Programming Assignment 2

Goal

- Get familiar with HTTP protocol. Understand how web API works.
- More practice on socket API.

Description

In this assignment, you will design and implement a dynamic web server that provides the following services.

/api/evalexpression

This API is to help clients evaluate arithmetic expressions and return them the results. The client is going to send an arithmetic expression in an HTTP `POST` request. See the example below where the client wants to evaluate expression `7+9-11+6`.

```
POST /api/evalexpression HTTP/1.0\r\n
Content-Length: 8\r\n
\r\n
7+9-11+6
```

And your server should return the following HTTP response.

```
HTTP/1.0 200 OK\r\n
Content-Type: text/html\r\n
Content-Length: 2\r\n
\r\n
11
```

Note: this API should only support simple arithmetic expressions:

- only integers
- '+' and '-' operators
- can have '(' and ')'

For all unsupported arithmetic expressions, your server should return an HTTP 400 response (bad request).

/api/gettime

This API is to help clients get the local time on the server. The client is going to send an HTTP `GET` request. See the example below.

```
GET /api/gettime HTTP/1.0\r\n
\r\n
```

And your server should return its local time in a human readable string format. For example, you can return a response like below.

```
HTTP/1.0 200 OK\r\n
Content-Type: text/html\r\n
Content-Length: 24\r\n
\r\n
Sun Sep 29 07:41:37 2019
```

You have the freedom to choose the time format you like.

/status.html

This regular HTML page, and should show the status information of your web server. It should contain

- The number of API calls for (`evalexpression` and `gettime`) during the last minute, last hour, last 24 hours, and lifetime.
- The most recent 10 expressions clients submitted to evaluate.

Your server needs to return a valid HTML page that is able to render successfully inside a browser. For instance, the page you return can look like

```
......
<h1>API count information</h1>
<h3>/api/evalexpression</h3>
```

```
<ul>
  <li>last minute: 2</li>
  <li>last hour: 10</li>
  <li>last 24 hours: 128</li>
  <li>lifetime: 314</li>
</ul>
<h3>/api/gettime</h3>
<ul>
  <li>last minute: 1</li>
  <li>last hour: 2</li>
  <li>last 24 hours: 2</li>
  <li>lifetime: 7</li>
</ul>
<h1>Last 10 expressions</h1>
<ul>
  <li>7+8-11</li>
  <li>1+1+1+1+1+1+1</li>
  ......
</ul>
......
```

You don't need to worry about persisting historical count, just need to collect these stats since the start of the server is sufficient.

Others

For all other URLs, your server needs to return an HTTP 404 response. (e.g. /api/whatisthis, /index.html, etc.)

Requirements

- You can only use the socket API we covered in class. You are not allowed to use any other higher level modules like `HttpServer`, `HttpUrlConnection`, etc.
    - When in doubt, please ask before you use it.
- Your server should be able to handle both HTTP/1.0 and HTTP/1.1 requests.
    - Don't need to handle multiple HTTP requests if the client is using version 1.1. Just close the TCP connection on the server side once the first request is handled.
- Server should be multithreaded.
- On Java implementation
    - Can only use java.io.**InputStream** to read from socket
    - Can only use java.io.**OutputStream** to write to socket
    - If max buffer size is 16 if using
        - `write(byte[] b, int off, int len)`
        - `read(byte[] b, int off, int len)`

Grading policy

100 points in total.

- [20 pts] Correctly parsing HTTP request and sending HTTP response
  - E.g. read/write socket according to the HTTP protocol, extract Content-Length, etc.
- [30 pts] Implement /api/evalexpression
  - Correctly parse the expression from HTTP POST request.
  - Able to evaluate expressions only involving integers, '+', and '-' correctly.
  - Correctly send HTTP response.
  - For unsupported expressions, correctly send HTTP 400 responses.
- [10 pts] Implement /api/gettime
- [30 pts] Implement /status
  - Can correctly show accumulated stats for /api/evalexpression and /api/gettime
  - Can correctly show the latest 10 expressions clients sent.
- [10 pts] For other URLs, return HTTP 404 response

