

Backend part

Contents

I.	"Present Deployment and CI Strategy"	1
I.	The system architecture and container solution	1
II.	Service orchestrator	1
II.	Files in the application.....	2
III.	Launch the application locally - Development part.....	2
I.	Launch the application	3
II.	Coding part	3

LINK: <https://github.com/Leamrn/CTFF>

- I. "Present Deployment and CI Strategy"
- I. 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.

- II. 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.

II. 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.
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.

III. Launch the application locally - Development part

I. 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

```
C:\Users\leame\Documents\CTF3>venv\Scripts\activate

(venv) C:\Users\leame\Documents\CTF3>docker-compose build
[+] Building 16.6s (15/15) FINISHED
=> [internal] load build definition from Dockerfile                                0.0s
=> => transferring dockerfile: 32B                                              0.0s
=> [internal] load .dockerignore                                                0.0s
=> => transferring context: 2B                                                  0.0s
=> [internal] load metadata for docker.io/library/python:3.9-slim-buster        1.5s
=> [internal] load build context                                                2.7s
=> => transferring context: 1.12MB                                             2.6s
=> [ 1/10] FROM docker.io/library/python:3.9-slim-buster@sha256:320a7a4250aba4249f458872adecf92eea88dc6abd2d76dc 0.0s
=> CACHED [ 2/10] WORKDIR /app                                                 0.0s
=> CACHED [ 3/10] COPY requirements.txt requirements.txt                       0.0s
=> CACHED [ 4/10] RUN pip install -r requirements.txt openpyxl                 0.0s
=> CACHED [ 5/10] RUN pip3 install --no-cache-dir -r requirements.txt openpyxl 0.0s
=> CACHED [ 6/10] RUN pip install flask_sqlalchemy                             0.0s
=> CACHED [ 7/10] RUN pip install pandas                                       0.0s
=> CACHED [ 8/10] RUN pip install matplotlib                                  0.0s
=> CACHED [ 9/10] COPY assets/css /assets/css                                 0.0s
=> [10/10] COPY . .                                                            9.9s
=> exporting to image                                                         2.2s
=> => exporting layers                                                         2.1s
=> => writing image sha256:18fa98d6258c67899de05053fb228f1476d638dd574fd043445bb11c87928f0a 0.0s
=> => naming to docker.io/library/ctf3-web                                    0.0s

(venv) C:\Users\leame\Documents\CTF3>docker-compose up
[+] Running 2/2
  ✓ Container ctf3-db-1   Created                                              0.0s
  ✓ Container ctf3-web-1  Recreated                                           0.3s
Attaching to ctf3-db-1, ctf3-web-1
ctf3-db-1 |
ctf3-db-1 | PostgreSQL Database directory appears to contain a database; Skipping initialization
ctf3-db-1 |
ctf3-db-1 | 2023-06-26 08:44:19.457 UTC [1] LOG:  starting PostgreSQL 13.10 (Debian 13.10-1.pgdg110+1) on x86_64-pc-linux-gn
2.1-6) 10.2.1
ctf3-db-1 | 2023-06-26 08:44:19.460 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
ctf3-db-1 | 2023-06-26 08:44:19.460 UTC [1] LOG:  listening on IPv6 address ":::", port 5432
ctf3-db-1 | 2023-06-26 08:44:19.466 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
ctf3-db-1 | 2023-06-26 08:44:19.480 UTC [27] LOG:  database system was shut down at 2023-06-24 10:47:15 UTC
```

