

DEMOCRATIC AND POPULAR REPUBLIC OF ALGERIA
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH
University of Science and Technology Houari Boumediene
Faculty of Computer Science
Specialty Software Engineering



Project Report of SI Lab

Title:

Eclinic - Streamlining Healthcare Management Web app

Subject: Systeme D'information

Lab Teacher: Ms. BOULKABOUL Sahar

Presented By:

Name: DEBBAL Lotfi

Group: 2

Matricula: 212131096256

Name: Ahmed TEHAR

Group: 4

Matricula: 212137037897

In: 28/01/2024

Academic Year: 2023/2024

Contents

<i>Introduction</i>	5
<i>System Architecture</i>	5
Technologies Used.....	5
IDE.....	5
Frontend.....	5
Backend:	7
<i>Database Schema</i>	7
MCD	8
MLD:	10
<i>Graphical interface</i>	10
Presentation	10
Sign in	11
Dashboard	11
Homepage	12
User management	13
Add user.....	13
User list.....	14
Edit user.....	15
Patient management	15
Add Patient.....	15
Patient list	16
Edit patient	17
Medical history	18
Appointment management	19
Add appointment.....	19
Appointment list for a patient	20
Edit appointment.....	21
Procedure application management	23
Add procedure application.....	23
Prescription management.....	25
Edit prescription.....	25

Medical staff management	26
Add medical staff	26
Medical staff list.....	27
Edit medical staff	28
Appointment list for a medical staff.....	29
Department management	30
Add department.....	30
Department list.....	30
Edit department.....	32
Speciality management	32
Add specialty.....	32
Speciality list	33
Edit speciality	34
Procedure management.....	35
Add procedure	35
Procedure list.....	36
Edit procedure.....	37
<i>Code structure</i>	37
Folders tree	37
Root folder	37
Dashboard	37
Templates.....	38
Pages	39
Views	39
EClinic.....	40
<i>Code screenshots</i>	40
<i>General code</i>	40
<i>Deployment process</i>	49
E-Clinic deployment	49
Preparing the app for deployment.....	49
Static files	49
Environment variables	50
Production database.....	50

Deploying the app to a webhosting service.....	51
Publishing to github.....	51
Creating the postgres database	52
Deploying the application	53
Configuring environment variables.....	55
<i>Lessons Learned</i>	56
<i>Conclusion</i>	57

Introduction

The main objective of this project is to create a management system for a surgical clinic with Django based on MERISE architecture, in order to practice what we have learned in SI module (systeme d'information).

The Eclinic web app aims to provide a comprehensive solution for managing patient appointments, medical records, and department-specific tasks, ultimately improving the efficiency of healthcare processes.

System Architecture

The Eclinic architecture comprises a frontend built with HTML, CSS, and JavaScript, a Django backend, and an SQLite database. This structure ensures a scalable and maintainable system.

Technologies Used

IDE

VSCODE:

Visual Studio Code, also commonly referred to as VS Code, is a source-code editor developed by Microsoft for Windows, Linux and macOS. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git.



Frontend

The frontend is implemented with HTML, CSS, and JavaScript. Tailwind CSS is used for styling, and DaisyUI components enhance the user interface. Chart.js is integrated to provide dynamic and interactive charts.

1. HTML:

Hypertext Markup Language (HTML) is the set of markup symbols or codes inserted into a file intended for display on the Internet. The markup tells web browsers how to display a web page's words and images.



2. CSS:

CSS (Cascading Style Sheets) is used to style and layout web pages — for example, to alter the font, color, size, and spacing of your content, split it into multiple columns, or add animations and other decorative features.



3. JavaScript

JavaScript is a scripting language that enables you to create dynamically updating content, control multimedia, animate images, and pretty much everything else.



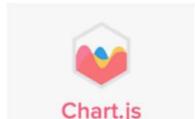
4. Tailwind CSS

Tailwind CSS is basically a utility-first CSS framework for rapidly building custom user interfaces. It is a highly customizable, low-level CSS framework that gives you all of the building blocks you need to build bespoke designs without any annoying opinionated styles you have to fight to override.



5. Chart.js

Chart.js is a free JavaScript library for making HTML-based charts. It is one of the simplest visualization libraries for JavaScript, and comes with the following built-in chart types: Scatter Plot. Line Chart.



6. DaisyUI :

daisyUI is a component templates library built on top of TailwindCSS. It provides us with short semantic classes composed from TailwindCSS utilities and a growing collection of convenient component templates that helps quickly build React components for our app.



Backend:

The backend is designed with Django, employing the Model-View-Controller (MVC) architecture.

1. Python (Django framework)

Django is utilized for backend development, ensuring efficient data handling and processing. Python scripts define models representing the database schema, views handle user requests, and forms manage data input and validation.



2. SQLite (development)

The SQLite database is employed to store patient information, appointments, and medical records. Django migrations are utilized to create and update the database schema.



We have used SQLite for creating tables of our database schema

3. PostgreSQL (deployment)

PostgreSQL is a powerful, open-source object-relational database system that uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads.

We used this SGDB for deploying our app as SQLite do not have this feature.

Database Schema

The database schema is well-structured to support patient information, appointments, medical records, and department-specific data.

The figure below shows our database schema design with MCD (Modele conceptual des donnees) diagram.

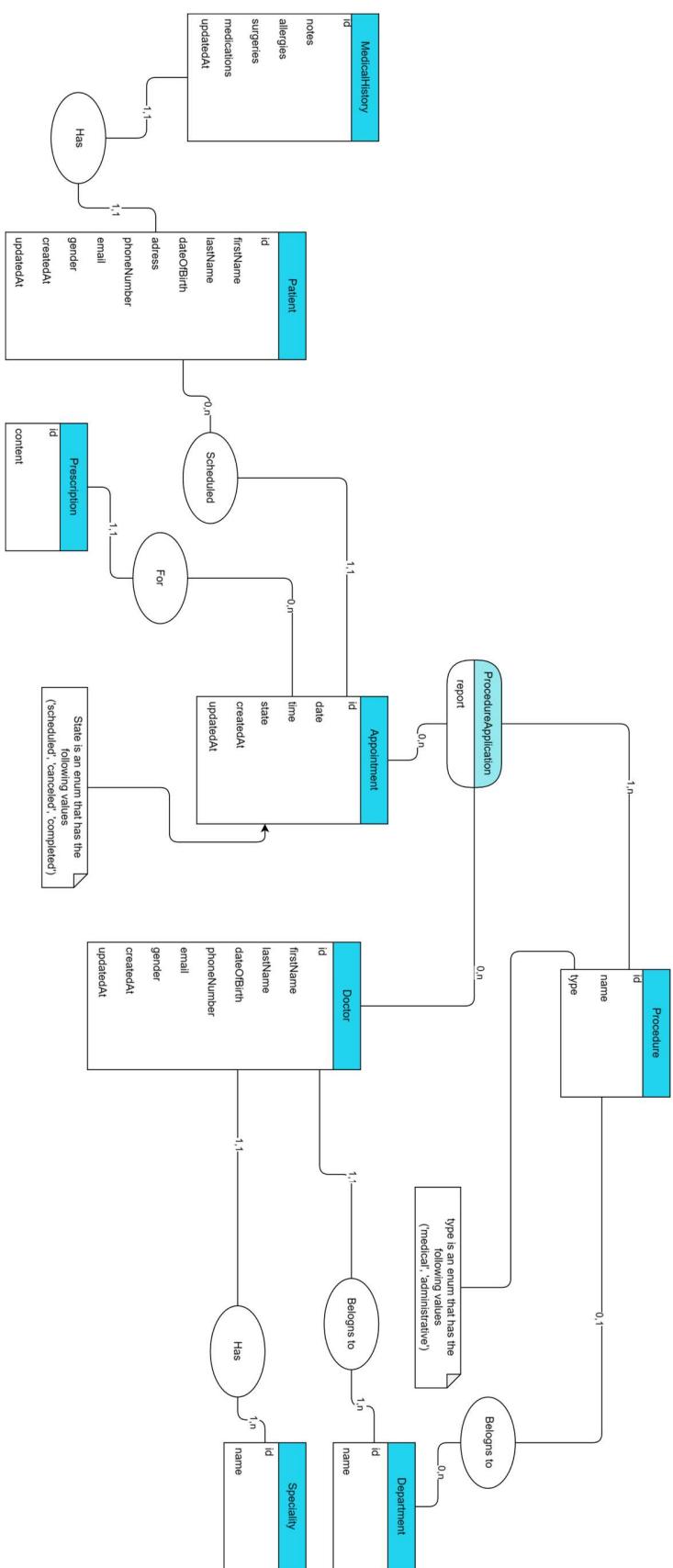
We used **MERISE** Method to design our database.

MERISE is a general-purpose modeling methodology in the field of information systems development, software engineering and project management. First introduced in the early 1980s, it was widely used in France, and was developed and refined to the point where most large French governmental, commercial, and industrial organizations had adopted it as their standard methodology.

Merise proceeds to separate treatment of data and processes, where the data-oriented view is modelled in three stages, from conceptual (**MCD**), logical (**MLD**)through to physical. Similarly, the process-oriented view passes through the three stages of conceptual, organizational and operational. These stages in the modelling process are paralleled by the stages of the life cycle: strategic planning, preliminary study, detailed study, development, implementation and maintenance. It is a method of analysis based on the entity-relationship model. By using Merise, you can design tables with relations to make a relational database.

Below we can see the MCD diagram and MLD model of our application

MCD



MLD:

Patient(**id**, firstName, lastName, dateOfBirth, address, phoneNumber, email, gender, createdAt, updatedAt)

MedicalStaff(**id**, *department**, *speciality**, firstName, lastName, dateOfBirth, address, phoneNumber, email, gender, createdAt, updatedAt)

MedicalHistory(**id**, *patient**, notes, allergies, surgeries, medications, updatedAt)

Department(**id**, name)

Speciality(**id**, name)

Procedure(**id**, name, type)

ProcedureApplication(**id**, *appointment**, *medicalStaff**, *procedure**, report)

Appointment(**id**, *patient**, timestamp, state, createdAt, updatedAt)

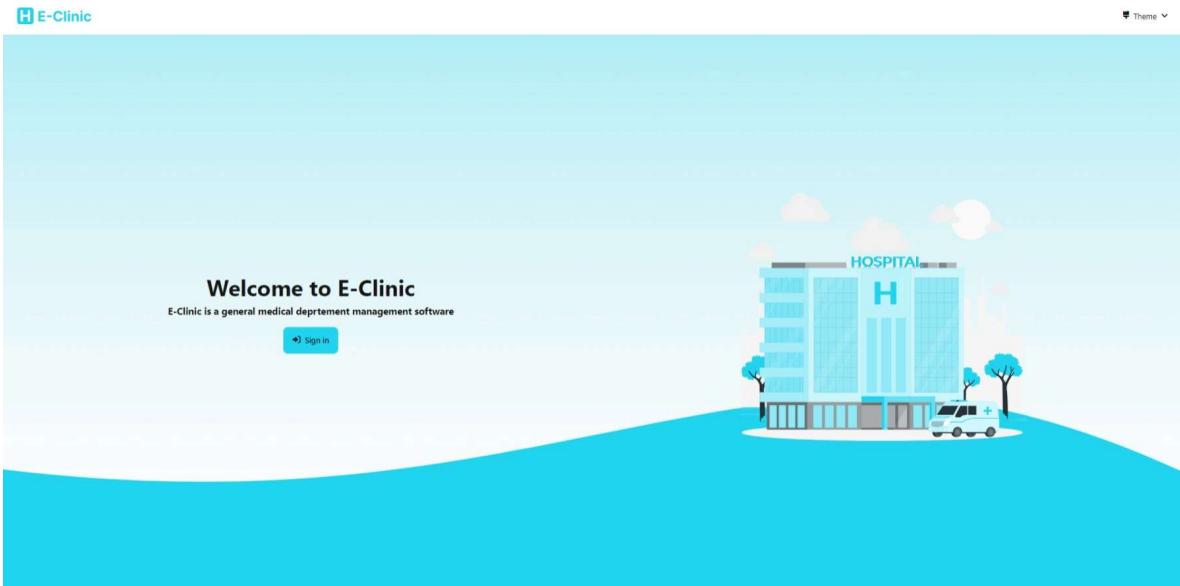
Prescription(**id**, *appointment**, content)

