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

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 SqlLite 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 sqlLite 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.

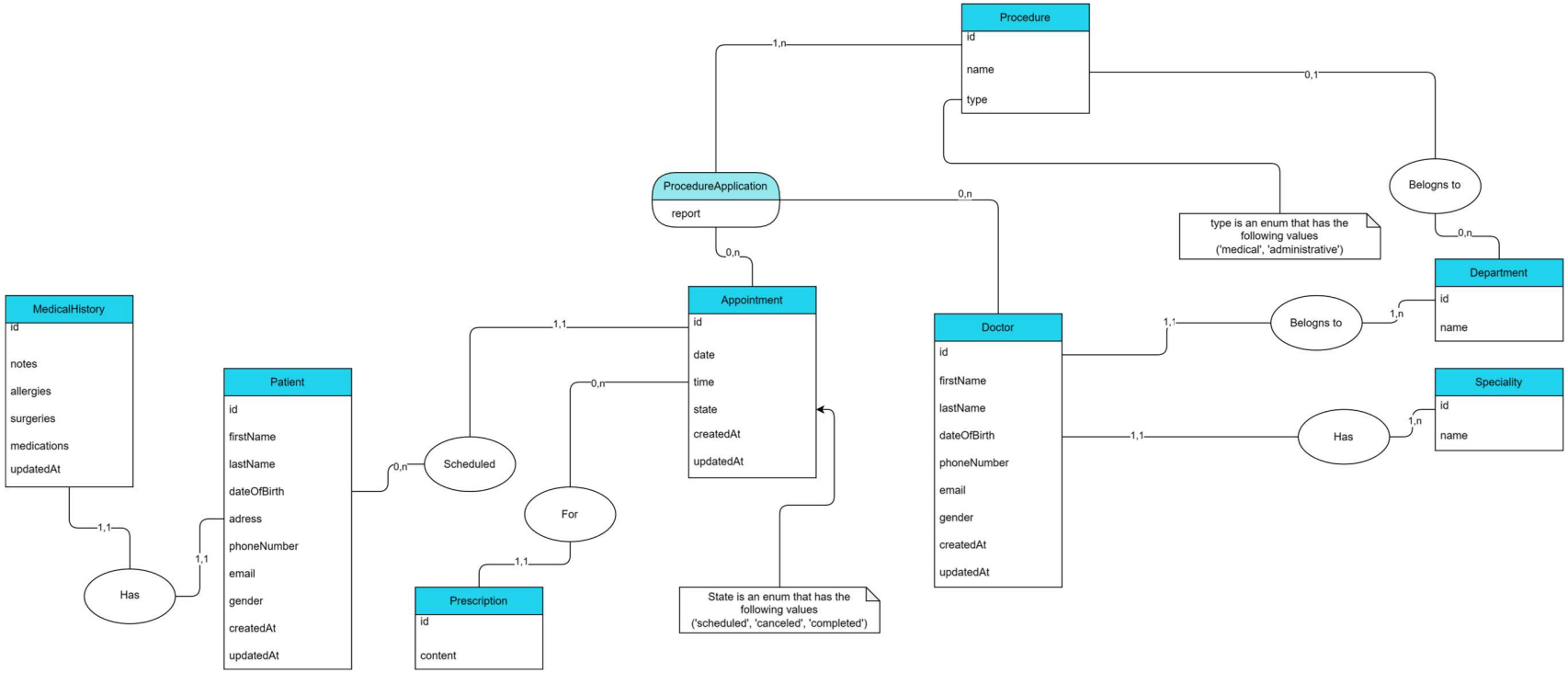
We used **MERISE** Method to design our database.

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


Procedure
id
name
type
1,n
0,1
ProcedureApplication
report
0,n
0,n
Belogns to
0,n
type is an enum that has the following values ('medical', 'administrative')
Department
id
name
MedicalHistory
id
notes
allergies
surgeries
medications
updatedAt
1,1
Has
1,1
Patient
id
firstName
lastName
dateOfBirth
adress
phoneNumber
email
gender
createdAt
updatedAt
0,n
Scheduled
1,1
Appointment
id
date
time
state
createdAt
updatedAt
0,n
For
1,1
Prescription
id
content
State is an enum that has the following values ('scheduled', 'canceled', 'completed')
Doctor
id
firstName
lastName
dateOfBirth
phoneNumber
email
gender
createdAt
updatedAt
1,1
Belogns to
1,n
1,1
Has
1,n
Speciality
id
name

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)

