Hi,
Because of time restriction for home assignment, my solution just solves the problem in a simple way. When the consumption scales up, it definitely struggles with many issues. To reach to the high availability and the stability, I would to propose some points as below:

- Use more worker processes in Fast API by Gunicorn: we can increase number of workers in config to leverage multiple processing in a single server. In my project, I only use one worker because I store data in dictionary and list format in memory. In Python, they are thread-safe, but in multiprocessing, it cannot resolve the racing condition.
- Scale up number of API instances and apply Load Balancer to serve a lot of requests from users. Load Balancer is responsible for directing requests to instances by some rules.
- To reach low latency, I would store data into in-memory database, it guarantees high performance for I/O operations. Besides, we should persist data into disk storage or a database. Accidentally, in-memory database could be restarted or crashed, data would be lost. Hence, we can recover data from persistent source.

Here is the architecture I want to propose: