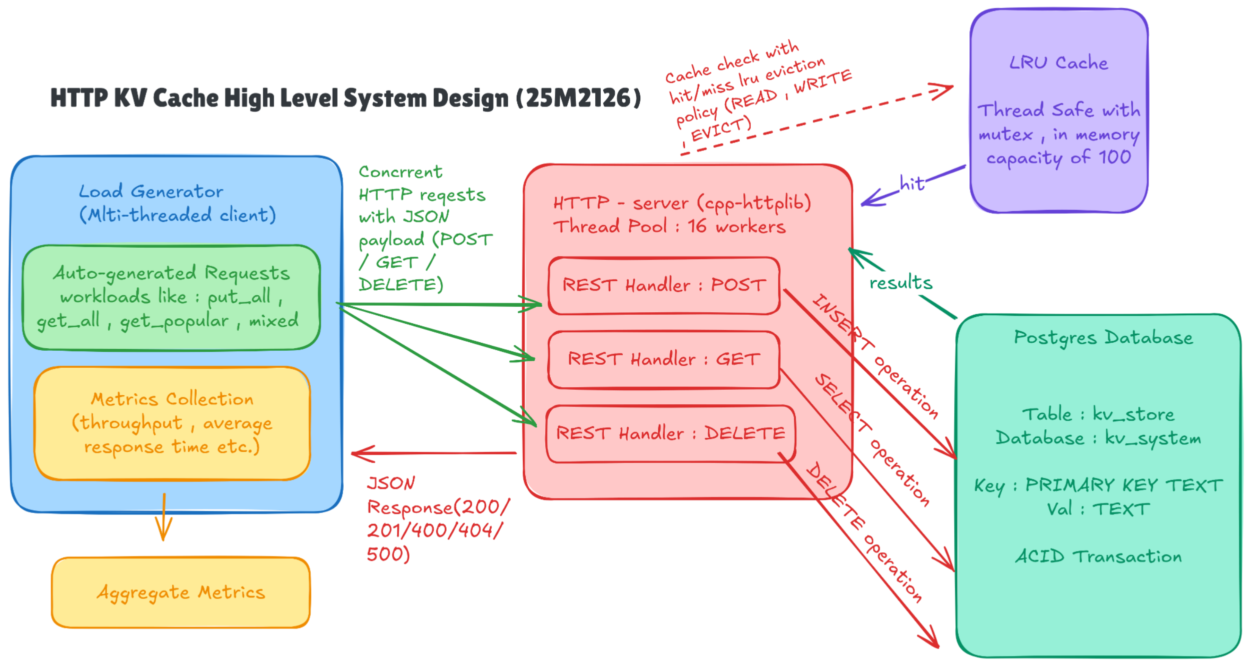
HTTP-based KV Cache Server

Description

This project implements a ***multi-threaded HTTP-based key-value (KV) store server*** in ***C++ with PostgreSQL*** as the persistent backend and an ***in-memory LRU cache*** for performance optimization. It supports RESTful CRUD operations (Create via POST, Read via GET, Delete via DELETE) and includes a closed-loop load generator for benchmarking throughput and latency under various workloads.

The system follows a client-server architecture with separated concerns: the server ***handles concurrent requests using a thread pool***, ***caches frequent accesses in memory***, and ***persists data to PostgreSQL***.

High Level System Design

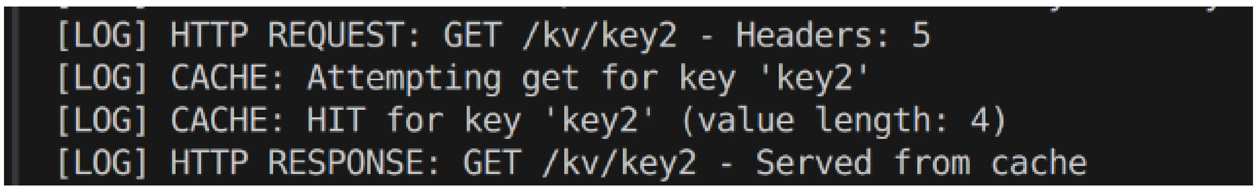


HTTP KV Cache High Level System Design (25M2126)
Load Generator (Mlti-threaded client)
Auto-generated Requests workloads like : put_all , get_all , get_popular , mixed
Metrics Collection (throughput , average response time etc.)
Aggregate Metrics
Concrrent HTTP reqests with JSON payload (POST / GET / DELETE)
JSON Response(200/ 201/400/404/ 500)
HTTP - server (cpp-httplib) Thread Pool : 16 workers
REST Handler : POST
REST Handler : GET
REST Handler : DELETE
Cache check with hit/miss lru eviction policy (READ , WRITE , EVICT)
LRU Cache
Thread Safe with mutex , in memory capacity of 100
hit
results
INSERT operation
SELECT operation
DELETE operation
Postgres Database
Table : kv_store Database : kv_system
Key : PRIMARY KEY TEXT Val : TEXT
ACID Transaction

