Persistent Key-Value Store

Architectural & API Reference

Pujith Sai Kumar Korlepara
IIT Bombay (ID: 25M0787)
Email: pujith@cse.iitb.ac.in / pujith22.sde@gmail.com
GitHub: pujith22 LinkedIn: in/pujith22
Portfolio: cse.iitb.ac.in/ pujith

1 Postman API Collection

An interactive Postman collection is published for quick exploration and manual testing of the API surface. You can import it directly via the shared workspace URL below or by copying the JSON specification.

Shared Collection URL: Postman Workspace Link

The local development server (see `main_server.cpp`) binds to host `localhost` on port `2222`. The collection above targets that base URL.

Repository

GitHub: https://github.com/pujith22/persistent-key-value-store

2 Purpose

A C++17 HTTP service providing a persistent key-value cache backed by PostgreSQL. It exposes a JSON-centric REST API for CRUD and bulk operations while maintaining latency via an inline in-memory write-through cache. Startup fails fast if persistence cannot be reached, enforcing durability guarantees.

3 High-Level Architecture

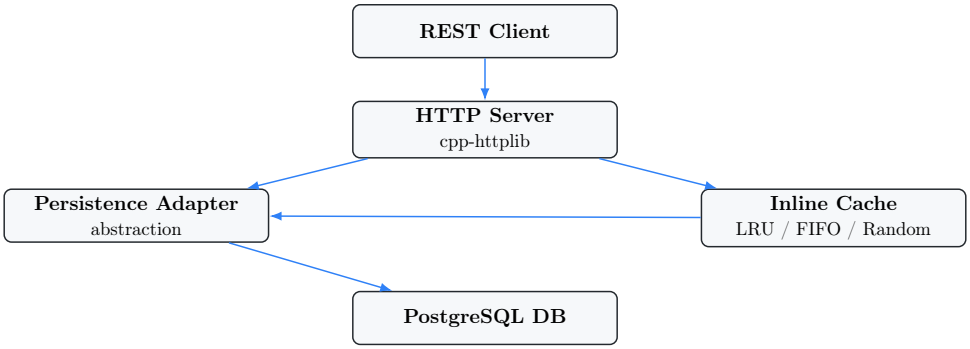


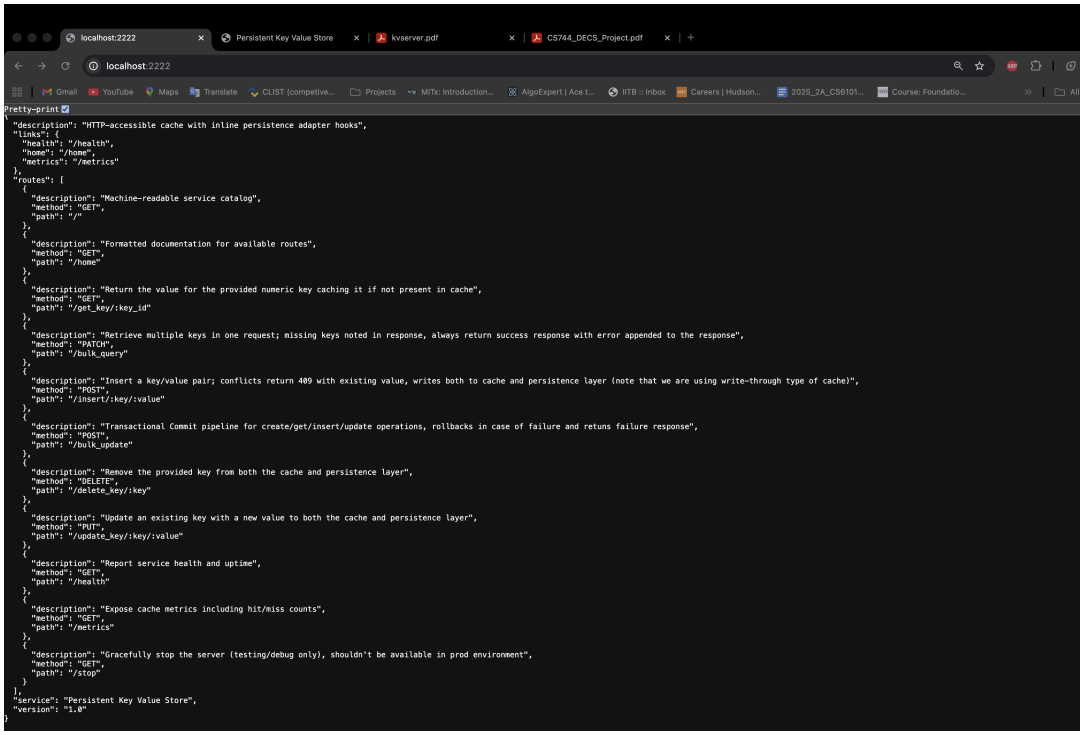
Figure 1: Compact layered architecture: request flow across server, cache, and persistence.

