Full Stack Development Lab Instructions
ENSF381: Lab10

Python Back-end

Created by: Mahdi Jaberzadeh Ansari (He/Him)
Schulich School of Engineering University of Calgary
Calgary, Canada
mahdi.ansari1@ucalgary.ca

Week 10, March 25/27, 2024

Contents

1 Introduction

1.1 Objectives

Lab 10 is designed to extend your practical experience using Python and Flask. We are going to create a Python server that provides CRUD (Create, Read, Update, Delete) operations for product information stored in a JSON file, along with serving images from a ”product-images” folder, which involves several steps. We will use Flask, a lightweight WSGI web application framework in Python, to accomplish this. Flask is well-suited for creating RESTful web services and is easy to set up for a simple server like this.

1.2 Prerequisites

1. Basic understanding of computer operations and algorithms.

We do not use a Git repository for this assignment, so you must deliver all your codes and the answer sheet file in a ZIP file via D2L.

1.3 Forming groups

- In this lab session, you **MUST** work with a partner (groups of three or more are not allowed).
- The main goal of working in a group is to learn:
  - how to do teamwork
  - how to not be a single player
  - how to not be bossing
  - how to play for team
  - how to tolerate others
  - how to behave with colleagues
  - how to form a winner team
  - and ...
- Working with a partner usually gives you the opportunity to discuss some of the details of the topic and learn from each other. Also, it will give you the opportunity to practice one of the popular methods of program development called pair programming. In this method, which is normally associated with the “Agile Software Development” technique, two programmers normally work together on the same workstation (you may consider a Zoom session for this purpose). While one partner, the driver, writes the code, the other partner, acting as an observer, looks over his or her shoulder, making sure the syntax and solution logic are correct. Partners should switch roles frequently in such a way that both have equivalent opportunities to practice both roles. **Please note that you MUST switch roles for each exercise.**
- When you have to work with a partner:
  - Choose a partner that can either increase your knowledge or transfer your knowledge. (i.e., do not find a person with the same programming skill level!)
  - Please submit only one lab report with both names. Submitting two lab reports with the same content will be considered copying and plagiarism.

1.4 Before submission

- For most of the labs, you will receive a DOCX file that you need to fill out the gaps. Make sure you have this file to fill out.
- All your work should be submitted as a single file in PDF format. For instructions about how to provide your lab reports, study the posted document on the D2L called How to Hand in Your Lab assignment.
- Please note that if it is group work, only one team member must submit the solution in D2L. For ease of transferring your marks, please mention the group member's name and UCID in the description window of the submission form.
- If you have been asked to write code (HTML, CSS, JS, Python, etc.), make sure the following information appears at the top of your code:
  - File Name
  - Assignment and exercise number
  - Your names in alphabetic order
  - Submission Date:

  Here is an example in Python:

```
"""
==========================================================
Name         : server.py
Assignment   : Lab 10, Exercise A,B,C
Author(s)    : Mahdi Ansari, William Arthur Philip Louis
Submission   : May 21, 2030
Description  : Flask.
==========================================================
"""
```

- Some exercises in this lab and future labs will not be marked. Please do not skip them, because these exercises are as important as the others in learning the course material.
- In courses like ENSF381, some students skip directly to the exercises that involve writing code, skipping sections such as "Read This First," or postponing the diagram-drawing until later. That's a bad idea for several reasons:
  - "Read This First" sections normally explain some technical or syntax details that may help you solve the problem or may provide you with some hints.
  - Some lab exercises may ask you to draw a diagram, and most of the students prefer to hand-draw them. In these cases, you need to scan your diagram with a scanner or an appropriate device, such as your mobile phone, and insert the scanned picture of your diagram into your PDF file (the lab report). A possible mobile app to scan your documents is Microsoft Lens, which you can install on your mobile device for free. Please make sure your diagram is clear and readable; otherwise, you may either lose marks or it could be impossible for TAs to mark it at all.
  - Also, it is better to use the Draw.io tool if you need to draw any diagram.
  - Drawing diagrams is an important part of learning how to visualize data transfer between modules and so on. If you do diagram-drawing exercises at the last minute, you won't learn the material very well. If you do the diagrams first, you may find it easier to understand the code-writing exercises, so you may be able to finish them more quickly.
- **Due Dates:**
  - You must submit your solution until 11:59 p.m. on the same day that you have the lab session.
  - Submissions until 24 hours after the due date get a maximum of 60% of the mark, and after 24 hours, they will not be evaluated and get 0.

1.5 Academic Misconduct/Plagiarism

- Ask for help, but don't copy.
  - You can get help from lab instructor(s), TAs, or even your classmates as long as you do not copy other people's work.
  - If we realize that even a small portion of your work is a copy from a classmate, both parties (the donor and the receiver of the work) will be subject to academic misconduct (plagiarism). More importantly, if you give exercise solutions to a friend, you are not doing him or her a favor, as he or she will never learn that topic and will pay off for this mistake during the exam or quiz. So, please do not put yourself and your friend in a position of academic misconduct.
  - You can use ChatGPT, but please note that it may provide similar answers for others too, or even the wrong answers. For example, it has been shown that AI can hallucinate, proposing the use of libraries that do not actually exist [1]. So, we recommend that you imagine ChatGPT as an advanced search engine, not a solution provider.
  - In order to find out who is abusing these kinds of tools, we will eventually push you toward the incorrect responses that ChatGPT might produce. In that case, you might have failed for the final mark and be reported to administration.
  - If we ask you to investigate something, don't forget to mention the source of your information. Reporting without reference can lead to a zero mark even by providing a correct answer.