Graphical interface

In this project we created an accessible UI that makes using the management system easier

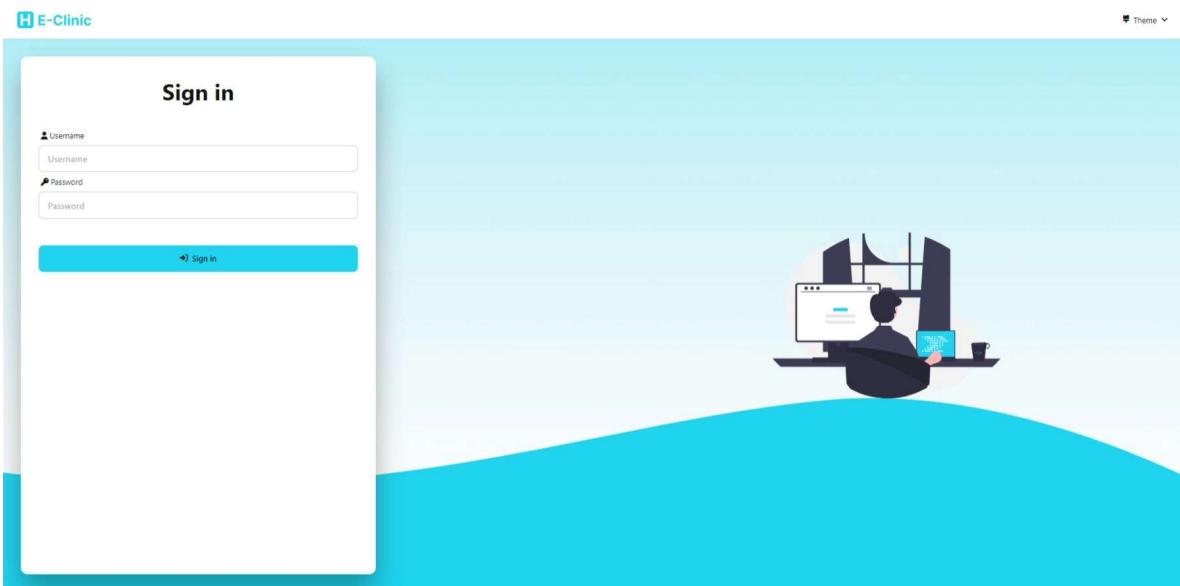
Presentation

This page describes what the app is about



Sign in

This page allows an administrator to sign in into the system

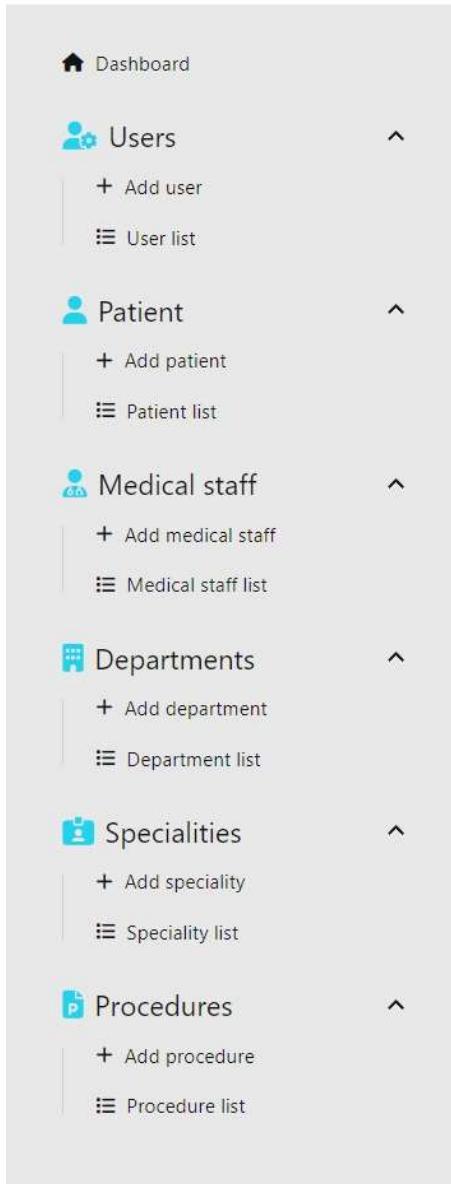


Dashboard

The dashboard is the place where admin manages the system and it has two navigation elements

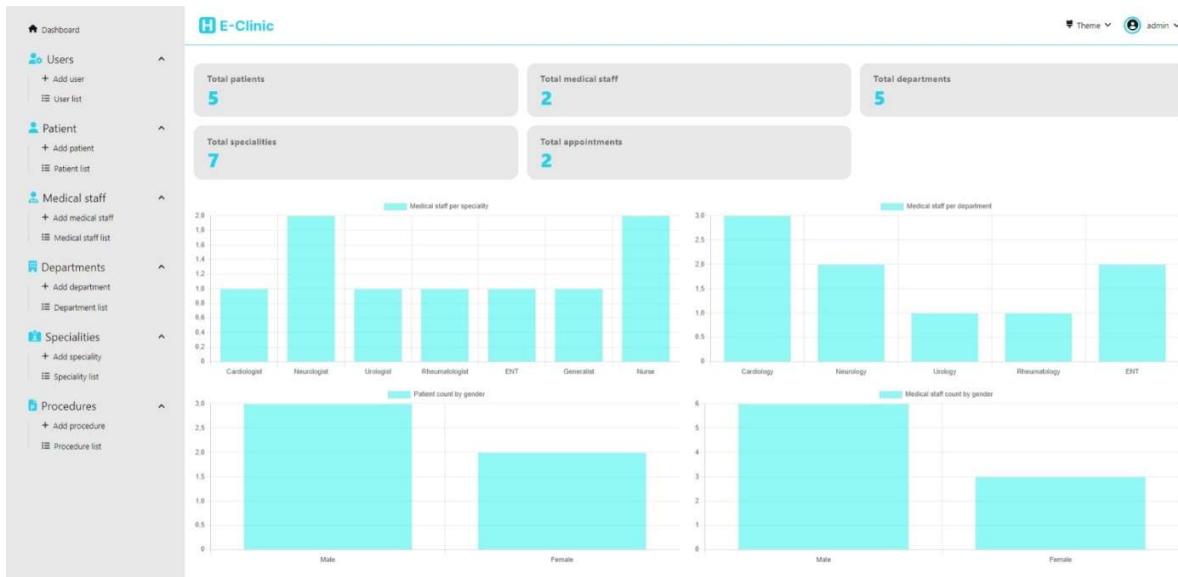
- The navbar shown the current user and allow us to sign out. Additionally we can also change the theme of the app

- The sidebar shown static navigation links (static means that it does not have a parameter passed in the URL)



Homepage

This page is the homepage of the system. It shows various statistics about every entity and some graphs.



User management

We have multiple pages to allow the user to manage the users that can access our system

Add user

This page allow us to add a new user to the system

The 'Add user' form fields include:

- Username:
- Password:
- Confirm password:

A large blue button at the bottom right says "+ Add user".

User list

This page allow us to view the list of all users and also allows us to delete a user and navigate to the page to edit a user and also search for users

The screenshot shows the 'User list' page of the E-Clinic application. The left sidebar contains navigation links for Dashboard, Users, Patient, Medical staff, Departments, Specialties, and Procedures. The 'Users' section is expanded, showing '+ Add user' and 'User list'. The 'User list' link is highlighted with a blue background. The main content area is titled 'User list' and features a search bar with placeholder 'Search users' and a magnifying glass icon. Below the search bar is a table with columns: Id, Username, Email, First Name, Last Name, Active, Staff, Superuser, and Date Joined. A single row is visible for the user 'admin'. To the right of the row, there is a context menu with options 'Edit' (with a pencil icon) and 'Delete' (with a trash bin icon). The top right corner of the page shows 'Theme' and 'admin'.

The screenshot shows the 'User list' page with a modal dialog box in the foreground. The dialog is titled 'Delete confirmation' and contains the message 'Are you sure you want to delete user "admin"?' Below the message are two buttons: 'Delete' (in red) and 'Close' (in grey). The background of the page is dimmed, and the user 'admin' is still listed in the table. The rest of the interface, including the sidebar and top navigation, remains the same as the first screenshot.

Edit user

(accessible from user list)

This page allow us to edit a user

The screenshot shows the 'Edit user' interface in the E-Clinic application. The left sidebar contains a navigation menu with links for Dashboard, Users (+ Add user, User list), Patient (+ Add patient, Patient list), Medical staff (+ Add medical staff, Medical staff list), Departments (+ Add department, Department list), Specialities (+ Add specialty, Speciality list), and Procedures (+ Add procedure, Procedure list). The main content area is titled '+ Edit user'. It features three checked checkboxes: 'Is superuser', 'Is staff', and 'Is active'. A 'Groups' section is empty. Below this is a 'Permissions' section containing several checkboxes for appointment-related permissions. Further down are input fields for 'Username' (admin), 'First name', 'Last name', and 'Email' (admin@admin.com). A 'Date joined' field is also present. At the bottom is a blue button labeled '+ Edit user'.

Patient management

We have multiple pages to allow the user to manage the patients in the system

Add Patient

This page allow us to add a new patient to the system

Add patient

First name	<input type="text"/>
Last name	<input type="text"/>
Date of birth	<input type="text"/> jj/mm/aaaa
Address	<input type="text"/>
Phone number	<input type="text"/>
E-mail	<input type="text"/>
Gender	<input type="button" value="-----"/>

+ Add patient

Patient list

This page allow us to view the list of all patient and search the list. It also allows us to delete a patient and navigate to the page to edit a patient, view patient history, view appointments for a patient, and add an appointment

Patient list

Id	First name	Last name	Date of birth	Address	Phone number	Email	Gender	Created at
be93aa51-a041-45a5-903f-616062315	patient1	patient1	June 18, 2004	Address of patient1	0500000001	patient1@patient1.com	MALE	Jan. 28, 2024, 9:52 a.m.
df8d9983-b864-4e85-9148-500fb96d7d5e	patient2	patient2	June 18, 2004	Address of patient2	0500000002	patient2@patient2.com	FEMALE	Jan. 28, 2024, 9:52 a.m.
4a210eb4bfc2	patient3	patient3	June 18, 2004	Address of patient3	0500000003	patient3@patient3.com	MALE	Jan. 28, 2024, 9:52 a.m.
4a5fa892-2e38-44cb-b90a-69dd95a5ab9	patient4	patient4	June 18, 2004	Address of patient4	0500000004	patient4@patient4.com	FEMALE	Jan. 28, 2024, 9:52 a.m.
4a5fa892-2e38-44cb-b90a-69dd95a5ab9	patient5	patient5	June 18, 2004	Address of patient5	0500000005	patient5@patient5.com	MALE	Jan. 28, 2024, 9:52 a.m.