4 Endpoint Catalog

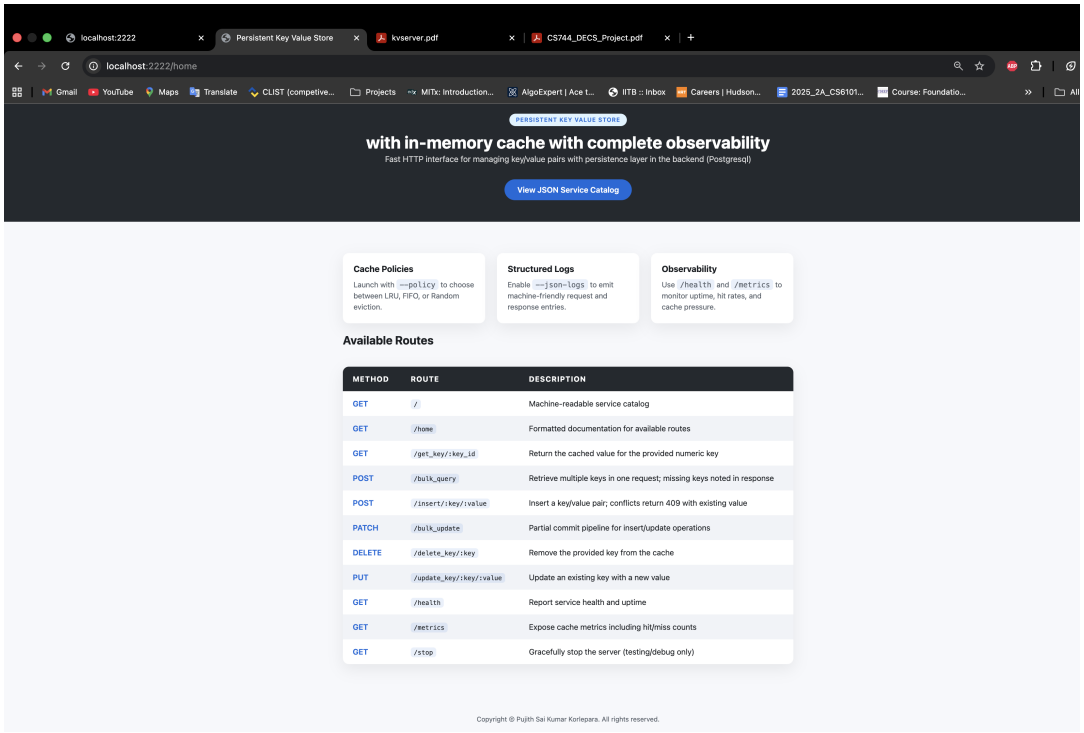
The service exposes the following HTTP endpoints.

Method	Path	Description
GET	`/`	Machine-readable service catalog
GET	`/home`	Formatted documentation for available routes
GET	`/get_key/:key_id`	Return the value for the provided numeric key, caching it if not present in cache
PATCH	`/bulk_query`	Retrieve multiple keys in one request; missing keys noted in response; always returns success with errors appended
POST	`/insert/:key/:value`	Insert a key/value pair; conflicts return 409 with existing value; write-through to cache and persistence
POST	`/bulk_update`	Transactional pipeline for create/get/insert/update; rolls back on failure and returns failure response
DELETE	`/delete_key/:key`	Remove the provided key from both cache and persistence layer
PUT	`/update_key/:key/:value`	Update an existing key with a new value in both cache and persistence layer
GET	`/health`	Report service health and uptime
GET	`/metrics`	Expose cache metrics including hit/miss counts
GET	`/stop`	Gracefully stop the server (testing/debug only; not for production)