Handling of two types of requests that follow different execution paths

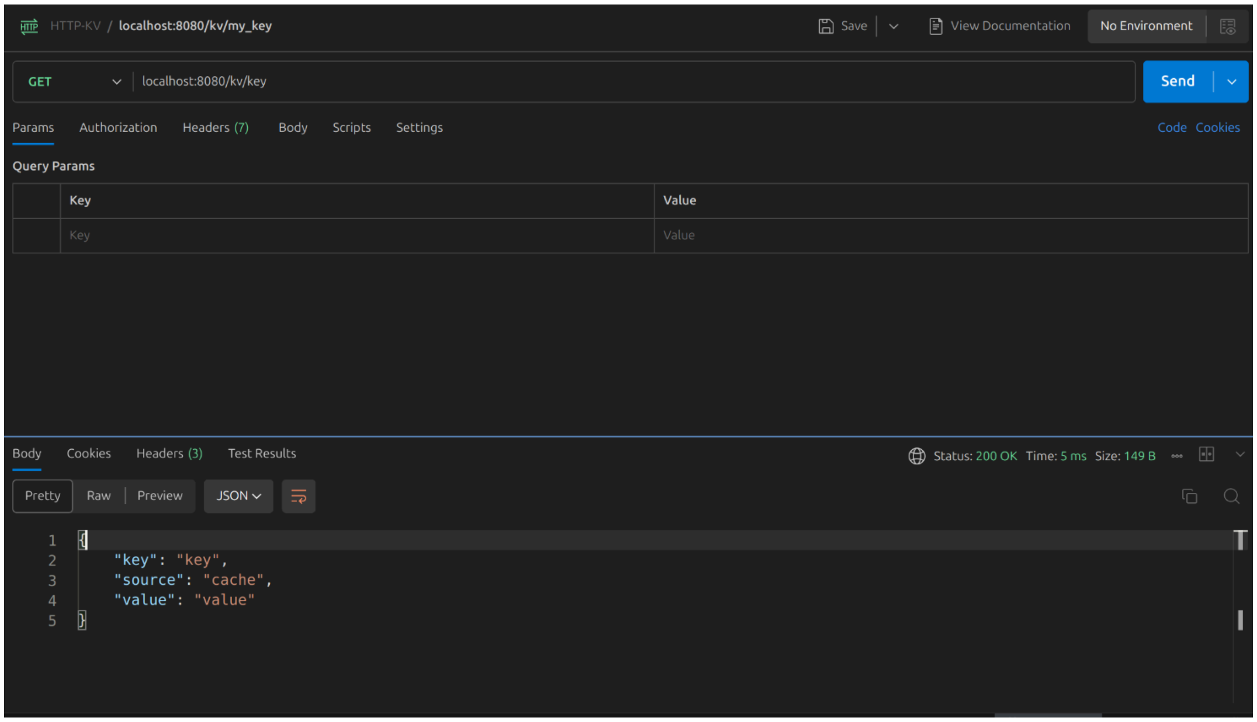
1. Execution Path 1 – Cache-Hit (Memory-Only)

Step	Code Reference	What happens
1.	`svr.Get(R"(/kv/(.+))", [](const httplib::Request& req, httplib::Response& res)`	GET Req received
2.	`auto cache_val = cache.get(key);`	Thread safe LRU cache lookup
3.	`if (cache_val) {...}`	Cache hit – value is returned directly from RAM; no DB connection is opened.
4.	`json j_res = {{"key", key}, {"value", *cache_val}, {"source", "cache"}};`	Response includes "source":"cache" for verification.



```
[LOG] HTTP REQUEST: GET /kv/key2 - Headers: 5
[LOG] CACHE: Attempting get for key 'key2'
[LOG] CACHE: HIT for key 'key2' (value length: 4)
[LOG] HTTP RESPONSE: GET /kv/key2 - Served from cache
```