1.6 Marking Scheme

- You should not submit anything for the exercises that are not marked.
- In Table 1, you can find the marking scheme for each exercise.

Table 1: Marking scheme

Exercise	Marks
A	20 marks
B	15 marks
C	15 marks
Total	50 marks

1.7 Complains

- Your grades will be posted one week following the submission date, which means they will be accessible at the subsequent lab meeting.
- Normally, the grades for individual labs are assessed by a distinct TA for each lab and section. Kindly refrain from contacting all TAs. If you have any concerns regarding your grades, please direct an email to the TA responsible for that specific lab.

[1] https://perma.cc/UQS5-3BBP

2 Exercise A (Making RESTful Back-end)

2.1 Preparing Your Environment

2.1.1 Install Python

First, ensure you have Python installed on your system. Python 3.6 or newer is required for Flask. You can download Python from the official website.

2.1.2 Read This First: Understanding Virtual Environments

A virtual environment in Python is a self-contained directory that contains a Python installation for a particular version of Python, plus a number of additional packages. Using a virtual environment allows you to manage dependencies for your project independently of other projects, avoiding conflicts between package versions.

Benefits of Virtual Environments:

- **Isolation**: Keeps dependencies required by different projects separate by creating isolated environments for them.
- **Manage Dependencies**: Easily manage project-specific dependencies without affecting the global Python installation.
- **Simplify Deployment**: By including only the project's dependencies, it simplifies deployment and setup on other machines or servers.

2.2 Creating a Virtual Environment

1. Create a folder in a desired path and name it `lab10`. Inside the `lab10`, create two more folders, `backend` and `frontend`. Now, you must have this structure:

```
/lab10
    /backend
    /frontend
```

2. Open your terminal or command prompt.
3. Navigate to your `backend` project directory: `cd path/to/lab10/backend`.
4. Run the following command to create a virtual environment named `venv`:

```
python -m venv venv
```

5. Activate the virtual environment:
   - On Windows run: `venv\Scripts\activate`
   - On Unix or MacOS run: `source venv/bin/activate`

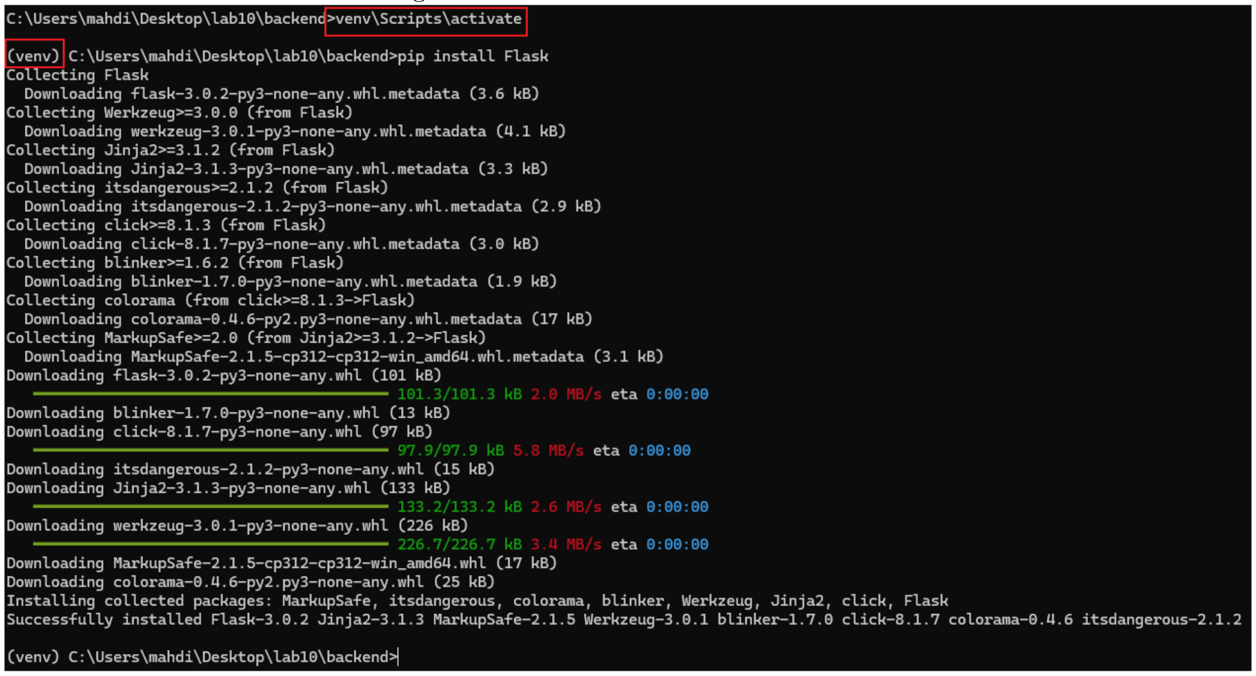
   Figure 1 shows how we activated the `venv` and performed the next step in a virtual environment.