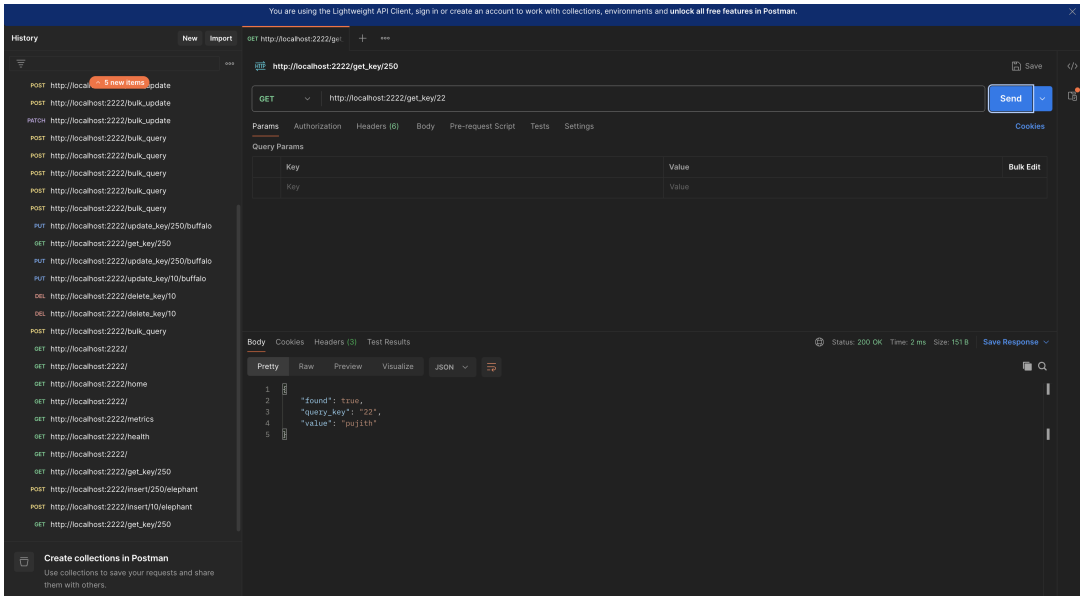
5 Screenshots



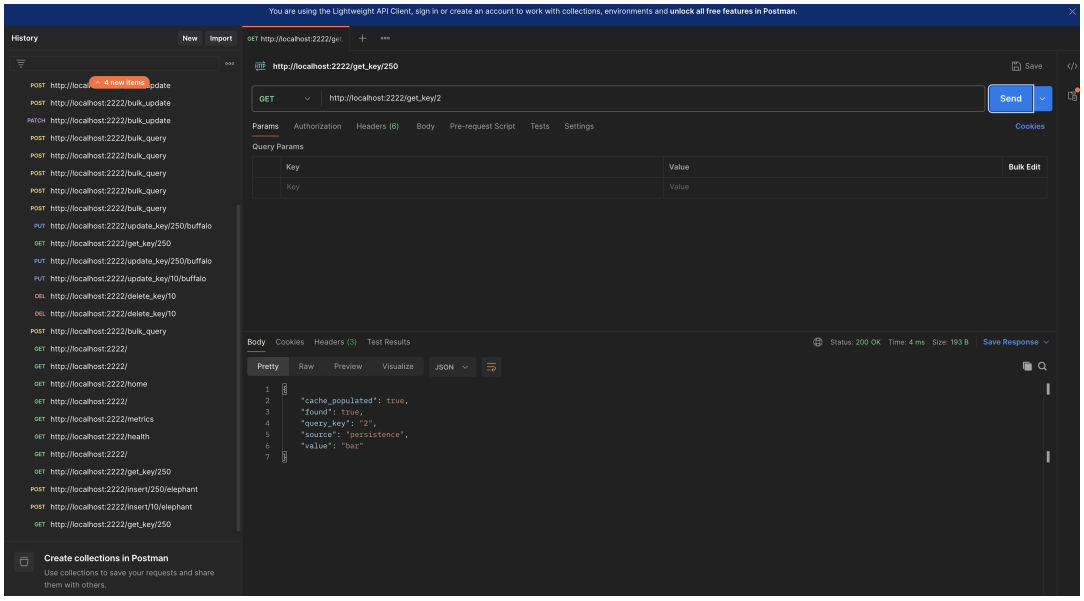
(a) Root endpoint (service catalog)