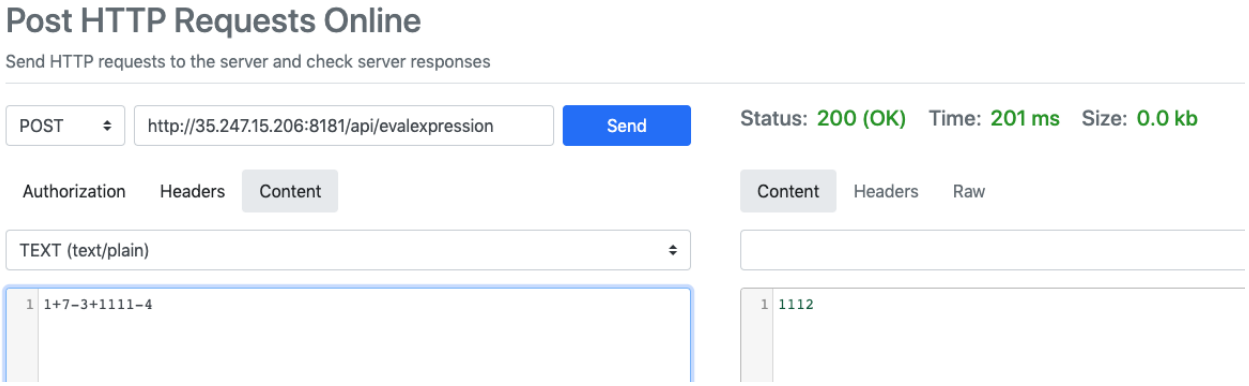
How to test your server

There are a few different ways to test your server implementation.

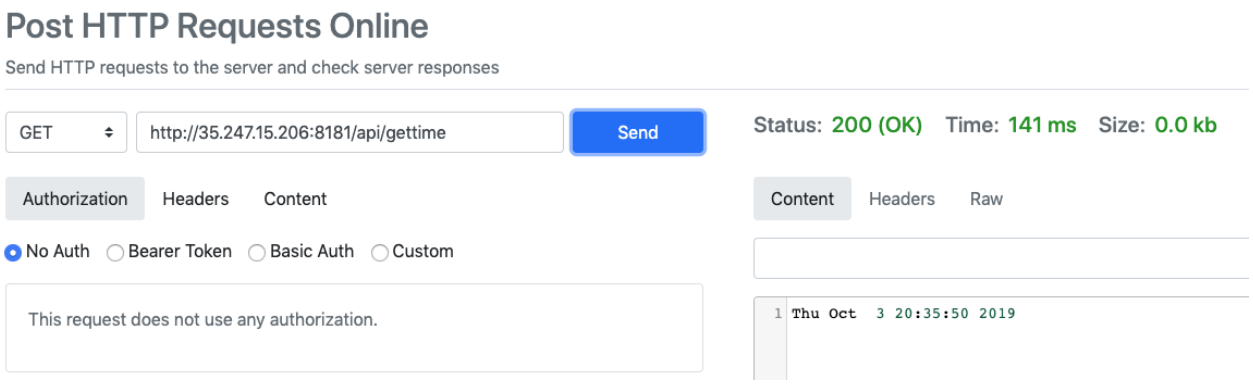
Online tools

You can use https://reqbin.com/ (or similar online tools) to send HTTP requests to different endpoints. To test your code using this method, you need to firstly run your code on a server with a public IP address.

For example, you can use it to send POST request, and test /api/evalexpression API.



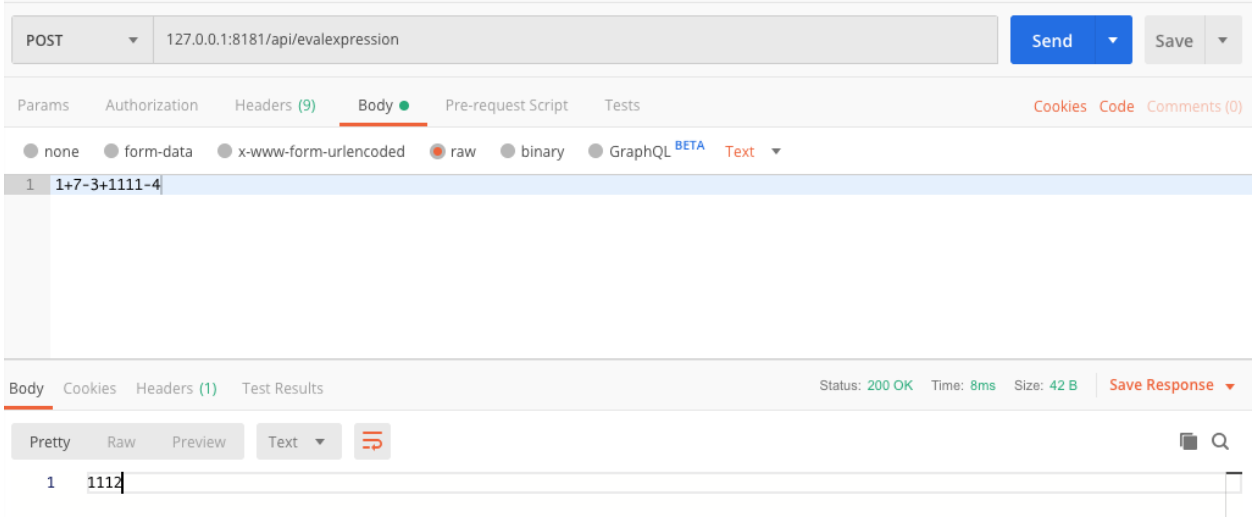
Similarly, you can use it to send GET request, and test /api/gettime API.



