Assignment 2

Computer Networks - an introduction

Author: John Herrlin
Email: jh222jx@student.lnu.se
Author: Rasmus Sjostrom
Email: rs222kp@student.lnu.se
Semester: VT2016
Area: Computer Science
Coursecode: 1DV701

Contents

1 Introduction

The idea of our project was to build a small web framework, since we already had to write the web server from scratch for the assignment. We figured this to be a great opportunity to develop something we could continue working on and learn more about since we consider it a very interesting field.

Our goals with this small framework is to keep things as simple and modular as possible, making it easy to expand or modify for further development.

We are very well aware of the fact that the database parts were not a part of the assignment, but to us it simply made sense to include this in the project. Therefore, we have built a small ORM with a SQLite database found in the db and domain modules. The reason for this was to be able to test and use our PUT and POST methods naturally, as if we were performing them on a real life web server.

We have saved the tcpclient in the code from the previous assignment. The reson for this is that we used it for fuzzy testing against the endpoints.

1.1 Workflow

The small enumeration figure below describes the workflow of the webserver, starting with a request and ending with a response being sent back.

1. Parse an incoming request and create a Request object
2. Send the request object to the URL handler, which will match the Request.uri with specified patterns
3. If the requested uri matches a pattern, the URL handler will call the corresponding View, forward the request as an argument
4. Depending on the called view, proper business logic will be executed. Thereon a response and body will be sent back to the client

By following this model, adding new URLs and a corresponding View for that URL is extremely easy.

2 Problem 1

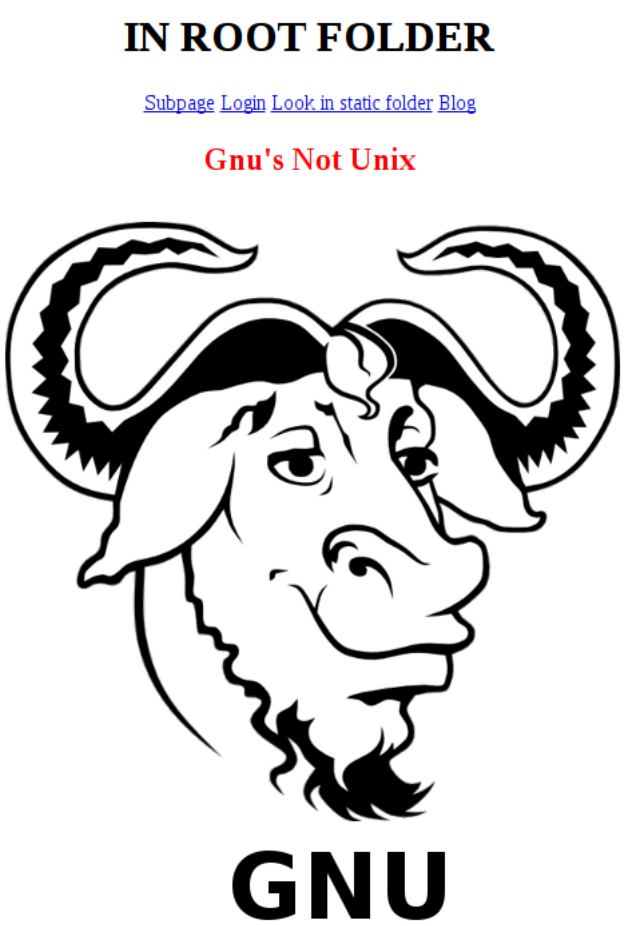

Figure 1: Index page with imgs and CSS.

```
nils@deb-cli:~/Git/Computer-Networks-1DV701/code/src/main/resources$ l
total 56
32 -rw-r--r-- 1 nils nils 32038 Feb 16 14:24 favicon.ico
 4 -rw-r--r-- 1 nils nils  1143 Feb 26 16:57 form.html
 4 -rw-r--r-- 1 nils nils   443 Feb 26 16:57 imageForm.html
 4 -rw-r--r-- 1 nils nils   608 Feb 26 16:57 index.html
 4 -rw-r--r-- 1 nils nils  1129 Feb 14 18:13 log4j.properties
 4 -rw-r--r-- 1 nils nils   156 Feb 16 14:42 mysite.html
 4 drwxr-xr-x 2 nils nils  4096 Feb 26 16:57 static
```

Figure 2: Resource folder, site root.