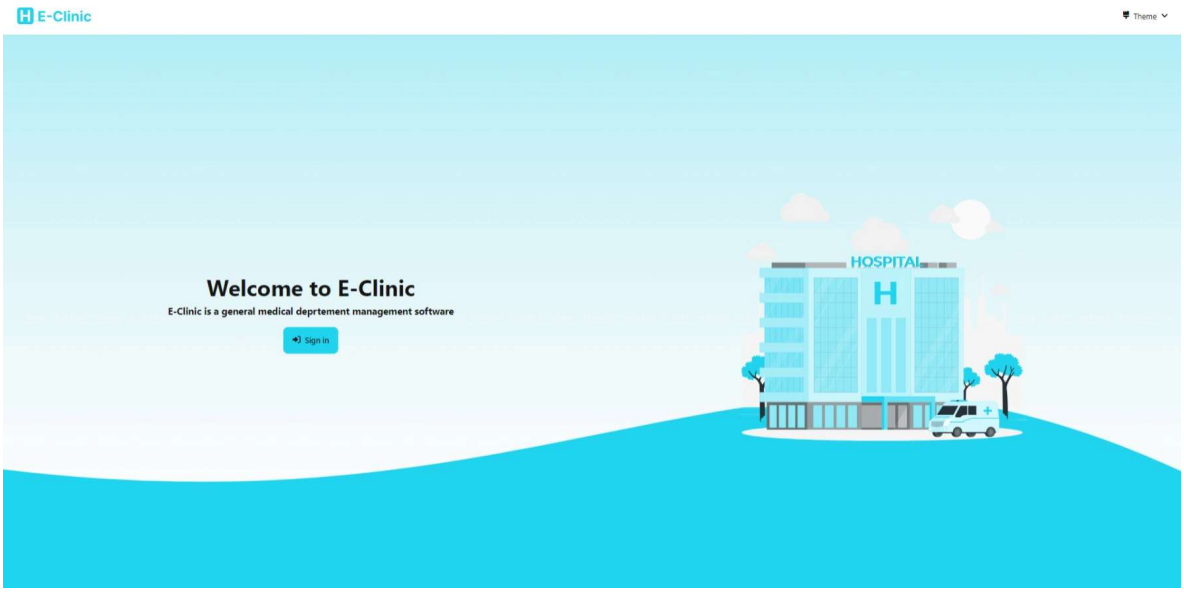
Prescription(**id**, *appointment**, content)