Figure 1: First method to run the application locally

II. 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.

```

# Definition of the TreatmentChemotherapy class
class TreatmentChemotherapy(db.Model):
    __tablename__ = 'treatment_chemotherapy'
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(50), default="", unique=True)

# Definition of the Patient class
class Patient(db.Model):
    __tablename__ = 'patient'
    id = db.Column(db.Integer, primary_key=True)
    #personal info
    first_name = db.Column(db.String(50), default="")
    last_name = db.Column(db.String(50), default="")
    gender = db.Column(db.String(10), default="")
    date_of_birth = db.Column(db.Date, default=None)
    email = db.Column(db.String(100), default="")
    op_date_primary_tumor = db.Column(db.Date, default=None)
    age_at_OP = db.Column(db.String(10), default="")
    BMI = db.Column(db.String(10), default="")
    weight = db.Column(db.Float, default=None)
    height = db.Column(db.Float, default=None)
    #medical info
    pre_existing_conditions = db.Column(db.String(10), default="")
    cardiac_diseases = db.Column(db.String(10), default="")
    pulmonary_diseases = db.Column(db.String(10), default="")
    urological_diseases = db.Column(db.String(10), default="")
    endocrine_diseases = db.Column(db.String(10), default="")
    previous_vascular_diseases = db.Column(db.String(10), default="")
    #About the tumor
    tumor_side = db.Column(db.String(10), default="")
    tumor_marker_CEA = db.Column(db.String(50), default="")
    tumor_marker_CA19_9 = db.Column(db.String(50), default="")
    preoperative_endoscopy = db.Column(db.String(20), default="")
    surgical_procedure = db.Column(db.String(30), default="")
    localization_OP = db.Column(db.String(10), default="")
    T_stadium = db.Column(db.String(10), default="")
    N_stadium = db.Column(db.String(10), default="")
    UICC_Stadium = db.Column(db.String(10), default="")
    grading = db.Column(db.String(10), default="")
    lymphangiosis_carcinomatous = db.Column(db.String(10), default="")
    vascular_invasion = db.Column(db.String(10), default="")
    perineural_invasion = db.Column(db.String(10), default="")
    tumor_diameter_in_cm = db.Column(db.String(10), default="")
    #treatment
    progression_free_survival_in_months = db.Column(db.String(10), default="")
    treatments = db.relationship('TreatmentChemotherapy', secondary='patient_treatment', backref='patients')

```

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

They are connected to each other in this way:

```

# Definition of the intermediate association table
patient_treatment = db.Table('patient_treatment',
    db.Column('patient_id', db.Integer, db.ForeignKey('patient.id'), primary_key=True),
    db.Column('treatment_id', db.Integer, db.ForeignKey('treatment_chemotherapy.id'), primary_key=True)
)

```

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

The third table is Admin for administrators.

```

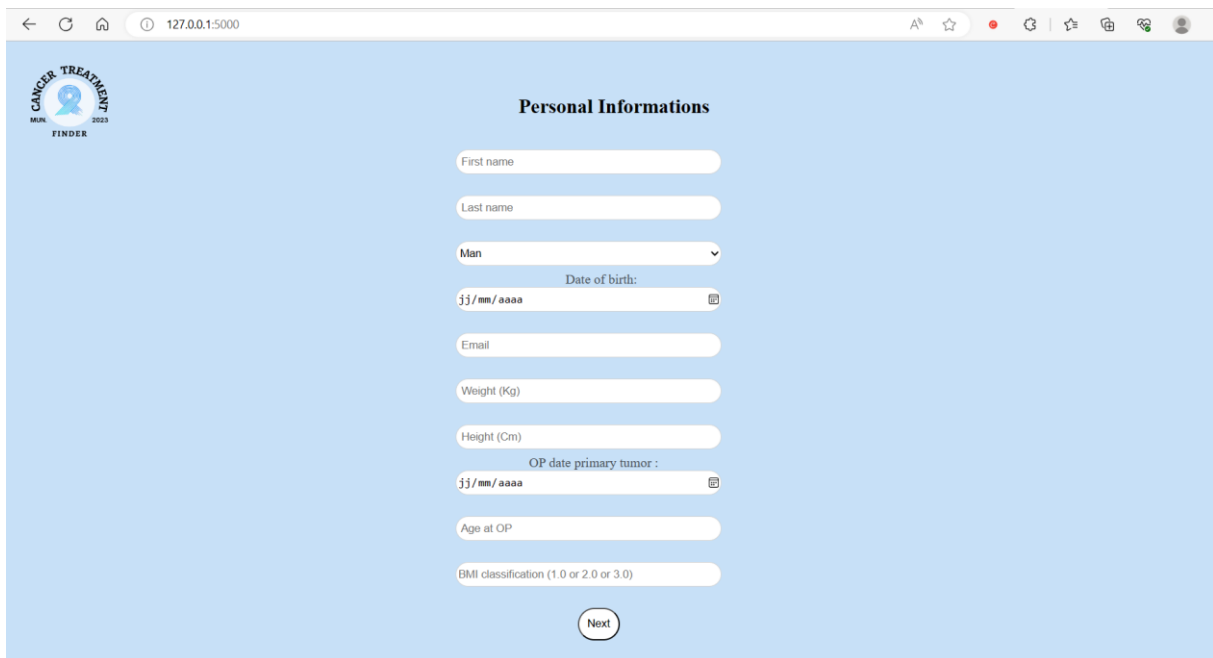
class Admin(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(50), unique=True, nullable=False, default='')
    password = db.Column(db.String(100), nullable=False, default='')
    is_superuser = db.Column(db.Boolean, nullable=False, default=True)

    def __repr__(self):
        return '<Admin %r>' % self.username

