Assignment 2
Upper Layers of the OSI-Model

Version: January 10, 2025

Prerequisites

1. Module 2 readings on Canvas
2. Module 2 lecture videos on Canvas
3. Run/Understand the following examples from the GitHub repository:
   - Network/SimpleGrabHttpURL
   - Network/SimpleGrabURL
   - Network/HTTP-JSON
4. Setup of a second device (second computer, AWS EC2) – see Canvas for details
5. Videos/Tutorials about Wireshark
6. Understand the lower layers of the OSI model (Module 1)

The learning outcomes of this assignment are:

- Understanding the purpose and function of the upper layer network protocols
- Understanding how these protocols differ (HTTP vs. HTTPs)
  - In particular, the differences in each protocols message traffic
- And as always, familiarization of terminal / command line usage
  - One of the most important skills to develop!

1 Understanding HTTP (20 points)

For this section, you will only need your web browser and Wireshark. (Wireshark only for taking a look at things yourself.) To understand GET HTTP requests a little better, we want to do a couple of GET requests through an API. (For this assignment, it is for the GitHub API.) The API works by nesting topics - where you will need to do a base call to get some data, use that data to perform the next call, and so forth.
To begin, go to the GitHub API documentation webpage. This will give you some information about the API. It might be overwhelming to read at first, but that is okay. What's interesting, however, is that you can make requests directly on your browser and not just from things like Java, Javascript, etc.
For instance, we use the List repositories for a user API call to get all public repos from a specific user. In your browser, use the following URL:

```
https://api.github.com/users/amehlhase316/repos
```

This should show you a JSON file for all of my public repos. (You can also use a different username if you like, maybe you should and check what you come up with)
Next, we want to look at one specific repo. To do this, we will use the *Getarepository* API call. For that, go to:

```
https://api.github.com/repos/amehlhase316/memoranda
```

This will give us JSON on the Memoranda project where I am the owner. You can use any other username and project you like (make sure it is public though).
Now, find and run a new call that gets all the commits on the default branch for a repository of your choosing. Any public repo that has some branches and commits on them is fine.

Deliverable (5 points): Paste the URL you used into your document, take a screenshot, and add it to your document. (The screenshot should show the call you made and its resulting JSON. If the JSON is too big, partially showing it is okay as well.)

From here, perform another call. In this call, add GET parameters/query arguments (to the call from above) that specifies a specific branch (not the default), and sets the per-page limit to 40 (so 50 commits are shown instead of just 30). Usually, APIs limit their response size and return one page. If you need more data you would then call the next page or increase the page limit (as we do here for simplicity - usually there is a limit to this "page limit").
If you are stuck on this, please go to the documentation and try to understand how this API call works.

Look for one more API call you can make on a public repo, any call that uses some query arguments is fine.

Deliverable (5 points): Paste the URLs you used into your document, take a screenshot of your result, and add it to the document (do this for all API calls you made).

Deliverable: Answer the following in your document (10 points):

1. Explain the specific API calls you used, include the information you needed to provide and include the link to the API documentation for that call.

2. Explain the difference between stateless and stateful communication.

You should take a look at Wireshark and check the communication that was going on with your calls. You do not have to document anything for me, but I advise you to take a detailed look and to do your best in understanding the traffic generated.

2 Set up your second system and run servers on it (70 points)

For this part, you will need to set up your second machine (AWS, which you should have already done). See the setup page on Canvas. We will call your local machine "first machine" and your second one (AWS) "second machine" in this document.

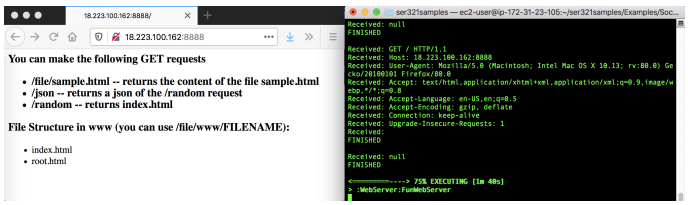
2.1 Getting sample code onto your systems (should be done already)

This is something that should be done already but is explained here in case you didn't do this yet.
Now that you have your second machine set up, you should make sure the example GitHub repo is available on both machines. I would advise you to fork the given repository and clone it on all the machines you want to work on - so you can still make changes to commit, push, and pull. This fork will be public - thus it is not for your assignment changes, but just for "playing" with the examples.
You can also, of course, download the ZIP and add it to both systems (but you won't be able to pull updates as easily). If you have not worked with GitHub before, I advise you re-read the GitHub review page on Canvas.