Graphical interface

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

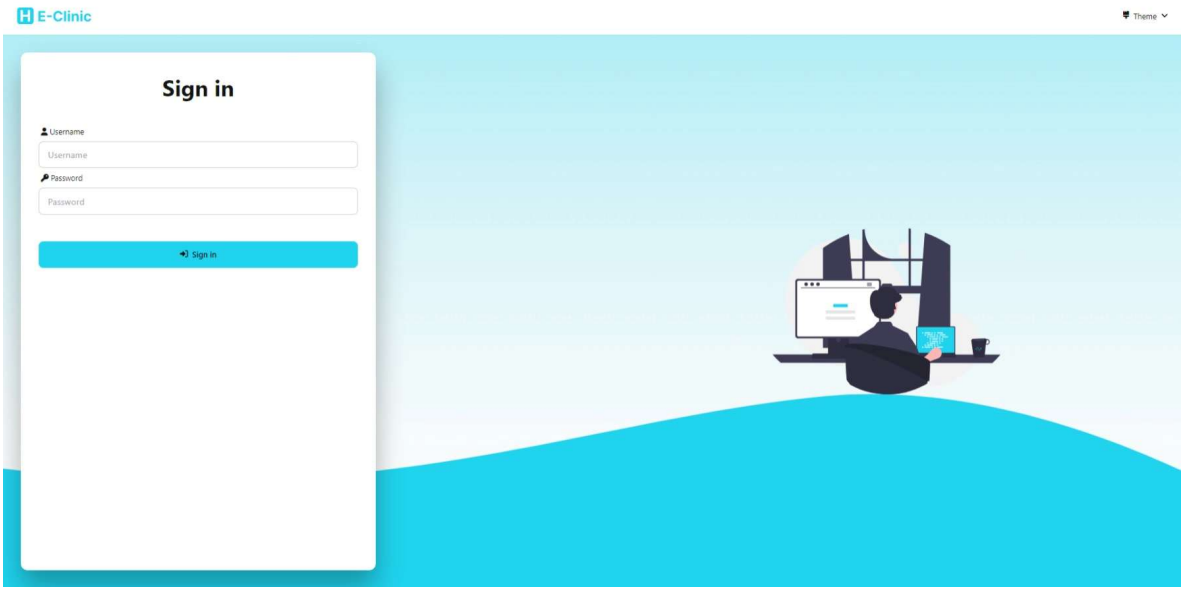
Presentation

This page describes what the app is about



Sign in

This page allows an administrator to sign in into the system

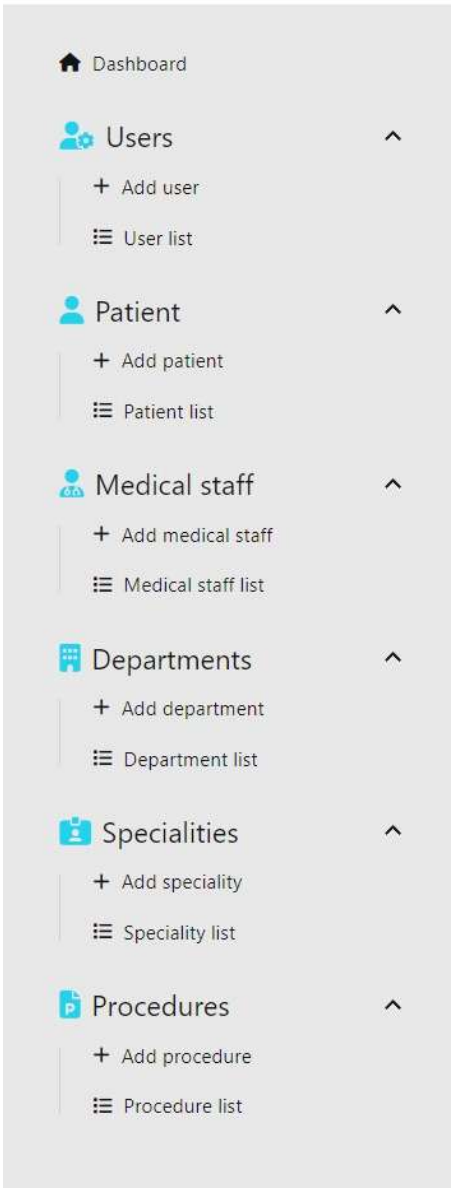


Dashboard

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

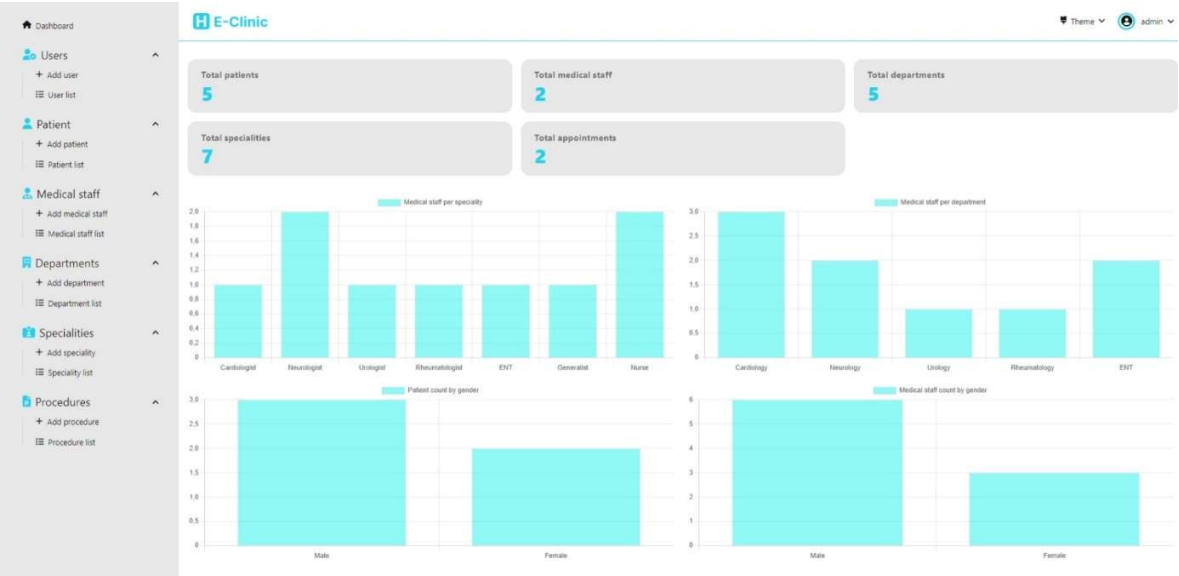
- The navbar shown the current user and allow us to sign out. Additionally we can also change the theme of the app

- 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.

