Generative AI usage and modifications:
- I used Generative AI to help create test data and test scripts (e.g., sample payloads and sanity checks against expected responses) and to assist with debugging by suggesting likely root causes for failing requests.
- I did not use AI to generate production service logic. Any AI-suggested snippets for tests/debugging Ire revieId and edited for correctness; I adjusted endpoints, ports, and expected status codes to match our implementation and the provided test cases.

Shortcuts and “this will do for A1” choices:
- Data storage is in-memory only. This meets A1 tests but will not survive restarts or scale to A2 requirements without a persistence layer.
- Inter-service error handling is minimal and focuses on the required status codes; it does not include retries, timeouts, or circuit breakers.
- Load balancing and discovery are simplified for a single-node LAN setup and will need a redesign for multiple instances in A2.

Pieces unlikely to require major rewrites for A2 (and why):
- Request/response schemas and endpoint structure for user, product, and order services. These match the spec and should remain stable as scale increases.
- Core validation logic for required fields and types. The rules are Ill-defined and will stay the same even if storage/backing services change.
- The workload parser’s command parsing. It already maps input lines to the specified API calls and can be reused while the backend scales.

I believe the above components are stable because they are defined by the public API contract and test cases, which are expected to carry forward. The main A2 work will be in persistence, scalability, and resilience rather than changing the public API behavior.