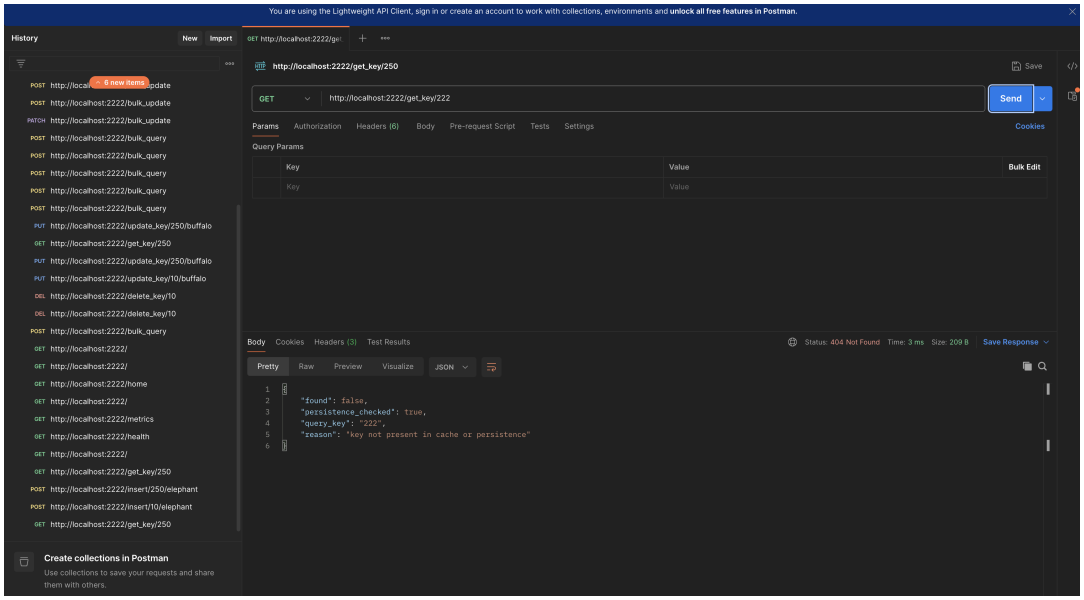
(b) Home HTML endpoint



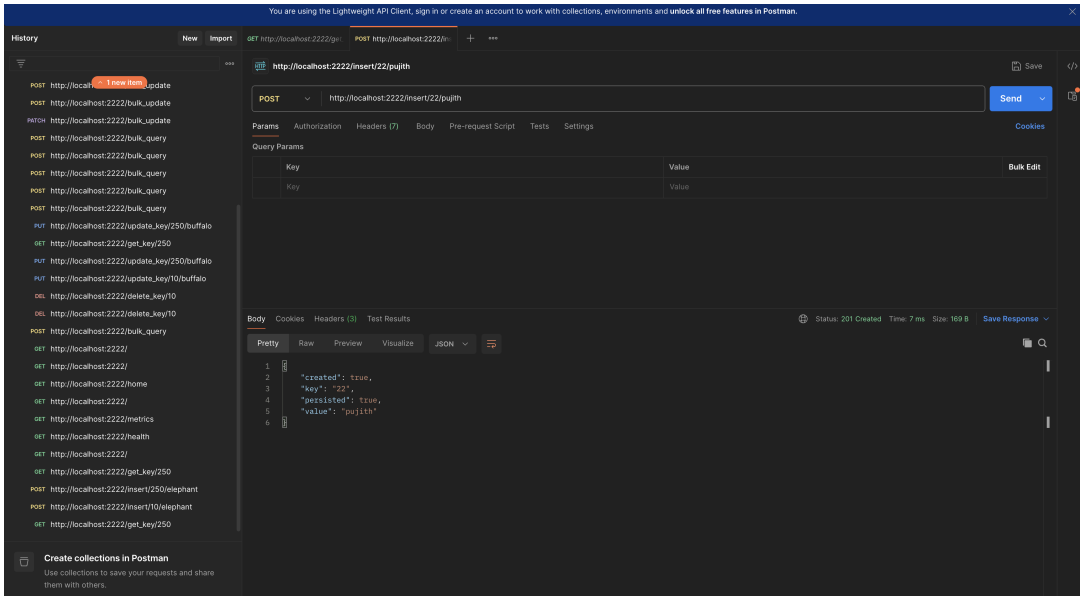
(a) Cache hit on get_key



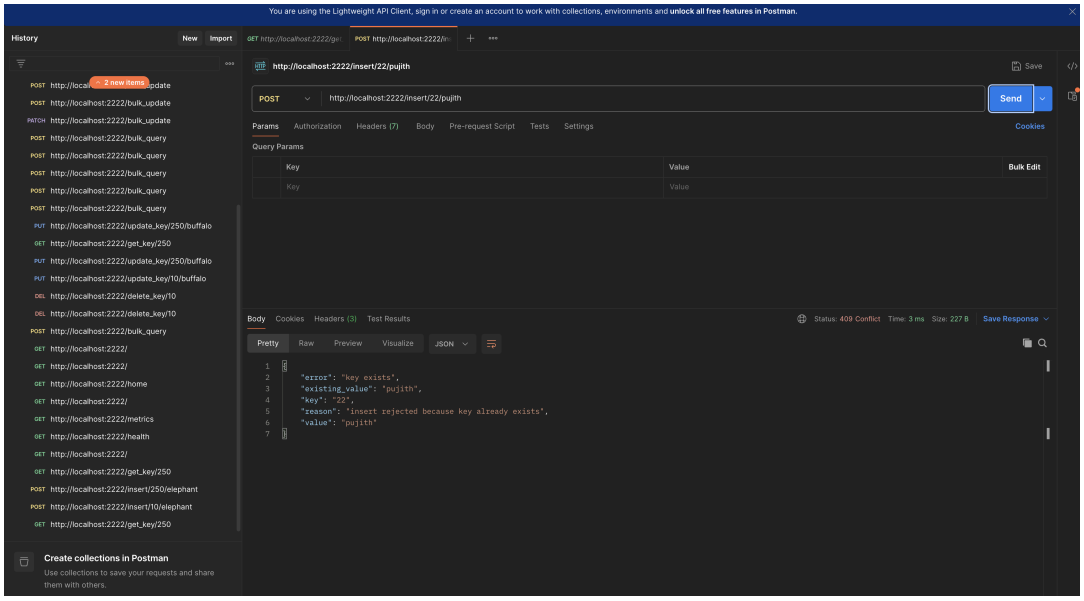
(b) Hydrating uncached key from persistence