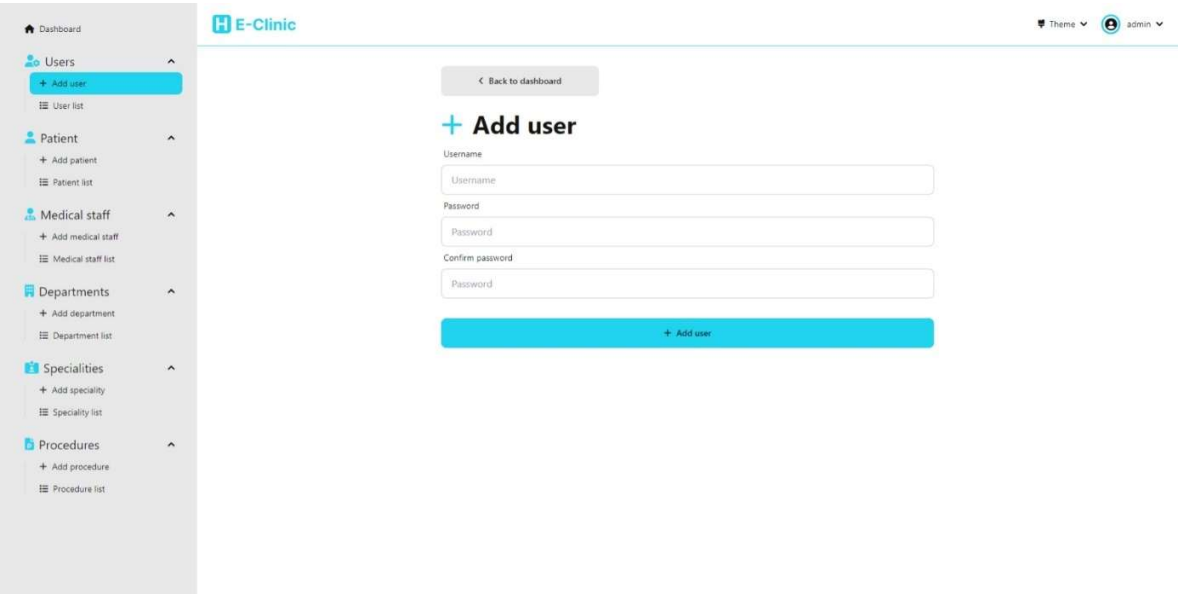


User management

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

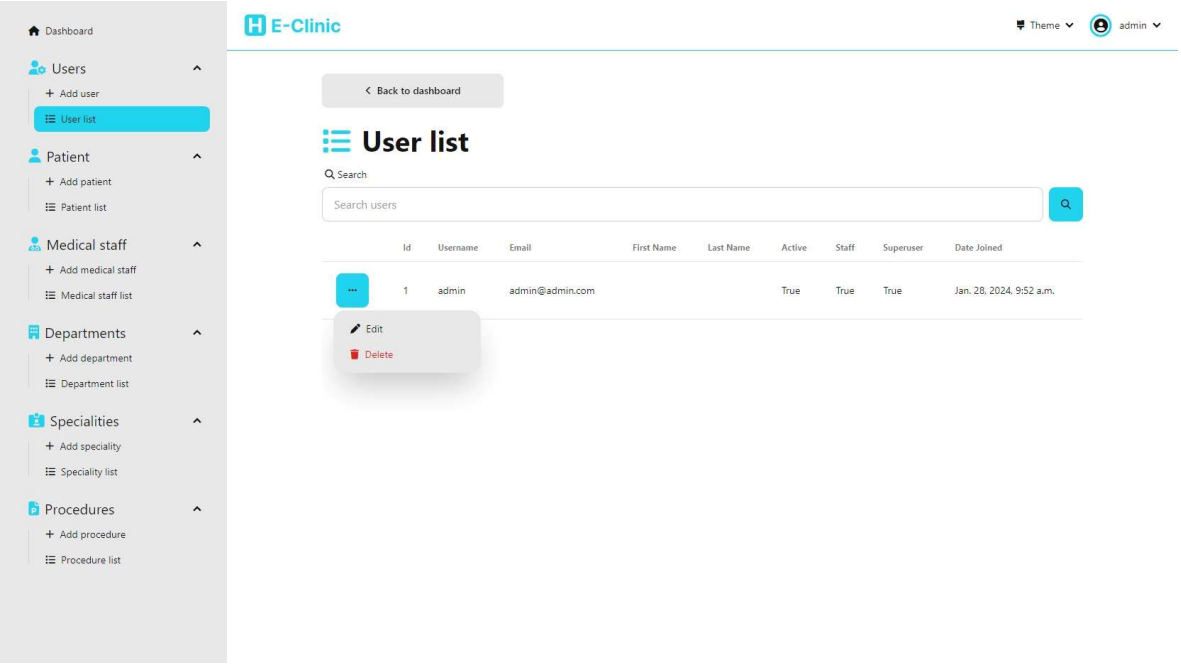
Add user

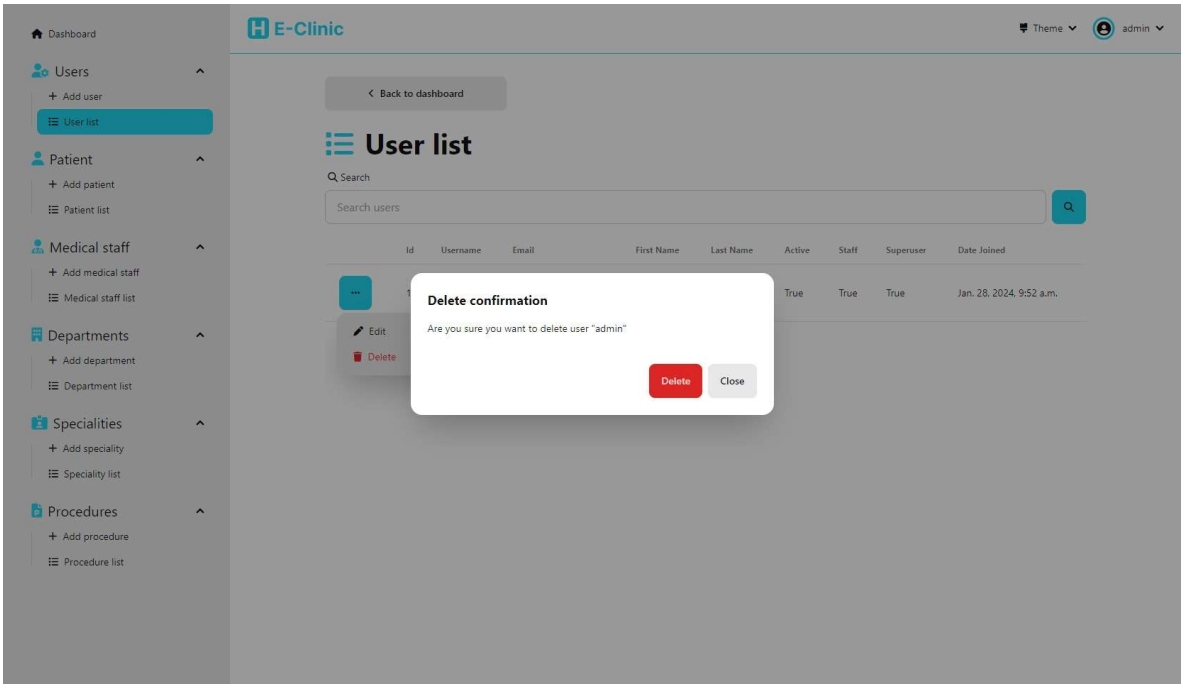