Logs showing req served from cache



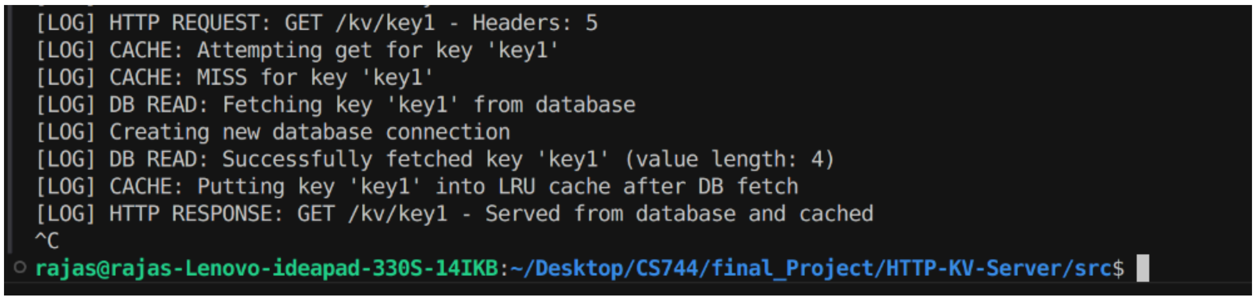
```
{
    "key": "key",
    "source": "cache",
    "value": "value"
}
```

Postman req for a key which was present in cache served from cache

2. Execution Path 2 – Cache-Miss → Disk (DB)

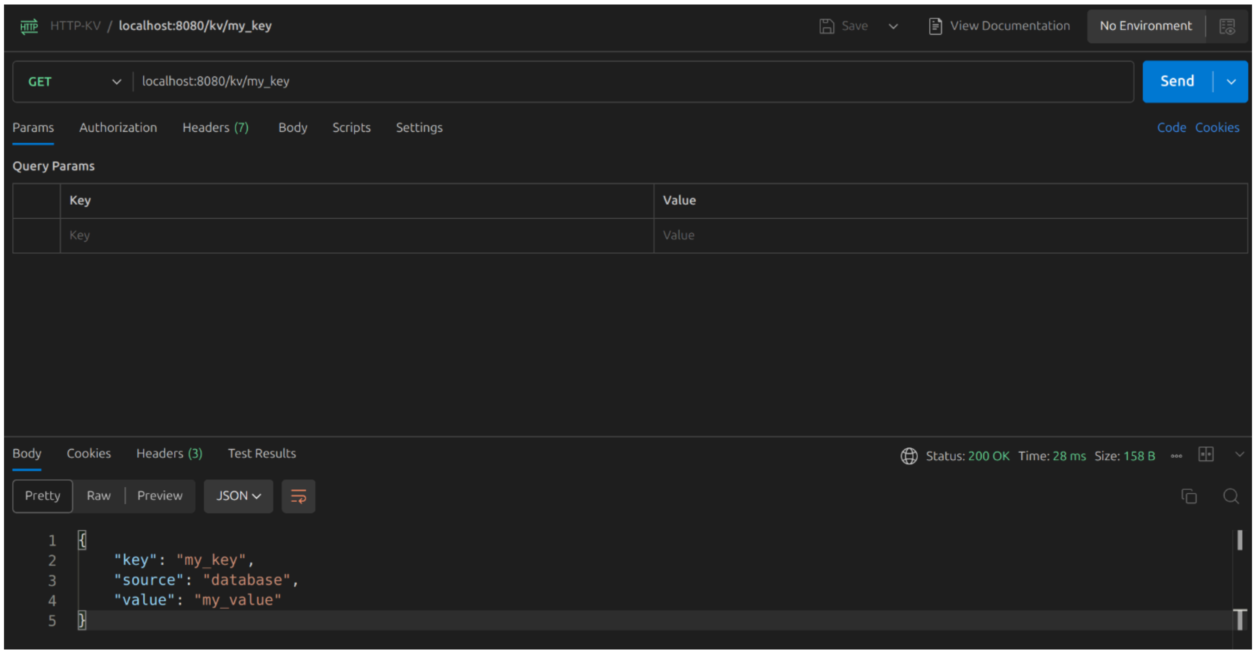
Step	Code Reference	What happens
1.	`svr.Get(R"(/kv/(.+))", [](const httplib::Request& req, httplib::Response& res)`	GET Req received
2.	`auto cache_val = cache.get(key);`	Thread safe LRU cache lookup
3.	`// 2. Cache Miss: Fetch from database` `auto db_val = db_read(key);`	Cache miss - db lookup required

4.	`if (db_val) {. . .}`	Value read from PostgreSQL (disk).
5.	`cache.put(key, *db_val);`	Populate cache; if capacity exceeded → LRU eviction
6.	`json j_res = {{"key", key}, {"value", *db_val}, {"source", "database"}};`	Response includes "source":"database" for verification.



```
[LOG] HTTP REQUEST: GET /kv/key1 - Headers: 5
[LOG] CACHE: Attempting get for key 'key1'
[LOG] CACHE: MISS for key 'key1'
[LOG] DB READ: Fetching key 'key1' from database
[LOG] Creating new database connection
[LOG] DB READ: Successfully fetched key 'key1' (value length: 4)
[LOG] CACHE: Putting key 'key1' into LRU cache after DB fetch
[LOG] HTTP RESPONSE: GET /kv/key1 - Served from database and cached
^C
rajas@rajas-Lenovo-ideapad-330S-14IKB:~/Desktop/CS744/final_Project/HTTP-KV-Server/src$
```