Figure 3: HTML document in static folder.

3 Problem 2

The table in 4 describes different types of response codes that the server sends back to the client. We table describes response codes that are both in the G task in Problem 1 and VG-task 1 in Problem 2. The reason to have everything in one table and not to split them up if for the simplicity and readability. Almost all of the response codes have figures and links to them are in the table.

3.1 403

We get this behavior when we try to post or put a form without being logged in. Login is done via the endpoint: /login.

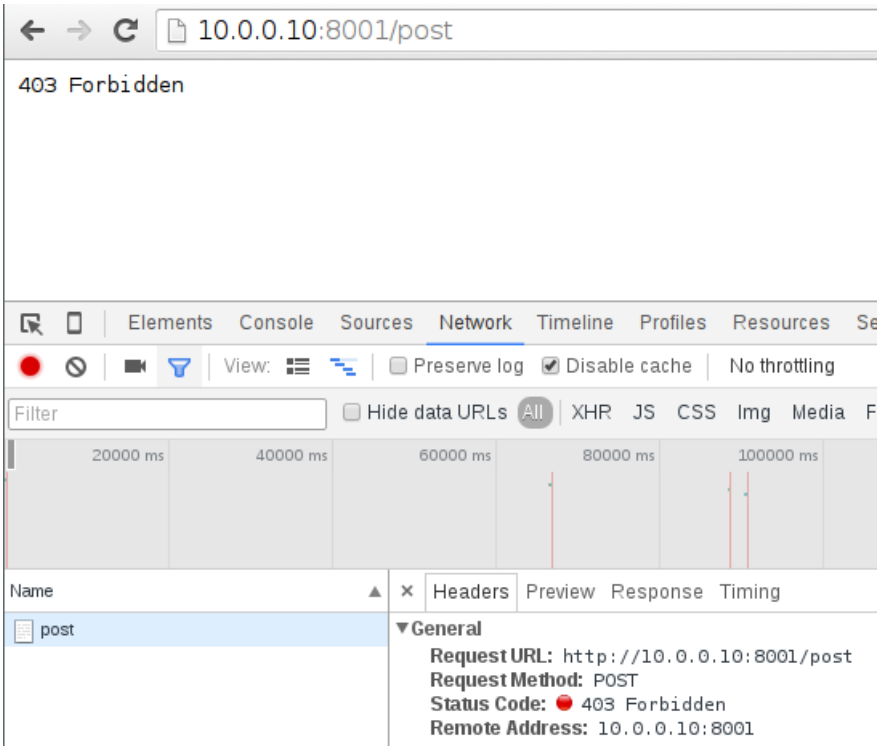

Figure 4: POST not allowed.

3.2 404

We get this behavior when we trying to access a endpoint that doesnt have any content.