This page allow us to add a new user to the system



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

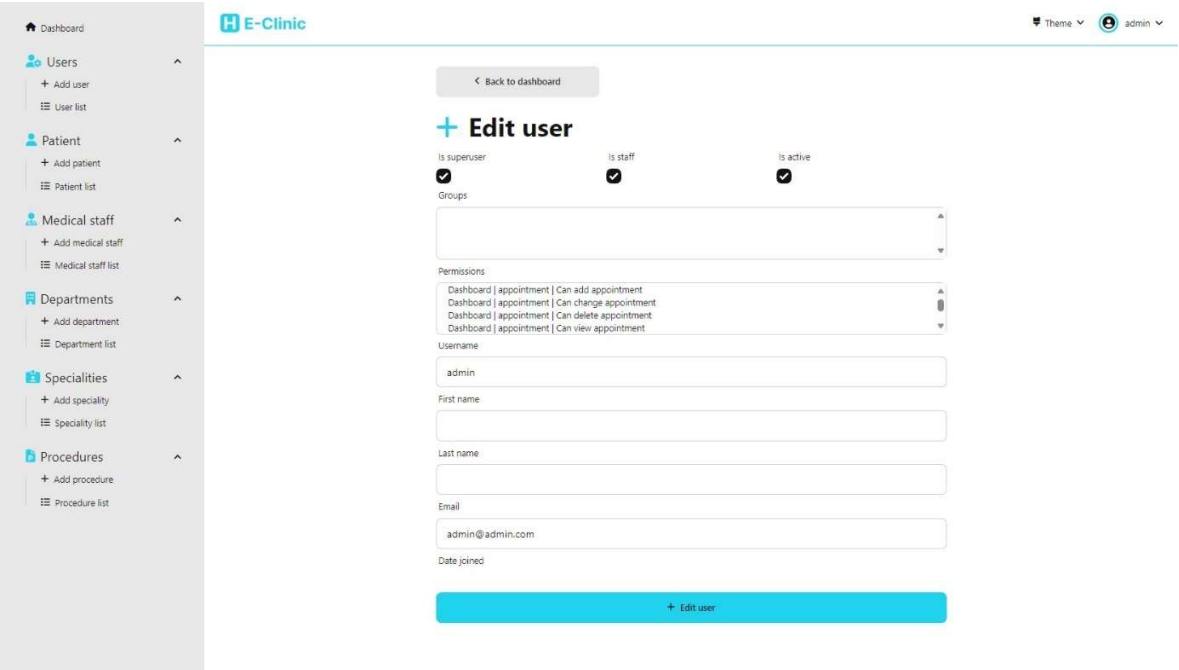




Edit user

(accessible from user list)