2.2 Running a Simple Java WebServer (10 points)

Now, you will need to work with the *Socket/WebServer/* directory. Copy this folder into YOUR Assignment2 directory (in your private assignment repo) before you make any major changes.
We want to run the *FunWebServer* task from Gradle in this example - with the server on your second machine (AWS). You should also start Wireshark on your first machine again if it is not currently running.
With your server running on your second machine, go to a web browser on your first machine and search: *<publicIPOfYourSecondMachine>:9000* (You can also change the port if you'd like - but make sure the port you use is opened up for TCP traffic on AWS). If everything works as intended, this should open a web page.

Deliverable (10 points): Take a screenshot of your web browser showing both the *<publicIPOfYourSecondMachine>:9000* and the web page itself. You should also take a screenshot of your second machine (either as one screenshot or two separate ones) and add it to your document under Task 2. The screenshot should look something like this:



2.3 Analyze what happens (10 points)

Wireshark should still be running in the background. Go to Wireshark, and create a filter that shows network traffic to and from your WebServer. Take a screenshot of your Wireshark capture and add it to your document.

Deliverable: In your document, answer the following (1-2 points each): 1 point each unless otherwise specified

1. What filter did you use? Explain why you chose that filter. something with port 9000 probably

2. What happens when you are on the /random page and click the "Random" button? Compare this to refreshing your browser. (You can also use the command line output that the WebServer generates to answer this.)one goes right to /json the other one first to /random and then /json 2 points

3. What types of response codes are you able to receive through different requests to your server? 200, 400, 404 should be the ones

4. Explain the response codes you receive and why you get them.explained the response codes well 2 points

5. When you do a *<publicIPOfYourSecondMachine>:9000*, take a look at what Wireshark generates as a server response. Are you able to find the data that the server sends back to you? (This should be the "Data" section of your response.)Should be able to see the data as plain text what is in the HTML file

6. Based on the previous question, explain why HTTPS is now more common than HTTP. HTTPs is more secure

7. In our case - what port does the server listen to for HTTP requests, and is that the most common port for HTTP? 9000 but most common for HTTP is 80

8. Which local port is used when sending different requests to the WebServer? would be some large port number eg. 13119, 45000 or similar

2.4 Setting up a "real" WebServer (10 points)

Stop your server, and do the following while on your second machine (it might need to be "apt-get" depending on your system setup).
Let's set up nginx so you have a real Web server running. Run the following:

```
sudo dnf install nginx --> when prompted, type y
sudo service nginx start --> to start the web server
```

Next, change the config file for the server. Note: All changes will be in the server ... section.

```
sudo vim /etc/nginx/nginx.conf
```

You need to enter the server_name and a location block into your config file. It should look something like this (with the correct IP from your host and port used in the Java file). This is only a sample snippet, but it shows the part that you need to change.

```
....
server {
    server_name  18.223.100.162;                    #CHANGE IP HERE
    root         /usr/share/nginx/html;
    location / {                                    #Add location block with correct port
            proxy_pass http://localhost:9000/;
    }
    # Load configuration files for the default server block.
    include /etc/nginx/default.d/*.conf;

    # redirect server error pages to the static page /40x.html
    #
    error_page 404 /404.html;
```

```
        location = /40x.html {
    }
....
```

Now, reload nginx with the configuration updates by using:

```
sudo service nginx restart
```

Your web server should be running, and your port 80 traffic will be re-directed to the port you specified in *location block* above.
Start your Java FunWebServer again, and ensure that Wireshark is also running. Now, go to your browser again. The old URL should still work but there should be another one that is working now, figure out how the URL changes.