2.3 Install Flask

With the virtual environment activated, install Flask using pip:

```
pip install Flask
```

Figure 1: Activated Virtual Environment



```
C:\Users\mahdi\Desktop\lab10\backend>venv\Scripts\activate

(venv) C:\Users\mahdi\Desktop\lab10\backend>pip install Flask
Collecting Flask
  Downloading flask-3.0.2-py3-none-any.whl.metadata (3.6 kB)
Collecting Werkzeug>=3.0.0 (from Flask)
  Downloading werkzeug-3.0.1-py3-none-any.whl.metadata (4.1 kB)
Collecting Jinja2>=3.1.2 (from Flask)
  Downloading Jinja2-3.1.3-py3-none-any.whl.metadata (3.3 kB)
Collecting itsdangerous>=2.1.2 (from Flask)
  Downloading itsdangerous-2.1.2-py3-none-any.whl.metadata (2.9 kB)
Collecting click>=8.1.3 (from Flask)
  Downloading click-8.1.7-py3-none-any.whl.metadata (3.0 kB)
Collecting blinker>=1.6.2 (from Flask)
  Downloading blinker-1.7.0-py3-none-any.whl.metadata (1.9 kB)
Collecting colorama (from click>=8.1.3->Flask)
  Downloading colorama-0.4.6-py2.py3-none-any.whl.metadata (17 kB)
Collecting MarkupSafe>=2.0 (from Jinja2>=3.1.2->Flask)
  Downloading MarkupSafe-2.1.5-cp312-cp312-win_amd64.whl.metadata (3.1 kB)
Downloading flask-3.0.2-py3-none-any.whl (101 kB)
   ---------------------------------------- 101.3/101.3 kB 2.0 MB/s eta 0:00:00
Downloading blinker-1.7.0-py3-none-any.whl (13 kB)
Downloading click-8.1.7-py3-none-any.whl (97 kB)
   ---------------------------------------- 97.9/97.9 kB 5.8 MB/s eta 0:00:00
Downloading itsdangerous-2.1.2-py3-none-any.whl (15 kB)
Downloading Jinja2-3.1.3-py3-none-any.whl (133 kB)
   ---------------------------------------- 133.2/133.2 kB 2.6 MB/s eta 0:00:00
Downloading werkzeug-3.0.1-py3-none-any.whl (226 kB)
   ---------------------------------------- 226.7/226.7 kB 3.4 MB/s eta 0:00:00
Downloading MarkupSafe-2.1.5-cp312-cp312-win_amd64.whl (17 kB)
Downloading colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Installing collected packages: MarkupSafe, itsdangerous, colorama, blinker, Werkzeug, Jinja2, click, Flask
Successfully installed Flask-3.0.2 Jinja2-3.1.3 MarkupSafe-2.1.5 Werkzeug-3.0.1 blinker-1.7.0 click-8.1.7 colorama-0.4.6 itsdangerous-2.1.2

(venv) C:\Users\mahdi\Desktop\lab10\backend>
```

2.4 Setting Up Your Project

Create the following structure in your `backend` project directory:

```
/lab10/backend
    /product-images
    products.json
    server.py
```

- **/product-images**: This folder will contain product images. Download the *product-images.zip* file via D2L and unzip it inside the `/product-images` folder. The zip file contains thirty folders named from *1* to *30*. We actually downloaded these pictures from the Dummy server.
- **products.json**: This file will store product data in JSON format. We downloaded and modified the JSON file from Dummy project, as well. You do not need to download the original one. Please replace your empty `products.json` file with the file we provided for you via D2L, or copy and paste the contents.
- **server.py**: This script will create the Flask server and define CRUD operations. You will write your server code in this file.

2.5 Implementing the Server

2.5.1 Initialize Flask App

In `server.py`, start by importing Flask and other necessary modules, then initialize your Flask app:

```python
from flask import Flask, request, jsonify, send_from_directory
import json
import os

app = Flask(__name__)
```

2.5.2 Load Products Function

Create a function to read the product data from `products.json`:

```python
def load_products():
    with open('products.json', 'r') as f:
        return json.load(f)['products']
```

2.5.3 CRUD Operations

Define Flask routes to implement CRUD functionality:

- **Read (GET)** - Fetch all products or a specific product by ID:

```python
@app.route('/products', methods=['GET'])
@app.route('/products/<int:product_id>', methods=['GET'])
def get_products(product_id=None):
    products = load_products()
    if product_id is None:
        # Return all products wrapped in an object with a 'products' key
        return jsonify({"products": products})
    else:
        product = next((p for p in products if p['id'] == product_id),
            None)
        # If a specific product is requested,
        # wrap it in an object with a 'products' key
        # Note: You might want to change this
        # if you want to return a single product not wrapped in a list
        return jsonify(product) if product else ('', 404)
```

- **Create (POST)** - Add a new product:

```python
@app.route('/products/add', methods=['POST'])
def add_product():
    new_product = request.json
    products = load_products()
    new_product['id'] = len(products) + 1
    products.append(new_product)
    with open('products.json', 'w') as f:
        json.dump({"products": products}, f)
    return jsonify(new_product), 201
```

2.5.4 Serving Images

Serve images from the “product-images” folder:

```python
@app.route('/product-images/<path:filename>')
def get_image(filename):
    return send_from_directory('product-images', filename)
```

