Homework 5

Introduction to Networks and Their Applications
Due: May 4th, 2017

In the previous homework, you had to develop a simple server that can serve static HTML files. In this homework, you will have to extend the capabilities of your static web server. As part of this homework, we will extend these capabilities to provide basic capabilities similar to those that web frameworks such as django or node.js provide. Specifically you have to:

- handle content supplied by the user via forms whose action is set to be HTTP POST
- generate dynamically HTML content with based on the values supplied by the user

The dynamic server will work as follows:

1. A user may access an HTML form that you have previously saved in the **static** folder. As in the previous assignment the **static** folder includes all the content that is static. An example of a simple form is shown below:

```
<html>
<form action="/r1.html" method="post">
    First name: <input type="text" name="fname"><br>
    Last name: <input type="text" name="lname"><br>
    <input type="radio" name="gender" value="male" checked> Male<br>
    <input type="radio" name="gender" value="female"> Female<br>
    <input type="radio" name="gender" value="other"> Other<br>
    <input type="submit" value="Submit">
</form>
</html>
```

   A detailed tutorial for writing forms is at `https://www.w3schools.com/html/html_forms.asp`. The form has several aspects worth highlighting:

   - The form will use HTTP POST to send the information that the user inputs to your server. The form specifies to use HTTP POST by setting the value of the attribute **method** to be *post*.
   - The action attribute indicates the URL with that will be used as part of the request URI in the HTTP POST.

- The form has a number of fields for which you can set what is the name and the value. Depending on the input type other attributes may also be used.
- One of the possible ways to encode the frame is to use `application/x-www-form-urlencoded`. In this case, the names and values will be saved as a long string in the body of the HTTP post method. The pairs of values are separated by **&** and the names and values by =.

Assuming that you fill out the form such that the first name is John, last name Doe, and gender male the browser will generate the following request:

```
POST /r1.html HTTP/1.1
Host: localhost:8001
Connection: keep-alive
Content-Length: 32
Cache-Control: max-age=0
Origin: http://localhost:8001
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_4)
Content-Type: application/x-www-form-urlencoded
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Referer: http://localhost:8001/f1.html
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.8
Cookie: Webstorm-ea36d52b=7cf702c4-f920-40c7-acfd-a25b1d77b36c

fname=John&lname=Doe&gender=male
```

Note that in order to handle HTTP GET in the previous assignment we just had to read the header. In this case, we have to also read the body of the HTTP POST not just its header. The length of the HTTP POST is provided in the **Content-Length** of the HTTP POST. Also, as part of the validation process you must make sure that the **Content-Type** is set to **application/x-www-form-urlencoded**.

2. The web server must now process the incoming HTTP POST request. This requires that you validate the request and parse the name-value pairs in the body of the HTTP request into a dictionary. As in the previous homework, you must handle concurrent clients. Additionally, you must handle a mix of HTTP GET and POST requests.

3. Once you constructed the dictionary you need to respond to it. You will use a simple templating language to generate dynamic html content that you will use in the reply. The language allows a name that is included in curly brackets to be replaced by its value. This is best seen as an example. Consider the template shown below.

```
<html>
 First name <b>{{fname}}</b><br/>
 Last name <b>{{lname}}</b><br/>
 Gender <b>{{gender}}</b><br/>
</html>
```

If the template is invoked using the dictionary {fname : John, lname : Doe, gender : male } will be rendered as:

```
<html>
 First name <b>John</b><br/>
 Last name <b>Doe</b><br/>
 Gender <b>male</b><br/>
</html>
```

If a name is missing from the dictionary, you will render its value to be the string **??UNKNOWN??**

4. Now that you generated the HTML page, you can included as part of an HTTP reply.

As usual, you have to submit your homework through ICON. The following resources may be useful in solving the homework:

- HTTP RFC (`https://tools.ietf.org/html/rfc2616`).
- Developer tools of any browser (I use Chrome)
- Postman extension for Chrome `https://www.getpostman.com/`