The screenshot shows the E-Clinic application interface. On the left is a sidebar with navigation links for Dashboard, Users, Patient, Medical staff, Departments, Specialities, and Procedures. The 'Patient' link is currently selected. The main content area is titled 'Patient list'. It features a search bar and a table with columns: Id, First name, Last name, Date of birth, Address, Phone number, Email, Gender, and Created at. A modal dialog box titled 'Delete confirmation' is displayed in the center, asking 'Are you sure you want to delete patient "patient1"?' with 'Delete' and 'Close' buttons. The table data includes five rows of patient information.

Id	First name	Last name	Date of birth	Address	Phone number	Email	Gender	Created at
be5a0d906a	patient1	patient1	June 18, 2004	Address of patient1	0500000001	patient1@patient1.com	MALE	Jan. 28, 2024, 9:52 a.m.
2d4ea1ac2e	patient2	patient2	June 18, 2004	Address of patient2	0500000002	patient2@patient2.com	FEMALE	Jan. 28, 2024, 9:52 a.m.
fcf4a3da-db58-4b15-a80e-4a210eb4bf2	patient3	patient3	June 18, 2004	Address of patient3	0500000003	patient3@patient3.com	MALE	Jan. 28, 2024, 9:52 a.m.
df8d9983-b864-4e85-9148-500fb95d7d5e	patient4	patient4	June 18, 2004	Address of patient4	0500000004	patient4@patient4.com	FEMALE	Jan. 28, 2024, 9:52 a.m.
4a5fa92-2e38-44cb-b90a-690d995a5ab9	patient5	patient5	June 18, 2004	Address of patient5	0500000005	patient5@patient5.com	MALE	Jan. 28, 2024, 9:52 a.m.

Edit patient

(accessible from patient list)

This page allow us to edit a patient

The screenshot shows the 'Edit patient' page. The sidebar on the left is identical to the previous screenshot. The main content area is titled 'Edit patient' with a subtitle 'Back to patient list'. It contains form fields for editing patient1's information: First name (patient1), Last name (patient1), Date of birth (18/06/2004), Address (Address of patient1), Phone number (0500000001), E-mail (patient1@patient1.com), and Gender (Male). At the bottom is a blue button labeled '+ Update patient'.

Medical history

(accessible from patient list)

This page allow us to edit the medical history of patient

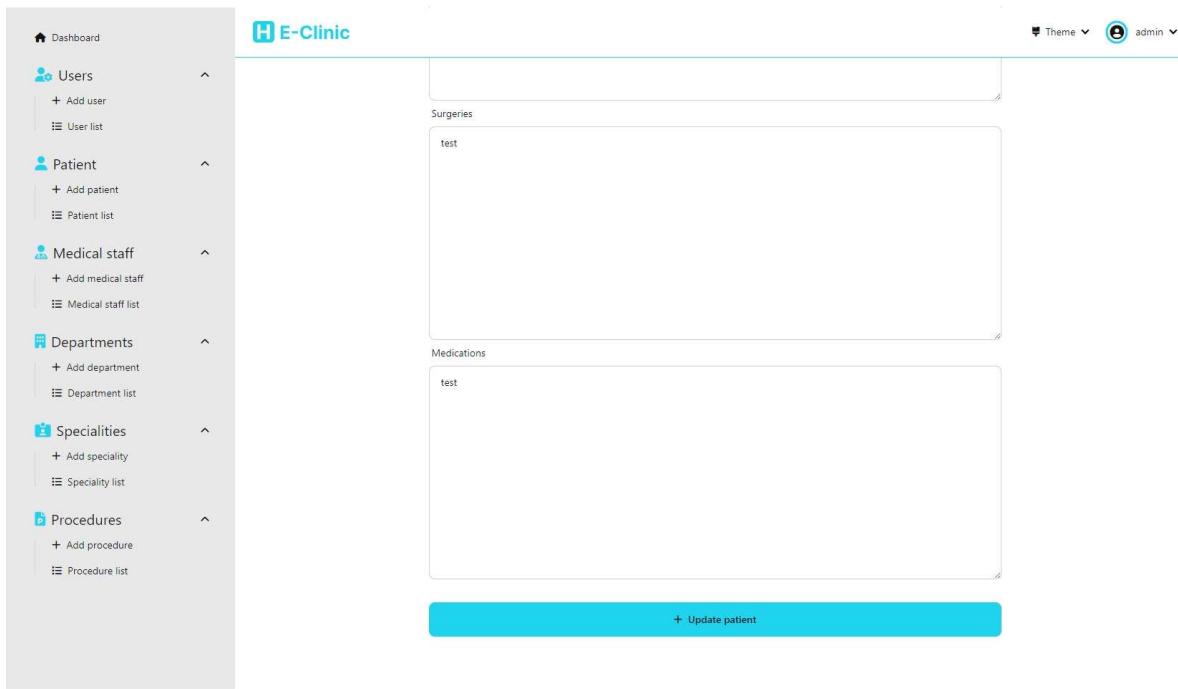
The screenshot shows the E-Clinic application interface. On the left is a sidebar with a navigation menu:

- Dashboard
- Users
 - + Add user
 - User list
- Patient
 - + Add patient
 - Patient list
- Medical staff
 - + Add medical staff
 - Medical staff list
- Departments
 - + Add department
 - Department list
- Specialties
 - + Add specialty
 - Specialty list
- Procedures
 - + Add procedure
 - Procedure list

The main content area is titled "Edit medical history" and shows the following fields:

- Notes**: A text area containing "test".
- Allergies**: A text area containing "test".
- Summaries**: A section at the bottom.

At the top right, there are "Theme" and "admin" dropdown menus.



Appointment management

We have multiple pages to allow the user to manage the appointments of the patients

Add appointment

(accessible from patient list)

This page allow us to add an appointment for a patient (accessible via the menu shown before)

Appointment list for a patient

(accessible from patient list)

This page allow us to view the list of all appointments for a given patient and search appointments. It also allows us to delete an appointment and navigate to the page to edit an appointment

ID	Timestamp	State
ead32697-e2bb-4751-9063-28b0fb9da7b	Jan. 29, 2024, 10:55 a.m.	SCHEDULED
e3624907-1aa3-4c58-a44d-e469c10fa109	Jan. 25, 2024, 10:57 a.m.	COMPLETED

Edit appointment

(accessible from appointment list)

This page allows us to edit an appointment for a given patient

It is composed of 3 part:

- The first part allows us to edit the general information about the appointment

The screenshot shows the E-Clinic application interface. On the left, there is a sidebar with the following navigation items:

- Dashboard
- Users
 - + Add user
 - User list
- Patient
 - + Add patient
 - Patient list
- Medical staff
 - + Add medical staff
 - Medical staff list
- Departments
 - + Add department
 - Department list

The main content area is titled "Edit appointment" and shows the following details:

- Patient: patient1 patient1
- Date & time: 25/01/2024 10:57
- State: Completed

A blue button at the bottom right says "+ Edit appointment".

- The second part allows us to view the list of medical procedures applied in that appointment and which medical staff is responsible for each one of them, It also allows us to delete a procedure application or add

E-Clinic

Theme admin

Applied procedures

Heart surgery

ID	Department	Specialty	First name	Last name	Date of birth
a9685bec-a42d-40ba-873b-aae2b45ba0afb1	Cardiology	Cardiologist	medicalstaff1	medicalstaff1	June 18, 2004
3924-49c2-aacb-ebb45ba0afb1	Cardiology	Nurse	medicalstaff6	medicalstaff6	June 18, 2004

Blood test

ID	Department	Specialty	First name	Last name	Date of birth
3924-49c2-aacb-ebb45ba0afb1	Cardiology	Nurse	medicalstaff6	medicalstaff6	June 18, 2004

Medical staff

ID	Department	Specialty	First name	Last name	Date of birth
a9685bec-a42d-40ba-873b-aae2b45ba0afb1	Cardiology	Cardiologist	medicalstaff1	medicalstaff1	June 18, 2004

Delete confirmation

Are you sure you want the procedure application "Heart surgery" done by "medicalstaff1 medicalstaff1"

Delete Close

Medical staff

ID	Department	Specialty	First name	Last name	Date of birth
3924-49c2-aacb-ebb45ba0afb1	Cardiology	Nurse	medicalstaff6	medicalstaff6	June 18, 2004

+ Add prescription

Procedure application management

Add procedure application

(accessible from edit appointment)

This page allows us to apply a procedure to a given appointment and assign the medical staff and the procedure that will be applied

- First we have the list of available medical staff, we can select only one at a time, We can also search for medical staff

ID	Department	Specialty	First name	Last name	Date of birth
a98b5bec-a42c-4dbd-873b-ae26bb0a57696	Cardiology	Cardiologist	medicalstaff1	medicalstaff1	June 1 2004
9a3e490e-1476-48b4-919c-91964cc8d1a2	Neurology	Neurologist	medicalstaff2	medicalstaff2	June 1 2004
700b94e3-1704-47a1-a30f-5cae8f72889b	Urology	Urologist	medicalstaff3	medicalstaff3	June 1 2004
0fedb666-1b2a-49c5-bfec-8d7d3e9fbe11	Rheumatology	Rheumatologist	medicalstaff4	medicalstaff4	June 1 2004

Once we select a medical staff, we need to chose the procedure to apply

ID	Department	Specialty	First name	Last name	Date of birth
a9885bec-a2d4-dba-873b-ae26b0a57696	Cardiology	Cardiologist	medicalstaff1	medicalstaff1	June 18, 2004

- The third part allows us to view the list of prescriptions given for that appointment, It also allows us to delete a prescription and navigate to the page that allows us to edit a prescription

ID	Content
3c936fc5-3924-49c2-aacb-ebb45ba0afb1	this is a test prescription medication 1 medication 2 medication 3 medication 4 medication 5
9e5b7c59-e1f0-437e-9e02-1f9eddd0f01d	this is a test prescription medication 1 medication 2 medication 3 medication 4 medication 5
89c5d149-adfd-4e38-8dff-ccaeef497e2a5	this is a test prescription medication 1 medication 2 medication 3 medication 4 medication 5

The screenshot shows the E-Clinic application interface. On the left is a sidebar with navigation links: Dashboard, Users (+ Add user, User list), Patient (+ Add patient, Patient list), Medical staff (+ Add medical staff, Medical staff list), Departments (+ Add department, Department list), Specialties (+ Add specialty, Specialty list), and Procedures (+ Add procedure, Procedure list). The main area displays a "Medical staff" table with one row of data:

ID	Department	Specialty	First name	Last name	Date of birth
3c936fc5-3924-492e-aac6-ebb45ba0afb1	Cardiology	Nurse	medicalstaff6	medicalstaff6	June 18, 2004

A modal window titled "Delete confirmation" asks if you want to delete a prescription from Jan. 25, 2024, 10:57 a.m. It contains three rows of prescription data:

ID	Content
9e5b7c59-e1f0-437e-9e02-1f9eddd0f01d	this is a test prescription medication 1 medication 2 medication 3 medication 4 medication 5
6c50a397-9b76-4059-962a-2eb3407e1731	this is a test prescription medication 1 medication 2 medication 3 medication 4 medication 5
89c5d149-add4-e38-8dff-ccae497e2a5	this is a test prescription medication 1 medication 2 medication 3 medication 4 medication 5