2.5.5 Running the Server

To run your Flask application, include the following code at the end of `server.py`:

```python
if __name__ == '__main__':
    app.run(debug=True)
```

2.6 Running Your Server

Execute the server script in your terminal that you activated virtual environment:

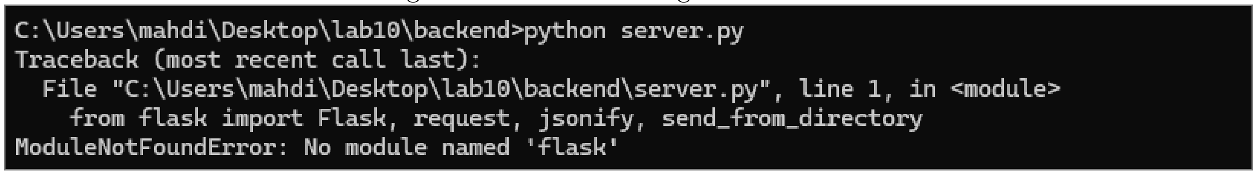
```
python server.py
```

Your Flask server will start, and you can interact with it via http://127.0.0.1:5000/products.

Note:

If you try to run your server on a terminal without activating your virtual environment, then most likely you will see an error like what has been shown in Figure 2.

Figure 2: Error in running outside of venv



```
C:\Users\mahdi\Desktop\lab10\backend>python server.py
Traceback (most recent call last):
  File "C:\Users\mahdi\Desktop\lab10\backend\server.py", line 1, in <module>
    from flask import Flask, request, jsonify, send_from_directory
ModuleNotFoundError: No module named 'flask'
```

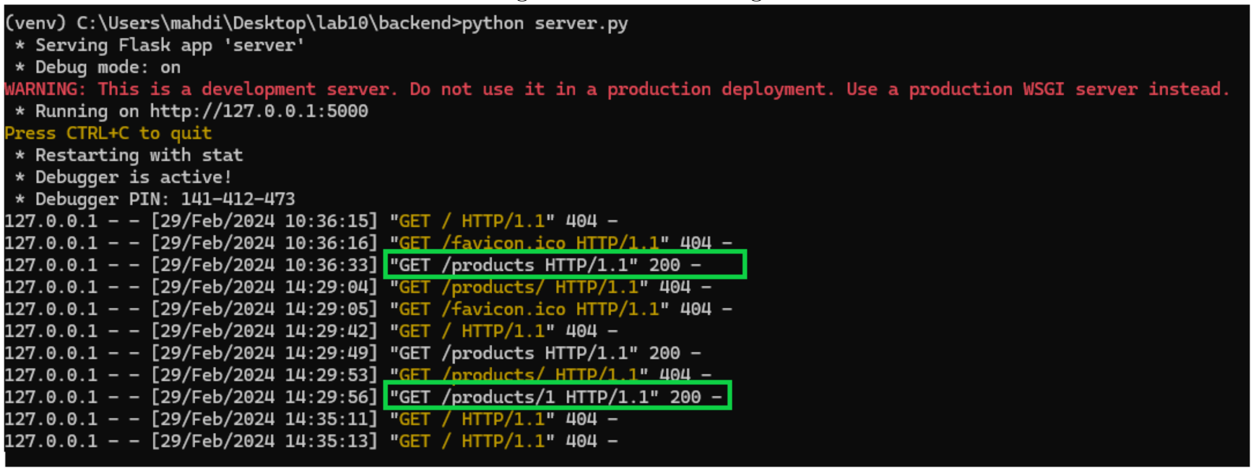
Note:

If you try to visit any of the following URLs: '`http://127.0.0.1:5000`' or '`http://127.0.0.1:5000/`' or '`http://127.0.0.1:5000/products/`' you will see a **Not Found** page.

Note:

When you try to visit different URLs related to this server, you must be able to see some logs in the terminal, like what has been shown in Figure 3.

