CSC301 Assignment 1 Writeup

Use of Generative AI

In my submission, I leveraged Generative AI tools, such as ChatGPT and Deepseek R1, to assist in the following areas:

1. **Code Structure and Design**: I used AI to generate initial scaffolding for UserService, ProductService, OrderService, and ISCS components. The AI provided an efficient way to kickstart the project, especially for setting up repetitive tasks like database initialization, request handling, and configuration loading.

4. **Workload Parser**: The AI-generated parser was adapted to support text-based formats in workload.txt. I modified it to dynamically read configurations and handle commands from a text file in the proper format given by the workload template file.

4. **Code Debugging and Refinement**: Several bugs, such as improper handling of response streams and specific Java syntax and functionality that I was unaware of, were addressed and solved with the help of AI-generated suggestions.

Modifications to AI-Generated Code

While the AI provided a strong foundation, significant modifications were made to tailor the code for our project:

• **Database Logic**: SQL queries were adapted to handle edge cases and proper API syntax.

• **Service Communication**: AI suggestions were refined to ensure seamless interaction between OrderService and ISCS, including error handling and fallback mechanisms.

• **Scalability Preparations**: Included configurations (e.g., dynamic IP/port loading) that were not part of the AI-generated code but are critical for A2's scalability requirements.

Shortcuts Taken for A1

Given the time constraints, I made some trade-offs to focus on passing the provided test cases for A1. These include:

1. **Single Instance per Service**: Our implementation does not yet support load balancing across multiple instances. A1 test cases only require single-instance functionality. Adding load balancing and distributed capabilities is deferred to A2 when scalability is tested.

2. **Simplified Inter-Service Communication**: ISCS acts as a basic router without advanced features like caching or asynchronous request handling. For A1, this simplistic approach suffices to pass the functional test cases and ensures correctness within the constraints.

3. **Error Handling Limitations**: Error responses are consistent but lack detailed logging or retries for failed inter-service requests. While sufficient for A1, this will require improvement to handle failures gracefully under high load in A2.

Code That Requires Major Rewrites for A2

1. **ISCS Service**:

• **Why**: It currently acts as a synchronous router without caching, load balancing, or failure recovery mechanisms.

• **What to Rewrite**: Introduce a task queue for asynchronous processing, implement caching for frequent requests, and add load balancing for scalability.

2. **Database Handling**:

• **Why**: SQLite is sufficient for A1 but will struggle with concurrent requests and large-scale operations in A2.

• **What to Rewrite**: Transition to a more robust database system like PostgreSQL or MySQL and implement connection pooling.

3. **Workload Parser**:

• **Why**: It is single-threaded and only processes sequential commands.

• **What to Rewrite**: Enhance it to handle concurrent workloads and output performance metrics (e.g., request processing time).

Code That Will Likely Not Require Major Rewrites

1. **Request Validation**:

• **Why**: The validation logic for user, product, and order commands is comprehensive and handles all defined edge cases.

2. **Service Configurations**:

• **Why**: The use of dynamic config.json files allows flexible IP and port configurations, which can scale to A2's requirements.

3. **Basic CRUD Operations**:

• **Why**: The create, update, delete, and get logic for UserService and ProductService is modular and efficient, adhering to RESTful principles.

Conclusion

My submission is designed to meet A1's requirements while providing a solid foundation for A2. Although I took shortcuts to simplify certain aspects, the modular design and dynamic configurations will ease future enhancements. Rewrites will primarily focus on scalability, database robustness, and distributed processing.