Report - To-do API with Flask, SQLite, and Fast-API

Overview of the API

This project is a simple Todo API built in Python. It started with Flask and used in-memory storage to hold a list of todo items. The API supports two endpoints:

```
GET /items: Returns a list of todo tasks
POST /items: Adds a new task to the list
```

Later, the API was upgraded to use SQLite for real data persistence. Finally, it was rewritten using FastAPI to support asynchronous performance and scalability.

How I Tested It

The API was tested using the following tools:

- `curl`: to send HTTP GET and POST requests
- `curl -w` and `time`: to measure request time and latency
- `Chrome DevTools`: to inspect response speed and payload size
- `htop`: to monitor CPU and memory usage during load
- A custom `multiprocessing` Python script to simulate 50 parallel users

What I Observed

- Regular request (GET/POST) took ~0.0045s
- During stress test (50 parallel POSTs), all requests succeeded in ~0.31s
- No CPU spikes or memory issues were observed
- SQLite handled write operations reliably
- Switching to FastAPI introduced async behavior, better suited for future scalability

What I Would Improve If Traffic Increased

- **Replace SQLite with PostgreSQL**:

  SQLite is suitable for simple, single-user scenarios, but not ideal under heavy concurrent load.

  PostgreSQL supports advanced features like **Write-Ahead Logging (WAL)**, which ensures data safety and better concurrency by writing changes to a log before applying them to the database — this makes it more robust in high-traffic environments.

- **Use asynchronous database access**:

  With FastAPI, we can use libraries like `SQLModel` or `databases` to make database operations fully async, which reduces blocking during I/O and improves performance under load.

- **Add caching using Redis**:

  Storing frequently accessed data (like the list of items) in Redis can reduce database hits and speed up responses.

- **Add unit testing coverage**:

  Writing unit tests using tools like `pytest` would ensure all endpoints work correctly and help catch bugs early as the API grows. This is crucial before deploying or adding new features.

- **Add rate limiting, input validation, and authentication**:

  These features would protect the API from misuse and make it safe for public or production deployment.