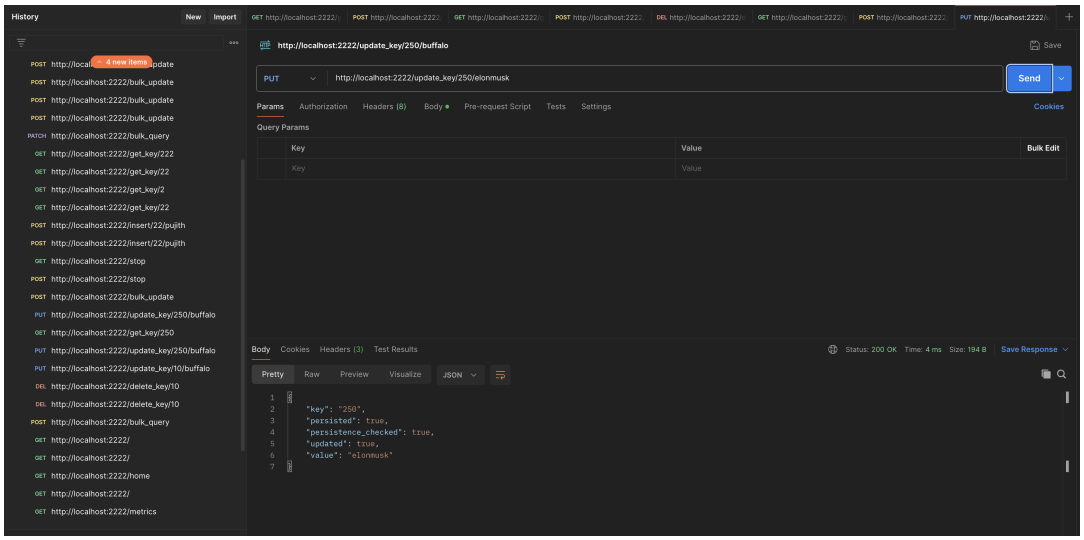
(a) Key not found scenario



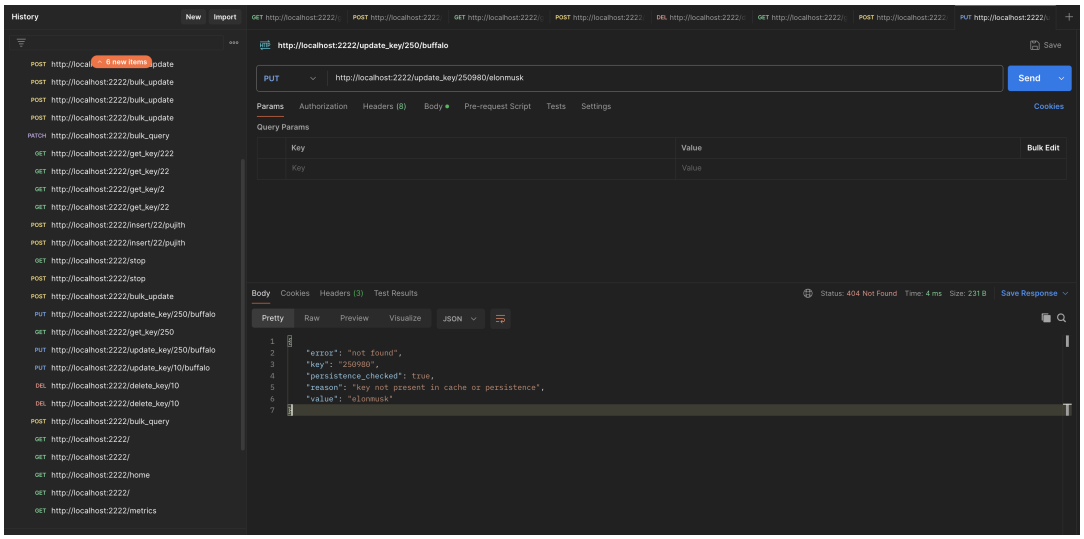
(b) Successful clean insert



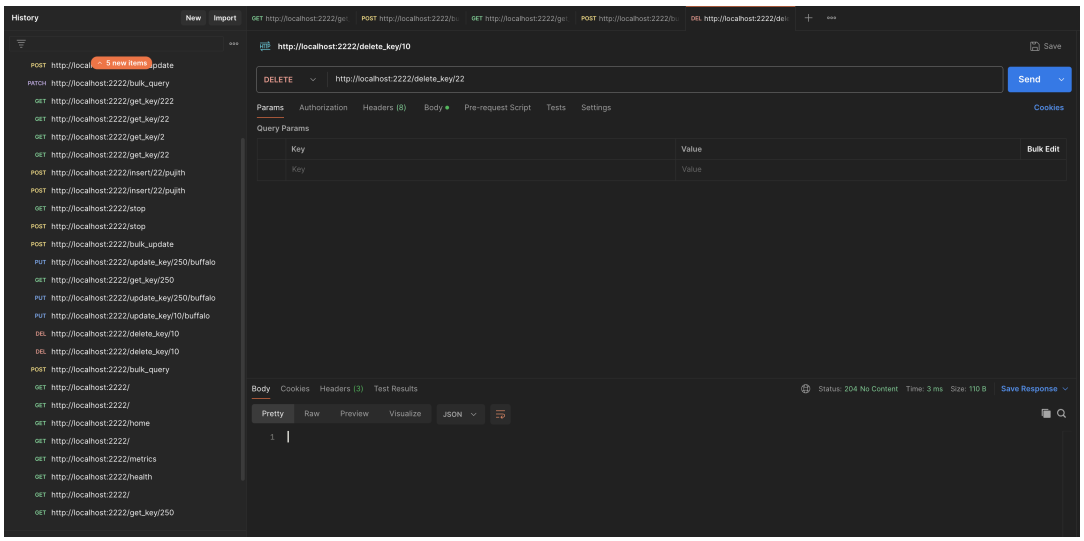
(a) Insert conflict (409)