```

Figure 4: Admin class from the app.py file

2. The main page is index.



The screenshot shows a web browser window with the URL 127.0.0.1:5000. The page has a light blue background and a logo in the top left corner that reads 'CANCER TREATMENT FINDER' with 'MINI' and '2023' below it. The main heading is 'Personal Informations'. Below this, there are several input fields: 'First name', 'Last name', a gender dropdown menu currently showing 'Man', 'Date of birth:' with a date picker showing 'jj/mm/aaaa', 'Email', 'Weight (Kg)', 'Height (Cm)', 'OP date primary tumor :' with a date picker showing 'jj/mm/aaaa', 'Age at OP', and 'BMI classification (1.0 or 2.0 or 3.0)'. At the bottom of the form is a 'Next' button.

Figure 5: Page 1 of the main page

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

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

The function ***create()*** calls 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 created in the ***create()*** function)

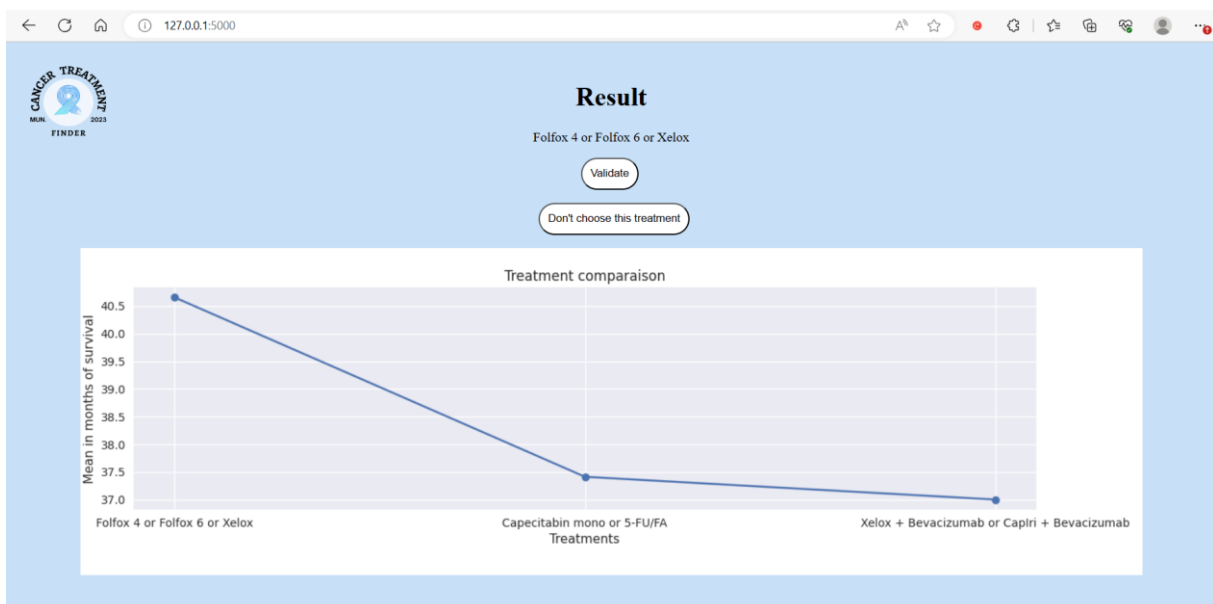


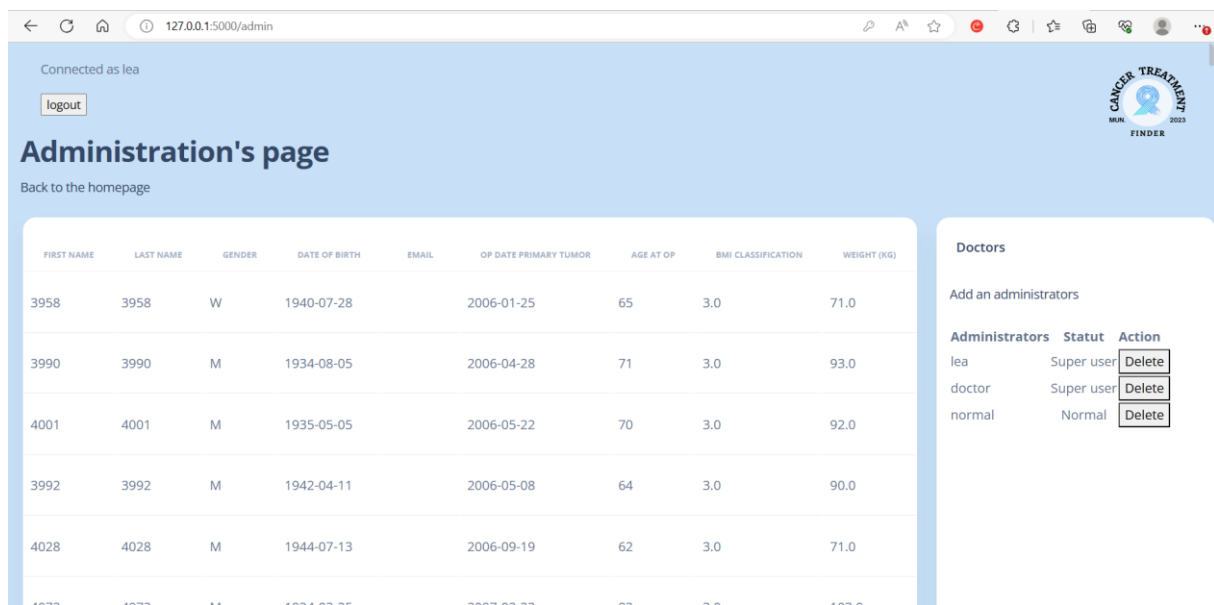
Figure 6: Last page of the main page: the result

