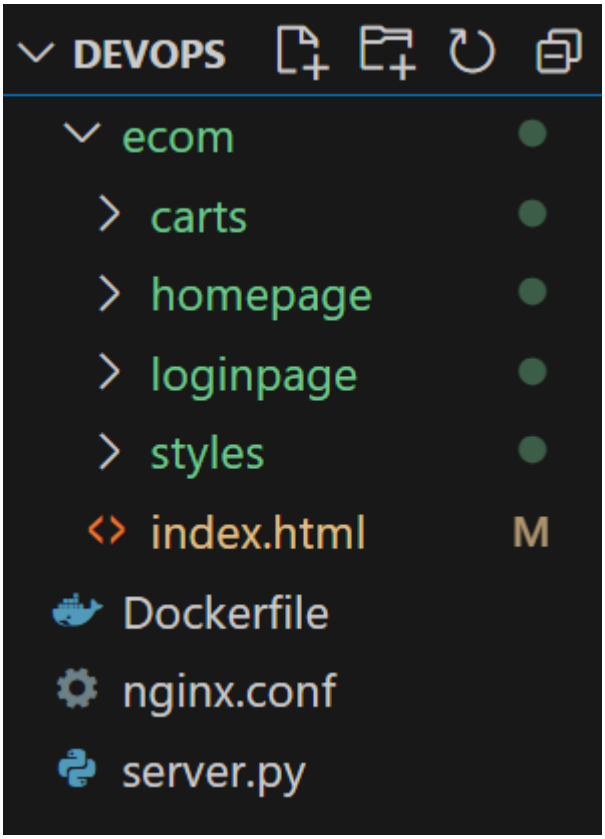
CREATING THE PROJECT AND CONTAINERIZATION USING DOCKER

1. Creating the actual project – Webpage
    - Here we are creating a webpage which basically falls under the category of E-commerce.
    - We will be dividing this task into 3 separate folders under the actual main folder
    - One for Login page, one for Homepage and the other for the Cart page
    - Here is the file structure
    - 

2. Creating the Python HTTP Service

- Here, for the purpose of serving the webpage, we are making the use of Python HTTP service, other alternatives like Nginx can also be preferred.
- Create a file named index.py to start a simple HTTP server and serve the static webpage:

```
import http.server
import socketserver
import os

PORT = 8080
DIRECTORY = "ecom"

class Handler(http.server.SimpleHTTPRequestHandler):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, directory=DIRECTORY, **kwargs)

    def do_GET(self):
        if self.path == '/homepage' or self.path=='/':
            self.path = '/homepage/index.html'
        elif self.path == '/login' :
            self.path = '/loginpage/login.html'
        elif self.path == '/cart':
            self.path = '/carts/cart.html'
        return http.server.SimpleHTTPRequestHandler.do_GET(self)

with socketserver.TCPServer(("", PORT), Handler) as httpd:
    print("Serving at port", PORT)
    httpd.serve_forever()
```

- http.server.SimpleHTTPRequestHandler automatically serves files in the current directory.
- The server will be available at http://localhost:8080.

3. Dockerize the application

- To run the static webpage in a container, we need to package it along with the Python HTTP service. This is done using Docker.

(1) Create a Dockerfile

- In the root of the project directory (ecom) as shown in the above picture, create a file named Dockerfile with the following content:

```
FROM python:3.9-slim

WORKDIR /app

# Copy the necessary files
COPY ./ecom /app/ecom
COPY ./server.py /app/server.py

# Expose port 8080
EXPOSE 8080

# Run the Python HTTP server
CMD ["python", "server.py"]
```

- Explanation of the Dockerfile:

- FROM python:3.9-slim: Uses a minimal Python 3.9 image.
- WORKDIR /app: Sets the working directory inside the container to /app.

- COPY ./ecom /app/ecom: Copies all files from the ecom directory into the container's /app/ecom directory.
- COPY ./server.py /app/server.py: Copies all content from the server file into the container's /app/server.py file.

- EXPOSE 8080: Exposes port 8080 for the Python HTTP server.
- CMD ["python", "app.py"]: Runs the Python application to start the HTTP server.

(2). Add the index.html and app.py to the Docker container:

Make sure both the index.html file and app.py are in the same directory as the Dockerfile.

