Generative Artificial Intelligence Usage:

Our group utilized Generative AI, such as ChatGPT and Genini, as learning aids and a powerful tool to enhance our abilities for working on this assignment. These tools were primarily used to support our understanding, debugging, and make us more efficient in our work.

1. Brainstorming and understanding:

We used Generative AI to assist us in reasoning about the main logic of this microservices architecture, specifying the purpose of each service and how individual services can communicate with each other. In particular, the AI assistant was helpful for clarifying the HTTP semantics, helping us to see the expected request and response patterns, and knowing what the exact structure of a basic HTTP Server should look like.

2. Creation of Helper Functions:

Generative AI was also very useful as a reference when creating small helper functions, such as parsing, hashing, and response formatting. We integrated the small helper functions it suggested and made modifications so that it can actually fit into the structures and restrictions of our main code, which provides huge convenience and efficiency for the service to work nicely.

3. Debugging and optimizing:

When we are testing out the code, we also use Generative AI as a debugging assistant during the testing phase. This process includes finding out the issue that's causing the incorrect HTTP status codes, the causes of the incorrect output, identifying logical errors in requesting validations, and analyzing what edge cases failed the test cases. Particularly, our discussion with AI helped us identify issues related to invalid input handling and unexpected incorrect responses. Then it made suggestions about how to resolve each of the problems we have encountered. This helped us to modify and adjust our code and finally fix the issues.

Modifications:

The AI outputs were not fully correct for our assignment, so we had to change a lot of parts. For example, the AI code sometimes used JSON libraries or more advanced parsing, but our assignment required manual parsing, so we rewrote those parts using simple string operations like split, indexOf, trimming, and removing quotes. Also, the AI code did not always match the exact API behavior in the test cases, so we adjusted responses to return the correct HTTP status codes like 200, 400, 404, 409, and we made sure the response JSON format matches what the provided **user_response.json** files expect. We also changed validation logic, like rejecting missing fields, handling empty strings correctly, and making sure duplicate IDs return the correct error code.

Shortcuts and A1 Limitations:

Because the time for Assignment 1 is limited, our group made some shortcuts in the implementation. These choices help us finish A1 on time and pass the test cases, but we know they are not good enough for Assignment 2 and larger scale system.

1. One shortcut is how we handle JSON parsing. We did not use any JSON library because the assignment instructions state that the lab machines will not have external libraries installed. Instead, we parse the request body using simple string operations like split, indexOf, trimming spaces, and removing quotes. This works for the simple and fixed input format in A1. However, this method is not safe or efficient. If the JSON format becomes more complex, or if there are nested objects or different spacing, our code may break. It is also slower when handling a very large number of requests.

2. Another shortcut is that we did not fully implement the ISCS as a complete and independent service. Instead, the communication logic between services is handled in a simple way inside our current structure. This is enough for A1 because the interactions between services are limited and predictable. However, for A2, this design will not scale well. A proper ISCS would be needed to handle service routing, failures, and more complex communication between services.

3. We also use a simple method to read values from config.json, such as extracting port numbers by searching strings. This works only because the config file format is fixed in A1. If the configuration becomes more complex in A2, this approach will be hard to maintain and easy to break.

Overall, these shortcuts were made intentionally so we can complete A1 within the time limit. We understand that many parts of this implementation will need to be rewritten or improved for A2.

A2 Scalability and Future Improvements:

Even though our implementation for Assignment 1 has many shortcuts, there are still some parts that we believe will not need a full rewrite for Assignment 2. These design choices can be improved step by step instead of being completely replaced.

1. First, our services are separated into UserService, ProductService, and OrderService. Each service runs independently and communicates using HTTP requests. Because of this separation, the logic of each service is already clear and modular. For Assignment 2, when the workload becomes much larger, we can scale each service separately without changing the overall structure too much.

2. Second, our use of a single entry point for requests is helpful for future work. Even though we did not fully implement a proper ISCS in A1, our current design already assumes that requests are routed in a controlled way. For A2, we can extend this idea by implementing a real ISCS or improving the existing routing logic to support load balancing, retries, and failure handling.

3. Third, the WorkloadParser written in Python is a part we believe will not need major changes. Python makes it easy to adjust the workload logic and add more complex testing. For A2, we can extend this script to generate higher load, parallel requests, or different workload patterns without rewriting it from scratch.

So, as a conclusion, while many low-level implementations in A1 are not scalable, the high-level structure of our system is still useful. With better JSON parsing, proper service coordination, and more efficient request handling, we believe our current design can be evolved to meet the requirements of Assignment 2.