Lebron2016x4 Project Report

By Harris Doan, Umair Khan, Jonathan Si, Eliot Yoon

Objective:

The goal of our project is to design and implement a modular, extensible HTTP web server. The server needed to support multiple handlers for different types of HTTP requests, while also adhering to best practices. This included separation of concerns, dynamic routing, dependency injection, and testability. We also wanted to extend the base functionality by implementing additional features such as sleep delays, health checks, an in-memory CRUD API, and finally support for Markdown file rendering.

Specific Contribution:

Custom-built Components:

- **Request/Response Infrastructure**: We build a robust HTTP parser and formatter for incoming requests and outgoing responses to valid requests.
- **Router**: Designed a longest-prefix matching router that maps URL paths to appropriate request handlers.
- **Handlers**:
    - EchoHandler: Simply echoes back incoming requests for testing purposes.
    - SleepHandler: Simulates latency by introducing configurable delays.
    - HealthHandler: Reports service health with a simple 200 OK.
    - CrudApiHandler: A lightweight in-memory CRUD API for storing/retrieving JSON blobs.
    - MarkdownHandler: Converts Markdown (.md) files into HTML responses in real time for displaying documentation on the web. Also provides a UI for live Markdown editing and previewing.

Third-Party Libraries:

- Integrated cmark (CommonMark C Library) to perform parsing of Markdown files and Markdown-to-HTML conversions.
- Used <filesystem> from C++17 to ensure secure file path resolution and canonicalization.

Tools:

- **Markdown Parsing (cmark)**: A new tool that we are utilizing for our new feature to our webserver project is the usage of the CMark library for parsing and converting Markdown to HTML. In order to eliminate the complexity and overhead of writing our own Markdown parser, we decided to use an external third-party library. CMark handles the proper handling of converting Markdowns for us which allows for us to create a proper HTTP response and serve the proper converted files.
- **Docker**: Configured Docker to streamline builds, testing, and portability across team members.
- **Google Cloud Shell/Google Cloud Platform**: Used to help develop, test, and deploy our web server in a live environment using GCP. Also set up a continuous build system for real time updates when pushing code to production.
- **CI/Testing**: Wrote unit and integration tests for handlers and functions using GoogleTest (gtest).

Methodology:

Key Classes/Files:

- **/src/server.cc** and **/src/session.cc**: Used to manage low-level TCP communication and multithreaded sessions.
- **/src/router.cc** Used to perform longest prefix-based route dispatching to registered handlers.
- **/src/handler_registry.cc**: Used to register our handler factories for dynamic instantiation of the necessary handler.
- **/src/markdown_handler.cc**: Our newest handler that adds further functionality to our modular web server. Dynamically parses and serves .md files as HTML responses without saving any rendered files.
- **/src/markdown_converter.cc**: Contains the logic to parse and convert Markdown files safely.

Testing:

- Added robust unit testing for:
  - URL path traversal handling
  - Valid and invalid Markdown rendering
  - CRUD operations (POST/GET/PUT/DELETE)
  - Static file serving and MIME-type resolution
- Example endpoint tests:
  - GET /api/42 returns a stored JSON on disk
  - GET /markdown/index.md returns a valid HTML body of the properly converted Markdown file

Evaluation:

Performance:

- Efficient request routing using our Router::get_routes()
- Markdown rendering is performed in-memory only when requested – no HTML file is ever written to disk.
- StaticHandler and MarkdownHandler ensure secure file access via canonical path resolution. This prevents unwanted path traversal.

Security:

- Markdown rendering uses CMARK_OPT_SAFE to strip away any raw HTML and prevent XSS.
- Path traversal attacks are mitigated by checking that resolved paths stay within the root.