Figure 3: Terminal Log

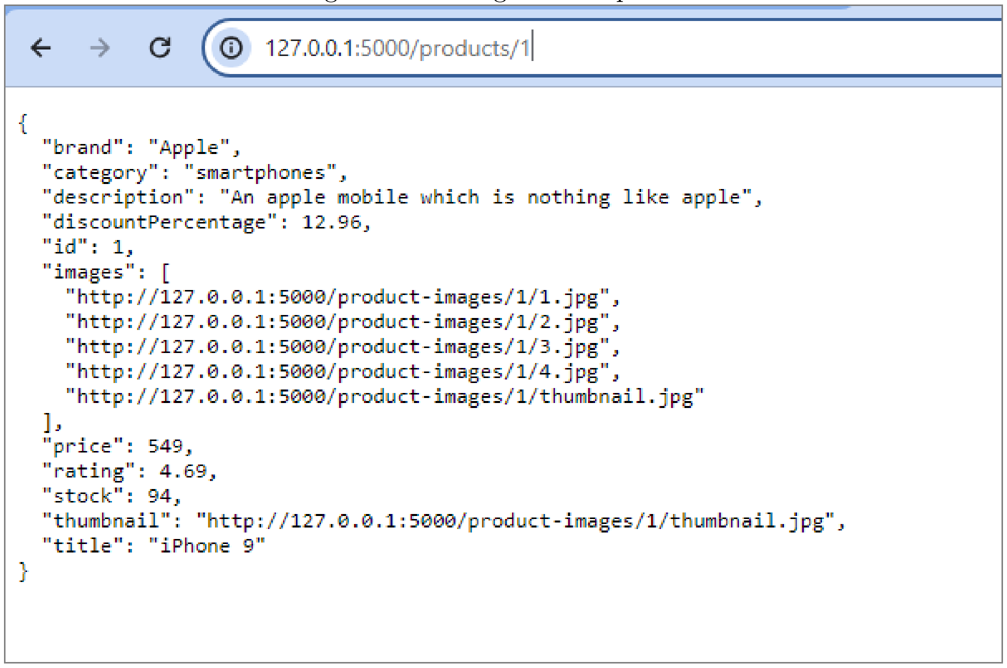


```
(venv) C:\Users\mahdi\Desktop\lab10\backend>python server.py
 * Serving Flask app 'server'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 141-412-473
127.0.0.1 - - [29/Feb/2024 10:36:15] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [29/Feb/2024 10:36:16] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [29/Feb/2024 10:36:33] "GET /products HTTP/1.1" 200 -
127.0.0.1 - - [29/Feb/2024 14:29:04] "GET /products/ HTTP/1.1" 404 -
127.0.0.1 - - [29/Feb/2024 14:29:05] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [29/Feb/2024 14:29:42] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [29/Feb/2024 14:29:49] "GET /products HTTP/1.1" 200 -
127.0.0.1 - - [29/Feb/2024 14:29:53] "GET /products/ HTTP/1.1" 404 -
127.0.0.1 - - [29/Feb/2024 14:29:56] "GET /products/1 HTTP/1.1" 200 -
127.0.0.1 - - [29/Feb/2024 14:35:11] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [29/Feb/2024 14:35:13] "GET / HTTP/1.1" 404 -
```

Note:

If you visit http://127.0.0.1:5000/products/1 you must be able to see detailed information about the first product, something similar to Figure 4.

Figure 4: Visiting the first product



```
{
  "brand": "Apple",
  "category": "smartphones",
  "description": "An apple mobile which is nothing like apple",
  "discountPercentage": 12.96,
  "id": 1,
  "images": [
    "http://127.0.0.1:5000/product-images/1/1.jpg",
    "http://127.0.0.1:5000/product-images/1/2.jpg",
    "http://127.0.0.1:5000/product-images/1/3.jpg",
    "http://127.0.0.1:5000/product-images/1/4.jpg",
    "http://127.0.0.1:5000/product-images/1/thumbnail.jpg"
  ],
  "price": 549,
  "rating": 4.69,
  "stock": 94,
  "thumbnail": "http://127.0.0.1:5000/product-images/1/thumbnail.jpg",
  "title": "iPhone 9"
}
```

2.7 More Tests

1. Visit http://127.0.0.1:5000/product-images/1/3.jpg, you must be able to see a picture from a phone. Take a screenshot of your browser's output, including its address bar. You must add it later to your answer sheet file.

2. Test your POST request via Postman and take a snapshot of its successful output. You must add it later to your answer sheet file.

2.8 You Tasks

1. **PUT and DELETE**: Implement PUT and DELETE similarly, ensuring you find the product by ID and then modify or delete it. Use `update_product` and `remove_product` as your function names, respectively. Please do not forget to follow Dummy project documentation; the routes and other specifications of your services must follow the details provided there.

2. **Test PUT and DELETE**: Try updating and deleting the product with ID 31 using Postman. Please note that the item that you have added in 2.7.2 must be accessible with ID 31! Don't forget to take screenshots to show you have implemented PUT and DELETE successfully. You must add them later to your answer sheet file.

2.9 Deliverable

1. Add your snapshot from 2.7.1 to your answer sheet file.

2. Add your snapshot from 2.7.2 to your answer sheet file.
3. Add your PUT snapshot from 2.8.2 to your answer sheet file.
4. Add your DELETE snapshot from 2.8.2 to your answer sheet file.

2.10 Deactivating the Virtual Environment

Once you are done working in the virtual environment, you can deactivate it by simply running:

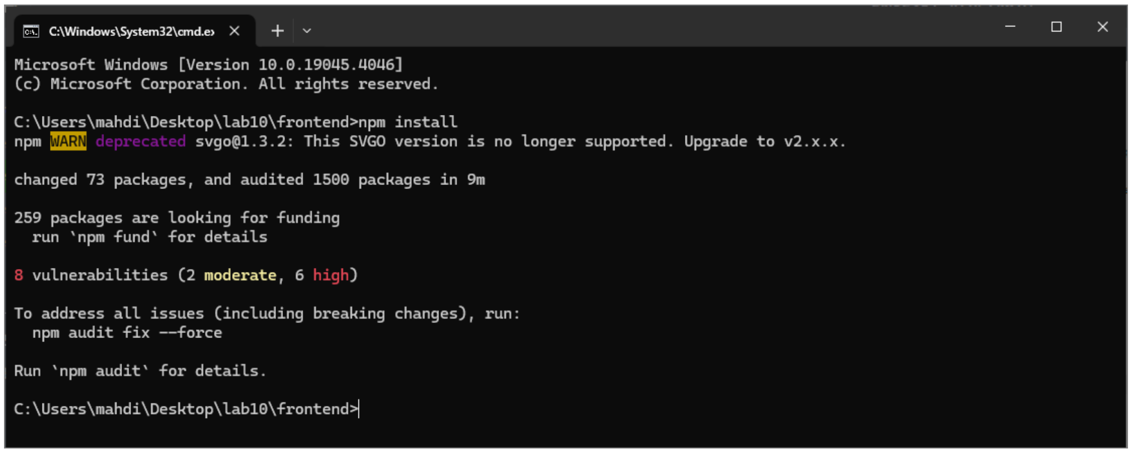
```
deactivate
```

This will revert to using your system's default Python interpreter and environment. **However, do not do it right now; we still need our server up and running for the next steps.**