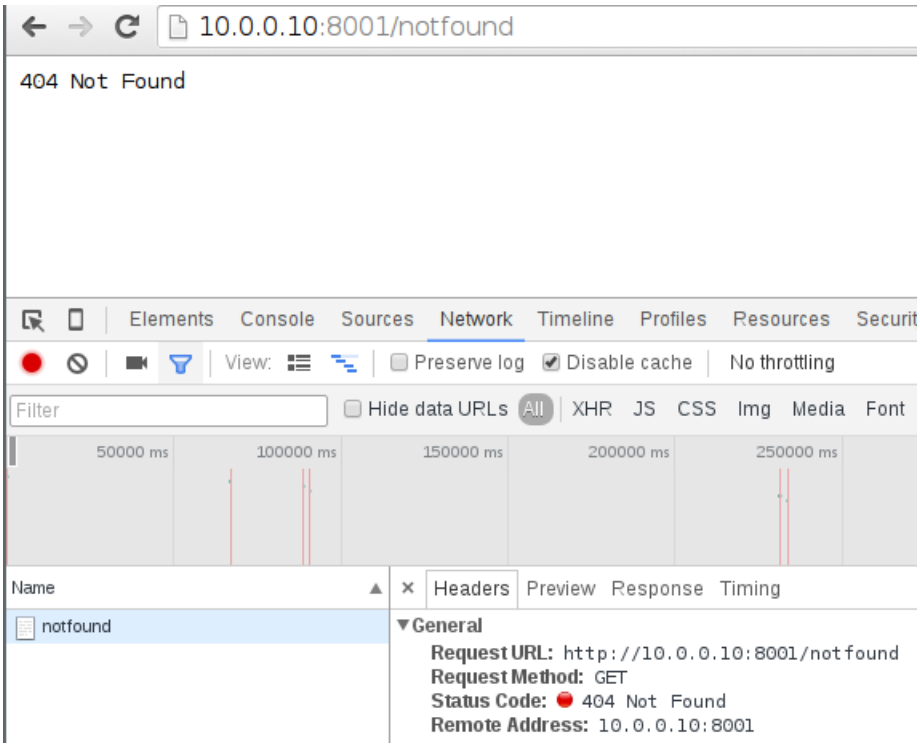

Figure 5: 404 Not Found.

3.3 500

This behavior it hard to get from a web broser as is sends correct requests. It appers if the server cant parse the request in a correct way it returns 500 Server Error. A screenshot of a telnet caption can be found here, Figure 5

3.4 VG-Task 1

All responsecodes thats implemented can be found in Figure 4

3.5 VG-Task 2

Both POST and PUT sends data to the server but are used in two different way. PUT is used to update a resource on a specific endpoint. In our server implementation we send data to /put/{blog-uuid}. The blog-uuid is the specific endoint where the resource should be updated. POST is also connected to a endpoint, but doesnt need to have a specific location. For example login, send data to the /login endpoint. The endpoint takes care of the data in the request, but we dont need to specify a specific id. In out implementation we send data to /post to create a new blog post. The important part is that PUT should have a unique id for the resource and not POST.

3.5.1 List all blog posts

This endpoint lists all of the blog posts that we have in the db.

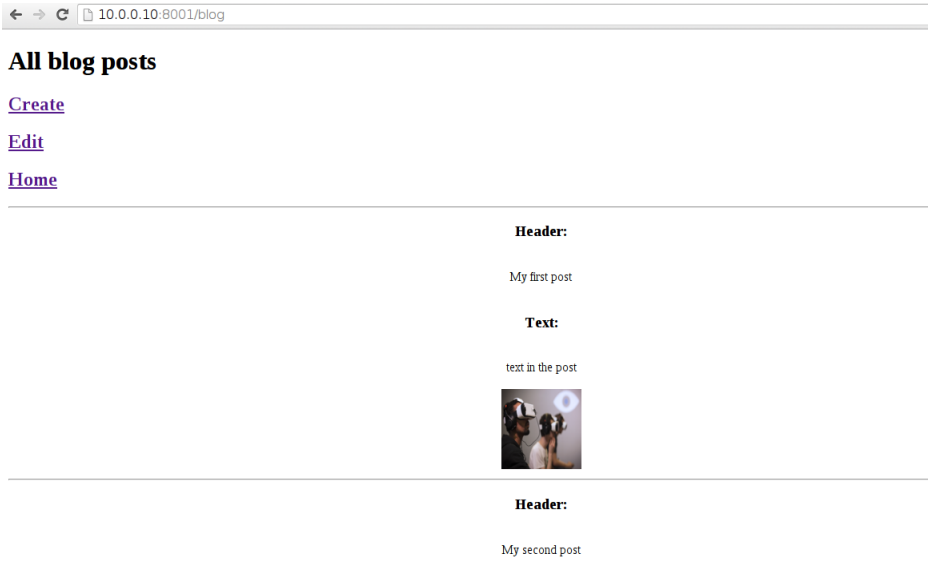

Figure 6: Method: GET, Endpoint: /blog, Comment: All blog posts in db.