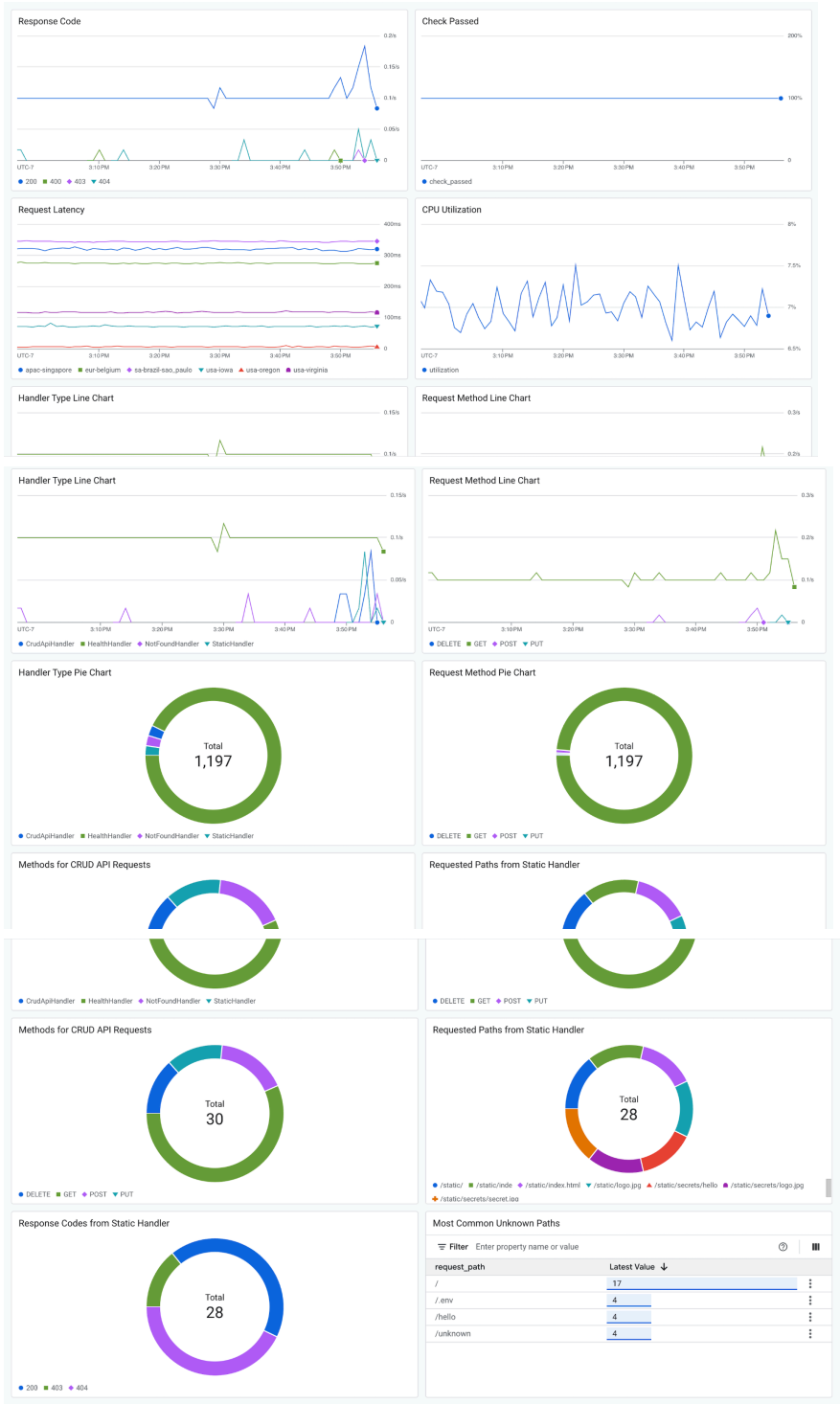
Viewing/Accessing the Web Server, Demos:

- You can view our deployed web server at the following URL: http://34.169.145.188
  - Do note: An empty path will return a 404. Please specify the static file path to the desired file you would like to view.
- To see more about what we have to offer you can go to:
  - Live Home Landing Page URL: http://34.169.145.188/static/home.html
  - Live Markdown Rendering URL: http://34.169.145.188/static/render.html
  - Live Markdown Demo URL: http://34.169.145.188/docs/demo.md
  - Live StaticHandler Demo URL: http://34.169.145.188/static/logo.jpg

How to Demo the Markdown Render Capability:

- To demo any static Markdown file, all you need to do is go to the path /docs/filename.md. In order to render user-generated Markdown files, you can go to the page at static/render.html, which contains a Markdown Renderer capable of live Markdown editing.

Images:


Response Code
Check Passed
Request Latency
CPU Utilization
Handler Type Line Chart
Request Method Line Chart
Handler Type Line Chart
Request Method Line Chart
Handler Type Pie Chart
Total
1,197
Request Method Pie Chart
Total
1,197
Methods for CRUD API Requests
Requested Paths from Static Handler
Methods for CRUD API Requests
Total
30
Requested Paths from Static Handler
Total
28
Response Codes from Static Handler
Total
28
Most Common Unknown Paths
Filter
Enter property name or value
request_path
Latest Value
/
17
/.env
4
/hello
4
/unknown
4