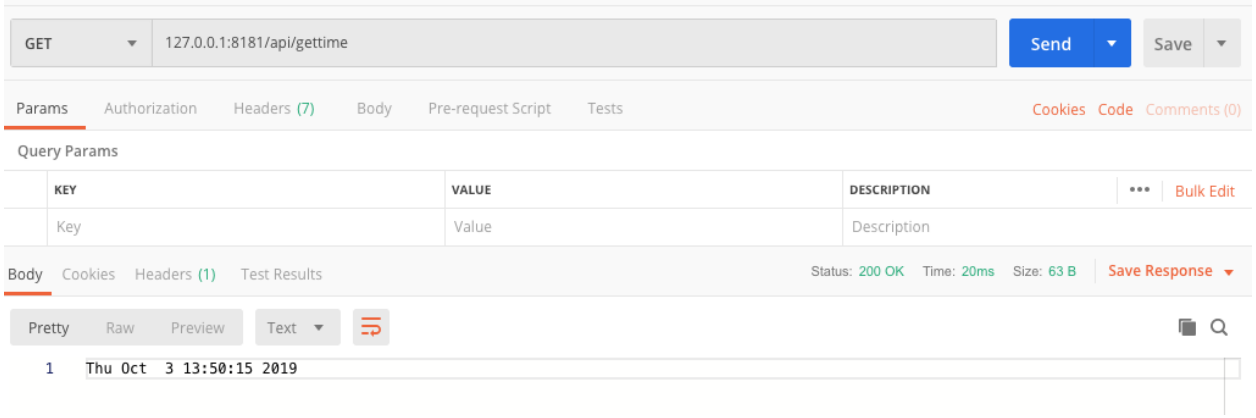
Postman

You can download Postman to get your implementations locally, without running time on a server with a public IP address. In this case, use a loopback IP address (`127.0.0.1`) and run your server on your laptop. Then you can use Postman to set HTTP requests.

For example, you can use it to send POST requests, and test /api/evalexpression API.



Similarly, you can use it to send GET requests, and test /api/gettime API.



Curl

Curl is a tool used to transfer data to or from a server. It supports HTTP protocol. You can test your implementation using

```
curl localhost:8181/api/gettime
curl -d '1+2+3-4' localhost:8181/api/evalexpression
```

Telnet

Alternatively, you can simply use telnet and test your code. See examples below.

```
[01:55:18] ~ $ telnet localhost 8181
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET /api/gettime HTTP/1.0

HTTP/1.0 200 OK
Content-Length: 24

Thu Oct  3 13:55:42 2019
```

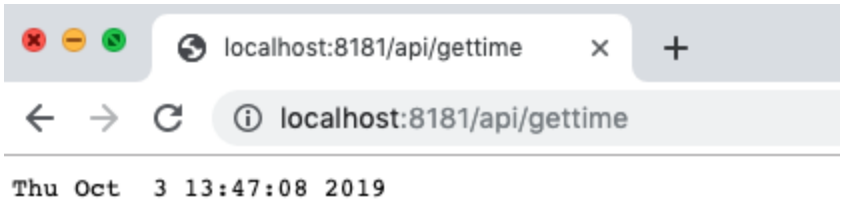
```
[01:58:34] ~ $ telnet 127.0.0.1 8181
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
POST /api/evalexpression HTTP/1.0
Content-Length: 12

1+7-3+1111-4
HTTP/1.0 200 OK
Content-Length: 4

1112
```

