Use of Generative AI

Through this assignment we consulted Generative AI tools as a supportive resource rather than direct usage of its code. The tools used were ChatGPT and GitHub Copilot.

Specific AI-Assisted Components:

SHA-256 Password Hashing:

The development of the hashSHA256() function within the UserServer.java file was assisted by the AI tool ChatGPT. The AI provided guidance on using Java's MessageDigest class and the byte to hex conversion logic. After implementing the hashSHA256() function, it was reviewed to ensure that it meets the security requirements of the assignment.

JSON String-to-Map Converter:

The StringToMap() function (in all three Java servers) was developed with AI assistance. We used ChatGPT to help us understand how to implement the stringToMap function using string manipulation. The idea behind the function was to parse JSON input to the server and use it accordingly, but we broke this down into a simpler version, using string manipulation. This was intentionally done for the simplicity of A1.

HTTP Method Semantics:

To answer the questions about the difference between GET and POST requests and how the body of the request is handled, Python and Java, GPT was used. This was helpful to debug the interaction between the ISCS (Python) and microservices (Java).

Javadoc Documentation:

GitHub Copilot (VS Code) was used to assist in drafting Javadoc comments for public classes and methods. All the generated documentation was reviewed to accurately reflect the implemented behavior and match the project specification.

The AI was not used to generate entire service implementations, as all of the architectural decisions and implementations were done by 2 of us. Some examples of the changes 2 of us made to the AI's source code include:

- changing the hash function output to uppercase strings, simplifying the JSON parser to handle only the flat object structure.
- Simplifying the JSON parser to handle only the flat object structure.
- Correcting Javadoc descriptions to accurately match the implemented method signatures and return types.
- Adjusting HTTP client code to properly handle received bodies in GET requests

Shortcuts Taken (and Why)

Given the time constraints of A1 and the explicit acknowledgement that parts of this submission may need revision for A2, several deliberate shortcuts were taken:

1. **Naive JSON Parsing**
   Additionally, the parsing of the request bodies has been done through simple string manipulation instead of by using a JSON parsing library like Gson or Jackson. Parsing through string manipulation is a hardcoded approach that would not work for more complicated JSON objects, but it does work for A1 and the test cases that will be used for validation. This approach was suggested by GPT.
2. **In-Memory Data Storage**
   All of the services currently use static HashMap structures to store users and products, which makes the service easier to implement and test, but would be entirely inadequate for a real-world scenario.
3. **Minimal Error Responses**
   The responses in cases of errors are empty. The reason behind this is precisely because of the test cases. In order to improve upon this, descriptive error messages could be presented to the user, which would describe the reasons for the error.
4. **No Caching**
   Instead of requesting user and product service via ISCS for each order placement, the order service may implement caching for user and product services with a limitation of how long to cache those services. That would significantly reduce latency and the overhead
5. **No Concurrency Control**
   The current implementation uses the default flow process for executing a HttpServer. There are no locks, synchronization blocks, or atomic operations protecting the shared HashMap data stores. This means concurrent order placements in the future could result in race conditions. Similar to reasoning before, the given A1's sequential test execution passes, but any concurrent workload would result in bugs.

Components Likely Requiring Major Rewrite for A2

The following components would require significant refactoring for A2:

1. **JSON Handling:**
   Replacing the string parsing with a proper JSON library (Gson, Jackson) to handle potentially complex payloads upcoming in A2.
2. **Persistence:**
   Moving from in-memory HashMap storage to durable storage (database or file based data parser and saver).
3. **Concurrency and Synchronization:**
   Adding thread-safe access to shared state using ConcurrentHashMap, synchronized blocks, or explicit locks to handle concurrent requests correctly.
4. **Error Handling:**
   Implementing descriptive error responses with appropriate HTTP status codes and meaningful error messages. For now these areas were kept simple for A1 to reduce complexity and implementation risk within the time constraints.

Components Likely Reusable for A2

The following code portions and design choices of the current submission are expected to carry forward with minimal conceptual changes:

1. **Service Boundaries:**
   The separation into UserService, ProductService, OrderService, and ISCS provides a clean microservices architecture. In addition to being given as a design pattern in the assignment instructions, a single responsibility rule makes it straightforward to extend or replace individual components.
2. **HTTP Routing Structure:**
   The endpoint design (/user, /product, /order with GET/POST methods) follows REST conventions and is consistent across services. Adding new endpoints or methods should be straightforward.
3. **Status Code Semantics:**
   The HTTP status codes (200, 400, 404, 409, etc.) closely follow the project specification and standard REST practices. This logic can also be preserved.
4. **Configuration Loading:**
   The config.json parsing and service address resolution pattern allows flexible deployment without hardcoded addresses.
5. **Startup Scripts:**
   The compilation and run scripts are portable and do not rely on user-specific paths or assumptions.

   These components represent deliberate design decisions rather than shortcuts. The service architecture, routing patterns, and status code handling should scale into A2 with minor improvement rather than full replacement.

We tried to focus on correctness and adherence to the A1 specification and test cases (payload and responses). Generative AI was used but responsibly. Any and all shortcuts we took were taken with the idea of understanding the costs but keeping the timeline in mind such decisions had to be taken.