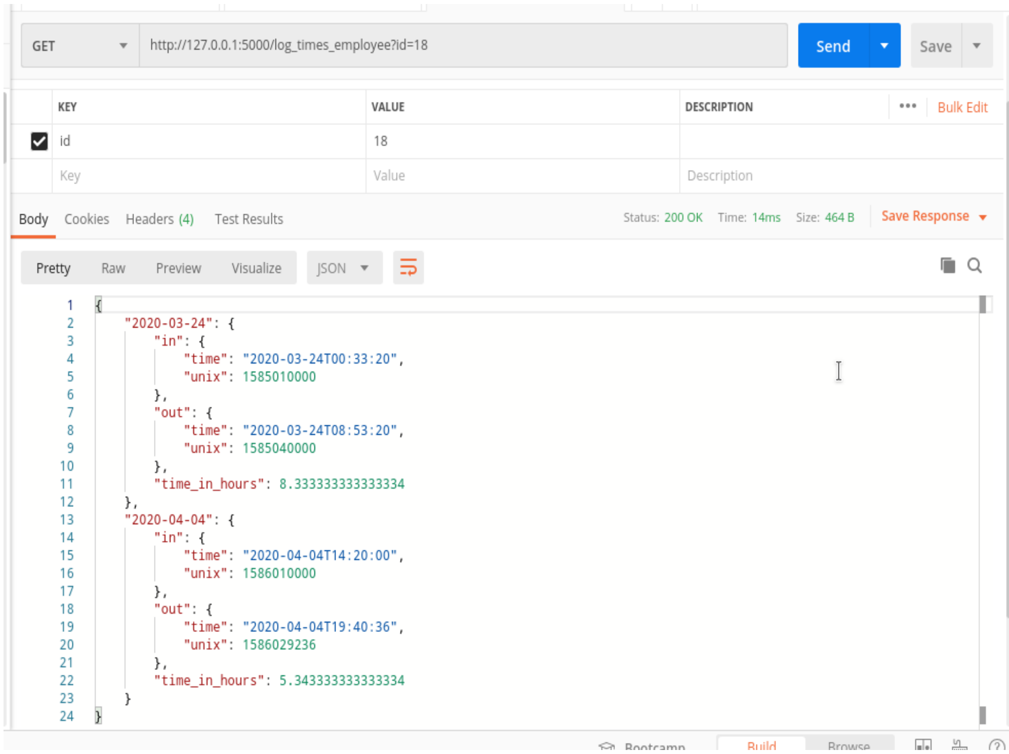
Patryk Stroński 242564

Lab 5, part 1

Intro

My RFID application is a monolith made of two important parts. The first part is RFID reader itself, which reads data and saves them to database, and the second one is a Flask server being a simple interface to fetch/edit user data in the application. The API is of REST type. Every information is saved in PostgreSQL database.

The application purpose is to track employees' working time. Each employee has the id card (called 'badge') with RFID number encoded in it. Each employee has one's own card so based on card uid (unique identifier) it is possible to track when employee comes and leaves work. The working time is generated into such report obtained from the application api:



Saved data:

The application has exactly two tables *users* and *card_logs*.

Users table stores name, last name and assigned badge uid of an employee. Uid is reflected as a string. Moreover the table is indexed with automatically incremented field *id*. The table declaration looks like this:

"CREATE TABLE IF NOT *EXISTS users(id SERIAL PRIMARY KEY,u_name VARCHAR(20), l_name VARCHAR(20), uid VARCHAR(100));"*

The *card_logs* stores the uid and time of card log. The time is stored as unix timestamp. Timestamp is the easiest way to store time very precisely and is easily parsable by python. The declaration of the table looks like this:

"CREATE TABLE IF NOT EXISTS card_logs(id SERIAL PRIMARY KEY,uid VARCHAR(100), log_time BIGINT);"

Read data:

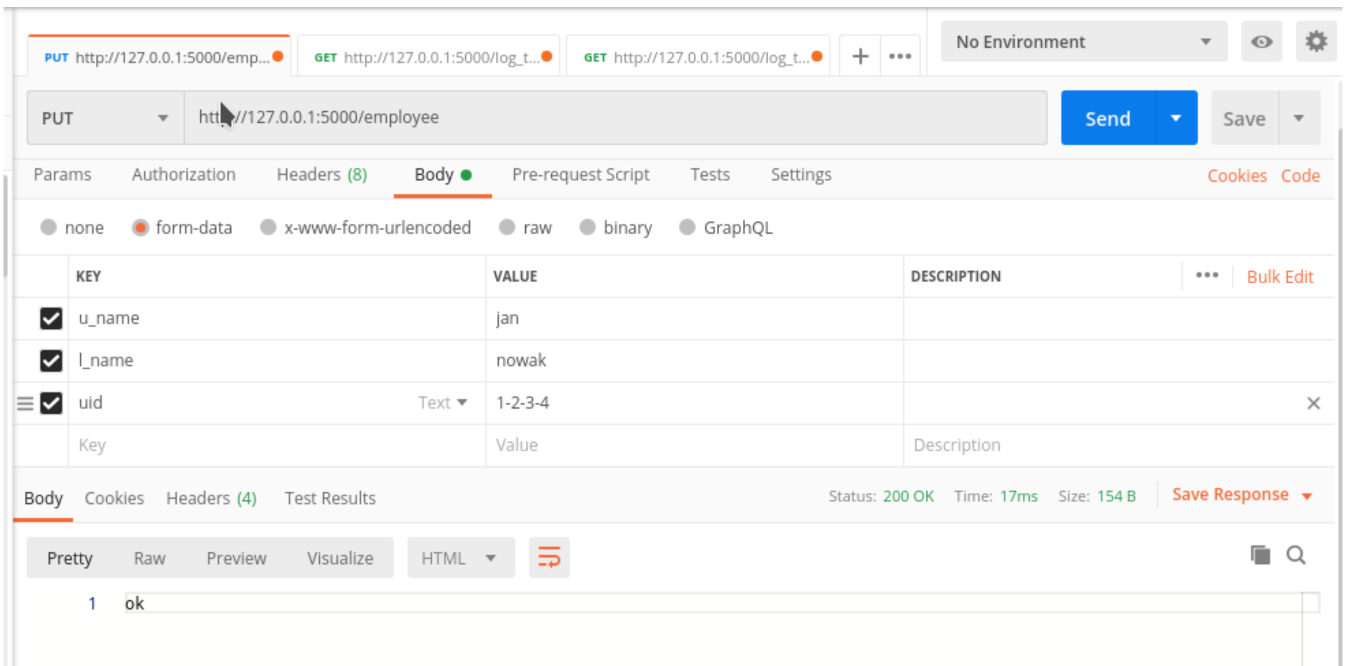
The data is read using MFRC522 driver the reader stops reading if CTRL+C or red button (on RaspberryPi) is pressed. Each time, when the badge is read, the reading time (as unixtime) and badge uid is saved to the database as new record.