4. Test the docker container locally

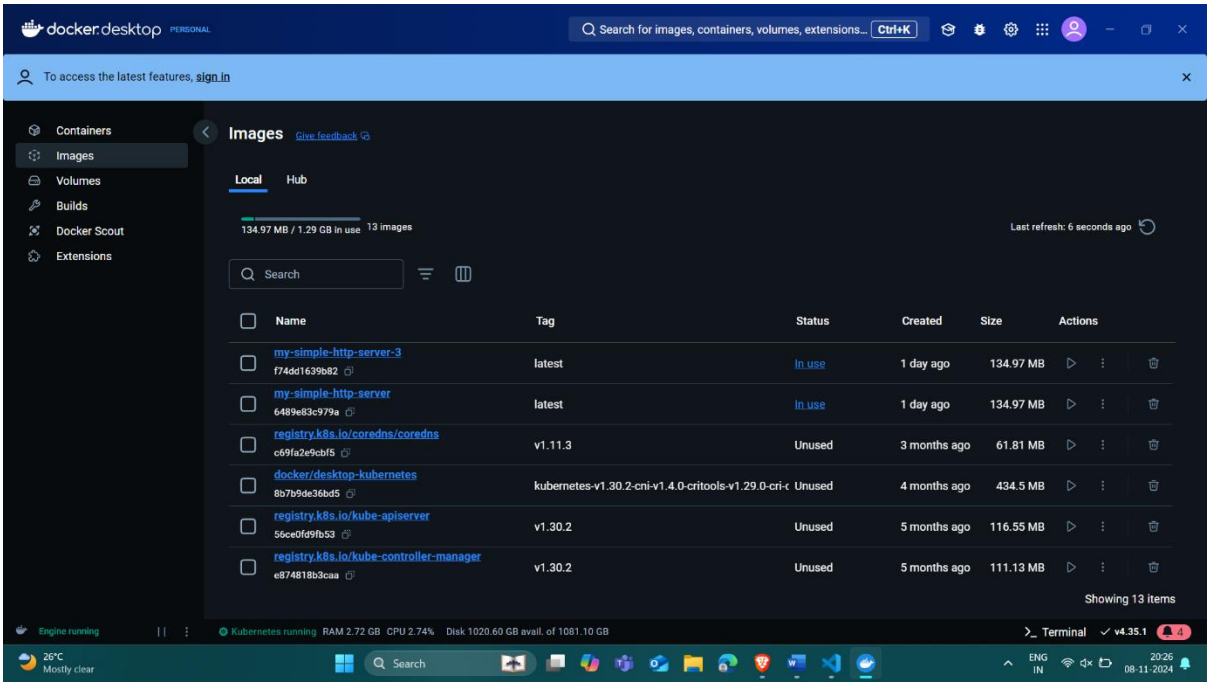
- Open a terminal in the project directory ecom and run:
- docker build -t my-simple-http-server .
- This command builds the Docker image and tags it as my-simple-http-server

- Run the docker container using the code:

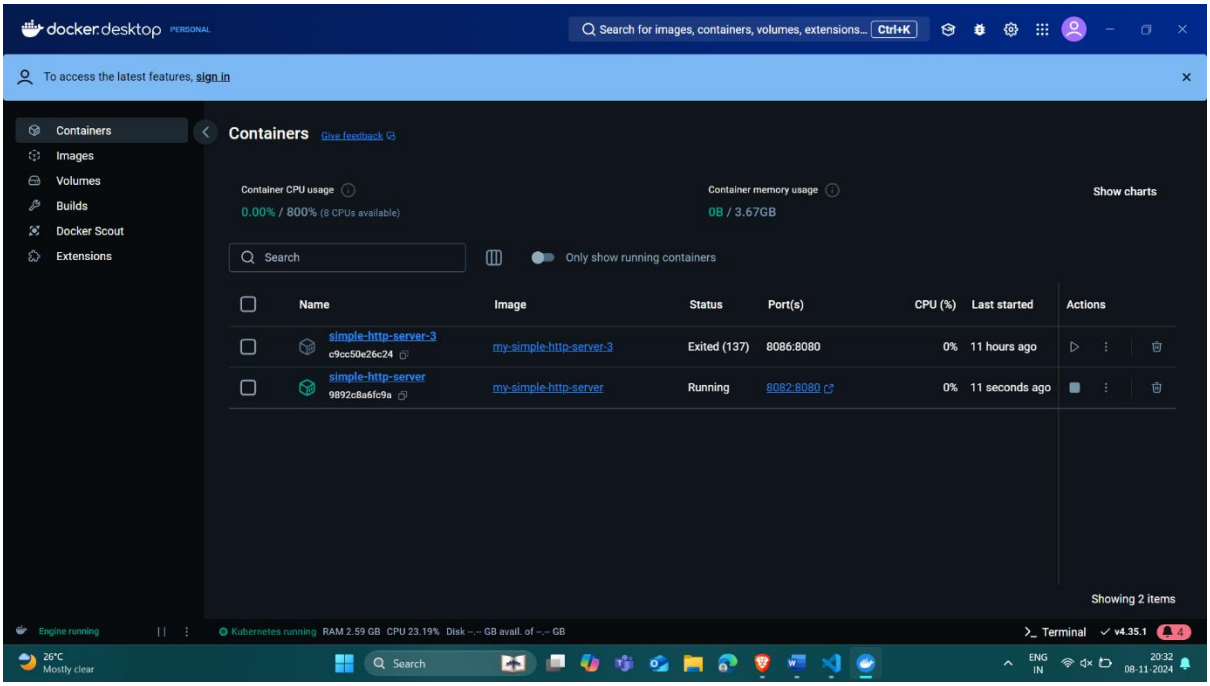
- docker run -p 8082:8080 simple-http-server

- Here, there is port mapping done, even though the server starts at the port 8080 originally, for us, the application will run on the port 8082.