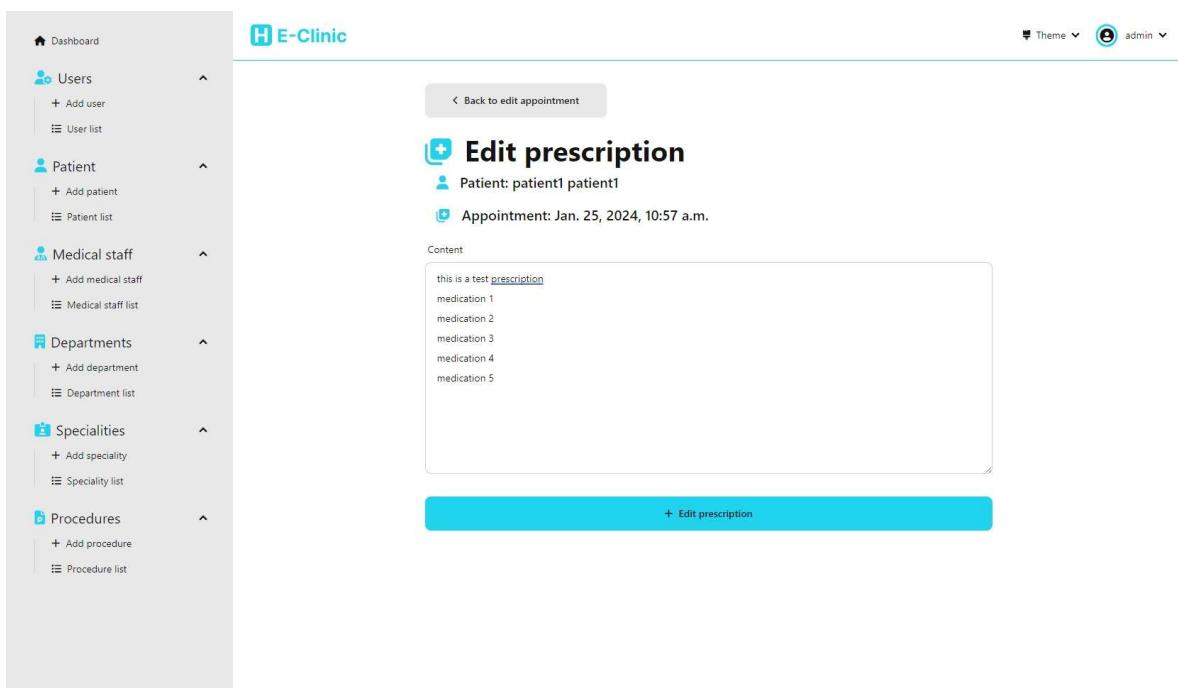
Buttons in the modal are "Delete" (red) and "Close".

Prescription management

Edit prescription

(accessible from edit appointment)

This page allows us to edit a prescription for a given appointment



Medical staff management

We have multiple pages to allow the user to manage the medical staff in our system

Add medical staff

This page allows us to add a new medical staff to the system

E-Clinic

Dashboard

- Users
 - + Add user
 - ≡ User list
- Patient
 - + Add patient
 - ≡ Patient list
- Medical staff
 - + Add medical staff
 - ≡ Medical staff list
- Departments
 - + Add department
 - ≡ Department list
- Specialities
 - + Add speciality
 - ≡ Speciality list
- Procedures
 - + Add procedure
 - ≡ Procedure list

Back to dashboard

Add medical staff

Department

Speciality

First name

Last name

Date of birth

Address

Phone number

E-mail

Gender

+ Add medical staff

Medical staff list

This page allows us to view the list of all medical staff in the system. It also allows us to delete a medical staff and also navigate to the page that allows us to edit a medical staff / appointment list for a given medical staff

ID	Department	Specialty	First name	Last name	Date of birth	Address	Phone number	Email
a9885bec-a42d-4cb4-873b-ae29b0b57569	Cardiology	Cardiologist	medicalstaff1	medicalstaff1	June 18, 2004	Address of medicalstaff1	0500000001	medicalstaff1@medicalstaff.com
700b94e3-1704-47a1-a30f-5cae87f2889b	Neurology	Neurologist	medicalstaff2	medicalstaff2	June 18, 2004	Address of medicalstaff2	0500000002	medicalstaff2@medicalstaff.com
0fedaa56-1b2a-49c6-bfec-8d7d3e9fb611	Rheumatology	Rheumatologist	medicalstaff4	medicalstaff4	June 18, 2004	Address of medicalstaff4	0500000004	medicalstaff4@medicalstaff.com
c798c87-633d-48b0-9667-173dcfb611	ENT	ENT	medicalstaff5	medicalstaff5	June 18, 2004	Address of medicalstaff5	0500000005	medicalstaff5@medicalstaff.com
3d3af65-3934-49c2-aacb-ebb45ba0afb1	Cardiology	Nurse	medicalstaff6	medicalstaff6	June 18, 2004	Address of medicalstaff6	0500000006	medicalstaff6@medicalstaff.com

Delete confirmation

Are you sure you want to delete medical staff "medicalstaff1" medicalstaff1?

Delete **Close**

ID	Department	Specialty	First name	Last name	Date of birth	Address	Phone number	Email
a9885bec-a42d-4cb4-873b-ae29b0b57569	Cardiology	Cardiologist	medicalstaff1	medicalstaff1	June 18, 2004	Address of medicalstaff1	0500000001	medicalstaff1@medicalstaff.com
9a3a409e-147c-48b4-919c-9fb64ccdd1a2					June 18, 2004	Address of medicalstaff2	0500000002	medicalstaff2@medicalstaff.com
700b94e3-1704-47a1-a30f-5cae87f2889b					June 18, 2004	Address of medicalstaff3	0500000003	medicalstaff3@medicalstaff.com
0fedaa56-1b2a-49c6-bfec-8d7d3e9fb611	Rheumatology	Rheumatologist	medicalstaff4	medicalstaff4	June 18, 2004	Address of medicalstaff4	0500000004	medicalstaff4@medicalstaff.com
c798c87-633d-48b0-9667-173dcfb611	ENT	ENT	medicalstaff5	medicalstaff5	June 18, 2004	Address of medicalstaff5	0500000005	medicalstaff5@medicalstaff.com
3d3af65-3934-49c2-aacb-ebb45ba0afb1	Cardiology	Nurse	medicalstaff6	medicalstaff6	June 18, 2004	Address of medicalstaff6	0500000006	medicalstaff6@medicalstaff.com
b1d2c4a1-b701-44d0-9334-49c2-aacb-ebb45ba0afb1	Cardiology	Generalist	medicalstaff7	medicalstaff7	June 18, 2004	Address of medicalstaff7	0500000007	medicalstaff7@medicalstaff.com

Edit medical staff

(accessible from medical staff list)

This page allows us to edit a given medical staff

Edit medical staff

Department: Cardiology

Speciality: Cardiologist

First name: medicalstaff1

Last name: medicalstaff1

Date of birth: 18/06/2004

Address: Address of medicalstaff1

Phone number: 0500000001

E-mail: medicalstaff1@medicalstaff1.com

Gender: Male

+ Update patient

Appointment list for a medical staff

(accessible from medical staff list)

This page allow us to view the list of all appointments for a given medical staff and search appointments. It also allows us to delete an appointment and navigate to the page to edit an appointment

The screenshot shows the E-Clinic application interface. On the left is a sidebar with navigation links: Dashboard, Users (with sub-links for Add user and User list), Patient (with sub-links for Add patient and Patient list), Medical staff (with sub-links for Add medical staff and Medical staff list), Departments (with sub-links for Add department and Department list), Specialties (with sub-links for Add specialty and Specialty list), and Procedures (with sub-links for Add procedure and Procedure list). The main content area is titled "Appointment list" and shows a table with two rows of appointment details. The columns are Id, Timestamp, and State. The first row has an Id of "ead32697-e2bb-4751-9863-28b0fb9da7b", a timestamp of "Jan 29, 2024, 10:55 a.m.", and a state of "SCHEDULED". The second row has an Id of "e3624907-1aa3-4c58-a44d-e469c10fa109", a timestamp of "Jan 25, 2024, 10:57 a.m.", and a state of "COMPLETED". A "Back to medical staff list" link is visible above the table.

Department management

We have multiple pages to allow the user to manage the departments in our system

Add department

This page allows us to add a new medical department to the system

The screenshot shows the E-Clinic application interface. The sidebar is identical to the previous screenshot. The main content area is titled "+ Add department" and contains a single input field labeled "Name" with the placeholder "Name". Below the input field is a blue button labeled "+ Add department". A "Back to dashboard" link is visible above the input field.

Department list

This page allows us to view the list of all departments in the system. It also allows us to delete a department and also navigate to the page that allows us to edit a department.

The screenshot shows the E-Clinic application interface. On the left is a sidebar with navigation links: Dashboard, Users, Patient, Medical staff, Departments, Specialties, and Procedures. The 'Departments' link is currently selected, highlighted with a blue background. The main content area is titled 'Department list'. It features a search bar at the top with placeholder text 'Search departments' and a magnifying glass icon. Below the search bar is a table with two columns: 'Id' and 'Name'. The table contains five rows of data:

Id	Name
dcf0a152-f61f-45ac-8d07-fa0a2cd2fc95	Cardiology
eb6b0-a8bf-4ee9-ab53-3f3899168d3c	Neurology
38339d99-75fd-48ab-b2d7-99b5ef8221de	Urology
9890ab88-39b0-4ad3-8555-4c8ba3c150ca	Rheumatology
e51ae6d5-0e3a-42a2-86ff-91d8bca08df1	ENT

This screenshot shows the same E-Clinic application interface as the previous one, but with a modal dialog box overlaid on the 'Department list' page. The dialog is titled 'Delete confirmation' and contains the message 'Are you sure you want to delete department "Cardiology"?' At the bottom right of the dialog are two buttons: a red 'Delete' button and a grey 'Close' button. The background of the main content area is dimmed to indicate that interaction with the underlying elements is disabled while the dialog is open.

Edit department

(accessible from department list)

This page allows us to edit a given department

The screenshot shows the E-Clinic software interface. On the left is a sidebar with the following navigation items:

- Dashboard
- Users
 - + Add user
 - User list
- Patient
 - + Add patient
 - Patient list
- Medical staff
 - + Add medical staff
 - Medical staff list
- Departments
 - + Add department
 - Department list
- Specialties
 - + Add specialty
 - Specialty list
- Procedures
 - + Add procedure
 - Procedure list

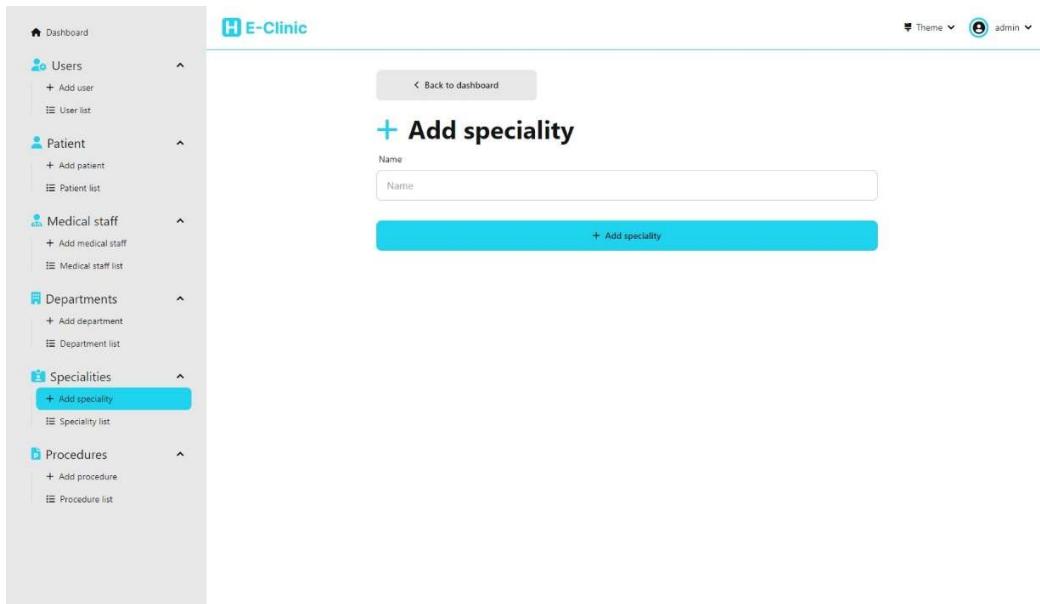
The main content area is titled "Edit department". It features a "Name" input field containing "Cardiology" and a blue "Edit department" button.

