Lab 04: Web protocols

The exercises marked with * are obligatory to complete before the upcoming weeks lecture.

The purpose of this lab is to get you started on utilizing API's. The lab is structured in a way, where the student is assisted through the tasks.

Remember to write useful commands in your cheat sheets.

Table of content

Exercise 01* **1**
- Facebook Developer Graph API Explorer 1
- cURL 2

Exercise 02* **2**

Exercise 03* **3**

Exercise 04 **4**

Exercise 05 **5**

Exercise 01*

In this exercise the student is going to examine the Facebook API with the purpose of getting the understanding about how a (REST) API works and the capabilities that an API like this offers.

Facebook Developer Graph API Explorer

The Facebook Developer Graph API Explorer (https://developers.facebook.com/tools/explorer/), is a tool made for testing, creating and debugging API calls.
To use the tool, it is required to create an app, which is basically to tell facebook that you would now like to use facebook's API. This makes you responsible for every call that is made through the created app.
With an app created, the API Explorer can be used. As default a User token is enabled, making it possible for the application to identify the user only by name and ID. To get access to information other than that, permission for the wished information must be included.

The access token is used to identify and authorize the user of the application, making it possible for us to use the API in an appropriate context.

The facebook API is accessible through https://graph.facebook.com/.
Remember that REST APIs use endpoints, so the above-mentioned access point needs to be specified.

cURL

The Linux Terminal tool `curl` is used to transfer data using a number of protocols. As we use the HTTPS protocol for the facebook API, this tool is useful.
We are using cURL to communicate with a HTTPS server with a GET method, requesting some data about the user of the app.

The tool, like many other Linux tools are used in the following fashion: `$ curl <OPTIONS> <URL-ENDPOINT>`
The only option we are interested in for now is the -H (or --header) which is used to include a header.

When using cURL to access the facebook API, you will need to include your token in the header of the request. To authorize yourself with the token use the following header: `“Authorization: Bearer <TOKEN>”`.

Your tasks:

1. Create a facebook app through *Facebook for Developers* (developers.facebook.com).
    a. Select “Log In” from the menu and login using your facebook credentials
    b. Select “My Apps” from the menu and select “Create App”.
    c. Select “For Everything Else”.
    d. Give the app a name and select “Create App ID”.
2. Open the Graph API Explorer through “Tools” in the menu.
3. Select “Get Token” and “Get User Access Token”.
4. Include permission for *user_birthday* through the *API Explorer*.
5. Generate and grab your token for authorization when using cURL.
6. Construct a URL requesting your birthday from the facebook API through an appropriate endpoint
7. Use cURL to create a GET request with a custom header, making Facebook authorize your request and responding with your birth date.

The exercise will return the following message:

```
{
   “birthday”: “01/01/2000”,
   “id”: “0000000000000000”
}
```

Exercise 1.5*

In this exercise you will use wireshark to inspect network traffic.
Your tasks

1. Open wireshark from the terminal with `sudo wireshark`
2. Start capturing on the interface connected to the internet (likely your wifi)
3. Visit a website, for example sdu.dk
4. Stop capturing in wireshark
5. Use filtering in wireshark to see only SYN, SYN/ACK and ACK messages. Hint: You can filter based on the tcp flags (for example: `tcp.flags==0x02`)
6. Can you explain what is happening?
7. Can you find any arp messages? (filter: `arp`) Why/why not?
8. Can you find any dns messages? (filter: `dns`) Why/why not?

Exercise 02*

In this exercise the student is going to create a server using python and a package named Flask. Using Flask, it is possible to create a custom API and set up specific routes. An example of a simple Flask app can be seen below.

```
from flask import Flask

# This sets up the application using the Flask object from the package flask.
app = Flask(__name__)

# Using decorators the route is associated with the following method.
@app.route('/')
def hello():
    return 'Greetings sire, the Docker universe awaits you!'

# This statement evaluates to true if this is the main python file. It starts up
the Flask app on localhost with the default port 5000.
if __name__ == '__main__':
    app.run(host='0.0.0.0')
```

In this task python version 3.x will be used. Installing packages can easily be done using the software pip, and when working with python 3.x on linux it is called pip3. To check whether pip is installed use `pip3 --version`. If it is not installed run `apt-get install python3-pip`. To install the package Flask simply run `pip3 install Flask`.

