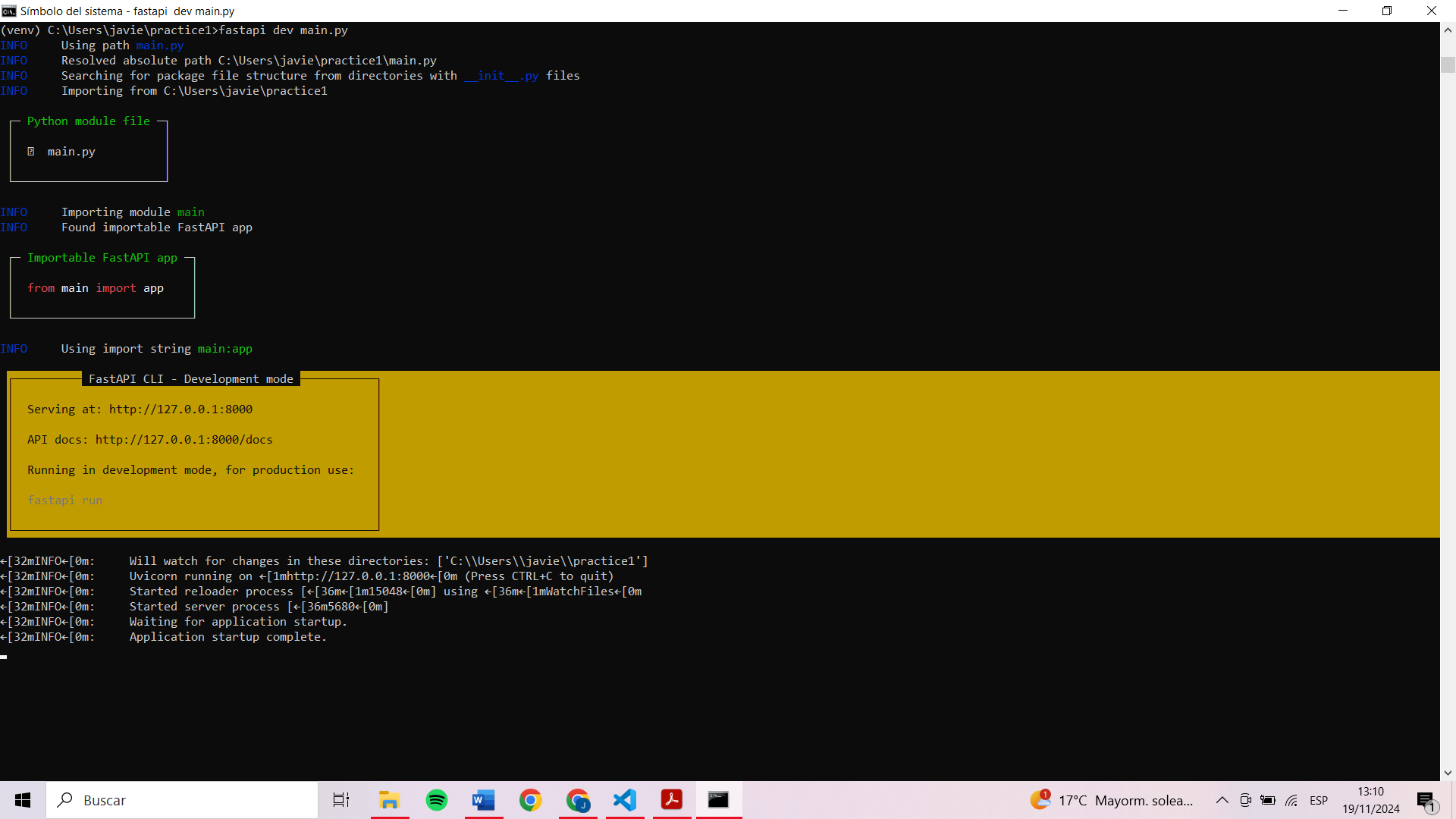
P1: API & Dockerization

Report

Following the steps that FastAPI shows in its web, we have installed all the necessary. After creating the main.py file, we run the server and the following result it's what we get:  


As we checked the creation of the API with a browser and random parameters, we can now continue.   
  
We began by restructuring our project to align with the FastAPI framework. Using the FastAPI GitHub repository as a reference, we modified the main.py file to serve as the entry point for our API. This file defines the API structure and serves as the main router for the endpoints.

Additionally, we integrated the functionality from Seminar 1 by adapting the S1.py script. With the help of AI, we updated and optimized the previous exercises to fit into the new API framework.

We installed Docker Desktop to enable containerized deployment.

A Dockerfile was created to define the container's setup, specifying the base image, environment, and the steps to build and run the API. Additionally, a requirements.txt file was created to list all necessary libraries and dependencies for the API to function correctly.

Using the tutorial from the FastAPI official documentation, we successfully built the Docker image for the API. After running the container, the API became accessible at <http://localhost:80> or <http://127.0.0.1:80>.

We created six endpoints, leveraging the work from Seminar 1 and extending its functionality. These endpoints were designed to interact with users through both HTML and API-based interfaces. When accessing <http://localhost:80> or <http://127.0.0.1:80>, users are presented with a main menu displayed in HTML format, which provides links to the various API endpoints.

(Photo menu)

Additionally, the API documentation is accessible at <http://localhost:80/docs> or <http://127.0.0.1:80/docs>through the Swagger UI, allowing users to interact with the API's endpoints in a structured and intuitive way.

Among the six endpoints, four provide interactivity, enabling users to perform specific tasks. These include value conversion, where users can input values for transformation, and image manipulation, which allows users to upload an image for resizing or applying a black-and-white filter. These interactive endpoints highlight the adaptability and enhanced functionality of the integrated work, offering a user-friendly experience for interacting with the API.

(Photo interact api)

**Commands Used for Running the FastAPI and Docker**

To run and manage the FastAPI application locally, we used the command fastapi dev main.py to start the API in development mode. When finished, the API could be stopped by pressing CTRL + C. These simple commands allowed for efficient testing and debugging during the development process.

For Docker, we ensured the application was containerized and ready for deployment by first building the Docker image with the command docker build -t myfastapiapp .. Once built, the container was run using docker run -d -p 80:80 --name myfastapiapp myfastapiapp, which started the application on port 80. To stop and remove the container, the command docker rm -f myfastapiapp was used. Additionally, if the image was no longer needed, it could be removed using docker rmi myfastapiapp, though this step was optional. These commands facilitated seamless container management and ensured a robust deployment pipeline.

Then we did the dockerization, first we have installed [Docker Desktop](https://www.docker.com/products/docker-desktop/), then we created a dockerfile, and created a requirements.txt to have installed all the libraries and programs that we need to run our api.

So then we have successfully built and dockeritzate using the api. Thanks to the following tutorial.

<https://fastapi.tiangolo.com/deployment/docker/#create-the-fastapi-code>

Then we have adapt our main.py and S1.py to include, the previous work that we have done in seminar 1, and thanks to the AI, we have adapt all the previous exercises.

Finally we have created 6 end points. Where when we run the docker file and we enter to:  
<http://localhost:80> or <http://127.0.0.1:80> we can observe a menu in html.  
(photo)

Then when we enter to <http://localhost:80/docs>or <http://127.0.0.1:80/docs>where we can interact with the functions that we have adapt, where 4 of them are more interactive at the time to put values to convert them, or select an image to resize o black & white.