Speciality management

We have multiple pages to allow the user to manage the medical staff specialties in our system

Add specialty

This page allows us to add a new specialty to the system



Speciality list

This page allows us to view the list of all departments in the system. It also allows us to delete a speciality and also navigate to the page that allows us to edit a speciality.

Speciality list		
	ID	Name
...	697069d5-0813-4b89-adb5-387a683f9bd5	Cardiologist
Edit Delete	16d2-1062-4635-877c-4414c3924a93	Neurologist
...	70ea5f80-69b2-449c-ad01-911094281f8d	Urologist
...	e0fc52f-b71e-41bb-968e-05c4ec8bcea8	Rheumatologist
...	5f44880f-c1e0-4353-a832-86af9ea68e44	ENT
...	44d7cbf2-7970-499a-b959-3651d953d1f8	Generalist
...	6949c071-0548-413d-a78e-188845c14a6a	Nurse

The screenshot shows the E-Clinic application interface. On the left is a sidebar with navigation links: Dashboard, Users (+ Add user, User list), Patient (+ Add patient, Patient list), Medical staff (+ Add medical staff, Medical staff list), Departments (+ Add department, Department list), Specialties (+ Add specialty, Specialty list - highlighted in blue), and Procedures (+ Add procedure, Procedure list). The main content area is titled "Speciality list". It features a search bar with "Search specialities" and a magnifying glass icon. Below the search bar is a table with columns "Id" and "Name". The table contains the following data:

Id	Name
ef05c52f-b71e-41bb-968e-05c4ec8bcea8	Cardiologist
5f4480f-c1e0-4353-a832-86af9ea68e44	Neurologist
44d7cbf2-7970-499a-b959-3651d953d1f8	Urologist
6949c071-0548-413d-a70e-188845c14a6a	Rheumatologist
	ENT
	Generalist
	Nurse

A modal dialog box titled "Delete confirmation" is displayed in the center, asking "Are you sure you want to delete specialty 'Cardiologist'?" with "Delete" and "Close" buttons.

Edit speciality

(accessible from speciality list)

This page allows us to edit a given speciality

The screenshot shows the E-Clinic application interface. On the left is a sidebar with navigation links: Dashboard, Users, Patient, Medical staff, Departments, Specialities, and Procedures. The 'Specialities' section is expanded, showing 'Add speciality' and 'Speciality list'. The 'Procedures' section is also expanded, showing 'Add procedure' and 'Procedure list'. The main content area has a header 'E-Clinic' with 'Theme' and 'admin' dropdowns. Below the header is a 'Back to speciality list' button. The title 'Edit speciality' is displayed with a pencil icon. A text input field contains the value 'Cardiologist'. At the bottom is a blue button labeled '+ Edit speciality'.

Procedure management

We have multiple pages to allow the user to manage the medical procedures in our system

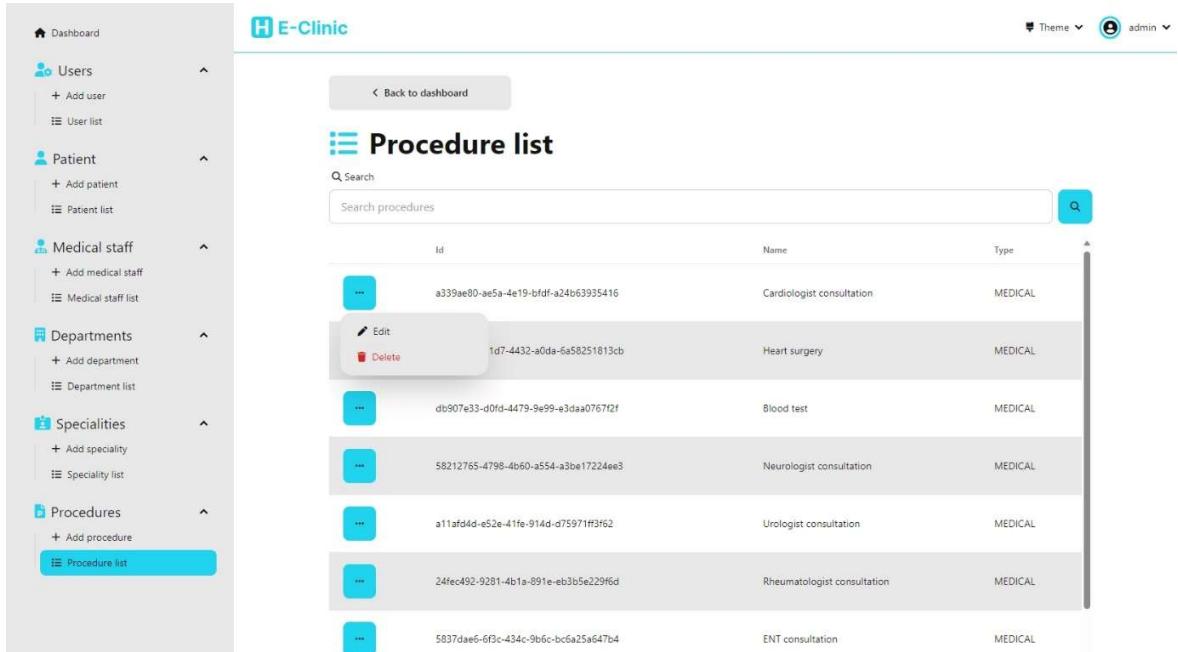
Add procedure

This page allows us to add a new procedure to the system

The screenshot shows the E-Clinic application interface. The sidebar is identical to the previous screenshot, with the 'Procedures' section expanded to show 'Add procedure' and 'Procedure list'. The main content area has a header 'E-Clinic' with 'Theme' and 'admin' dropdowns. Below the header is a 'Back to dashboard' button. The title '+ Add procedure' is displayed with a plus sign icon. There are three input fields: 'Name' (empty), 'Department' (empty dropdown menu), and 'Type' (empty dropdown menu). At the bottom is a blue button labeled '+ Add procedure'.

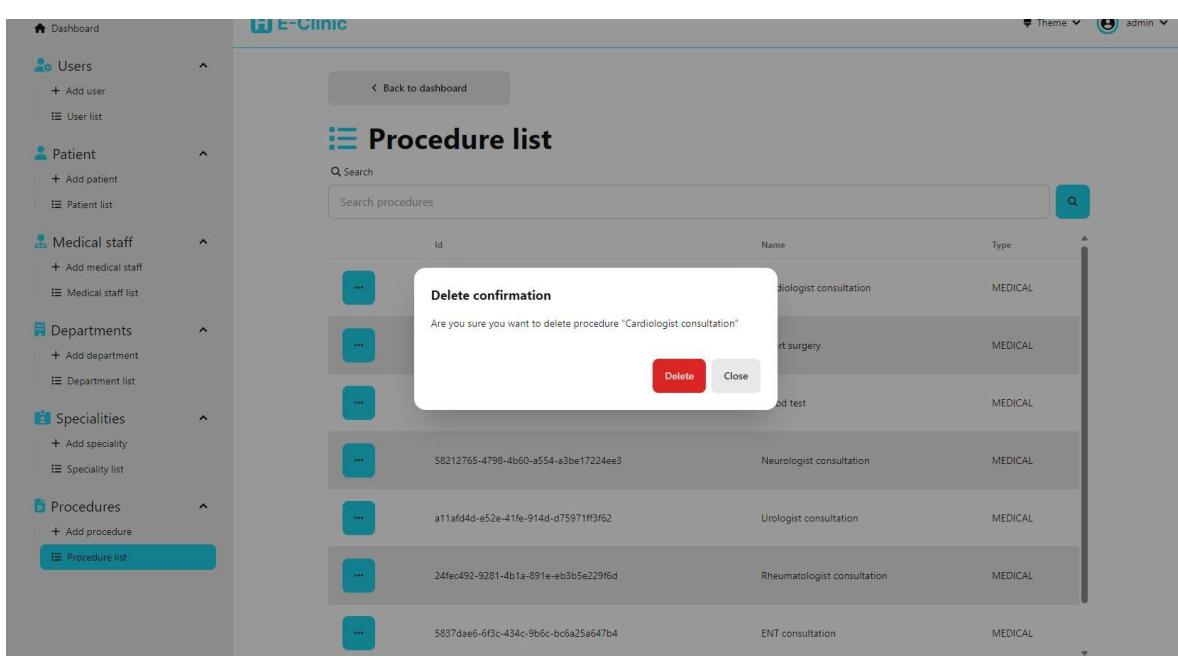
Procedure list

This page allows us to view the list of all procedures in the system. It also allows us to delete a procedure and also navigate to the page that allows us to edit a procedure.



The screenshot shows the 'Procedure list' page in the E-Clinic application. The left sidebar contains navigation links for Dashboard, Users, Patient, Medical staff, Departments, Specialities, and Procedures. The 'Procedures' section is currently selected, with 'Procedure list' highlighted. The main content area displays a table titled 'Procedure list' with columns for Id, Name, and Type. The table lists seven procedures: Cardiologist consultation, Heart surgery, Blood test, Neurologist consultation, Urologist consultation, Rheumatologist consultation, and ENT consultation, all categorized as MEDICAL. Each row includes an 'Edit' button and a 'Delete' button. A search bar at the top allows filtering by procedure name.

Id	Name	Type
a339ae80-ae5a-4e19-bfdf-a24b63935416	Cardiologist consultation	MEDICAL
1d7-4432-a0da-6a58251813cb	Heart surgery	MEDICAL
db907e33-d0fd-4479-9e99-e3da0767f2f	Blood test	MEDICAL
58212765-4798-4b60-a554-a3be17224ee3	Neurologist consultation	MEDICAL
a11af4d4d-e52e-41fe-914d-d75971ff3f62	Urologist consultation	MEDICAL
24fec492-9281-4b1a-891e-eb3b5e229f6d	Rheumatologist consultation	MEDICAL
5837dae6-6f3c-434c-9b6c-bc6a25a647b4	ENT consultation	MEDICAL



The second screenshot shows the same 'Procedure list' page, but with a modal dialog box overlaid. The dialog is titled 'Delete confirmation' and asks, 'Are you sure you want to delete procedure "Cardiologist consultation"?' It contains two buttons: a red 'Delete' button and a grey 'Close' button. The background table and sidebar are visible but dimmed.

Edit procedure

(accessible from procedure list)

This page allows us to edit a given procedure

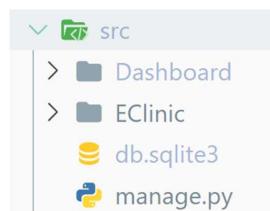
The screenshot shows the E-Clinic application interface. On the left is a sidebar with navigation links: Dashboard, Users (+ Add user, User list), Patient (+ Add patient, Patient list), Medical staff (+ Add medical staff, Medical staff list), Departments (+ Add department, Department list), Specialties (+ Add specialty, Specialty list), and Procedures (+ Add procedure, Procedure list). The main content area is titled 'Edit procedure'. It contains fields for 'Name' (Cardiologist consultation), 'Department' (Cardiology), and 'Type' (Medical). A blue button at the bottom right says '+ Edit procedure'.