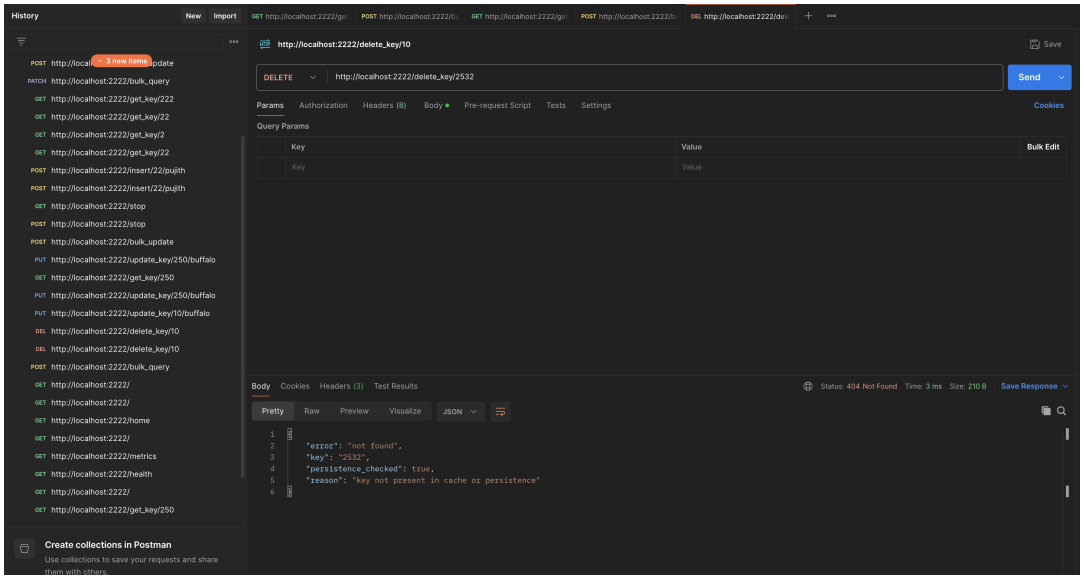
(b) Successful update operation



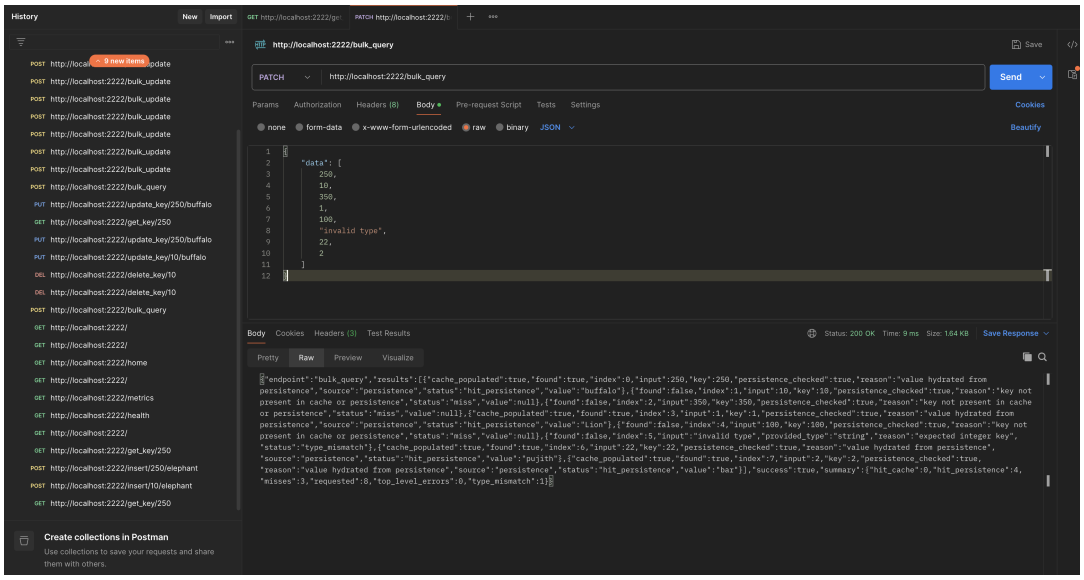
(a) Update when key absent (404)



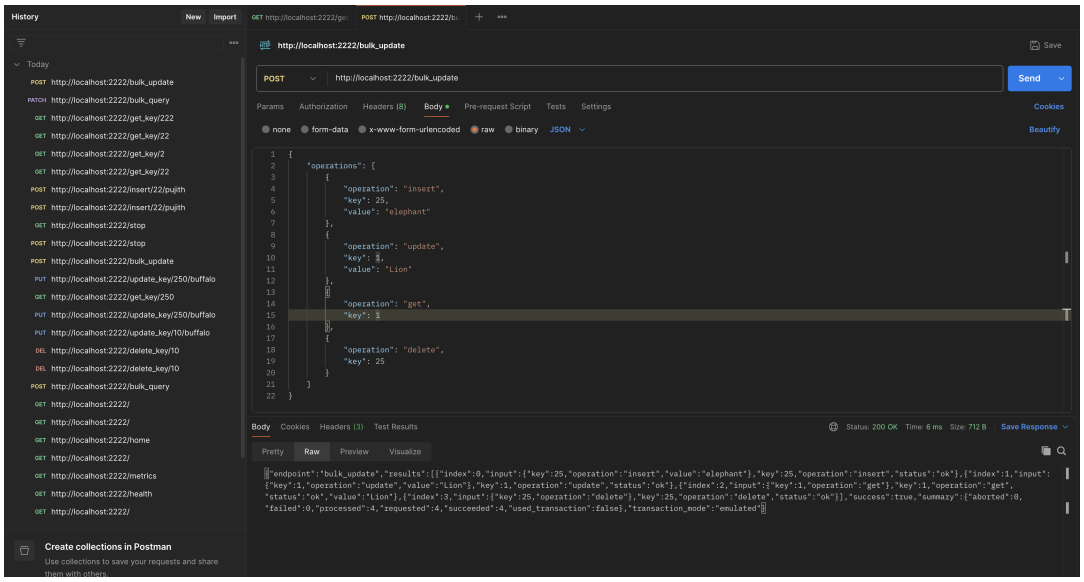
(b) Successful deletion (204)