3 Exercise B (Solving CORS Error)

3.1 Your Task

1. Find your solution for Lab 8, or download the `frontend.zip` file that we provided.
2. Copy and paste the front-end code into the '`/lab10/frontend`' folder. Either from your Lab 8 records or from the `frontend.zip` file.
3. If the '`node_modules`' folder does not exist in your '`lab10/frontend`' folder, you need to run '`npm install`' to install the required Node.js modules. It might take a few minutes, but wait until you see that the installation has finished, as shown in Figure 5.



```
Microsoft Windows [Version 10.0.19045.4046]
(c) Microsoft Corporation. All rights reserved.

C:\Users\mahdi\Desktop\lab10\frontend>npm install
npm WARN deprecated svgo@1.3.2: This SVGO version is no longer supported. Upgrade to v2.x.x.

changed 73 packages, and audited 1500 packages in 9m

259 packages are looking for funding
  run `npm fund` for details

8 vulnerabilities (2 moderate, 6 high)

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.

C:\Users\mahdi\Desktop\lab10\frontend>
```

Figure 5: Reinstalling Node Modules

4. Open a new terminal and run your React development server using `npm start`.
5. Visit the list view http://localhost:3000/products. Confirm that it works, and you can see the list of products.
6. Update the old code from Lab 8 and use the address of the new local server instead of the online dummy server. You only need to update the '`src/services/appService.js`' file and replace the *BASE_URL* value with '`http://localhost:5000/products`' or "http://127.0.0.1:5000/products'. Please note that `localhost` and `127.0.0.1` are equal, however, you might have problem in MacOS, therefore, use the second one in that case.
7. Visit the list view http://localhost:3000/products again. What will you see? You must probably see something like Figure 6, which is called CORS error.

3.2 Solve CORS Error

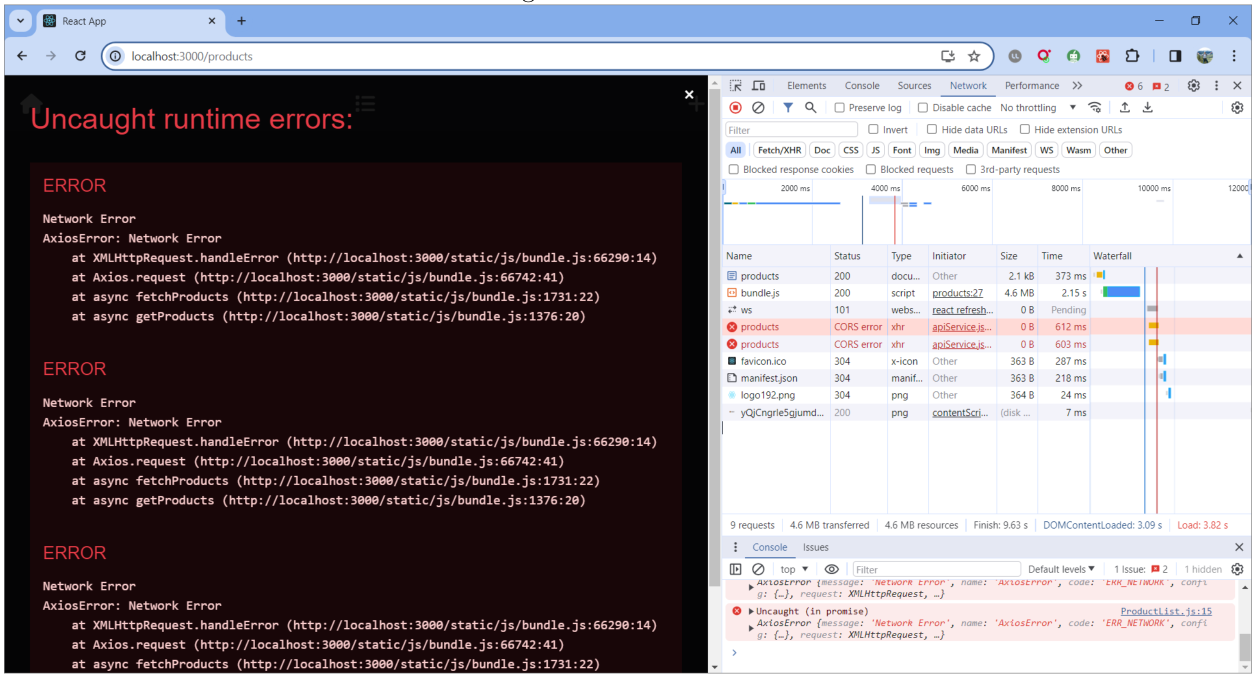
To resolve the CORS (Cross-Origin Resource Sharing) error when accessing your Flask server from a React application (or any other client-side framework), you need to allow cross-origin requests on your Flask server. This can be done using the Flask-CORS extension, which makes it simple to add CORS support to your Flask app.

