Single-Server Key-Value Store

Mayank Mangipudi

Spring 2026

Design Decisions

For this assignment, we used Python (high-level language allows for more focus on system design vs. low-level details, and easier to import libs to Python), and FastAPI (easy to use, lightweight, quick to set up). The data is stored in a Python dictionary, which keeps the program simple and efficient.

The server is able to handle multiple requests at the same time. We used a global lock to ensure that only one thread can modify the store at a time. This prevents problems when two requests try to modify the same key at the same time. This approach is simple and works well for a single-server system, even though it would not scale well to a larger system.

To avoid losing data when the server restarts, we added some persistence using a JSON file. When the server starts, it loads any existing data from the disk. Whenever a key is modified, the store is saved back to the file. This allowed us to keep the design simple while still supporting the persistence.

The server handles errors as follows. If a client tries to get or delete a key that does not exist, the server returns an error. If a PUT request does not include a value, then we reject the request.

We also added logging to record all operations along with their timestamps. This helped with debugging and also seeing how the server behaves during concurrent requests.

Challenges and Assumptions

One challenge was making sure the shared data stayed consistent when we had multiple requests at once. We solved this by adding locking around all operations that modify the store. Another challenge was adding persistence without making the system too complicated. We assumed that the server runs on a single machine and the workload is small for testing purposes.

Possible Improvements

We could improve performance by reducing disk writes or using better locking. We could lock by key instead of just setting a global lock.