Backend CPP Developer Recruitment
Practical Task

Callstack

January 2025

1 Practical task

1.1 Introduction

The task ahead of you to solve is to implement a basic REST API server for a simplified telemetry service, without the persistence layer.

The server you will be building will be a simplified version that only mocks performing some tasks. Its goal is to track the interaction times within consecutive screens on fixed-length user paths in the application. For instance, consider a case of a shopping cart area where the user first clicking a button (starting the path) from which the time is counted until they enter their name, click next, insert their address, etc. For the purpose of this task, consider the length of this path to be 10, meaning each time your program should accept events being 10 time duration numeric values expressed in seconds.

The endpoints are specified in section 1.2.

1.2 Technical requirements

Feel free to use any libraries you prefer - there is no preference as to the tech stack. In case you don't have any idea to start, we recommend to try using Boost as shown in this small server example.

Your project should be prepared using CMake as the build tool.

The specification of endpoints is as follows:

1.2.1 Get mean path length

Endpoint path: `/paths/{event}/meanLength` where `event` is the event name

HTTP method: `GET`
Request:

```
{
    resultUnit: "seconds" | "milliseconds",
    startTimestamp?: number,
    endTimestamp?: number
}
```

where `startTimestamp` and `endTimestamp` are - respectively - optional start and end timestamps by which to filter the event records to calculate the mean length of; if either is not passed, the missing filter is inactive and should not limit the results.

Response: `{ mean: number }`

1.2.2 Save event

Endpoint path: `/paths/{event}` where `event` is the event name

HTTP method: `POST`
Request: `{ values: number[], date: number }`
Response: `{ }`
This endpoint should save the submitted entity to a store (for the purpose of this task, it may be a `std::map<std::string, std::vector<...>>` mapping event names to vectors being the event data - no complex / durable persistence is required).

1.3 Grading criteria

- code style
- memory management
- OOP - your project's components should be structured in classes and use hermetization properly
- language constructs (nice to have lambdas, smart pointers, containers)
- functionalities (listed below)
- architecture: feel free to use your preferred backend code architecture (the setup can be simple but should demonstrate the knowledge of a widely-used concept)
- async networking: the server should use asynchronous IO or threading and properly lock resources
- request parsing
- compliance with technical requirements in section 1.2

Note 1: all endpoints shall respond with a proper HTTP status (e.g. `200 OK`) and have the `Content-Type` header set to `application/json`.

Note 2: persistence of data is not the goal of this project.

Note 3: the project should not be optimized to handle a large volume of requests / large frequent of requests (as a telemetry server would normally do).