Here's how you can modify your Flask server to accept requests from all origins.

3.2.1 Install Flask-CORS

First, you need to install the Flask-CORS extension. As we are using a virtual environment, make sure it's activated, then run:

Figure 6: CORS Error



Uncaught runtime errors:

```
ERROR

Network Error
AxiosError: Network Error
    at XMLHttpRequest.handleError (http://localhost:3000/static/js/bundle.js:66290:14)
    at Axios.request (http://localhost:3000/static/js/bundle.js:66742:41)
    at async fetchProducts (http://localhost:3000/static/js/bundle.js:1731:22)
    at async getProducts (http://localhost:3000/static/js/bundle.js:1376:20)

ERROR

Network Error
AxiosError: Network Error
    at XMLHttpRequest.handleError (http://localhost:3000/static/js/bundle.js:66290:14)
    at Axios.request (http://localhost:3000/static/js/bundle.js:66742:41)
    at async fetchProducts (http://localhost:3000/static/js/bundle.js:1731:22)
    at async getProducts (http://localhost:3000/static/js/bundle.js:1376:20)

ERROR

Network Error
AxiosError: Network Error
    at XMLHttpRequest.handleError (http://localhost:3000/static/js/bundle.js:66290:14)
    at Axios.request (http://localhost:3000/static/js/bundle.js:66742:41)
    at async fetchProducts (http://localhost:3000/static/js/bundle.js:1731:22)
```

Name	Status	Type	Initiator	Size	Time
products	200	docu...	Other	2.1 kB	373 ms
bundle.js	200	script	products:27	4.6 MB	2.15 s
ws	101	webs...	react refresh...	0 B	Pending
products	CORS error	xhr	apiService.js...	0 B	612 ms
products	CORS error	xhr	apiService.js...	0 B	603 ms
favicon.ico	304	x-icon	Other	363 B	287 ms
manifest.json	304	manif...	Other	363 B	218 ms
logo192.png	304	png	Other	364 B	24 ms
yQjCngrle5gjumd...	200	png	contentScri...	(disk ...	7 ms

9 requests | 4.6 MB transferred | 4.6 MB resources | Finish: 9.63 s | DOMContentLoaded: 3.09 s | Load: 3.82 s

```
AxiosError {message: 'Network Error', name: 'AxiosError', code: 'ERR_NETWORK', config: {…}, request: XMLHttpRequest, …}
Uncaught (in promise)                                                     ProductList.js:15
AxiosError {message: 'Network Error', name: 'AxiosError', code: 'ERR_NETWORK', config: {…}, request: XMLHttpRequest, …}
```

```
pip install flask-cors
```

3.2.2 Configure CORS in Your Flask App

After installing Flask-CORS, you need to import `CORS` from `flask_cors` and initialize it with your Flask app instance. Modify your `server.py` file to include CORS support:

```
from flask import Flask, request, jsonify, send_from_directory
from flask_cors import CORS  # Import CORS
import json
import os

app = Flask(__name__)
CORS(app)  # Enable CORS for all domains on all routes

# The rest of your Flask app code goes here
```

By calling `CORS(app)`, you enable CORS for all routes and all origins. If you need to restrict access to specific origins or configure other CORS behaviors, the Flask-CORS extension provides options to do so. For example, to allow only requests from specific origins, you can pass the `origins` parameter:

```
CORS(app, origins=["http://example.com", "http://localhost:3000"])
```

This configuration allows requests only from `http://example.com` and `http://localhost:3000`.

3.2.3 Run Your Flask App

With CORS configured, run your Flask app as you normally would. Your React application should now be able to communicate with your Flask server without encountering CORS errors.

3.2.4 Additional Notes

Security Considerations: While enabling CORS for all origins (`CORS(app)`) is convenient for development, it's important to restrict origins to trusted sites in production environments to prevent unwanted cross-origin requests. Some educational websites, like Dummyjson.com, do not care about security and it is fine based on their purpose. That is the reason you haven't seen this error in Lab 8.