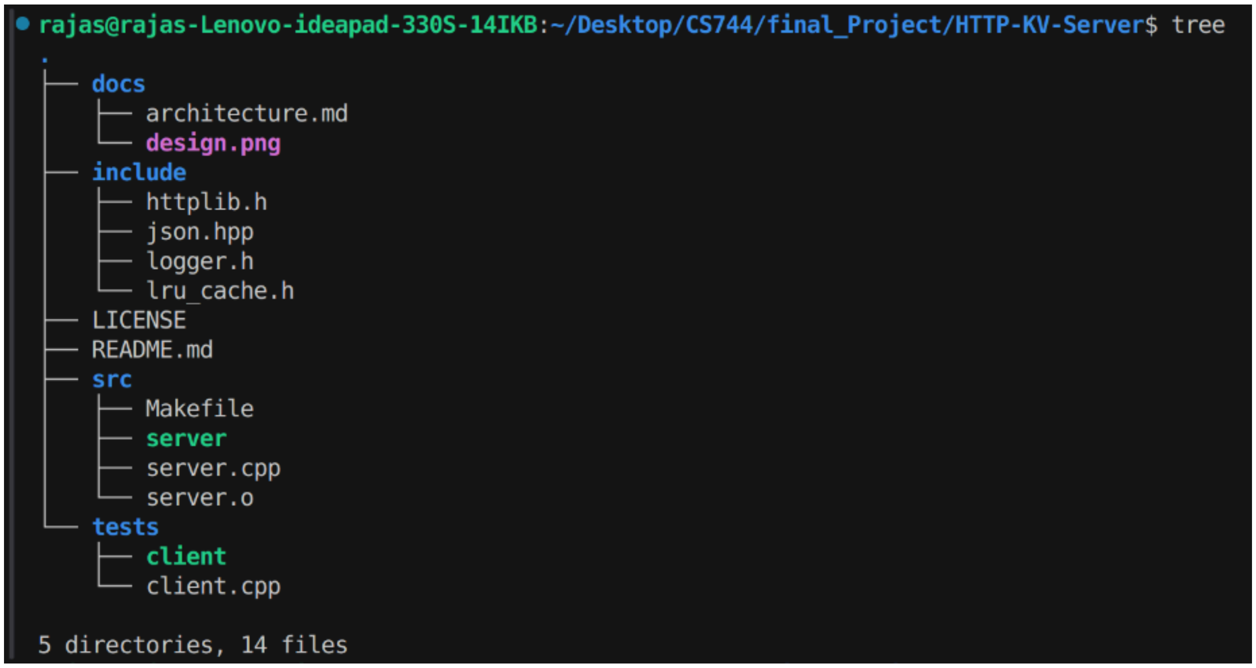
Logs showing req served from database on cache miss



```
{
    "key": "my_key",
    "source": "database",
    "value": "my_value"
}
```

Postman req for a key which was not present in cache served from database

Overall Architecture & Repository Quality



```
● rajas@rajas-Lenovo-ideapad-330S-14IKB:~/Desktop/CS744/final_Project/HTTP-KV-Server$ tree
.
├── docs
│   ├── architecture.md
│   └── design.png
├── include
│   ├── httplib.h
│   ├── json.hpp
│   ├── logger.h
│   └── lru_cache.h
├── LICENSE
├── README.md
├── src
│   ├── Makefile
│   ├── server
│   ├── server.cpp
│   └── server.o
└── tests
    ├── client
    └── client.cpp

5 directories, 14 files
```

Tree command showing directory structure

Points on Code Base Quality

Point No.	Criterion	Implementation
1.	Modular layout	include/ (public headers) src/ (implementation) docs/ (design) tests/ (client test)
2.	Build system	Makefile (make all, make clean) one-command reproducible build

3.	Documentation	README.md (setup, DB init, curl examples), docs/architecture.md (component diagram)
4.	Observability	JSON "source" field, added LOGS for every event like cache miss/hit db fetch etc.
5.	Security	Store db credentials , PORT etc. in .env file

Github Link : https://github.com/rajaspaunikar/HTTP-KV-Server
Name : Rajas Paunikar
Roll No: 25M2126