This page allow us to edit a user

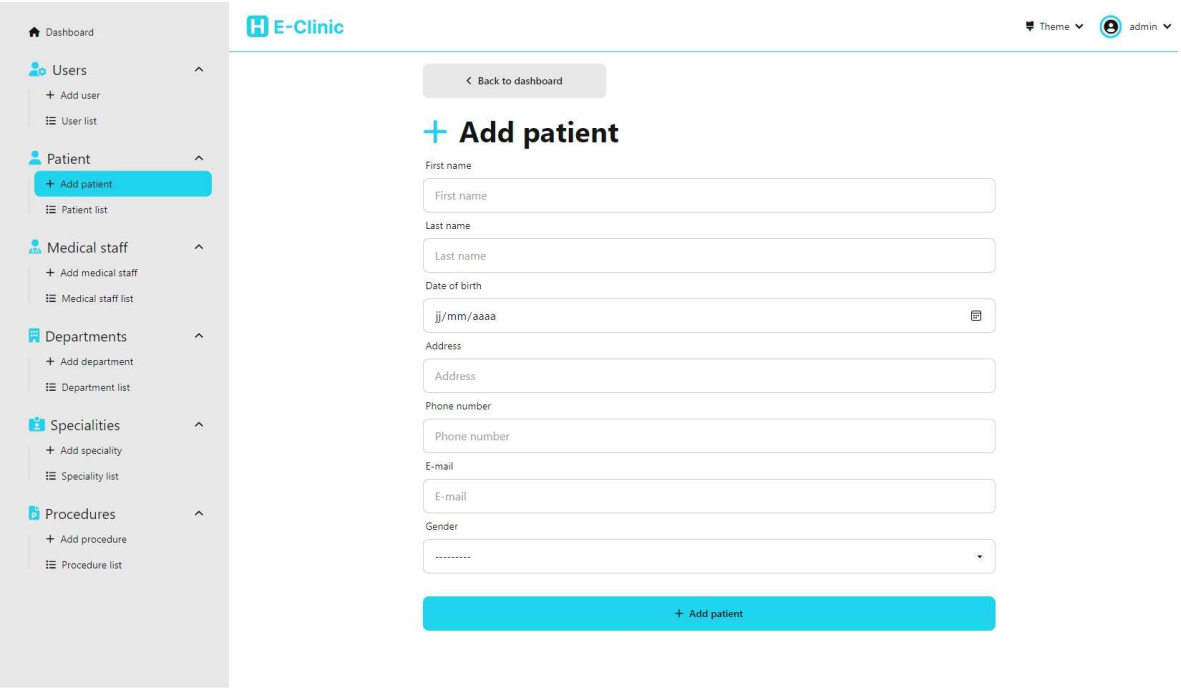


Patient management

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