Your tasks:

1. Create a server in python using flask.
   Hint: Inspiration can be found in the code snippet above.
   More information can be found here:
   https://flask.palletsprojects.com/en/1.1.x/quickstart/
2. Enable the route `/welcome` and give it the response:
   "Welcome to this awesome page!"
3. Run the script and access it at http://localhost:5000/welcome.

The exercise will show the following message in the browser:

```
Welcome to this awesome page!
```

Exercise 03*

In this exercise the student is going to use the server previously created and run it in a container.

Your tasks:

1. Modify the server file from the previous exercise so that the route /welcome takes a name as a parameter, and the app responds with:
   "Welcome to this awesome page <*NAME*>!"
   where <*NAME*> is the argument.
2. Create a Dockerfile that does the following:
   a. Uses the python image as base
   b. Installs the package Flask
      Hint: This can be done in similar fashion as installing it on linux
   c. Copies the server file
   d. Runs the python server

   Hint: More information on using the python image can be found on Docker Hub: https://hub.docker.com/_/python
3. Build an image from the Dockerfile. Remember to give the build a tag.
4. Run the built image and forward the container's port to port 7000.
   Hint:The server is exposing port 5000 in the container.
5. Access the server running in the container at http://localhost:7000/welcome/James

The exercise will show the following message in the browser:

```
Welcome to this awesome page James!
```

Exercise 04

In this exercise the student is going to contact a weather API and get the temperature of a given city.

A public API for weather is MetaWeather which has several routes. Other public APIs can be found here: https://github.com/public-apis/public-apis/blob/master/README.md

When using the API, the response is in the format of JavaScript Object Notation (JSON). Two of them are necessary for getting the temperature of a given city:

Route 1

- `https://www.metaweather.com/api/location/search/?query=<CITY>`
  where <CITY> e.g. could be London, Copenhagen or Cairo

This route returns information about the city. The information needed is the **woeid** of the city which is unique and used to identify the city when requesting for weather data.

Route 2

- `https://www.metaweather.com/api/location/<WOEID>/`
  where <WOEID> is the id of the city

This route returns information about a city based on the woeid given. It returns several sets of data each one beginning with the parameter “id”. The information about temperature is stored in the parameter called “the_temp”

Your tasks:

1. Use the first route to get the information about a city.
2. Identify the <WOEID>.
3. Use the second route to get the weather data of a city.
4. Identify the “the_temp” of the first data set

Exercise 05

In this exercise the student is going to use the server previously created and the weather API to create a custom api which returns the temperature of a requested city.

Python and JSON

Python can request for data and interpret it as JSON using the following method:

```
# imports the  necessary libraries
import requests                                                # Install using pip

def get_data_from_url_request(url_endpoint):
    response = requests.get(url_endpoint)                      # get url response
    data = response.json()                                     # read it as json
    return data
```

Your tasks:

1. Modify the server file from the previous exercise and create a new route /weather which takes a city as an argument, and make it respond with:
   “The temperature in *<CITY>* is: <TEMPERATURE>”
   where *<CITY>* is the argument and <TEMPERATURE> is the value returned from the weather api.
2. Use the argument `<CITY>` to request for the woeid.
   Hint: The woeid can be extracted from the data using `data[0]['woeid']`

3. Use the woeid retrieved to request for the temperature.
   Hint: The temperature can be extracted from the data using
   `data['consolidated_weather'][0]['the_temp']`
4. Use the Dockerfile from Exercise 3 to build an image. Remember to give the build a tag.
5. Run the builded image and forward the container's port to port 7000.
   Hint:The server is exposing port 5000 in the container.
6. Get the current temperature in Copenhagen by accessing the server running in the container at http://localhost:7000/weather/Copenhagen

The exercise will show the following message in the browser:

```
The temperature in Copenhagen is: <TEMPERATURE>
```