Project guidelines

2024-07-17, v 0.2

Introduction

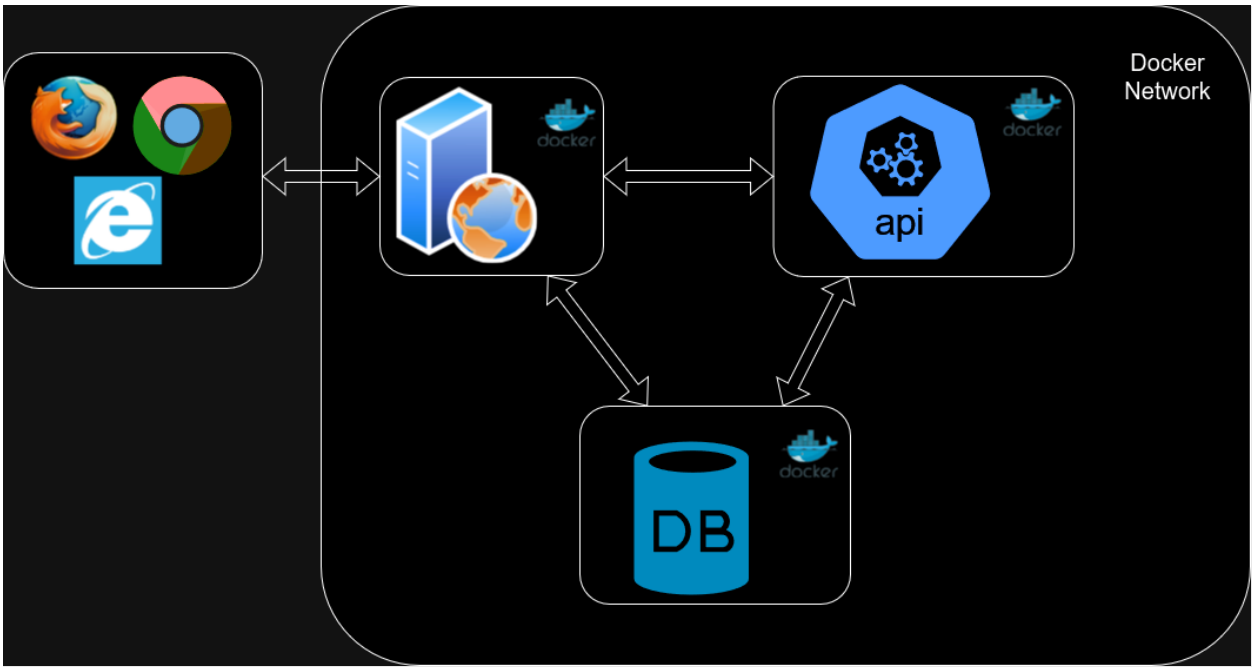
Welcome to the last part of the course!

This is your time to shine and bring everything you learned this semester to create a great final project, pay close attention to the **Project Architecture**, **the new interface (We added and removed some instructions from it)** and follow the guidelines for the technical part.

Afterwards you will take the technical skeleton and give your personal spin on it (add more functionalities to the project) **but** make sure to not change the technical skeleton.

Project Architecture

The project will be based on the homework but with slightly different architecture. The design of the project can be seen in the following diagram:



Project Components

As you can see from the diagram, the project consists of the following three components:

Database

The database should be a MongoDB database running inside a docker container (see the Docker video to see how to set up such a container). It should save its persistent data to a named docker volume.

Image Classification API:

- The API is a simple REST API.
- It must implement the requirements in the interface0.3.md document.
    - Note that (unlike in the HW) this API is completely stateless, i.e. each request contains all the information necessary to process it. This means that there are no sessions and no authentication.
- To classify the images, the API should either use a web API or a locally run ML model.
- The server can (and probably has to) save any data that must be persistent to the database.
- The API server must run inside a docker container.
- The API server must be run using gunicorn using multiple workers to allow parallel processing of requests.