3.5.2 Create blog posts

This endpoints provides a form for creating a new blog post. When submiting the form the data is send to the /post endpoint and the form data in the request body. The request method is POST. The server parses the data and stores it in the db. If the submit is correct the server sends back a document that redirect us to the /blog endpoint.

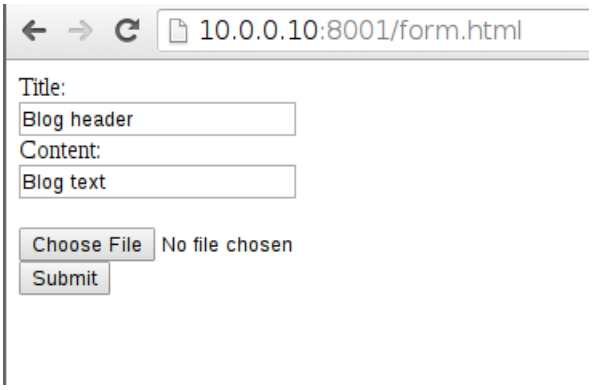

Figure 7: Method: GET, Endpoint: /form.html, Comment: Create blog post.

3.5.3 Update blog posts

From this endpoint we can edit all of the blog posts. The update is made with a PUT request to the endpoint /put/blog-uuid. The server checks if we have a field in the form called _method with the value put. If the server finds that is sets the request as a PUT. The server then tries to parse the data and update the blog entry by its uuid. Response is a html document that redirects the user to the /blog endpoint.

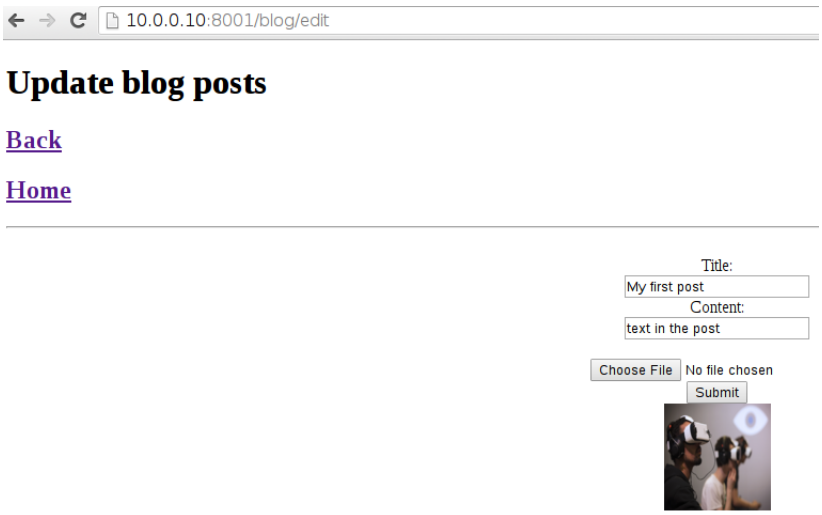

Figure 8: Method: GET, Endpoint: /blog/edit, Comment: Update blog posts.

4 Problem 3

Requesting a image from a URI within telnet gives back the compiled code of the image.

Code	Usage
200	Many places where server handles response in a correct way.
201	When success with POST on /post Figure 5
202	When /login success Figure 5
205	When /logout success. Figure 5
400	When not PUT on /put . Figure 5
403	When trying /post or /put before logged in (/login) before Figure 5
404	When static file not found OR if url doesn match any Figure 5
405	When not POST on /post Figure 5
500	If we cant parse the request Figure 5

Figure 9: Response codes and their usage.