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.

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



Connected as lea
logout

Administration's page
Back to the homepage

FIRST NAME	LAST NAME	GENDER	DATE OF BIRTH	EMAIL	OP DATE PRIMARY TUMOR	AGE AT OP	BMI CLASSIFICATION	WEIGHT (KG)
3958	3958	W	1940-07-28		2006-01-25	65	3.0	71.0
3990	3990	M	1934-08-05		2006-04-28	71	3.0	93.0
4001	4001	M	1935-05-05		2006-05-22	70	3.0	92.0
3992	3992	M	1942-04-11		2006-05-08	64	3.0	90.0
4028	4028	M	1944-07-13		2006-09-19	62	3.0	71.0
4073	4073	M	1974-02-25		2007-02-22	67	3.0	102.0

Doctors

Add an administrators

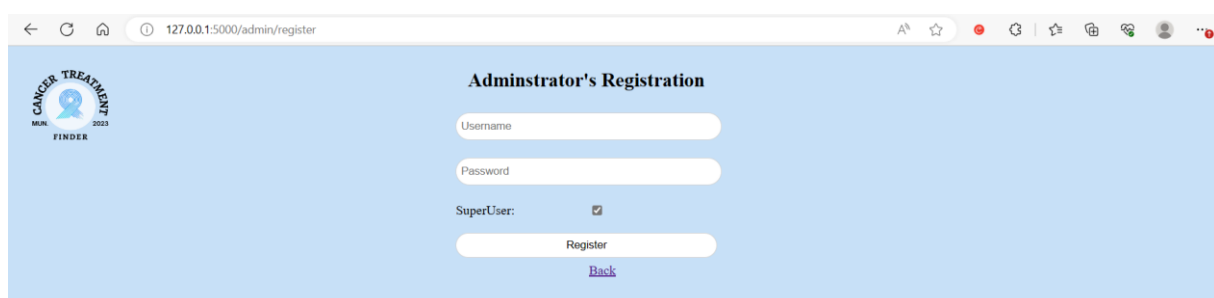
Administrators	Statut	Action
lea	Super user	Delete
doctor	Super user	Delete
normal	Normal	Delete

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*.

4. And only Superuser can add a new administrator by clicking on the button "Add an administrator".



Administrator's Registration

Username

Password

SuperUser: ☒

[Back](#)

Figure 8: The administrator's registration page

Here the page associated with the python function **register_admin()**.