Deliverable: In your document, answer the following:

1. What is the URL that you can now use to reach the main page?
2. Check the traffic to your WebServer. What port is the traffic going to now? Is it the same as before, or is it (and should it) be different? Should be 80 now
3. Is it still using HTTP, or is it now using HTTPS? Why?still HTTP
4. Could you change your security settings on AWS now?yes port 9000 could be removed now
5. Take a screenshot of your web browser, your second machine, and the port number on Wireshark. This should be similar to the screenshot you took before (but also with Wireshark), and add it to your document for this task. Note: If we are unable to see that you reached the WebServer with the "different URL", we will not know if you actually set up the server correctly. Therefore, please make sure that it shows up if you want points for this task.

If you'd like, you can stop your nginx again:

```
sudo service nginx stop
```

This should bring you back to where you were previously (without your nginx WebServer).

2.5 Setting up HTTPS (5 points)

The commands often seem to change here so this is extra credit if you can make it work. Important: Only attempt this if you are ready for some extra steps and some research!!! If you'd like, you can create a copy of the current nginx config file in case you want to go back (and make things easier later on).
Start your nginx again:

```
sudo service nginx start
```

Now, to make our traffic secure, we want our communication to go through HTTPS. This will require some more work and additional steps, but it is worth it to understand what's actually going on. (Note: Work on your second machine for this.)
To do this, get a domain. Here, we use DuckDNS, but you can use something else if you'd like. In the end, we just want to get a domain name - which will resolve to your IP address. (Remember the DNS portion from the lectures.)
In a web broswer, visit DuckDNS on your main machine (or other DNS services if you like). Then:

1. Login with whatever method you prefer
2. Enter a sub-domain –> click "Add Domain"
3. Select "Install" from the top navigation bar
4. DuckDNS/Install:
   - Choose "linux cron"
   - Select the domain you created
   - Go to your second machine and follow the linux cron instructions. Note: When saving the crontab, use (esc –> :wq! –> Enter) instead of the instruction's (CTRL+o –> CTRL+x).

With your domain set up, we can now continue with the certificate setup. On your second machine, run the following command to change your nginx config info:

```
sudo vim /etc/nginx/nginx.conf
```

Change the server name from the IP you had to your newly created DuckDNS domain name. (e.g. ser321.duckdns.org) Save the changes and reload nginx. When you go to DuckDNS.org, you should see your domain name listed - and it should now show your second machine's IP as the IP address. (Basically, your domain "perfectname.duckdns.org" gets resolved to your machine's IP address.)
Now, you should already be able to use *yourdomain.duckdns.org* to reach your server (This should of course still use HTTP and work after starting your Java program again). Stop the Java program before continuing. We need to set up a certificate - and to do this, we can use Certbot.
Run the following:

```
sudo dnf install python3-certbot-nginx
sudo certbot
```

Domain names are automatically detected from the nginx configuration.
When prompted –> Choose the name
When prompted –> Enter an email address
When prompted –> Type a to agree
When prompted –> Type n to decline emails

Your certificate should now be set up. To verify, you should be able to go to *perfectname.duckdns.org* and get to your website through HTTPS. Check both Wireshark and your web browser to see that your traffic is going through HTTPS.
Take a screenshot of your web browser, your second machine, and the port number on Wireshark. This should be similar to the screenshot you took before (but also with Wireshark), and add it to your document. (We need to see that your web browser now shows HTTPS for you to receive extra credit.)

Deliverable: In your document, answer the following:

1. What port is your traffic going through now?
2. Can you still find the plain text responses that were found with HTTP?

If you want to stop the nginx, just call:

```
sudo service nginx stop
```

Your domain should not work anymore, and you should only be able to use the IP address to access your Java server again.
OPTIONAL: The following can be used in the event you want to remove the certificate. (You can do this - but when nginx is stopped, you will only have the "normal traffic" again anyways.)
To remove the certificate (so only the nginx WebServer works) - first remove the certificate, and then update the nginx config. (This is easy if you copied the config file from before - in which you can just replace it now.) If you did not copy it, then open the file and remove everything related to the certificate and port 443. Also, if you do not want to use the domain name anymore, do:

```
sudo certbot revoke --cert-name YOUR_DOMAIN
sudo vim /etc/nginx/nginx.conf
```

DONE.