Web Server

- The web server will allow access to the image classification API using HTML pages.
- The web server must implement registration, login, and logout.
- The web server must only allow access to the API services to users that have registered and logged in.
- The web server must use browser cookies (via Flask sessions) to keep a user logged in during a session.
    - Note that sessions depend on the browser settings and the expiration time of the cookies and we do not give specific requirements for how long the user should remain logged in.
- The web server must allow access to all the services given by the API
    - Note that we do not define how exactly the user should be able to access these services (which endpoints, how to navigate between them etc.). This is up to you.

    - For example, you need to let the user choose synchronous or asynchronous classification. You can implement this with a toggle button on the upload page, or have different endpoints for the different methods or any other solution you can think of.
- The web server can save any data that must be persistent (e.g. login details) to a database. This will be a different database instance from the one used by the API server.
- The server must be capable of serving multiple clients concurrently.
- The server shall run inside a docker container.

Networking

Since the API and the DB will not require authentication, they must not be generally accessible. Instead, all communication between the DB, API and the web server must be done using a docker network (see docker 101 tutorial for details on how to set this up, and read more in the docker docs). Only the web server shall be accessible from outside the docker network.

Stress Testing

We will check the image classification endpoint (which is common to all implementations). Let's say that one model inference job takes T seconds.

Running 6 concurrent image inference jobs should complete successfully in < 1.1* T Seconds.

To enable this test, your web server must implement **POST /classify_image** with the same semantics as the API's "/upload_image".

Example:

Your server completes the classification task in 7 seconds (average time).

Our test code will run 6 concurrent requests, and all of them should be completed in less than 1.1*7 seconds.

Completion is when the GET /result returns "completed"

Monitoring

How do you know if your server is up or crashed? Add a third-party monitor such as https://uptimerobot.com/ (strongly recommended)

Containers

You should create Dockerfiles for any containers that use images not hosted in public dockerhub repositories.

You should create a docker-compose yaml file that handles file mounting, network settings, and any other setting required for starting up all the necessary containers. Running your project must be as simple as running "*docker compose up*".

Virtual Machines

Each team will be allocated a virtual machine. You will need to install docker so that your project can run on the virtual machine. See tutorial.

Version Control

All the files needed for your project must be saved to a private github repository. Make sure to make proper use of git as discussed in class. All team members should contribute to the project.

Grading

Technical implementation – 75 %

Containers, Interface, Servers, DBs ,Git usages, Testing

Report quality – 5%

You can write the report in Hebrew or in English

You cannot generate the report using LLMs or similar tools. We will not accept reports generated by these LLMs. You can use these tools to fix your grammar but ensure that the text isn't filled with LLM "fluffy words" that make it difficult to read and understand the main points.

Creativity & Complexity – 20%

What are we looking for in Creativity and Complexity of a project?

In terms of scoring, we will score projects by their **Complexity** and their **Creativity**, not by the look of the HTML and CSS pages and not by the ML\DL algorithms that you are deploying.

The focus itself should be on the backend of the server and the functionality that it serves. Some ideas might be a collaborative application that X amount of user team up and create something together or an application that does some parallel processing and calls some different functions to produce something, anything is possible, think of something that will be interesting for you to build that has a focus on the backend aspects and topic learned in class.

Yes - you can create more containers, Yes you can create more servers, Yes you can add more algorithmics, yes you can add more endpoints, yes , yes , yes. As We said before, it's up to you to create something that will be interesting!

Submission

- A PDF report describing the work you've done on the project should include the following sections: Introduction, details on the utility of the server, and technical implementation details describing how you achieved the creative part of the project.

  The report should be 3 pages max, and have an appendix (that isn't count towards the 3-page max) which includes:
  1. A reflection on what were the most challenging aspects of the project.
  2. Images (**it's highly recommended to add images to the appendix**)
  3. A list of all end points and their short descriptions.

- Video recorded from the mobile\pc showcasing the project and the services that your server serves. **Narrating or talking over the recording is recommended.**
- **Private GitHub Repo with permission for cohenstav1@gmail.com,** orifa@campus.technion.ac.il, cnoam@technion.ac.il

Important dates

For the project presentation, you will be asked to create a "digital poster," meaning one PowerPoint slide that will present what you have done in the project up to the presentation and your overall plan. More details will be released at the start of August.

11.8 Project presentation 1.

18.8 Project presentation 2.

20.8 Project submissions.

GOOD LUCK!