Quick Start

Run `./gradlew bootRun` from the top-level directory to start the server. It will be hosted at http://localhost:8080. The server accepts a GET request at http://localhost:8080/github/{username}, where the username is the username for which we want to retrieve Github data. It will return

- 200 with the requested information if successful
- 400 if there is an IllegalArgumentException, such as
    - Invalid username
- 404 if there is a ResourceNotFoundException, such as
    - The provided username wasn't located by Github
- 422 if there is a BadUpstreamResponseException, such as
    - The body returned from the request couldn't be parsed.

Architecture

The architecture follows a Controller-Service-Repository pattern, though somewhat loosely on the "repository" part. By loosely, I mean it's not being persisted to the device, and it can be cleared out with interaction.

Decisions

I went with Java/Spring Boot because of my experience with them both. My second choice would have been Go, but I haven't used Gin before and didn't want to spend precious time learning a new library.

I decided on a Controller-Service-Repository pattern, as it most tightly fit the requirements. Even as the project grows (get data from bitbucket, gitlab, etc), this pattern would fit nicely with what the end goal would be. Beyond that, I chose a cache instead of a HashMap to back the repo simply to prevent the memory footprint from growing out of control. If this was a long-running service, I might go with something more permanent.

While it's required to have the profile data returned, repos for that user could potentially be null. In keeping with this, a failure to retrieve the repos for a user simply logs such and continues on as if the user had none. I wanted to return as much information to the user as possible.

I didn't add tests for the controller. Outside of the Spring setup, the function simply calls another and returns the result in response form. I don't typically find it useful to write a test to confirm a function simply called another function.

I went back and forth on the status code for the "request body unable to be parsed" path. I considered

- 422 (Unprocessable Entity) - I landed on this, and it makes the most sense to me, as we've gotten back a body but were unable to parse it for an unknown reason.
- 502 (Bad Gateway)/503 (Service Unavailable) - I thought about these for a second because I wanted to try to communicate to the user that it's not my service that's having an issue but an upstream one.
- 419 (I'm a teapot) - it really shouldn't happen, as we've gotten a 200 back at that point, so we should be able to expect the body to be in the form we expect.

Future Improvements

Given more time, some features I would add/tweaks I would make are

- Docker-ize the container for easier running
- Read environment variables for
    - Github Token
    - Max size of the cache
    - Expiration duration of the cache
- Use something like PowerMock to test/mock private functions