating: Computes and returns the average rating for a product.

## 2.5 Database Migration

This part of the setup ensures that the database schema is created and updated through migrations using Flask-Migrate.

- Commands: The migration commands (flask db init, flask db migrate, flask db upgrade) manage database schema changes.

# 3. Frontend

The frontend is built using React and includes components for submitting feedback and viewing submission history. It interacts with the Flask backend via API calls to perform these tasks.

## 3.1 FeedbackForm.js

This React component renders the form that allows users to submit feedback on a product.

- Form Fields: Includes fields for selecting a product, providing a rating (1-5 stars), and writing comments.  
- Axios: Handles the submission of the feedback data to the backend via an HTTP POST request.  
- State Management: Utilizes useState and useEffect hooks to manage form input and API calls.

## 3.2 SubmissionHistory.js

This component displays the feedback submission history for a given user.

- Fetch Feedback: Uses Axios to retrieve feedback data from the backend and displays it in a list format.  
- Dynamic Rendering: Displays feedback details such as product ID, rating, comments, and submission date.

## 3.3 App.js

The main application component that brings together the feedback form and the submission history.

- Component Composition: Combines FeedbackForm and SubmissionHistory into a cohesive UI.  
- Styling: Basic CSS is applied to structure the application and ensure user-friendly interactions.

## 3.4 App.css

Defines basic styling for the application, ensuring a clean and responsive UI.

# 4. Database

The application uses SQLite for development purposes. However, the database configuration is flexible, and you can switch to a more robust solution (like PostgreSQL) by updating the SQLALCHEMY\_DATABASE\_URI in the config.py file.

- SQLAlchemy ORM: Manages the interaction between the app and the database.  
- Flask-Migrate: Handles database schema updates and migrations.

# 5. Running the Application

## 5.1 Backend Setup

1. Install the required dependencies: pip install -r requirements.txt.  
2. Set up the database by running the following commands:  
 - flask db init  
 - flask db migrate -m "Initial migration"  
 - flask db upgrade  
3. Start the Flask backend: flask run.

## 5.2 Frontend Setup

1. Navigate to the frontend folder (customer-feedback-frontend) and install dependencies: npm install.  
2. Start the frontend application: npm start.