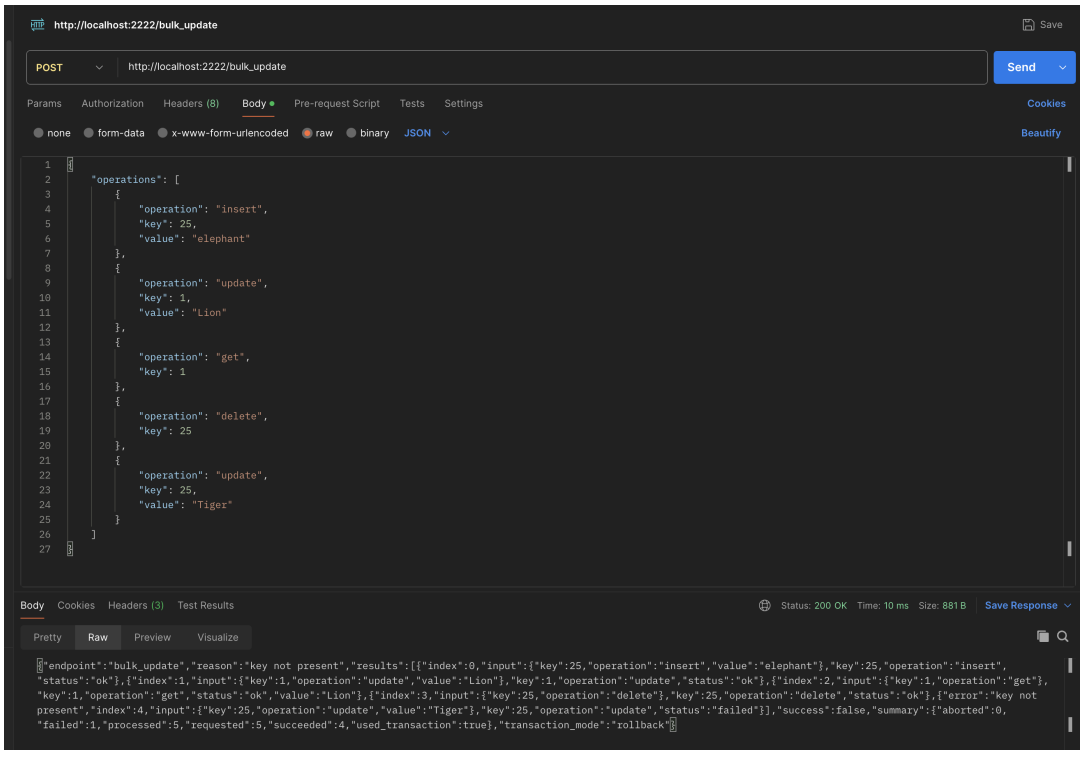
(a) Deletion when key not found



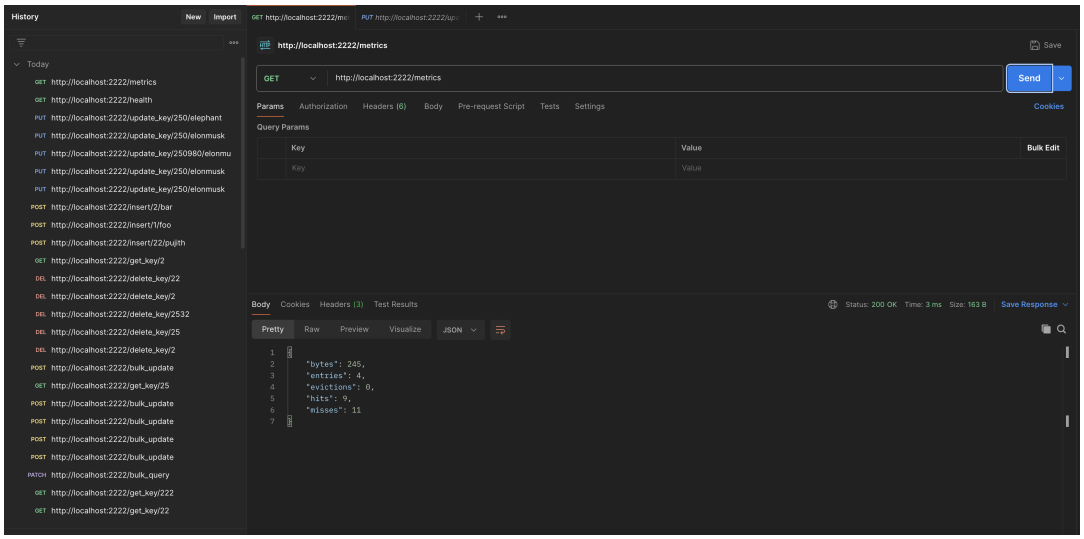
(b) Bulk query results with mixed statuses



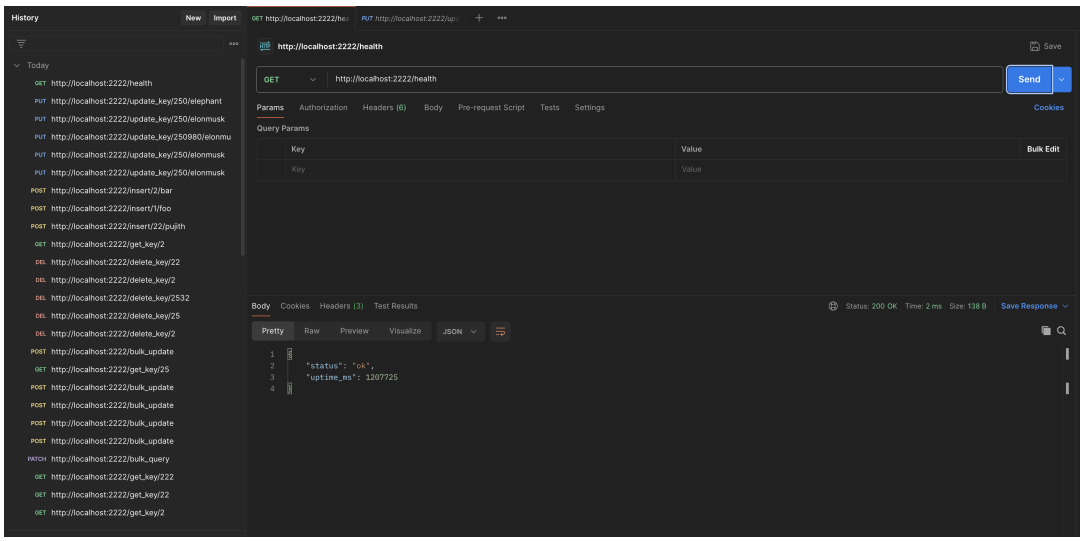
(a) Bulk update transaction success report