2.6 Some programming on your WebServer (35 points)

This should be done whether you have nginx running or not! This can also be done locally on your working machine, I would even advise doing this locally. I'd advise spending some time on the WebServer code and playing around with it, such that you understand what happens and how things work.
In the WebServer code, you will see a couple ToDo's. These are things you need to implement and some further details will be explained here. Changes need to be implemented on your private repo and not on the example repo.

2.6.1 Multiply (5 points)

Check out the if-statement for the *multiply* case. This is a normal GET request with two parameters. Your task is to add some error handling in the case that the user does not provide the correct inputs, just one input, etc. You can handle it as you see fit (e.g. set default values and print a message), but your server should not crash and should respond in a good way. (This is true for the whole assignment. DO NOT let your server crash. Also do not just put one big try/catch around everything and return a generalized error)
It is important that you create an appropriate header response with a good error code. You can find explanations about error codes here.
In your document: Explain what you decided to do, which error code you used, and why.
Some codes they might choose 406 if they want to say that the server cannot handle it due to wrong values (no default values used)
400 Bad Request if they think that is a bad request
These are hopefully the ones they would choose.

2.6.2 GitHub (9 points)

You see another if-statement waiting for *github*?. In there, you will see a fetch that pulls information from GitHub. (This part is about parsing a JSON response and understanding it.)

Implement your WebServer, such that when calling:
$host : PORT/github?query = users/amehlhase316/repos$, you should get a response that contains all of my public repos (which is not a lot). Parse the JSON in your code and respond with some data. The data you should return (of each repo) is its full_name, ID, and login of the owner. (Go by what this assignment wants displayed, as the given code in the example repo says something else.)
The webpage should display this information in some way. (It does not have to be pretty.) Make sure you include good error handling and that we cannot crash your server with the wrong request (which we will try). You should include good error codes - not just one big try-and-catch!

2.6.3 Make your own requests (14 points - 7 for each request)

In your WebServer, add two more request types that it can handle. (This should be similar to the Multiply/GitHub requests.) You can do any request you'd like, but they should fulfill the following requirements:

1. The request should provide more functionality than just a new version of Multiply (e.g. Do not do an add/subtract/divide request.)
2. The request should get at least two arguments and use them for something.
3. The request should include proper error handling. We should not be able to crash your server with incorrect inputs/calls, and any error message should be appropriately provided (with good error codes).
4. The request should be explained on the main web page, and an example should be given on how we can use your request. (Please provide an exact URL with example data that we can copy and paste.)

2.6.4 WebServer for Everyone (5 points)

Now, figure out how you can keep your WebServer running - even when closing the terminal window to AWS. Then, post the link to your server (with your public IP address and port) to #servers on Slack.

2.6.5 Test other WebServers (2 points - one point for each server you test)

Test 2 other servers in the #servers channel, and give a valuable comment for each one you test. (You can do this up to 2 days after the due date). No valuable comment -> No points.
To receive credit: add the links to your Slack comments and to your URL where you posted your server to Canvas as comment on your submission please. Slacks seach function is a little annoying and sometimes does not find everything. We are working on a better way for this for future assignments.

3 Submission

Submit your link to your GitHub repo Assignment2 folder on Canvas and also zip up that whole directory and submit it on Canvas.

In your Assignment 2 folder, include your PDF *UpperLayers_asurite.pdf* with all of your answers, explanations, and screenshots. No Wireshark captures are needed.
Please, add the WebServer directory into the Assignment2 folder as well. Name the directory "WebServer" - and it should include all necessary files for us to run your code and view your implementation. Please leave the port at 9000. We will run the code by going to your WebServer directory and calling *gradle FunWebServer*. If this does not work, you will not receive points for the server!!!
Your folder should look like the following:

```
- Assignment2
  - UpperLayers\_asurite.pdf
  - /Webserver
    - /src
    - /www
    - build.gradle
    - README.md
```