NOTE: This part I haven't tested due to lack of proper equipment

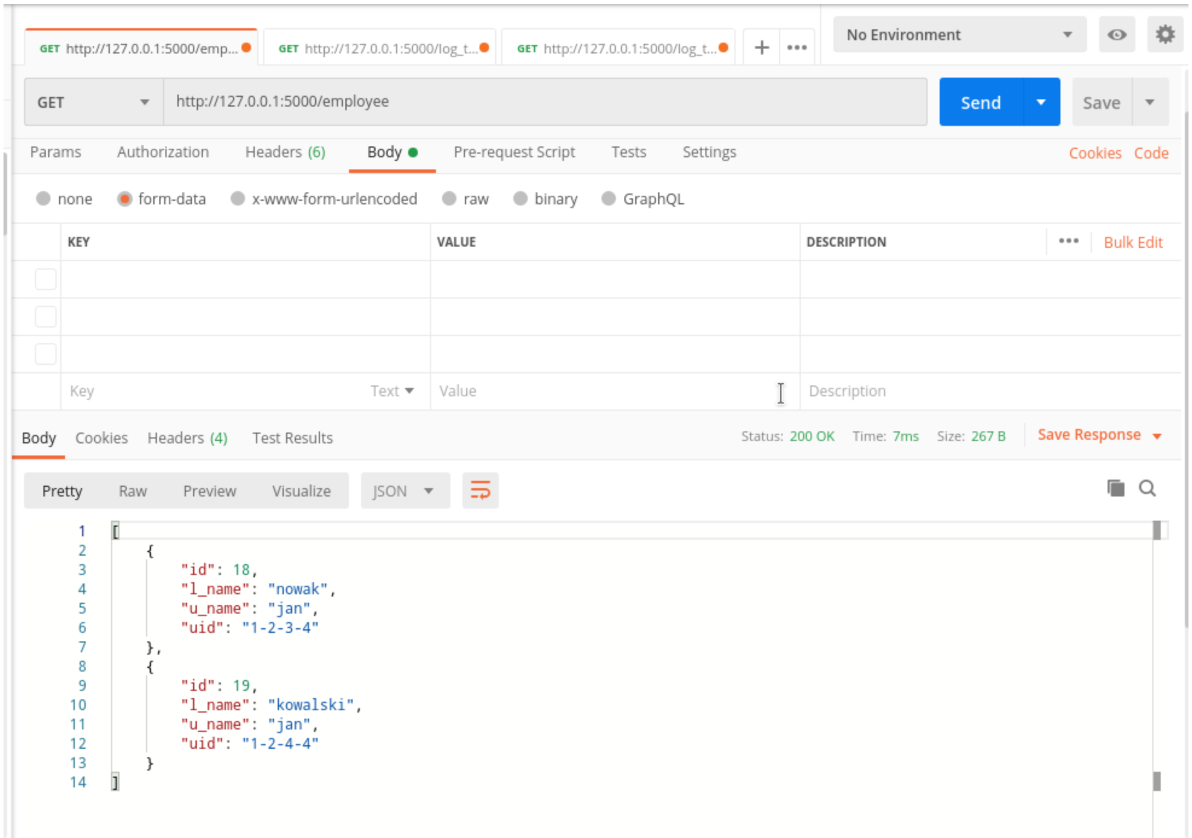
Employee management

In order to manage employees the simple server is running. This part will be separated in part 2 of the laboratory. There are the following endpoints:

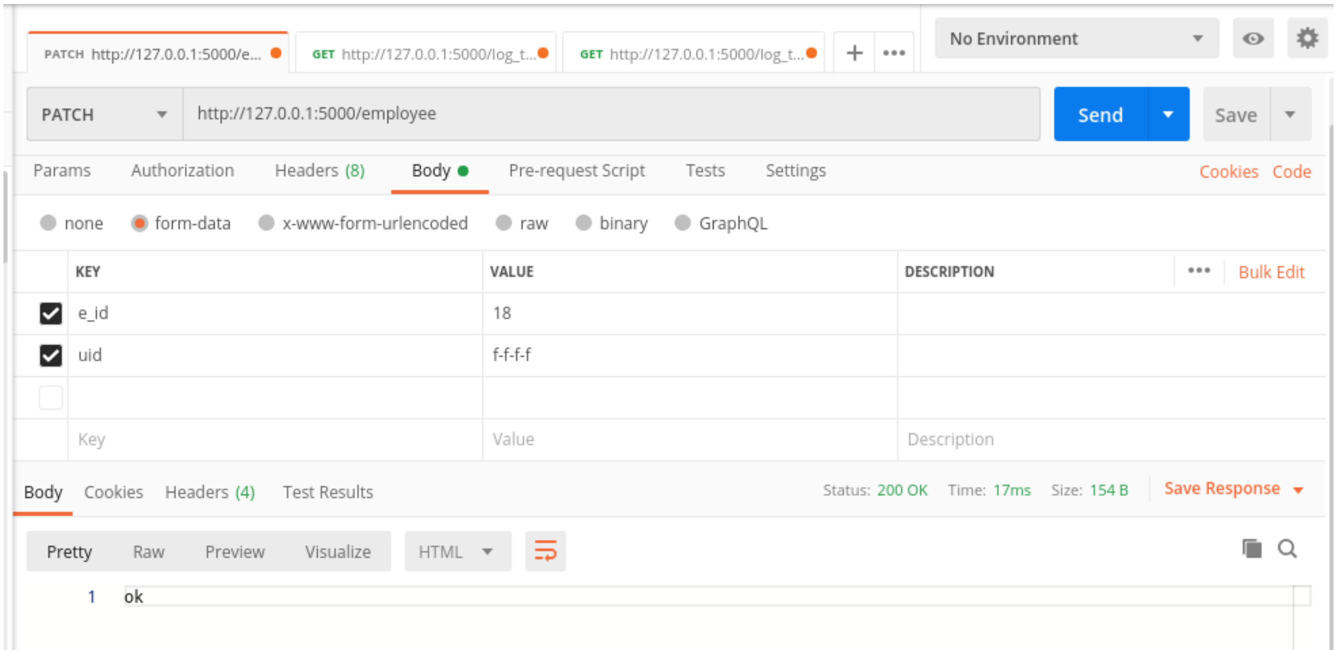
PUT /employee – adds new employee.It is required to specify in JSON, te following fileds: u_name, l_name, uid. From those fields employee name, last name and card uid are read. Example using Postman:

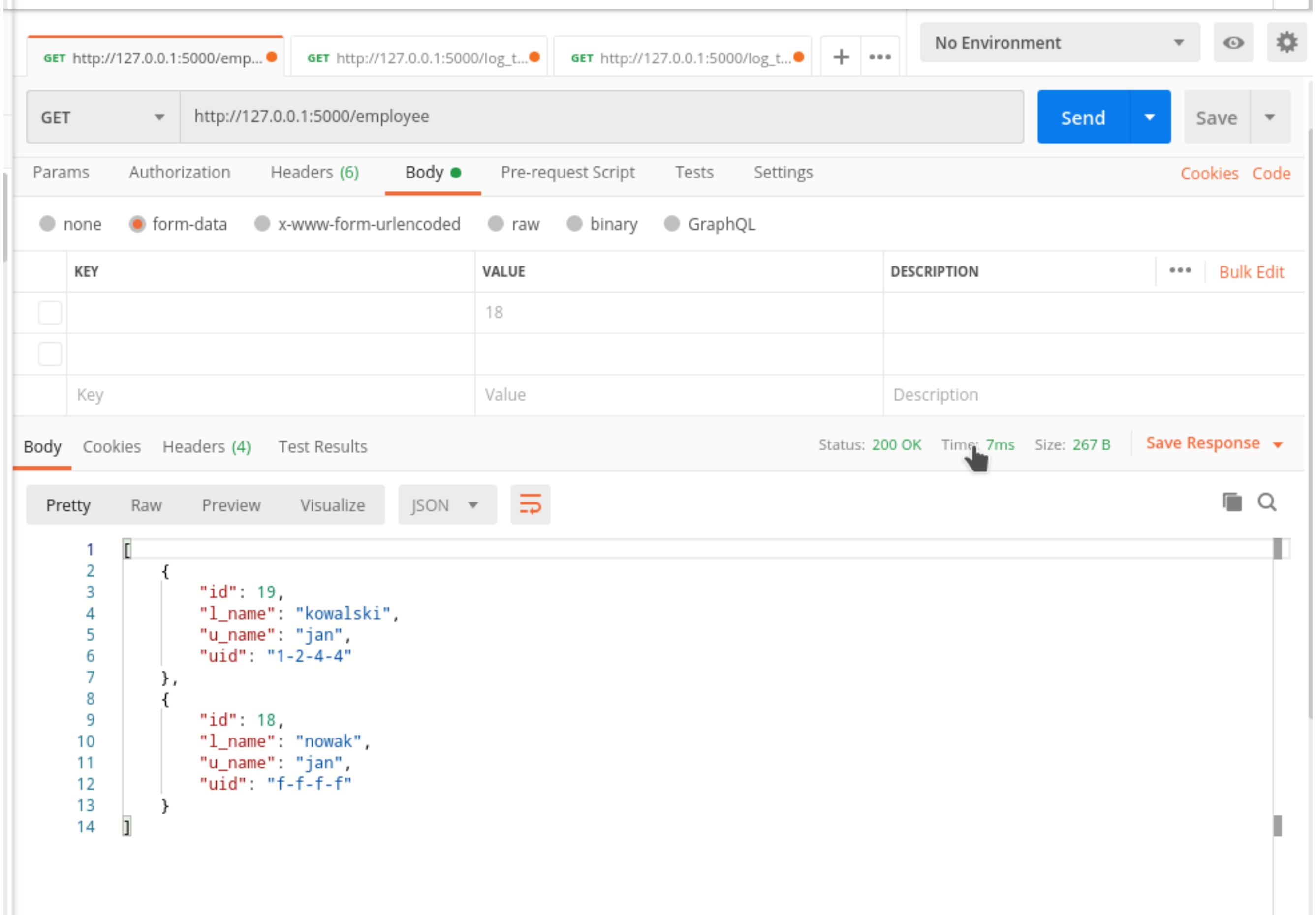


GET /employee – gets the list of all employees

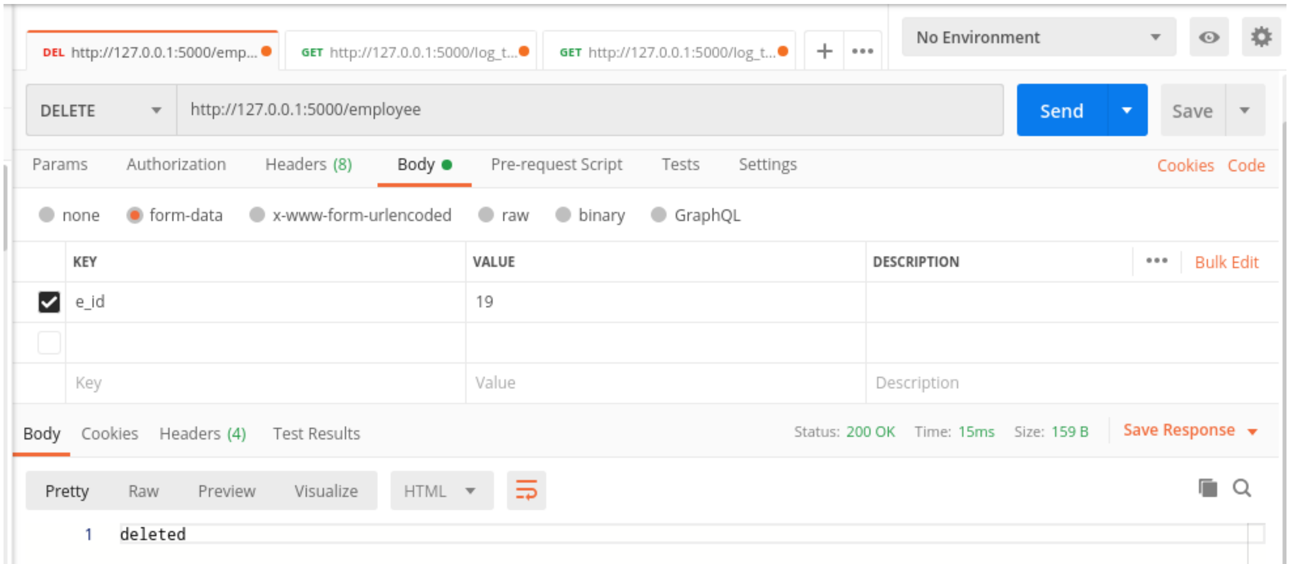


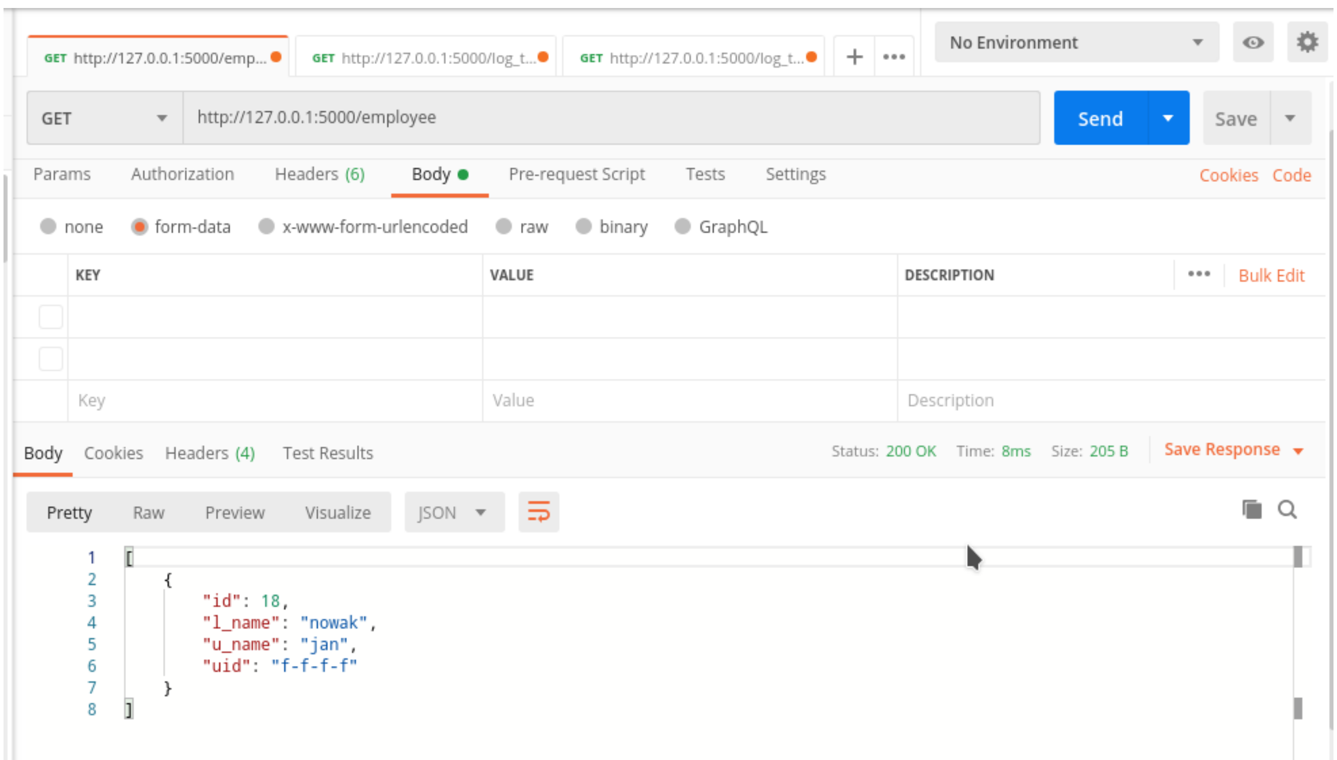
PATCH /employee – by providing e_id (employee id as shown in GET) and new or empty uid the system edits the users uid (assigns new or leaves the field empty).