5 Screenshots

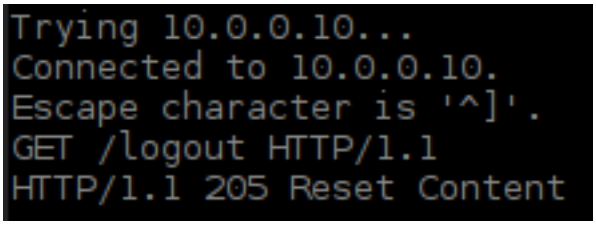

Figure 10: Method: GET, Endpoint: /logout, Response: 205 Reset Content

Figure 11: Method: GET, Endpoint: /login, Response: 202 Accepted

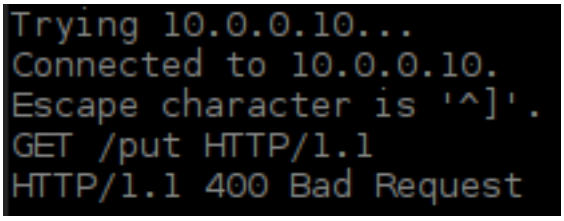

Figure 12: Method: GET, Endpoint: /put, Response: 400 Bad Request

Figure 13: Method: POST, Endpoint: /post, Response: 201 Created

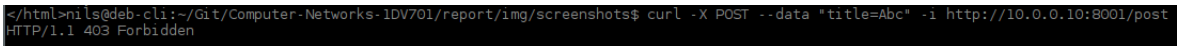

Figure 14: Method: POST, Endpoint: /post, Response: 403 Forbidden

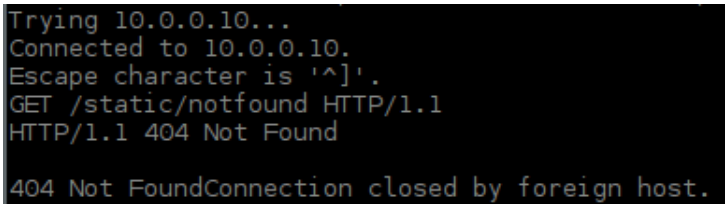

Figure 15: Method: GET, Endpoint: /static/notfound, Response: 404 Not Found

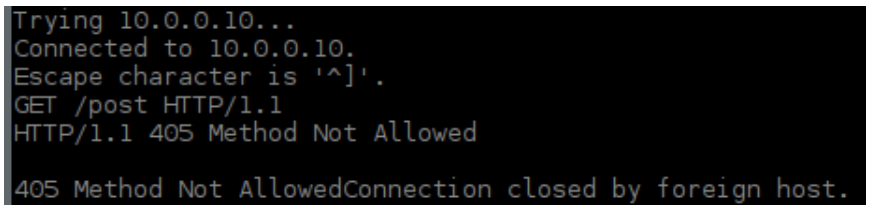

Figure 16: Method: GET, Endpoint: /post, Response: 405 Method Not Allowed

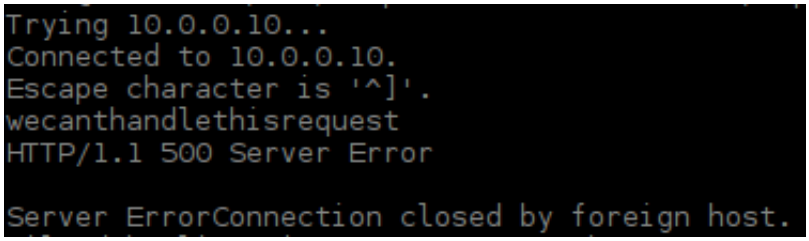

Figure 17: Method: ?, Endpoint: ?, Response: 500 Server Error

6 How To Run/Build

Import the project as a maven project in an IDE. Download maven dependencies. Run se.jherrlin.run.Main with arguments: -m tcpserver