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

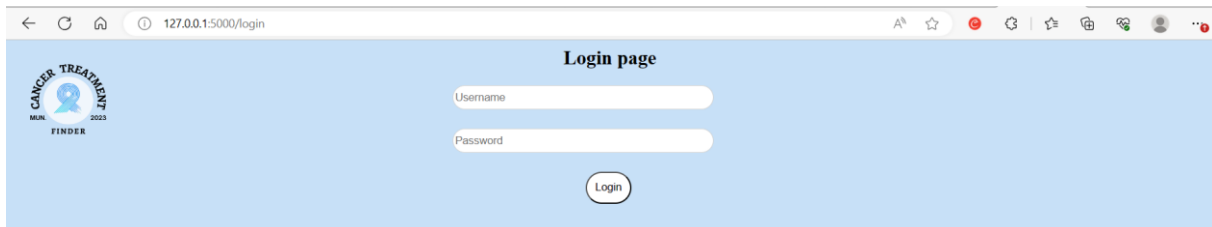


Figure 9: The login page

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

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

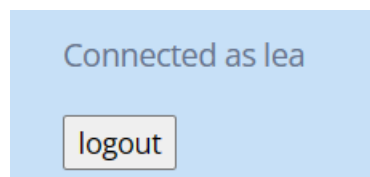


Figure 10: Logout button

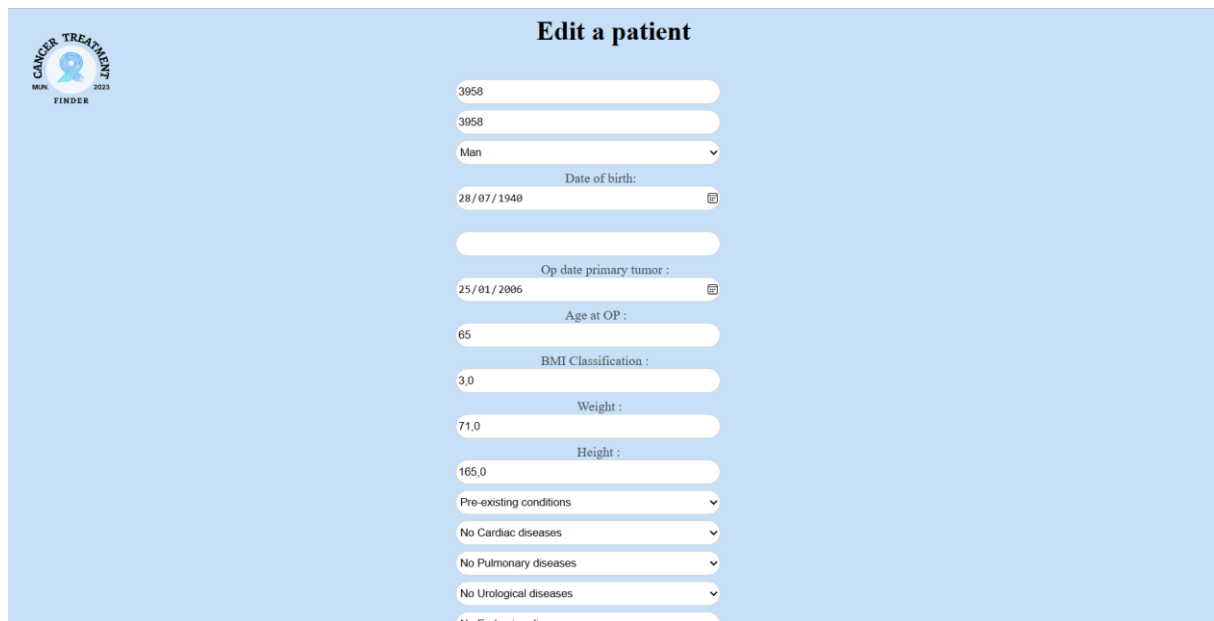
The button is associated to the function **logout()**.

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

Pn0	2.8	Xelox + Bevacizumab or Caplri + Bevacizumab	44	Delete
Pn0	5	Folfox 4 or Folfox 6 or Xelox	7	Edit Delete
Pn0	2.4	Xelox + Bevacizumab or Caplri + Bevacizumab	119	Edit Delete
Pn0	-	Folfox 4 or Folfox 6 or Xelox	-	Edit Delete

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).



The screenshot shows a web interface titled "Edit a patient" with a light blue background. In the top left corner is a logo for "CANCER TREATMENT MUNICH FINDER 2023". The form contains several input fields and dropdown menus for patient data:

- Two text input fields, both containing the value "3958".
- A dropdown menu for gender, currently set to "Man".
- A date input field labeled "Date of birth:" containing "28/07/1940".
- An empty text input field.
- A date input field labeled "Op date primary tumor :" containing "25/01/2006".
- A text input field labeled "Age at OP :" containing "65".
- A text input field labeled "BMI Classification :" containing "3,0".
- A text input field labeled "Weight :" containing "71,0".
- A text input field labeled "Height :" containing "165,0".
- A dropdown menu for "Pre-existing conditions".
- A dropdown menu for "No Cardiac diseases".
- A dropdown menu for "No Pulmonary diseases".
- A dropdown menu for "No Urological diseases".
- A partially visible dropdown menu for "No Endocrine diseases".

Figure 12: The edit page

8. 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()***.