CS601-01/CS601-02: Lab 6[1]: RESTful API Servers

Due Date: Nov 3, 11:59pm (60 pts)

For this lab, you will write two webservers that provide an API to access hotel data:

- A webserver that uses raw sockets to communicate with clients
- A webserver based on Jetty/servlets

Both servers will provide a Restful API to access hotel information (general hotel info and reviews). Requirements for each of the two servers and the API are described below:

RawSocketsHttpServer

Your server must explicitly use raw sockets for this lab. Your server should be able to handle Http GET requests from clients. If a client sends any other type of Http request, the server should return the response code: `405 Method not allowed`.

- Your basic server should be general, application independent, **not** specific to the hotel data app (except for the provided API).

- You may *not* use any libraries for parsing HTTP messages including headers and the body of the message.

- Your server should be **multithreaded.** You should create a welcoming socket to listen for client connections; once a new client request comes in, your program should create a connection socket for this client and add a new Runnable task to the WorkQueue (or feel free to use ExecutorService). In the run method of the Worker you should get the input stream of the socket and read the client request. Your code needs to parse a client's request (including the header), and send an appropriate response back to the client (see the section on the API).

JettyHttpServer

This server should provide the same service, but should be written using Jetty&servlets. See examples on using Jetty and servlets in the **servlets** folder in the **Examples** repository on github.

API for accessing hotel data

Both servers need to provide a Restful API to access hotel data. Both servers need to provide the following API endpoints:

[1] The assignment is modified from the original assignment of Prof. Rollins.

- **/hotelInfo** This endpoint is for accessing general information about a particular hotel (name and address) It requires the **hotelId** parameter like in the following sample request: http://localhost:8080/hotelInfo?hotelId=25622

  The response to the GET request sent to /hotelInfo that includes a **valid** hotel id (that exists in the ThreadSafeHotelData) should be a JSON file in the following format:

  ```
  {
  "success" : true,
  "hotelId": "25622",
  "name": "Hilton San Francisco Union Square",
  "addr": "333 O'Farrell St.",
  "city": "San Francisco",
  "state": "CA",
  "lat": "37.786160",
  "lng": "-122.410180"
  }
  ```

  If the provided hotelId does not exist in the HotelData, the response should be a JSON file in the following format:

  ```
  {
  "success" : false,
  "hotelId": "invalid"
  }
  ```

- **/reviews** This endpoint is for accessing reviews for a given hotel. It requires two parameters: **hotelId** and **num** (the number of reviews to return) as in the following request: http://localhost:8080/reviews?hotelId=10323&num=2

  If the hotelId parameter corresponds to a **valid** hotel, your webserver should return a JSON in the following format:

  ```
  {
  "success": true,
  "hotelId": "10323",
  ```

```
 "reviews": [
    {
     "reviewId": "aXdsoJShow25vnla",
      "title": "Nice clean hotel",
      "user": "Bob15",
      "reviewText": "The location is perfect, close to all attractions. Lots of good places to eat nearby.",
      "date": "09:05:16"
    },
    {
     "reviewId":  "uqGBAsfnsHah47h",
      "title": "Overpriced!",
      "user": "HelenTravels",
      "reviewText": "Nice, but expensive for the area. A bit noisy.",
      "date": "10:19:16"
    }
 ]
}
```


If the hotelId that was provided does not exist or the **num** parameter is invalid, the response should be:


```
{
"success" : false,
"hotelId": "invalid"
}
```


- **/attractions**

This end point is for accessing tourist attractions nearby a particular hotel. It requires the **hotelId** parameter and a **radius (in miles)** like in the following sample request: http://localhost:8080/attractions?hotelId=25622&radius=2

that should return tourist attractions within a radius of 2 miles from hotel with id=25622. If the request is valid, the webserver should return a json file in the same format as in lab 4 (as returned by Google Places API).

Implementation

There is no starter code for this lab. You might want to start with your lab 3, part 2 code (or your lab 3 part 1 code depending on whether you plan to use WorkQueue or ExecutorService). You will also use some code from Lab 4 for this lab. Before you start your server, you need to load all the hotel data (both general hotel info and reviews) into your data structures from the input files. For getting tourist attractions, you can either first store attractions info in the map or in the files prior to running the servers, or access this info on the fly from Google Places API). I recommend the former.

For an extra credit, you can get tourist attractions from a website like tripadvisor by writing a **scraper**. Discuss with the instructor prior to implementation.

Deployment

Every student needs to deploy their two servers on the **microcloud node** assigned to them (your servers should be listening for connections on the ports that were assigned to you). Microcloud nodes are on the departmental private server subnet (10.0.1.0/24). The gateway for this network is *angry.cs.usfca.edu* (10.0.1.1). The DNS names for the nodes range from mc01.cs.usfca.edu through mc12.cs.usfca.edu, although in our class we will only use nodes mc07 to mc12).

The pdf with the microcloud assignment is posted under Modules on Canvas. Make sure your server will continue running, even after you log out of the node. The instructions on running your servers on the microcloud node will be provided in a separate file under Assignments->Lab 6.

Javadoc Comments

You are required to add javadoc comments to your code (above each class and above each public method).

Tests

A test file will be provided by the instructor on Monday. The test will send requests to a webserver and check the responses. The test file will be in Python and will require a Python **requests** package.

Submission:

Apart from deploying your servers on the microcloude node, your lab6 code should also be submitted to your private lab6-username repo on github. Please keep pushing your code to github as you work on the lab.

This is a larger lab and I expect to see **at least 10 commits in your repo**. Your github history should be meaningful - I should be able to look at it and understand how you

iteratively developed your code, how you fixed bugs etc. Commit messages should also be meaningful - they should describe what you accomplished in the specific commit.

Several students in the class will be asked to come for a code review for lab 6. Inability to explain your design or your code will result in a 0 for the assignment.

No collaboration is allowed on this project. You may not discuss implementation details with anybody except for the instructor and the TAs.