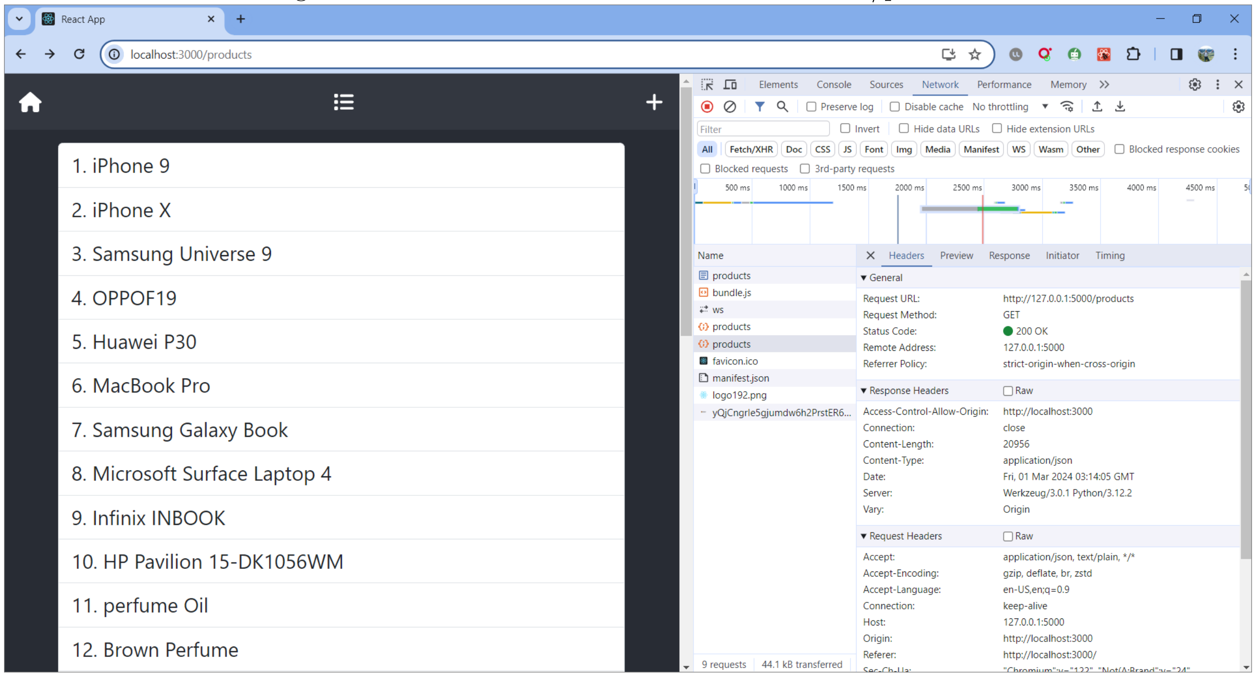
3.3 Testing

After making these changes, test your Flask server with your React application to ensure that the CORS issue is resolved. You can use development tools in your browser to inspect network requests and confirm that CORS headers are correctly applied to responses from your server.

3.4 Deliverable

1. Submit your final `server.py` file as part of your final submission. Please do not add the contents of your `server.py` file to the answer sheet file.
2. Take a screenshot of your browser, like what I did in Figure 7, and add it to your answer sheet file. It must show the GET call to `/products`. Also, do not forget to activate the header tab for the request before making your snapshot.

Figure 7: Evidence for successful GET call to local /products



1. iPhone 9
2. iPhone X
3. Samsung Universe 9
4. OPPOF19
5. Huawei P30
6. MacBook Pro
7. Samsung Galaxy Book
8. Microsoft Surface Laptop 4
9. Infinix INBOOK
10. HP Pavilion 15-DK1056WM
11. perfume Oil
12. Brown Perfume

```
products
bundle.js
ws
products
products
favicon.ico
manifest.json
logo192.png
yQjCngrle5gjumdw6h2PrstER6...

9 requests   44.1 kB transferred
```

```
▼ General
Request URL:                   http://127.0.0.1:5000/products
Request Method:                GET
Status Code:                   ● 200 OK
Remote Address:                127.0.0.1:5000
Referrer Policy:               strict-origin-when-cross-origin

▼ Response Headers             ☐ Raw
Access-Control-Allow-Origin:   http://localhost:3000
Connection:                    close
Content-Length:                20956
Content-Type:                  application/json
Date:                          Fri, 01 Mar 2024 03:14:05 GMT
Server:                        Werkzeug/3.0.1 Python/3.12.2
Vary:                          Origin

▼ Request Headers              ☐ Raw
Accept:                        application/json, text/plain, */*
Accept-Encoding:               gzip, deflate, br, zstd
Accept-Language:               en-US,en;q=0.9
Connection:                    keep-alive
Host:                          127.0.0.1:5000
Origin:                        http://localhost:3000
Referer:                       http://localhost:3000/
```

4 Exercise C (Solving Content-Type Error)

4.1 Fix one more problem

1. You have updated the *BASE_URL* in the previous section. However, if you try to add a new product or update an existing one, you will get an error in the console as *Did not attempt to load JSON data because the request Content-Type was not 'application/json'.*

2. We have to update the `addProduct` and `editProduct` functions in the *apiService.js* file, and add the content type to header requests. Here is the code for the `addProduct` method.

```
export const addProduct = (product) => {
    return axios.post(`${BASE_URL}/add`, JSON.stringify(product), {
        headers: {
            'Content-Type': 'application/json',
        },
    });
};
```

3. Modify `editProduct` function as well.

By now, your front-end application must also be functional. Just yet, we have not provided a mechanism for uploading pictures to the server. If you are interested, you can try to extend the code in both the back-end and the front-end.

4.2 Deliverable

Follow the following scenario and make a video or gif file:

1. Visit the listing page.
2. Add a new product.
3. Visit the listing page again.
4. Visit the detail page of the new product (i.e., id = 31).
5. Edit the name of the newly created product.
6. Visit the listing page again; the name of the last product must be updated.
7. Delete the newly created product.
8. Visit the listing page again; the newly created product must be gone.
9. Visit one of the products; it must have detailed information and a thumbnail picture.

Submit your video beside the *server.py* file and the answer sheet file. We do not need your front-end code.