PR Lab 1

Deadline: 2 weeks; until 18.10.2024;

This lab focuses on web scrapping, the fundamentals of HTTP/S protocol, understanding the TCP transport layer beneath it, serialization/deserialization, and a little bit of Docker.

Web scraping, to use a minimal definition, is the process of processing a web document and extracting information from it. You can do web scraping without doing web crawling.

Web crawling, to use a minimal definition, is the process of iteratively finding and fetching web links starting from a list of seed urls. Strictly speaking, to do web crawling, you have to do some degree of web scraping (to extract the URLs.)

For this lab, we will focus on web scraping. The idea is to select an e-commerce website such as 999.md, darwin.md, smart.md, asos.com, etc. The site should have a list of products to scrape. The products should contain a **name** (string) and **price** (int). Additionally, you will need to scrape each product's link to extract additional details (for example, the color or any field). The rest is described in the conditions.

Please see the conditions on the next page

Grade	Conditions
1	Exist
2	Select your website and make an HTTP GET request.[1] It should return a valid HTTP response with HTML content/body :)
3	Use an HTML parser to extract the name and price of the products.
4	Extract the product link. Scrape the link further and extract one additional piece of data from it
5	Before storing the product information in your data models/structures, add two validations to the data.[2]
6	Process the list of products using Map/Filter/Reduce functions.[3] Map the price to EUR. If it's already in EUR, convert it to MDL. Filter the products within a range of prices. Use reduce to sum up the prices of the filtered products. Attach that sum to the new data model/structure along with the filtered products. Attach a UTC timestamp as well.
7	Instead of using the library from grade 2, use a TCP socket for connection, send an HTTP-formatted request, select the HTTP response body, and pass it further to the scraping logic.
8	Implement serialization logic for the data models in both JSON and XML formats without using external libraries. You must manually construct the strings/byte array that represent these formats. Just print/log them.
9	Create your custom serialization technique that can serialize and deserialize data.[4]
10	Install Docker on your system. Follow the task described in the reference[5]

[1] https://hackernoon.com/http-made-easy-understanding-the-web-client-server-communication-yz783vg3

[2] For example: remove whitespaces from string fields and ensure that the price represents only integer data type or any other validation.

[3] https://www.geeksforgeeks.org/map-reduce-and-filter-operations-in-python/

[4] The goal is to convert language-specific objects (like lists, dictionaries, strings, and integers) into interpretable bytes/strings, and then implement a deserialization process to convert that data back into the original objects. Be creative and explore different methods.

[5] For now, focus only on the HTTP server; you can explore Docker in more depth later. Follow these steps to pull the HTTP server using Docker and run it locally. First, we need to make sure Docker is installed on your computer. Open your terminal and run the following command:

```
docker --version
```

Next, you'll need to create a file named `docker-compose.yml`. This file tells Docker how to set up services/containers. Create a new file and add the following content:

```
version: '3.8'
services:
  lab1-http-server:  # Name of your container
    image: nicug/server-pr-lab1:latest  # The Docker image to use
    ports:
      - "8000:8000"  # Connect the port 8000 on your computer to port 8000 in the container
    environment:  # Set environment variables
      - PORT=8000
```

Make sure your terminal is in the same folder as your `docker-compose.yml` file. Then run the following command to start the server:

```
docker compose up -d
```

`-d` means "detached mode," which runs the server in the background. If you want to see the server logs in your terminal, you can run the command withou `-d`:

```
docker compose up
```

To stop the server at any time, press `Ctrl + C` (if it is not in detached mode). When you want to stop the server in detached mode, run this command:

```
docker compose down
```

This command will stop and remove the container.

The server should now be running on port 8000, as specified in your `docker-compose.yml` file. You can use an application like **Postman** to send requests to your server. Here's what you need to do:

- Make sure the server is running and accessible at `http://localhost:8000`.
- Guess the riddle of the server, try out a POST request to `/upload` route. Study the returned HTTP status codes and messages.
- Once guessed, send two **POST requests** to the server at the endpoint `/upload`:
  - The first request should send your serialized data in **XML format**. Make sure to set the header `Content-Type` to `application/xml`.
  - The second request should send your serialized data in **JSON format**. Set the header `Content-Type` to `application/json`.

Both requests should receive a **200 status code** in response, indicating that everything is working correctly.