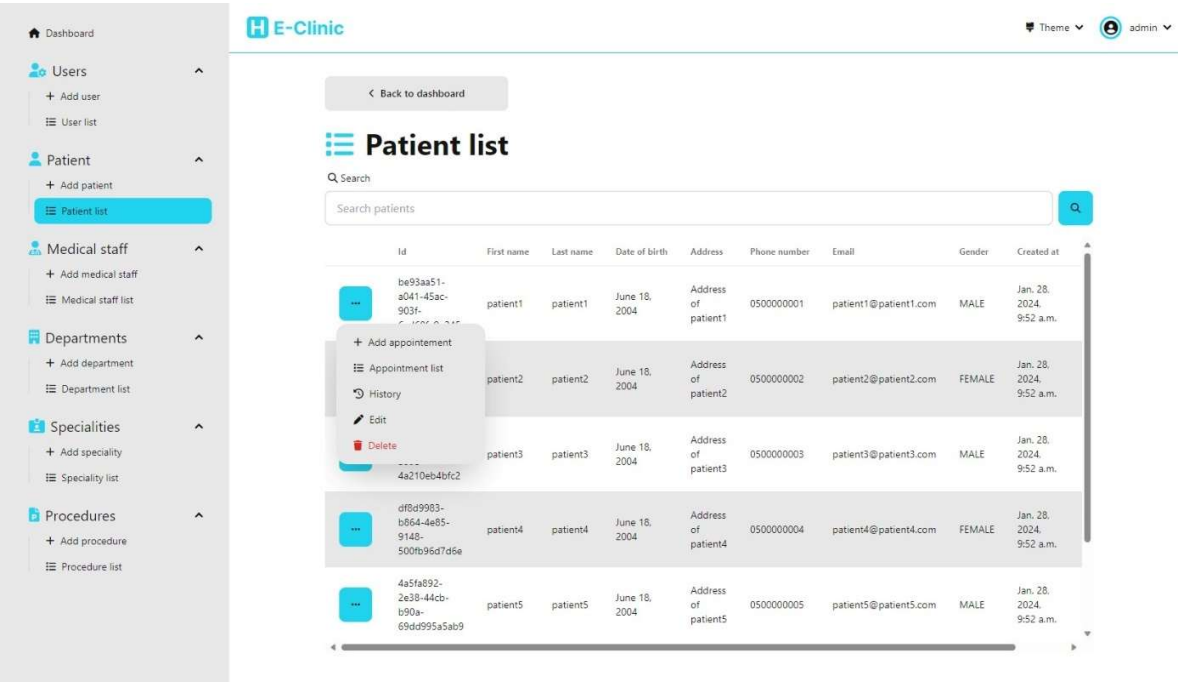
Add Patient

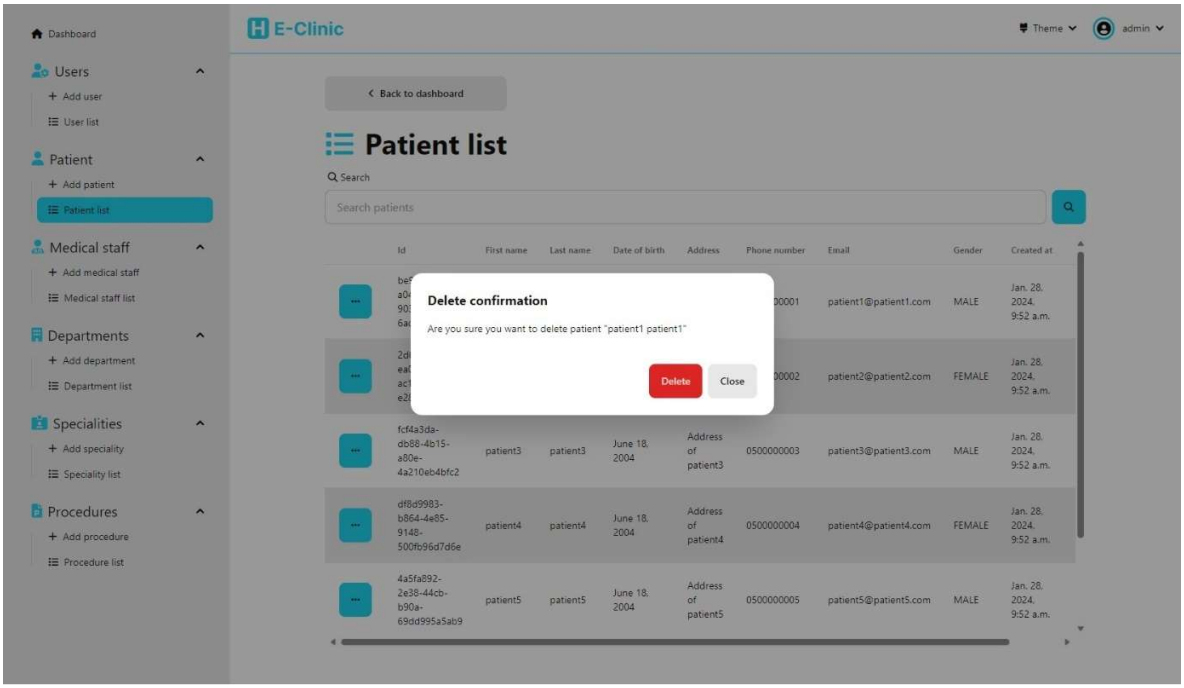
This page allow us to add a new patient to the system