Code structure

Folders tree

Root folder

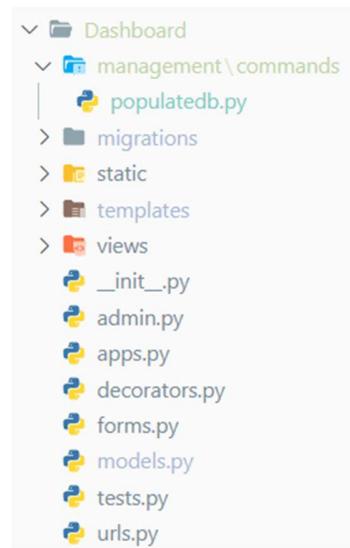
- Dashboard: Contains the dashboard app
- EClinic: Contains config files for the main project
- db.sqlite3: The database stored locally for the sqlite DBMS
- manage.py: used to run commands on the project



Dashboard

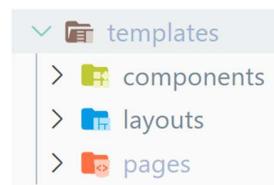
- commands: contains a script to populate the database with test data
- migrations: contains db migrations

- static: contains static assets like images and svgs
- templates: contains Django templates of the app
- views: contains views for the app
- `__init__.py`: declares the folder as a package
- `admin.py`: used to prepare the admin site, mainly used to register models
- `apps.py`: Contains config for the app
- `decorators.py`: Contains utility python decorators that are used in the app
- `forms.py`: Contains Django form for the app
- `models.py`: Contains Django model for the app
- `test.py` Contains test code
- `urls.py`: Contains URLs configuration for the app



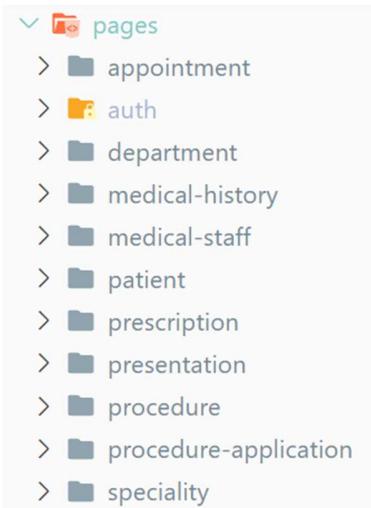
Templates

- components: Contains html code that will be reused elsewhere
- layouts: Contains layouts
- pages: Contains the different pages that will be displayed in the app



Pages

- appointments: Contains appointment related pages
- auth: Contains authentication related pages
- department: Contains department related pages
- medical-history: Contains medical history related pages
- medical-staff: Contains medical staff related pages
- patient: Contains patient related pages
- prescription: Contains prescription related pages
- presentation: Contains pages used only for presentation purposes
- procedure: Contains procedure related pages
- procedure-application: Contains procedure applications related pages
- specialty: Contains specialty related pages



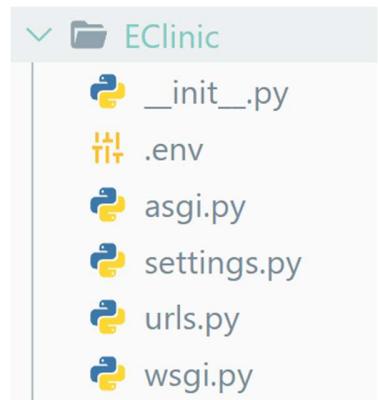
Views

- `__init__.py`: declares the folder as a package
- `appointment.py`: Contains appointment related views
- `auth.py`: Contains authentication related views
- `department.py`: Contains department related views
- `medicalHistory.py`: Contains medical history related views
- `medicalStaff.py`: Contains medical staff related views
- `patient.py`: Contains patient related views
- `prescription.py`: Contains prescription related views
- `presentation.py`: Contains presentation related views
- `procedureApplication.py`: Contains procedure applications related views
- `procedures.py`: Contains procedure related views
- `speciality.py`: Contains specialty related views



EClinic

- `__init__.py`: declares the folder as a package
- `.env`: Contains environment variables for the application
- `asgi.py` & `wsgi.py`: used for serving the app in production
- `settings.py`: Contains project settings
- `urls.py`: Contains the project's URLs configuration



Code screenshots

General code

Dashboard URL configuration

```
from django.urls import path
from .views import medicalStaff, presentation, auth, patient, medicalHistory, department, procedure, speciality, appointment, procedureApplication, prescription

urlpatterns = [
    path('', presentation.index, name='index'),
    path('dashboard', presentation.dashboard, name='dashboard'),

    path('auth/sign-in/', auth.signIn, name='sign-in'),
    path('auth/sign-out/', auth.signOut, name='sign-out'),
    path('auth/users/', auth.userList, name='user-list'),
    path('auth/users/add/', auth.addUser, name='add-user'),
    path('auth/users/edit/<str:userId>', auth.editUser, name='edit-user'),
    path('auth/users/delete/<str:userId>', auth.deleteUser, name='delete-user'),

    path('dashboard/patients/', patient.patientList, name='patient-list'),
    path('dashboard/patients/add/', patient.addPatient, name='add-patient'),
    path('dashboard/patients/edit/<str:patientId>', patient.editPatient, name='edit-patient'),
    path('dashboard/patients/delete/<str:patientId>', patient.deletePatient, name='delete-patient'),

    path('dashboard/patients/edit/<str:patientId>/history', medicalHistory.editMedicalHistory, name='edit-medical-history'),

    path('dashboard/medical-staff/', medicalStaff.medicalStaffList, name='medical-staff-list'),
    path('dashboard/medical-staff/add/', medicalStaff.addMedicalStaff, name='add-medical-staff'),
    path('dashboard/medical-staff/edit/<str:medicalStaffId>', medicalStaff.editMedicalStaff, name='edit-medical-staff'),
    path('dashboard/medical-staff/delete/<str:medicalStaffId>', medicalStaff.deleteMedicalStaff, name='delete-medical-staff'),

    path('dashboard/departments/', department.departmentList, name='department-list'),
    path('dashboard/departments/add/', department.addDepartment, name='add-department'),
    path('dashboard/departments/edit/<str:departmentId>', department.editDepartment, name='edit-department'),
    path('dashboard/departments/delete/<str:departmentId>', department.deleteDepartment, name='delete-department'),

    path('dashboard/specialities/', speciality.specialityList, name='speciality-list'),
    path('dashboard/specialities/add/', speciality.addSpeciality, name='add-speciality'),
    path('dashboard/specialities/edit/<str:specialityId>', speciality.editSpeciality, name='edit-speciality'),
    path('dashboard/specialities/delete/<str:specialityId>', speciality.deleteSpeciality, name='delete-speciality'),

    path('dashboard/procedures/', procedure.procedureList, name='procedure-list'),
    path('dashboard/procedures/add/', procedure.addProcedure, name='add-procedure'),
    path('dashboard/procedures/edit/<str:procedureId>', procedure.editProcedure, name='edit-procedure'),
    path('dashboard/procedures/delete/<str:procedureId>', procedure.deleteProcedure, name='delete-procedure'),

    path('dashboard/appointments/patient/<str:patientId>/add', appointment.addAppointment, name='add-appointment'),
    path('dashboard/appointments/edit/<str:appointmentId>', appointment.editAppointment, name='edit-appointment'),
    path('dashboard/appointments/patient/<str:patientId>', appointment.appointmentListPatient, name='appointment-list-patient'),
    path('dashboard/appointments/medical-staff/<str:medicalStaffId>', appointment.appointmentListMedicalStaff, name='appointment-list-medical-staff'),
    path('dashboard/appointments/delete/<str:appointmentId>', appointment.deleteAppointment, name='delete-appointment'),
    path('dashboard/appointments/apply-procedure/<str:appointmentId>', procedureApplication.applyProcedure, name='apply-procedure'),

    path('dashboard/procedure-applications/delete/<str:procedureApplicationId>', procedureApplication.deleteProcedureApplication, name='delete-procedure-application'),

    path('dashboard/prescriptions/patient/<str:appointmentId>/add', prescription.addPrescription, name='add-prescription'),
    path('dashboard/prescriptions/patient/<str:prescriptionId>/edit', prescription.editPrescription, name='edit-prescription'),
    path('dashboard/prescriptions/patient/<str:prescriptionId>/delete', prescription.deletePrescription, name='delete-prescription'),
]
```

Dashboard forms

```
 5
 4 class SignInForm(forms.Form):
 5     username = forms.CharField()
 6     password = forms.CharField()
 7
 8 class SearchForm(forms.Form):
 9     search = forms.CharField(required=False)
10
11 class PatientForm(forms.ModelForm):
12     class Meta:
13         model = Patient
14         fields = '__all__'
15
16 class MedicalHistoryForm(forms.ModelForm):
17     class Meta:
18         model = MedicalHistory
19         fields = '__all__'
20
21 class MedicalStaffForm(forms.ModelForm):
22     class Meta:
23         model = MedicalStaff
24         fields = '__all__'
25
26 class DepartmentForm(forms.ModelForm):
27     class Meta:
28         model = Department
29         fields = '__all__'
30
31 class SpecialityForm(forms.ModelForm):
32     class Meta:
33         model = Speciality
34         fields = '__all__'
35
36 class ProcedureForm(forms.ModelForm):
37     class Meta:
38         model = Procedure
39         fields = '__all__'
40
41 class ProcedureApplicationForm(forms.ModelForm):
42     class Meta:
43         model = ProcedureApplication
44         fields = '__all__'
45
46 class AssignMedicalStaffForm(forms.Form):
47     medicalStaffId = forms.CharField()
48     remove = forms.BooleanField(required=False)
49
50 class AppointmentForm(forms.ModelForm):
51     class Meta:
52         model = Appointment
53         fields = '__all__'
54
55 class PrescriptionForm(forms.ModelForm):
56     class Meta:
57         model = Prescription
58         fields = '__all__'
```

