Back-end part

Contents

[I. "Present Deployment and CI Strategy" 1](#_Toc139042434)

[I. The system architecture and container solution 1](#_Toc139042435)

[II. Service orchestrator 2](#_Toc139042436)

[II. Files in the application 3](#_Toc139042437)

[III. Launch the application locally - Development part 4](#_Toc139042438)

[I. Launch the application 4](#_Toc139042439)

[II. Coding part 5](#_Toc139042440)

# "Present Deployment and CI Strategy"

## The system architecture and container solution

As part of my project, I have opted to incorporate **Docker** as my container solution to achieve these goals.

Why Docker?

Docker has gained immense popularity in recent years due to its ability to simplify application deployment, enhance scalability, and improve collaboration among developers. By choosing Docker, I can leverage its numerous advantages to streamline my project's development and deployment processes.

How Docker Fits into the System Architecture?

In my system architecture, Docker plays a pivotal role by providing a consistent and isolated environment for my application's components. I can divide my application into smaller, self-contained services, each running within its own Docker container. This approach, known as microservices architecture, offers several benefits.

Firstly, microservices enable independent scalability..

Secondly, Docker facilitates a modular and decoupled architecture.

Furthermore, Docker simplifies the deployment process.

## Service orchestrator

As part of my project, I have selected **Docker Compose** as my service orchestrator to simplify the deployment and management of my containerized application.

Why Docker Compose?

Docker Compose is a powerful tool that enables the definition and management of multi-container applications. It allows me to declare the various services, their configurations, and their dependencies in a simple YAML file. By choosing Docker Compose, I can leverage its capabilities to streamline the deployment and coordination of my containerized application's components.

How Docker Compose Fits into the System Architecture?

In my system architecture, Docker Compose acts as the orchestrator that manages the deployment and coordination of the containerized services. By utilizing Docker Compose, I can define and configure the various services that compose my application, such as web servers, databases, message queues, and more.

# Files in the application

1. **Folder: Templates:** The "templates" folder contains the HTML templates used for rendering the user interface of my application. These templates serve as the foundation for generating dynamic web pages by incorporating data from my backend code.
2. **Folder: venv:** The "venv" folder is commonly used to store a virtual environment for my Python application. A virtual environment is an isolated Python runtime environment that allows you to install and manage project-specific dependencies without conflicting with the system-level Python installation or other projects. It ensures consistency across different development environments and simplifies dependency management.
3. **File: app.py:** The "app.py" file is the main entry point for my application's backend code. It contains the implementation of my application's routes, business logic, and other server-side functionalities. This file defines the behavior of my application and handles incoming requests from the user interface or API endpoints.
4. **File: docker-compose.yml**: The "docker-compose.yml" file is used to define and configure multiple Docker containers that make up my application's services. It specifies the desired services, their configurations, networking settings, and dependencies.
5. **File: Dockerfile:** The "Dockerfile" is a text file that contains instructions for building a Docker image of my application. It defines the environment, dependencies, and steps required to create a containerized version of my application.
6. **File: requirements.txt**: The "requirements.txt" file is commonly used in Python projects to list the dependencies required by my application. Each line of the file specifies a package name and version, allowing me to easily install and manage the necessary libraries and modules. Tools like pip use the requirements.txt file to resolve and install the specified dependencies.
7. **File: README:** The "README" file is a text file that provides essential information and instructions about my application.

# Launch the application locally - Development part

## Launch the application

Steps to run the application.

* Go on the virtual environment.

On window: ***“venv\Scripts\activate”***

On Mac/Linux : ***“source venv/bin/activate’’***

First method:

* Write ***“docker-compose build”.***
* Write ***“docker-compose up”.***

Second method:

* Write ***“flask run”.***

More information on the file README.txt

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

*Figure 1: First method to run the application locally*

## Coding part

The functions are all in the file app.py. To see the structure of these functions is possible in the app.py file.

1. Different tables

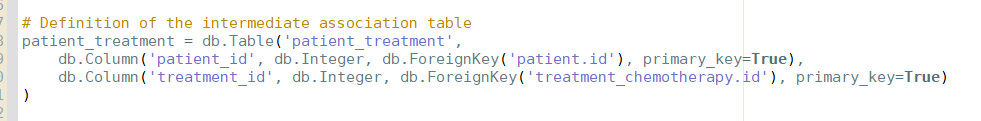
In the system, we have 3 tables, one for the patient and another one for the treatment.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

*Figure 2: TreatmentChemotherapy and Patient class from the app.py file*

They are connected to each other in this way:



*Figure 3: Connection between TreatmentChemotherapy and Patient class from the app.py file*

The third table is Admin for administrators.

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

*Figure 4: Admin class from the app.py file*

1. The main page is index.

*Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Description générée automatiquement*

*Figure 5: Page 1 of the main page*

The page is associated with the python function ***index()*** and ***create()***.

The function ***index()*** checks is the user is connected or not. And the function ***create()*** can create a new patient in the database.

The function ***create()*** call the function ***trouver\_meilleur\_traitement()*** to find the best treatment.

When we put all patient information, we have the result: the treatment, with the more chance to survive. We have a graphic to compare with other treatments. (The graphic is create in the ***create()*** function)

*Une image contenant texte, capture d’écran, Tracé, diagramme

Description générée automatiquement*

*Figure 6: Last page of the main page: the result*

The way to choose the best treatment is based on patients with the same profile and the result is only their survival time.

The system here does not take all the criteria and only takes the survival time as a result (age is not taken into account). Moreover, it is only an average calculation.

Unfortunately, this is due to lack of data. The excel table provided had only 245 patients.

1. The next page is admin to see all the database.

*Une image contenant texte, capture d’écran, logiciel, nombre

Description générée automatiquement*

*Figure 7: Administration’s page : an overview of the database*

The page is associated with the python function ***admin()*** and ***admin\_delete().***

We can see all the data of patient and all the administrators. We can only delete an administrator, if we are login with an Admin with the status: *Superuser.*

1. And only Superuser can add a new administrator by clicking on the button “Add an administrator”.

*Une image contenant texte, Police, logiciel, capture d’écran

Description générée automatiquement*

*Figure 8: The administrator’s registration page*

Here the page associated with the python function ***register\_admin().***

1. To access to the admin page you need to be login.

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Description générée automatiquement

*Figure 9: The login page*

The page is associated with the python function ***login().***

1. The button *logout* in the admin page, allows the user to logout.

*Une image contenant texte, Police, capture d’écran, conception

Description générée automatiquement*

*Figure 10: Logout button*

The button is associated to the function ***logout().***

1. In the admin page, we can edit or delete the patient from the database with the function ***edit()*** and ***delete()***

*Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement*

*Figure 11: A part of the admin page where we can see edit and delete button*

We can edit any criteria of the patient. We can also add the result of an patient (time of survival).

*Une image contenant texte, capture d’écran, conception

Description générée automatiquement*

*Figure 12: The edit page*

1. Some functions have been used for setting up the system. For example, a function was created to create patients in the database from the excel file provided by the Munich hospital.

This function is named ***create\_patient\_from\_excel().***