- Test the webpage:
- Open your web browser and go to http://localhost:8082. We can see the webpage served by our Python HTTP server.



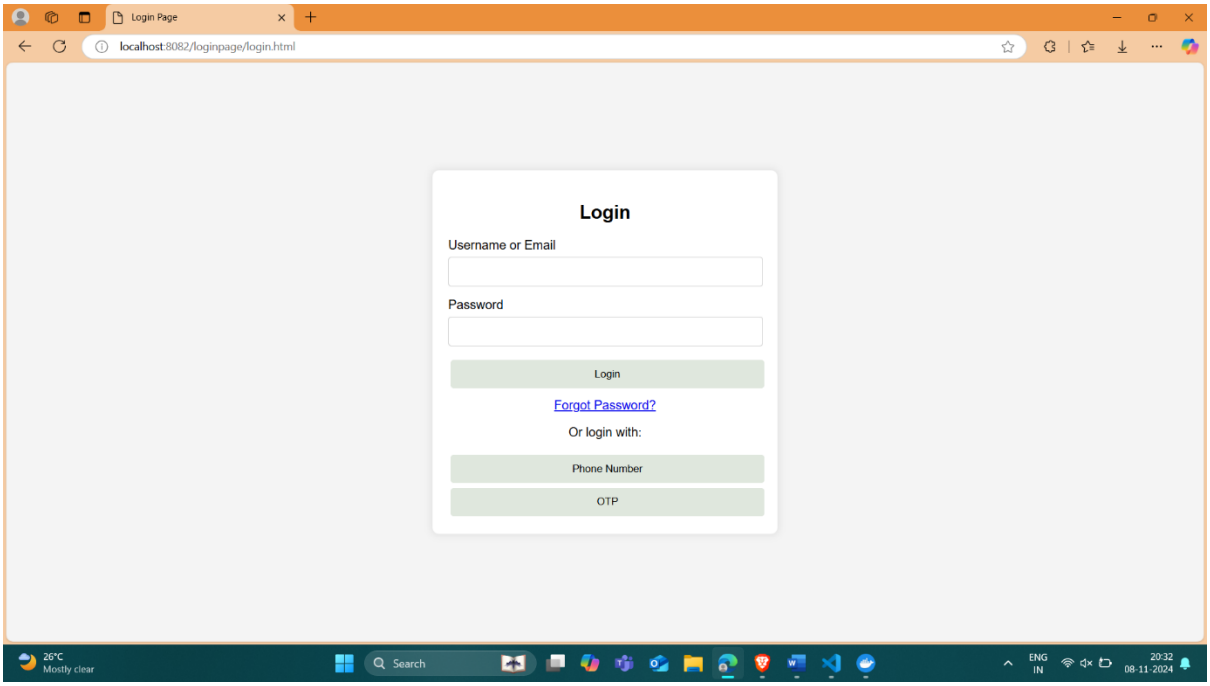
- Here, we can see that out of all these images, we have created an image called as “my-simple-http-server”