Dashboard models

```
● ● ●
1  from django.db import models
2  import uuid
3  from django.core.validators import EmailValidator, RegexValidator
4
5  # Create your models here.
6  class Genders(models.TextChoices):
7      MALE = 'MALE'
8      FEMALE = 'FEMALE'
9
10 class ProcedureType(models.TextChoices):
11     MEDICAL = 'MEDICAL',
12     ADMINISTRATIVE = 'ADMINISTRATIVE'
13
14 class AppointmentState(models.TextChoices):
15     SCHEDULED = 'SCHEDULED'
16     COMPLETED = 'COMPLETED'
17     CANCELED = 'CANCELED'
18
19 class Patient(models.Model):
20     id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
21     firstName = models.CharField(max_length=256, null=False)
22     lastName = models.CharField(max_length=256, null=False)
23     dateOfBirth = models.DateField(null=False)
24     address = models.CharField(max_length=256, null=False)
25     phoneNumber = models.CharField(max_length=20, null=False, unique=True, validators=[RegexValidator(r'^(\d{10,14})$')])
26     email = models.CharField(max_length=254, null=False, unique=True, validators=[EmailValidator()])
27     gender = models.CharField(max_length=10, choices=Genders.choices, null=False)
28     createdAt = models.DateTimeField(auto_now_add=True)
29     updatedAt = models.DateTimeField(auto_now=True)
30
31     def __str__(self):
32         return f'{self.lastName} {self.firstName}'
33
34 class MedicalHistory(models.Model):
35     id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
36     patient = models.OneToOneField('Patient', on_delete=models.CASCADE)
37     notes = models.TextField(null=True, blank=True)
38     allergies = models.TextField(null=True, blank=True)
39     surgeries = models.TextField(null=True, blank=True)
40     medications = models.TextField(null=True, blank=True)
41     updatedAt = models.DateTimeField(auto_now=True, blank=True)
42
43 class MedicalStaff(models.Model):
44     id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
45     department = models.ForeignKey('Department', on_delete=models.CASCADE)
46     speciality = models.ForeignKey('Speciality', on_delete=models.CASCADE)
47     firstName = models.CharField(max_length=256, null=False)
48     lastName = models.CharField(max_length=256, null=False)
49     dateOfBirth = models.DateField(null=False)
50     address = models.CharField(max_length=256, null=False)
51     phoneNumber = models.CharField(max_length=20, null=False, unique=True, validators=[RegexValidator(r'^(\d{10,14})$')])
52     email = models.CharField(max_length=254, null=False, unique=True, validators=[EmailValidator()])
53     gender = models.CharField(max_length=10, choices=Genders.choices, null=False)
54     createdAt = models.DateTimeField(auto_now_add=True)
55     updatedAt = models.DateTimeField(auto_now=True)
56
57     def __str__(self):
58         return f'{self.lastName} {self.firstName}'
```

```

59
60 class Department(models.Model):
61     id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
62     name = models.CharField(max_length=256, null=False, unique=True)
63
64     def __str__(self):
65         return f"{self.name}"
66
67 class Speciality(models.Model):
68     id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
69     name = models.CharField(max_length=256, null=False, unique=True)
70
71     def __str__(self):
72         return f"{self.name}"
73
74 class Procedure(models.Model):
75     id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
76     name = models.CharField(max_length=256, null=False, unique=True)
77     department = models.ForeignKey('Department', on_delete=models.CASCADE, null=True, blank=True)
78     type = models.CharField(max_length=20, choices=ProcedureType.choices, null=False)
79
80     def __str__(self):
81         return f"{self.name}"
82
83 class ProcedureApplication(models.Model):
84     id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
85     appointment = models.ForeignKey('Appointment', on_delete=models.CASCADE)
86     medicalStaff = models.ForeignKey('MedicalStaff', on_delete=models.CASCADE)
87     procedure = models.ForeignKey('Procedure', on_delete=models.CASCADE)
88     report = models.TextField(null=True, blank=True)
89
90     class Meta:
91         unique_together = ('appointment', 'medicalStaff', 'procedure')
92
93 class Appointment(models.Model):
94     id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
95     patient = models.ForeignKey('Patient', on_delete=models.CASCADE)
96     timestamp = models.DateTimeField(null=False)
97     state = models.CharField(max_length=15, choices=AppointmentState.choices, default=AppointmentState.SCHEDULED, blank=True)
98     createdAt = models.DateTimeField(auto_now_add=True)
99     updatedAt = models.DateTimeField(auto_now=True)
100
101 class Prescription(models.Model):
102     id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
103     appointment = models.ForeignKey('Appointment', on_delete=models.CASCADE)
104     content = models.TextField()
105

```

A utility decorator

A utility decorator that prevents a signed in user from going to a page where only unauthenticated users can see

```
● ● ●
1 from django.shortcuts import redirect
2
3 def login_not_required(view_func):
4     def wrapper(request, *args, **kwargs):
5         if request.user.is_authenticated:
6             return redirect('dashboard')
7         else:
8             return view_func(request, *args, **kwargs)
9
10    return wrapper
11
```

Explanation of a view

In this project views mostly do basic operations on the models. So to avoid redundancy we are only going to explain the working of the appointment views as a typical example of how we implemented the other views' code.

Appointment views

Add appointment view

```
● ● ●
1 @login_required(login_url='sign-in')
2 def addAppointement(request: HttpRequest, patientId: str):
3     patient = get_object_or_404(Patient, id=patientId)
4     form = AppointmentForm(initial={'patient': patient})
5     if request.method == 'POST':
6         data = request.POST.copy()
7         data.appendlist('patient', patient)
8         form = AppointmentForm(data=data)
9     if form.is_valid():
10        appointment = form.save()
11        messages.success(request=request, message='Appointment created successfully')
12        return redirect('edit-appointment', appointmentId=appointment.id)
13    context = { 'form': form, 'patient': patient }
14    return render(request, 'pages/appointment/add-appointment.html', context=context)
```

Explanation

In this view, marked with the `@login_required` decorator to ensure authentication, we handle the creation of appointments for a specific patient. The patient is retrieved from the database using the provided `patientId`. An `AppointmentForm` instance is then generated with the 'patient' field pre-populated with the obtained patient information. The view accommodates both GET and POST requests. In the case of a

POST request, the form data is processed, and if valid, a new appointment is saved to the database. A success message is added, and the user is redirected to the 'edit-appointment' view with the ID of the newly created appointment. If the request method is GET or the form is invalid, the form and patient details are passed to the 'add-appointment.html' template for rendering.

Edit appointment view

```
● ● ●
1 @login_required(login_url='sign-in')
2 def editAppointment(request: HttpRequest, appointmentId: str):
3     appointment = get_object_or_404(Appointment, id=appointmentId)
4     patient = get_object_or_404(Patient, id=appointment.patient.id)
5     procedureApplications = ProcedureApplication.objects.filter(appointment=appointment.id).order_by('procedure')
6     prescriptions = Prescription.objects.filter(appointment=appointment)
7
8     form = AppointmentForm(instance=appointment, initial={'patient': patient})
9     if request.method == 'POST':
10         data = request.POST.copy()
11         data.appendlist('patient', patient)
12         form = AppointmentForm(data=data, instance=appointment)
13         if form.is_valid():
14             form.save()
15             messages.success(request=request, message='Appointment updated successfully')
16             return redirect('appointment-list-patient', patientId=patient.id)
17     context = { 'form': form, 'patient': patient, 'procedureApplications': procedureApplications, 'prescriptions': prescriptions, 'appointment': appointment }
18     return render(request, 'pages/appointment/edit-appointment.html', context=context)
```

Explanation

In this view, decorated with `@login_required` to enforce authentication, we manage the editing of a specific appointment identified by the provided `appointmentId`. The corresponding appointment and patient objects are retrieved from the database using the appointment's ID and the patient's ID associated with the appointment. Additionally, the view fetches related procedure applications and prescriptions for the appointment, ordering the procedure applications by the 'procedure' field. An `AppointmentForm` instance is created, setting both the instance and initial data to the appointment and patient details, respectively. The view accommodates both GET and POST requests. In the case of a POST request, the form data is processed. The 'patient' field is appended to the data to ensure it aligns with the expected form structure. The form is then instantiated with this modified data and the appointment instance. If the form is valid, the changes are saved to the appointment, a success message is added, and the user is redirected to the 'appointment-list-patient' view for the corresponding patient. If the request method is GET or the form is invalid, the form, patient details, procedure applications, prescriptions, and the appointment itself are passed to the 'edit-appointment.html' template for rendering. This view provides a comprehensive interface for editing appointments, displaying related information, and maintaining a structured approach to Django web application development.

Delete appointment view

```
● ● ●
1 @login_required(login_url='sign-in')
2 def deleteAppointment(request: HttpRequest, appointmentId: str):
3     appointment = get_object_or_404(Appointment, id=appointmentId)
4     patient = appointment.patient
5     appointment.delete()
6     return redirect('appointment-list-patient', patientId=patient.id)
```

Explanation

In this Django view, decorated with `@login_required` to ensure user authentication, the focus is on deleting a specific appointment identified by the provided `appointmentId`. The view begins by fetching the appointment object from the database using the ID. The associated patient object is then obtained from the appointment. Subsequently, the appointment is deleted from the database. Finally, the user is redirected to the '`appointment-list-patient`' view for the patient associated with the deleted appointment.

Appointment list for a medical staff

```
● ● ●
1 @login_required(login_url='sign-in')
2 def appointmentListMedicalStaff(request: HttpRequest, medicalStaffId: str):
3     medicalStaff = get_object_or_404(MedicalStaff, id=medicalStaffId)
4
5     procedure_applications_for_medical_staff = ProcedureApplication.objects.filter(medicalStaff=medicalStaff)
6     appointments_for_medical_staff = [pa.appointment for pa in procedure_applications_for_medical_staff]
7
8     length = len(appointments_for_medical_staff)
9
10    context = { 'appointments': appointments_for_medical_staff, 'medicalStaff': medicalStaff, 'length': length }
11    return render(request, 'pages/appointment/appointment-list-medical-staff.html', context=context)
```

Explanation

In this Django view, tagged with `@login_required` to ensure user authentication, the primary objective is to list appointments associated with a specific medical staff member identified by the provided `medicalStaffId`. The view starts by fetching the `MedicalStaff` object from the database using the provided ID. Subsequently, it retrieves procedure applications for the given medical staff, extracting the corresponding appointments. The length of the resulting list of appointments is then calculated. The view constructs a context dictionary containing the list of appointments, the medical staff member, and the length of the appointment list. Finally, it renders the '`appointment-list-medical-staff.html`' template, passing the context for rendering.

Appointment list for a patient

```
● ○ ●
1 @login_required(login_url='sign-in')
2 def appointmentListPatient(request: HttpRequest, patientId: str):
3     patient = get_object_or_404(Patient, id=patientId)
4     form = SearchForm(request.GET)
5     appointments = []
6     if form.is_valid():
7         search = form.cleaned_data['search']
8         appointments = Appointment.objects.filter(
9             Q(id__icontains=search) |
10            Q(timestamp__icontains=search) |
11            Q(state__icontains=search),
12            patient=patient,
13        ).order_by('-timestamp')
14     else:
15         appointments = Appointment.objects.filter(
16             patient=patient,
17         ).order_by('-timestamp')
18     context = { 'appointments': appointments, 'form': form, 'patient': patient }
19     return render(request, 'pages/appointment/appointment-list-patient.html', context=context)
```

Explanation

In this Django view, protected by the `@login_required` decorator to ensure user authentication, the primary goal is to list appointments associated with a specific patient identified by the provided `patientId`. The view begins by fetching the `Patient` object from the database using the provided ID. Additionally, it initializes a `SearchForm` instance with the GET data from the request. The view then proceeds to handle the search functionality. If the form is valid, meaning that a search term is provided, the search term is extracted from the form's cleaned data. The appointments are filtered based on the search term and patient, using the `Appointment` model fields (ID, timestamp, and state), and ordered by descending timestamp. If the form is not valid, all appointments for the specified patient are fetched and ordered by descending timestamp. Finally, the view constructs a context dictionary containing the list of appointments, the form, and the patient, and renders the '`appointment-list-patient.html`' template with this context. This view offers a user-friendly interface for patients to view their appointments, with the added flexibility of a search feature, contributing to the overall functionality and user experience of a Django web application.

Deployment process

The deployment process involves cloning the repository, installing dependencies, configuring the SQLite database, running migrations, creating a superuser for administrative access, and starting the development server.

E-Clinic deployment

Preparing the app for deployment

Running a Django app on a production differs greatly from the local development environment so we have to make some changes

Static files

Django framework necessitates the collection of static files to ensure a seamless and efficient deployment of web applications. Static files encompass essential components such as stylesheets, JavaScript files, images, and other assets that contribute to the overall presentation and functionality of a web application. When a Django project is deployed, these static files need to be organized, centralized, and served efficiently to enhance the user experience. The process of collecting static files involves consolidating these resources from various locations within the project and placing them in a designated directory.

- First we specify the “STATIC_ROOT” in “settings.py” which is the folder that will contain all the static files of our application

```
STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')
```

- Serving static files as the project is currently configured can cause URL issues in production which will make our static files inaccessible. In our case we will use a Django tool called “Whitenoise”

Whitenoise is essential for Django because it simplifies the handling of static files during deployment. With Whitenoise, Django can directly serve static files, eliminating the need for an additional server configuration. This makes deployment easier and reduces potential complications. Whitenoise also optimizes performance by compressing and caching static assets, leading to faster load times. On the other hand, not using Whitenoise might require relying on a separate web server for static files, introducing complexity and potentially slowing down the application.