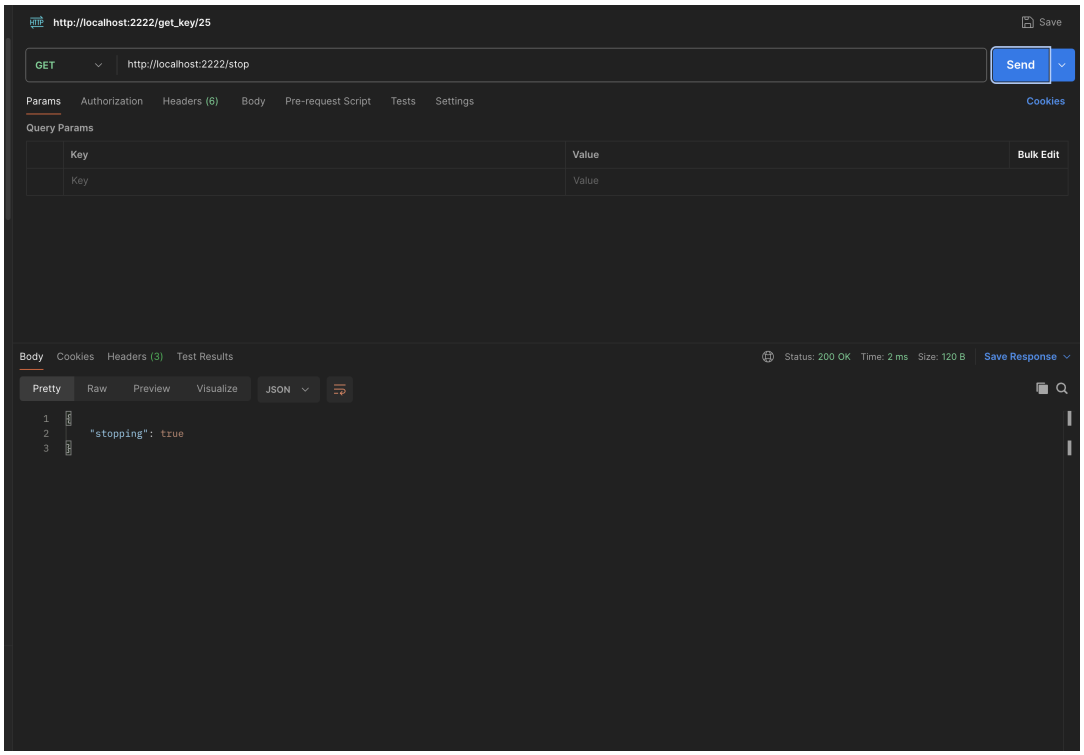
(b) Bulk update transaction failure and rollback



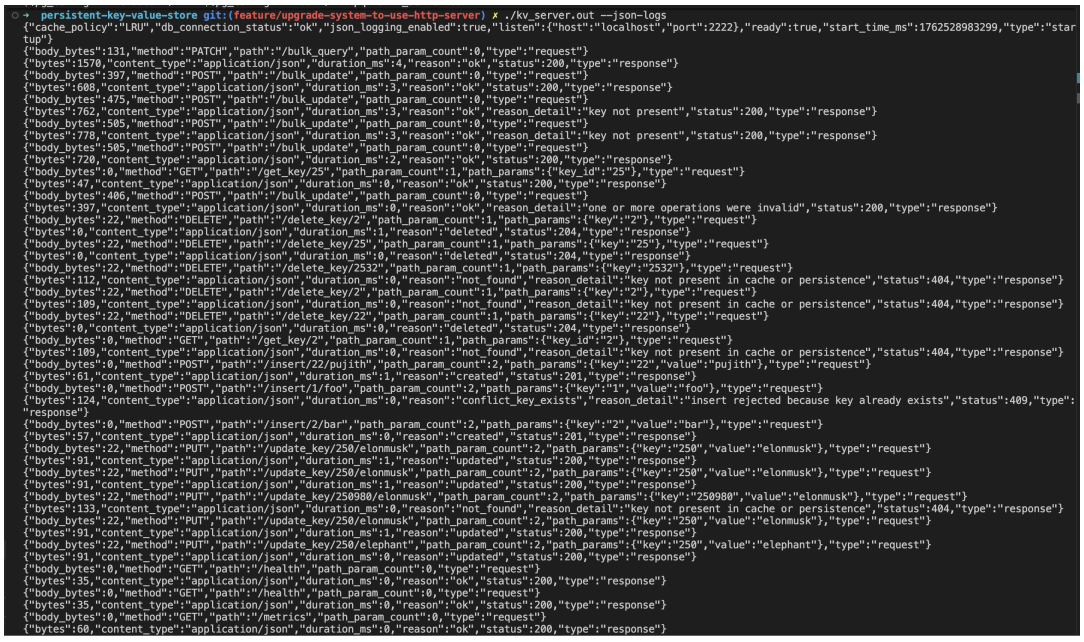
(a) Metrics endpoint showing cache statistics



(b) Health endpoint with uptime



(a) Stop endpoint (dev only)



(b) Structured JSON logging output sample

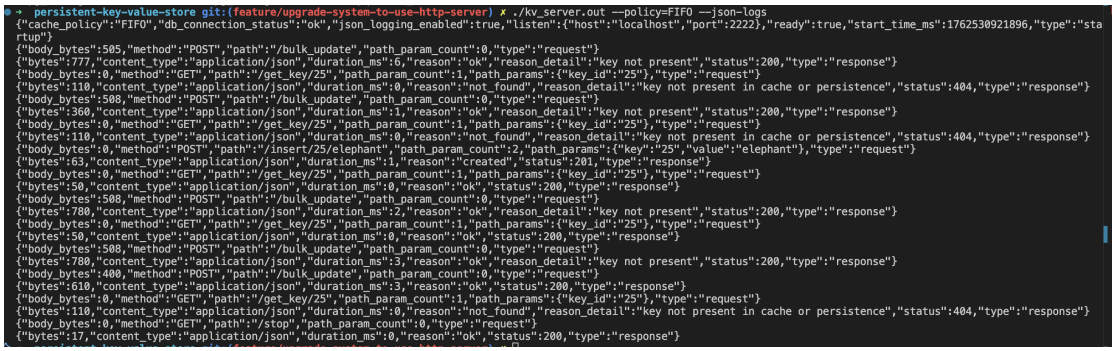


Figure 11: Server startup with custom cache eviction policy.

6 Credits

Third-party libraries: nlohmann/json and cpp-httplib. Licensed under MIT (see project LICENSE).