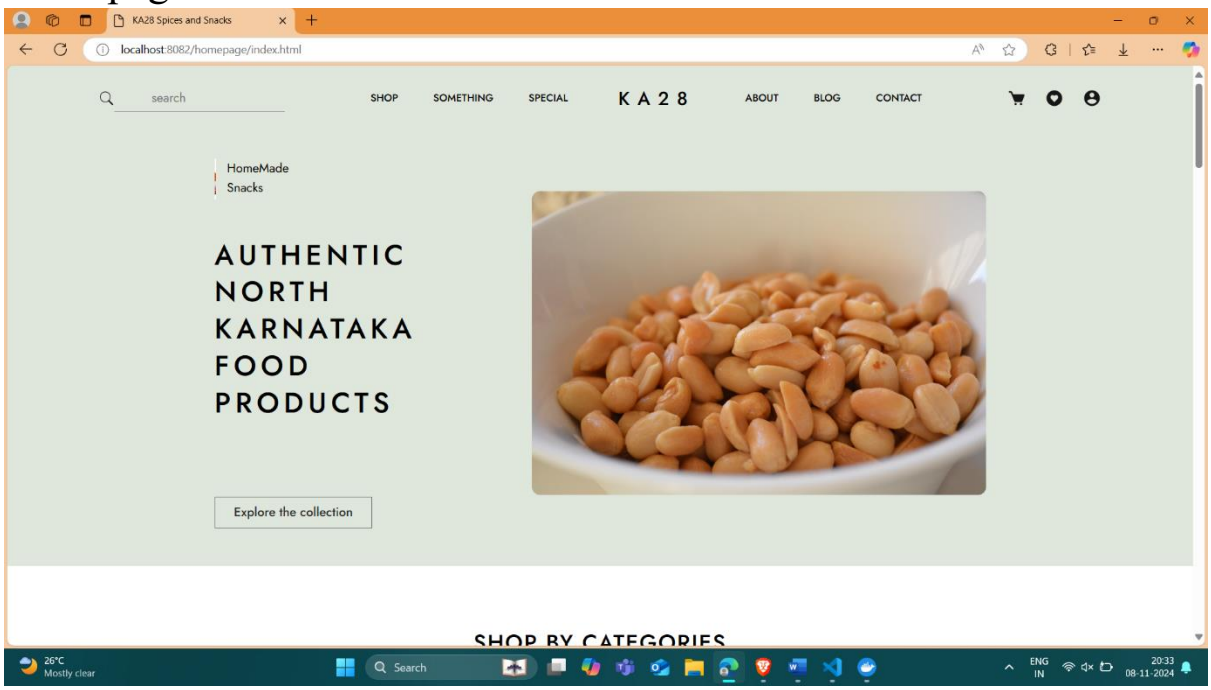
- In the above image you can see that the container has been created named as "simple-http-server" which runs with the image of "my-simple-http-server" which is being mapped on port 8082. This is the actual container which we have just created a while back.

➢ After clicking on the port 8082:8080 (or) going to the localhost:8082, we will get the following results

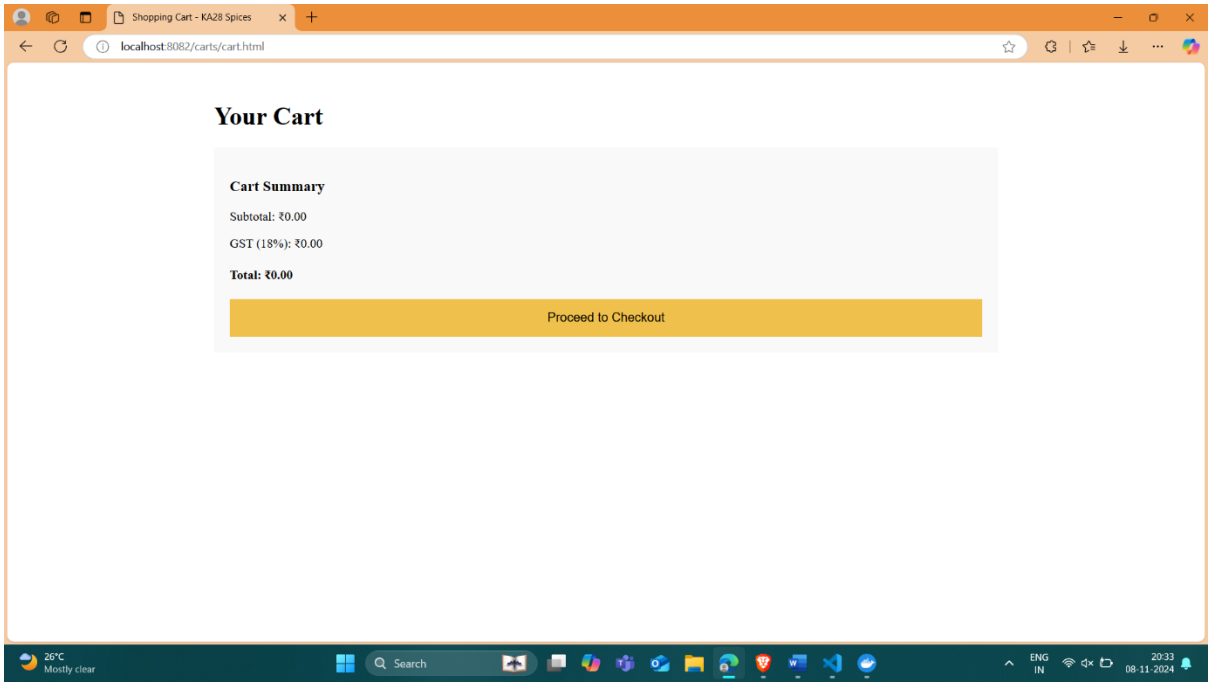
➢ Login Page:

➢ 

❖ Homepage:

❖ 

❖ Cart Page:



And hence, our webpage application is containerized and is running on the local system successfully.

This can also be pushed into Docker Hub if we want to make it accessible to the outside world.