Browser

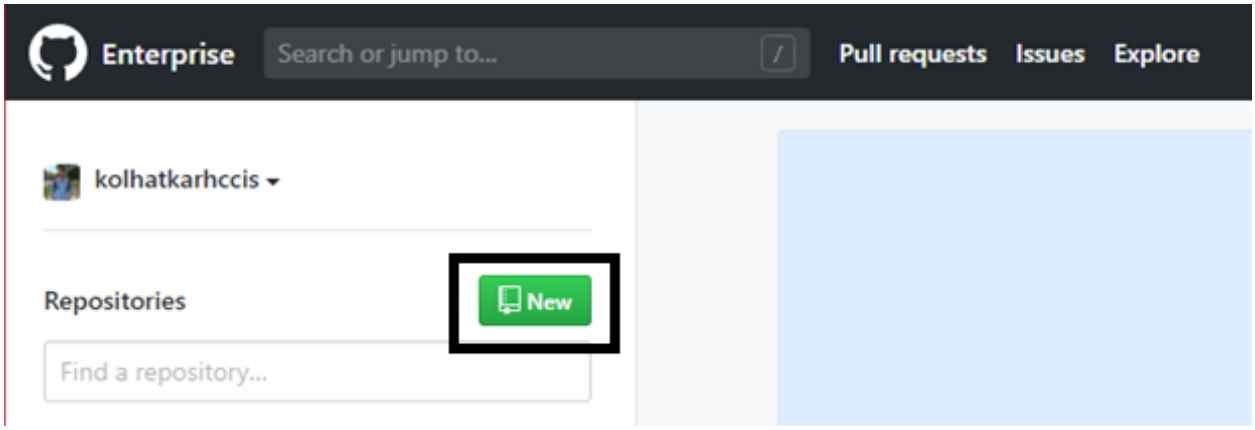
You can also do some testing using just browser. It is very easy to test GET request in this way. For example, use browser to test /api/gettime as below.



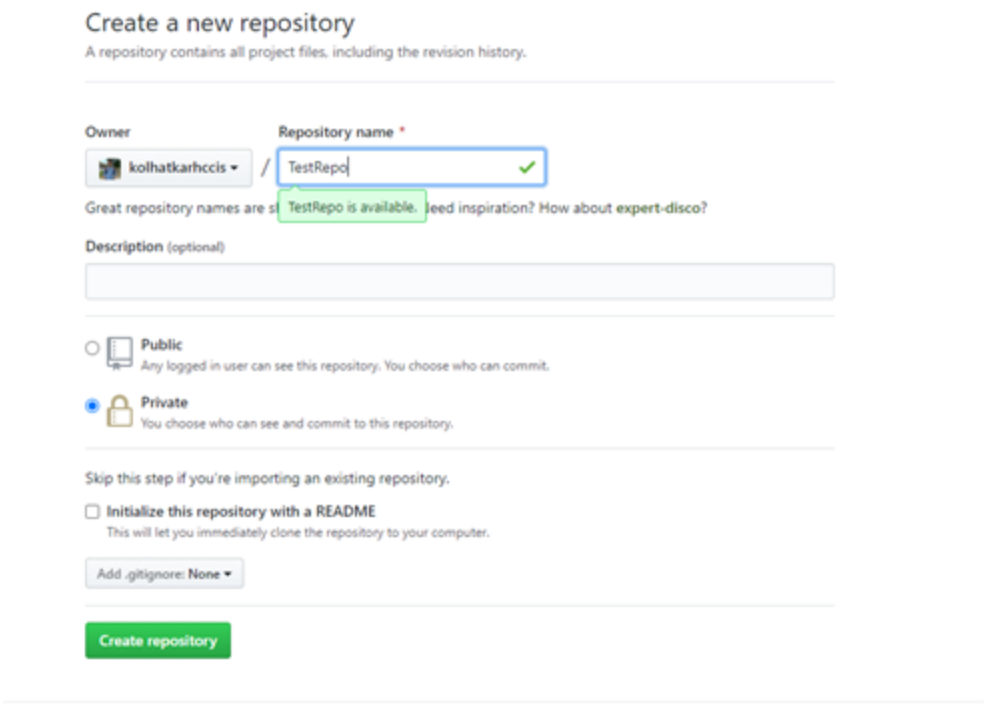
Submission instruction

Submissions will be done by pushing the code to your own private repository on Khoury Github website. To create a private repository on the Khoury website, follow the instructions given below,

- Login to Khoury Github site using Khoury credentials. (https://github.ccs.neu.edu/)
- Once you login, you can create a private repo by clicking 'New' -> Give repository name



- Make sure that you select the repository as a Private repository.



- Once the repository is created, you need to add TAs as Collaborators to your private repo. You can do that by going to ‘Settings’ tab and selecting Collaborators on the left.