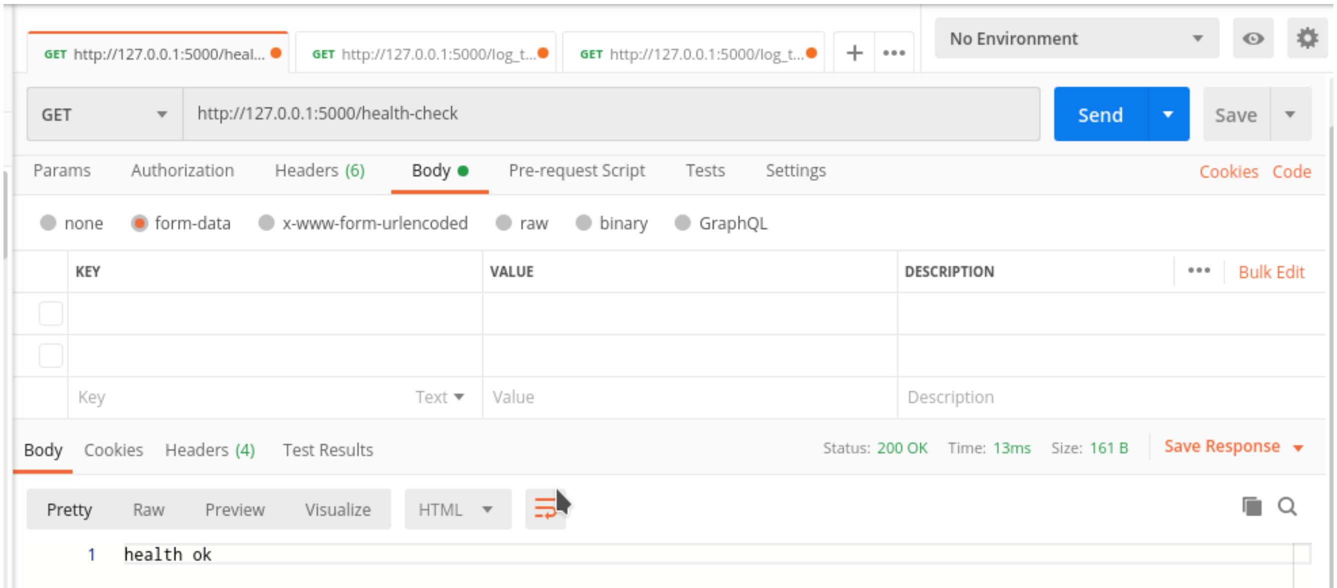


DELETE /employee – by providing e_id to this endpoint, the employee is deleted

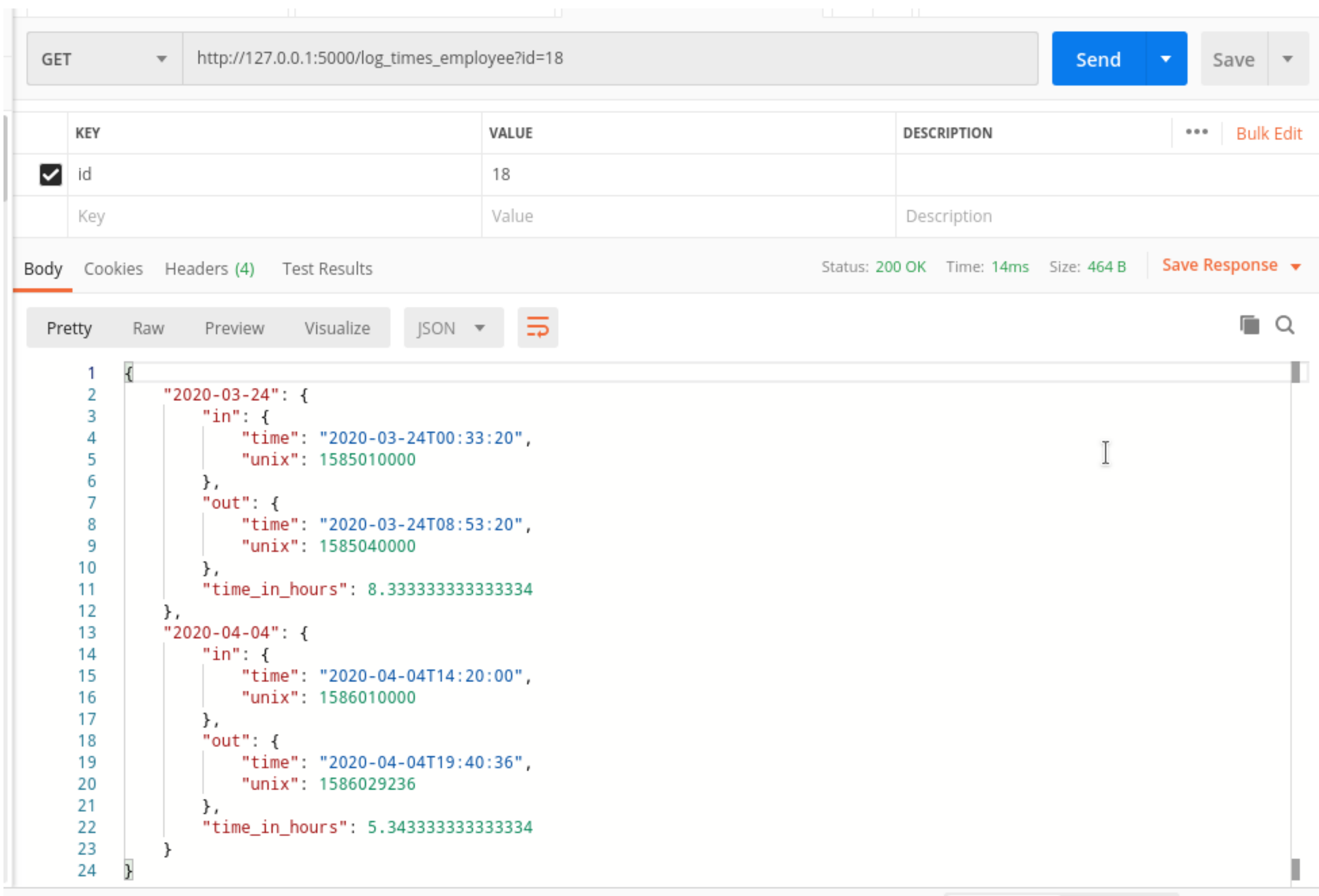




GET /health-check – this endpoint is only used to check whether server is running. If it is, it sends “health ok”.



GET /log_times_employee – this endpoint gets the report of when did the employee come to work and when did the one left the work



How to run the application

In order to run the application, you need to have PostgreSql installed and running. The configuration regarding the database, user, host and passwords are stored in db_management.py file. You need to edit them before running to suit your set-up. Apart from it, all modules from file dependencies.txt need to be installed using pip and python in version 3.6 or higher. Then, using terminal, use *cd* to go to directory where the application is, and *index.py* is stored. First, export an environment variable . In RaspberryPi or Linux machine;

$ export FLASK_APP=index.py

And afterwards

$flask run

The application should be running like this:

```
* Serving Flask app "index.py"
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```