Writeup

For this assignment, the generative AI that was incorporated into working on the assignment is GitHub Copilot, for inline suggestions. Copilot allows for inline code prediction through the context of the rest of the codebase and suggests a range of predictions, ranging from “it is reading my mind” to “not even close”. One of its strengths as a tool is offloading the hassle of writing repetitive structural code, such as writing javadocs. The initial suggestions usually need some tweaking but the subsequent suggestions are much better because of the initial context.

Another strength in its utility lies in the syntax of programming languages, as remembering how to translate an idea into code requires a lot of google searches and reading documentation on how to use libraries / languages. With copilot, the focus on how to write something is alleviated with inline suggestions, allowing developers to spend their time more on the logic. This has personally helped me pick back up the Java syntax rather quickly after not using it for many years.

However, its utility as a tool is limited for higher level architecture. It sometimes suggests not the most efficient or ideal solutions when there is no context to work off of, so its utility is proportional to the building blocks i.e. the quality of the context you give it. This means that many of the smaller snippets were generated with copilot but the architecture was with our own efforts.

A large shortcut that was taken in this assignment was the use of data persistence in text files. While the requirements allow for this, it is not ideal to primarily store data in the microservice memory as it is not fault tolerant and only vertically scalable. Concurrency and load balancing was kept in mind when designing the system, specifically with the ISCS but not accounted for in the code. By using a database for transactions, concurrency should require a moderate amount of rewrites by re-organizing the service handlers into threads.

If docker were a requirement for assignment 2, containerizing the application should account for horizontal scalability and durability by persisting the database data in volumes. Some system strengths that were prioritized given the criteria is query caching in the order service for performance. By using in memory dictionaries (similar to redis) the order service caches user and product queries into the ordering service, and invalidates the cache when a mutation event occurs. While the caching could be more aggressive by implementing it in the ISCS as well, the architecture is in place for future improvements.

However, the logic flow should not be susceptible to large changes through the use of early returns, exception handling and abstracting larger chunks of reusable code into private methods. This allows us to change up the method implementations without largely affecting the service logic.