COMPREHENSIVE REPORT

Building Microservices with FastAPI and Streamlit

1. Introduction

The objective of this project was to develop a set of microservices using FastAPI, with a focus on managing products, reviews, and offers, and to create a Streamlit application for presenting these products. The assignment required the integration of database storage, API documentation, unit testing, pagination, static code analysis, and code linting.

2. Approach

2.1. Project Setup

- **Environment Setup:** Installed necessary packages including FastAPI, SQLAlchemy, and Streamlit.
- **Database Configuration:** Used SQLite for simplicity and SQLAlchemy ORM for database interaction.
- **Schema Design:** Defined three primary models: Product, Review, and Offer, each with relevant fields.

2.2. FastAPI Microservices

- **Create Product API:** Endpoint to create a new product.
- **Fetch Product API:** Endpoint to retrieve a product by its ID.
- **Update Product API:** Endpoint to update product attributes.
- **Create Review API:** Endpoint to add a review for a product.
- **Create Offer API:** Endpoint to add an offer for a product.
- **List Reviews with Pagination API:** Endpoint to list reviews for products with pagination.
- **Delete Product API:** Endpoint to remove old products.

2.3. Streamlit Application

- **Product Listing:** Displayed a list of products.
- **Product Details:** Showed detailed information about a selected product.
- **Offers:** Presented available offers related to products.
- **Reviews:** Listed reviews for products.

2.4. Testing and Validation

- **Unit Tests:** Created comprehensive unit tests for each API endpoint using `pytest` and `TestClient`.
- **Static Code Analysis:** Used SonarQube for static code analysis to ensure code quality and adherence to best practices.
- **Linting:** Utilized `pylint` to maintain code consistency and style.

2.5. Data Population

- **Fake Data Generation:** Generated 10,000 records using `Faker` to populate the database for testing and demonstration purposes.

3. Challenges Faced

3.1. Database Transactions

- **Challenge:** Ensuring atomic transactions in SQLite, especially during bulk data insertion.
- **Solution:** Utilized SQLAlchemy's session management to handle transactions effectively and implemented proper error handling.

3.2. Pagination

- **Challenge:** Implementing efficient pagination for large datasets.
- **Solution:** Integrated FastAPI Pagination library to handle pagination seamlessly.

3.3. API Documentation

- **Challenge:** Providing clear and comprehensive API documentation.
- **Solution:** Leveraged FastAPI's built-in support for OpenAPI specifications and automatically generated Swagger UI.

3.4. Data Consistency

- **Challenge:** Maintaining data integrity and consistency during concurrent operations.
- **Solution:** Used database constraints and ensured proper locking mechanisms in the ORM layer.

3.5. Testing and Coverage

- **Challenge:** Achieving high test coverage and writing meaningful unit tests.
- **Solution:** Developed thorough test cases covering various scenarios and edge cases to ensure robustness.

4. Lessons Learned

4.1. FastAPI and SQLAlchemy Integration

- FastAPI's seamless integration with SQLAlchemy made it easier to manage database operations, but careful session management is crucial to avoid common pitfalls.

4.2. Importance of Documentation

- Auto-generated documentation via Swagger UI was incredibly valuable for understanding and testing APIs, highlighting the importance of clear API documentation.

4.3. Data Handling

- Generating large volumes of test data using `Faker` was beneficial for testing performance and pagination features, underscoring the importance of realistic test data.

4.4. Code Quality Tools

- Using tools like SonarQube and `pylint` significantly improved code quality, highlighting the need for continuous integration of code analysis tools in the development workflow.

4.5. User Interface with Streamlit

- Streamlit provided a simple yet powerful way to create interactive user interfaces for data presentation, emphasizing the ease of building front-end applications with modern tools.

5. Conclusion

This project provided comprehensive exposure to building microservices with FastAPI, integrating databases using SQLAlchemy, and developing a user interface with Streamlit. The challenges faced were valuable learning experiences, particularly in managing database transactions, implementing pagination, and ensuring high code quality. The project's success demonstrates the effectiveness of combining modern web frameworks and tools to create robust, scalable, and maintainable applications.

Korivipalli Pavithra,

Emp ID:-46275083.