To add Whitenoise to our app we run

```
pip install whitenoise
```

And we add it to our “settings.py” in the middleware list

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    "whitenoise.middleware.WhiteNoiseMiddleware",
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

Environment variables

By default some values in the “settings.py” are written in the code directly which means we can’t change them in a production environment. So we read them from the local environment variables

In our case we will use a package called “django-environ” so we run

```
pip install django-environ
```

and we edit our “setting.py”

```
SECRET_KEY = env('SECRET_KEY')
DEBUG = False if env('DEBUG') == 'False' else True
ALLOWED_HOSTS = env('ALLOWED_HOSTS').split(" ")
```

Production database

By default Django used “sqlite” as a database, which is only intended for local development. However in production we need a read DBMS, in our case we will use “PostgreSQL”

To use it with Django we will need the appropriate package

```
pip install psycopg2
```

And we need to configure our app to use it production, otherwise it will use sqlite

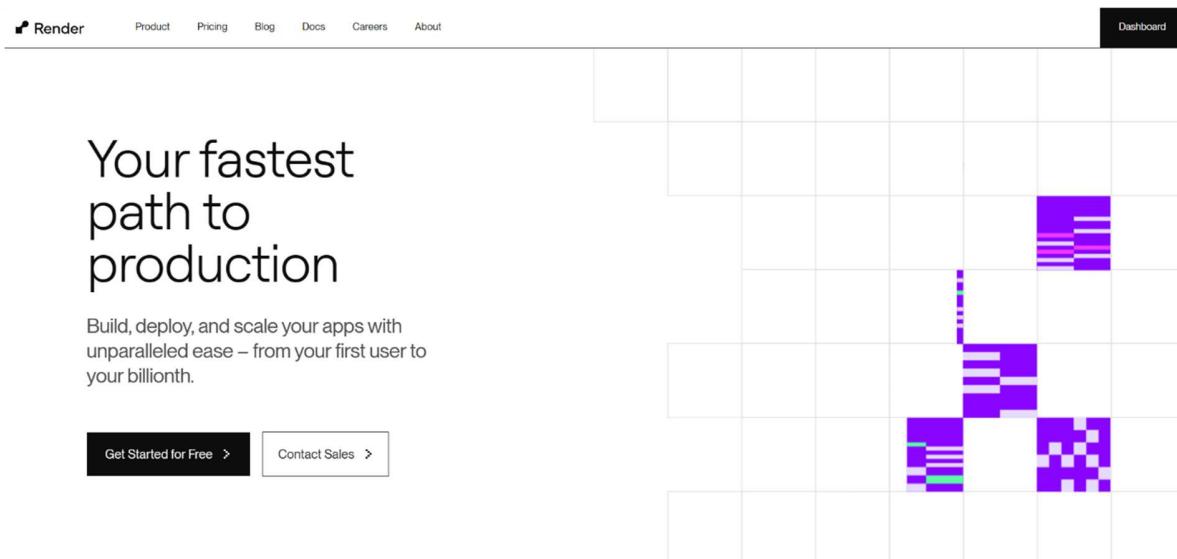
```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': env('DB_NAME'),
        'USER': env('DB_USER'),
        'PASSWORD': env('DB_PASSWORD'),
        'HOST': env('DB_HOST'),
        'PORT': env('DB_PORT'),
    } if env('DEBUG') == 'False' else {
```

```
'ENGINE': 'django.db.backends.sqlite3',
'NAME': BASE_DIR / 'db.sqlite3',
}
```

Now our app is ready to run on a production server

Deploying the app to a webhosting service

Render.com

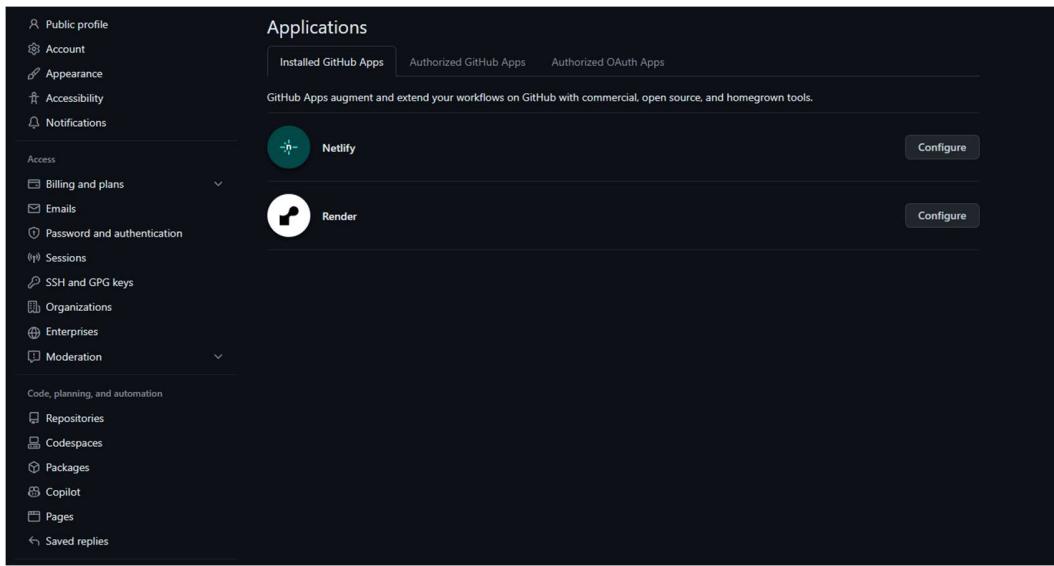


Render emerges as a cloud application hosting platform aimed at simplifying the deployment and management of web apps, APIs, and background jobs for developers. Utilizing a serverless architecture, Render facilitates a code-centric workflow, relieving developers of infrastructure concerns. Renowned for its scalability, speed, reliability, and security, Render leverages a global network and provides developer-friendly tools, including a generous free plan.

Publishing to github

In order to deploy our app to “render.com” we first need to publish it to a github account that will be used to log into “render.com”

We also need to allow access to the repository of our project for the hosting provider



Creating the postgres database

New PostgreSQL

[Read the docs](#)

Name

A unique name for your PostgreSQL instance.

eclinic-postgres

Database Optional

The PostgreSQL `dbname`

eclinic

User Optional

admin

Region

The [region](#) where your PostgreSQL instance runs. Services must be in the same region to communicate privately and you currently have services running in Frankfurt.

Frankfurt (EU Central)

PostgreSQL Version

15

Datadog API Key Optional

The API key to use for sending metrics to Datadog. Setting this will enable Datadog monitoring.

Instance Type

For hobby projects

Free
\$0 / month

256 MB (RAM)
0.1 CPU
1 GB (Storage)

⚠️ Upgrade to enable more features
Free instances do not support backups. A free database expires 90 days after creation. Only one free PostgreSQL instance can be active for any given Render user or team.

Deploying the application

Create a new Web Service

Connect a Git repository, or use an existing image.

How would you like to deploy your web service?

- Build and deploy from a Git repository

Connect a GitHub or GitLab repository.

- Deploy an existing image from a registry ADVANCED

Pull a public image from any registry or a private image from Docker Hub, GitHub, or GitLab.

Next

Create a new Web Service

Connect your Git repository or use an existing public repository URL.

Connect a repository

 Search...

 lotfi165 / EClinic  · 2 hours ago

Connect

You are deploying a web service for [lotfi165/EClinic](#).

You seem to be using Django, so we've autofilled some fields accordingly. Make sure the values look right to you!

Name	<input type="text" value="EClinic"/>
Region	<input type="text" value="Frankfurt (EU Central)"/>
Branch	<input type="text" value="main"/>
Root Directory	<small>Optional</small>
	Defaults to repository root. When you specify a <code>root directory</code> that is different from your repository root, Render runs all your commands in the specified directory and ignores changes outside the directory.
Runtime	<input type="text" value="Python 3"/>
Build Command	<pre>\$ pip install -r requirements.txt && cd src && python manage.py collectstatic</pre>
Start Command	<pre>\$ cd src && gunicorn EClinic.wsgi:application</pre>

Configuring environment variables

Environment Variables

Set environment-specific config and secrets (such as API keys), then read those values from your code. [Learn more.](#)

Key	Value	Action
DEBUG	*****	
ALLOWED_HOSTS	*****	
SECRET_KEY	*****	
DB_NAME	*****	
DB_USER	*****	
DB_PASSWORD	*****	
DB_HOST	*****	
DB_PORT	*****	

[↳ Create Environment Group](#) [+ Add Environment Variable](#) [Save Changes](#)

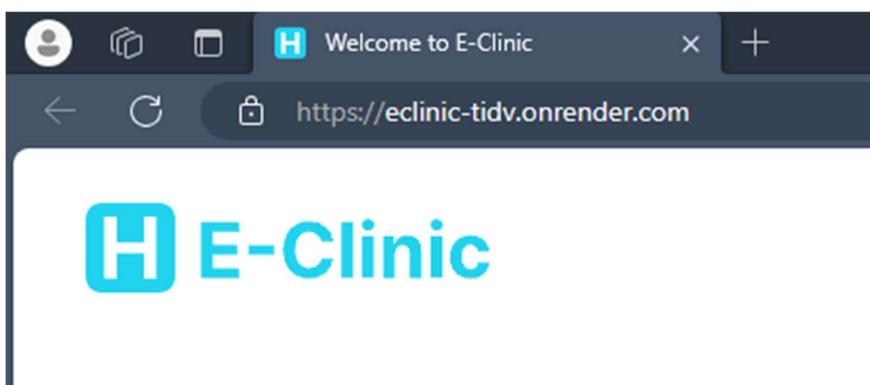
All logs Search Live tail GMT+1

```
Jan 27 06:07:23 PM  => Uploading build...
Jan 27 06:07:31 PM  => Build uploaded in 7s
Jan 27 06:07:31 PM  => Build successful 🎉
Jan 27 06:07:37 PM  => Deploying...
Jan 27 06:07:53 PM  => Using Node version 20.10.0 (default)
Jan 27 06:07:53 PM  => Docs on specifying a Node version: https://render.com/docs/node-version
Jan 27 06:07:59 PM  => Running 'cd src && gunicorn EClinic.wsgi:application'
Jan 27 06:08:02 PM  [2024-01-27 17:08:02 +0000] [41] [INFO] Starting gunicorn 21.2.0
Jan 27 06:08:02 PM  [2024-01-27 17:08:02 +0000] [41] [INFO] Listening at: http://0.0.0.0:10000 (41)
Jan 27 06:08:02 PM  [2024-01-27 17:08:02 +0000] [41] [INFO] Using worker: sync
Jan 27 06:08:02 PM  [2024-01-27 17:08:02 +0000] [42] [INFO] Booting worker with pid: 42
Jan 27 06:08:08 PM  => Your service is live 🎉
```

The app is now deployed and can be accessed via an url provided by the hosting service

you can access the app via

<https://eclinic-tidv.onrender.com/>



Lessons Learned

In this project we learned how to make a database design using the MERISE method. We also learned how to implement it using Django. In addition, we know now how to deploy a Django based web application and how to structure projects and make a clean code. On top of that, we gained experience on teamwork and how to split tasks for each member. Finally, we learnt how to present a project report academically and professionally.

Conclusion

The Eclinic web app successfully achieves its objectives, providing an efficient solution for managing healthcare processes. The project report highlights key achievements and outlines future considerations both academically and professionally.