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

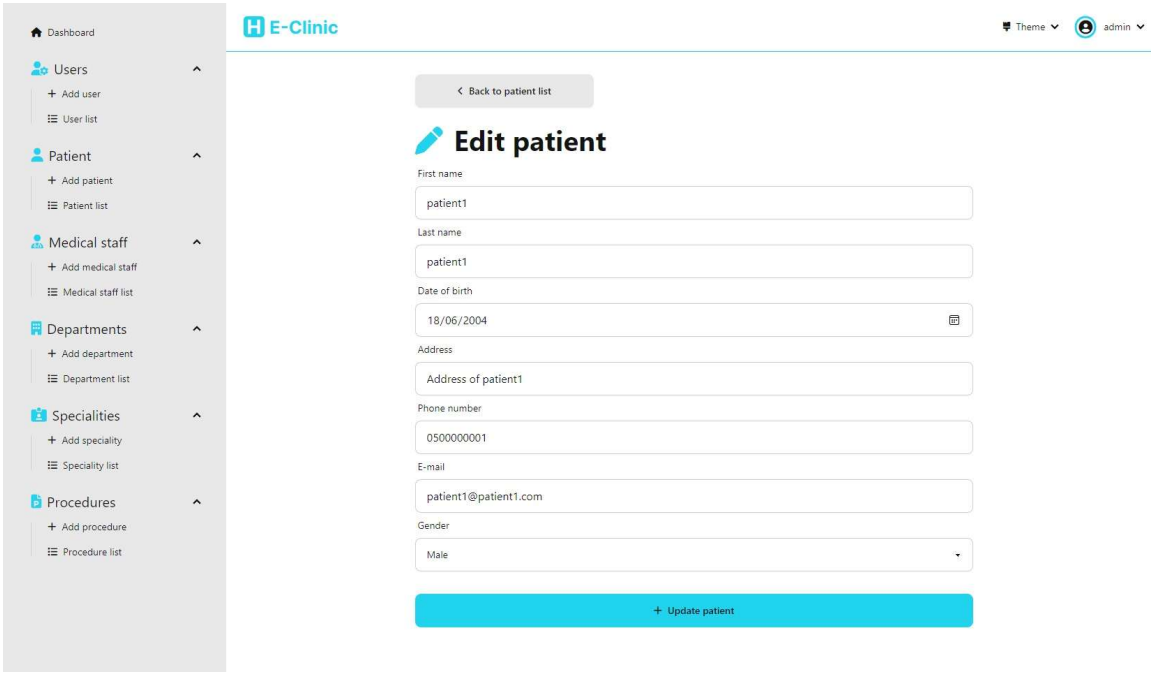




Edit patient

(accessible from patient list)

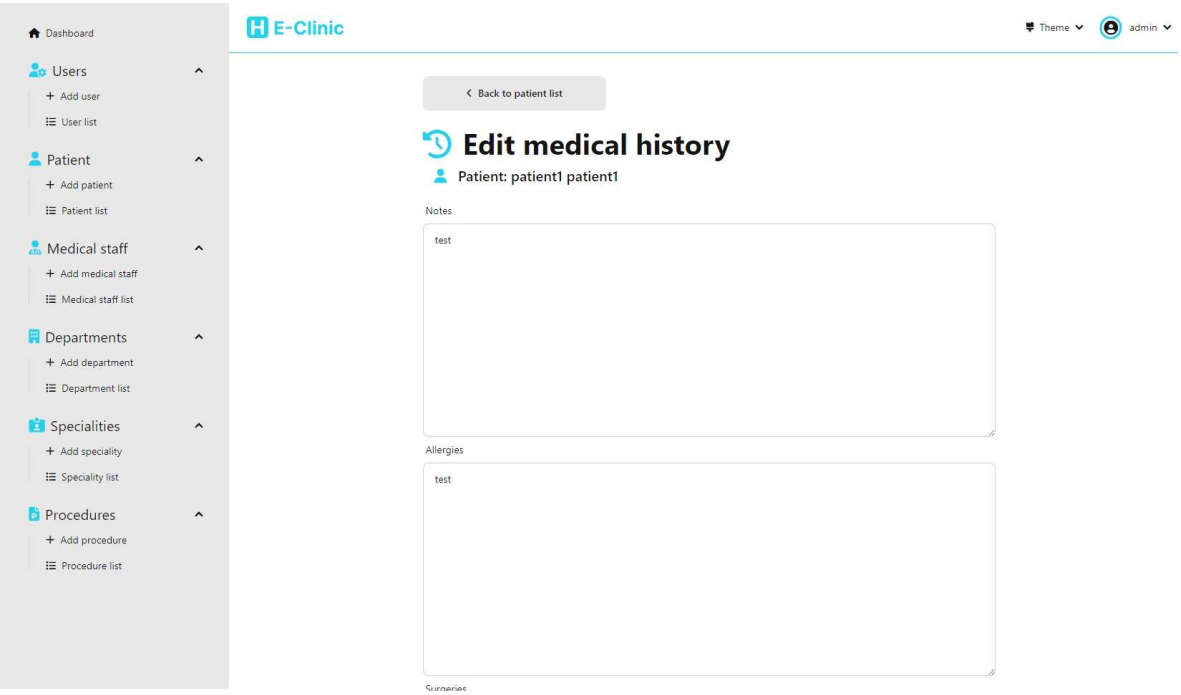
This page allow us to edit a patient

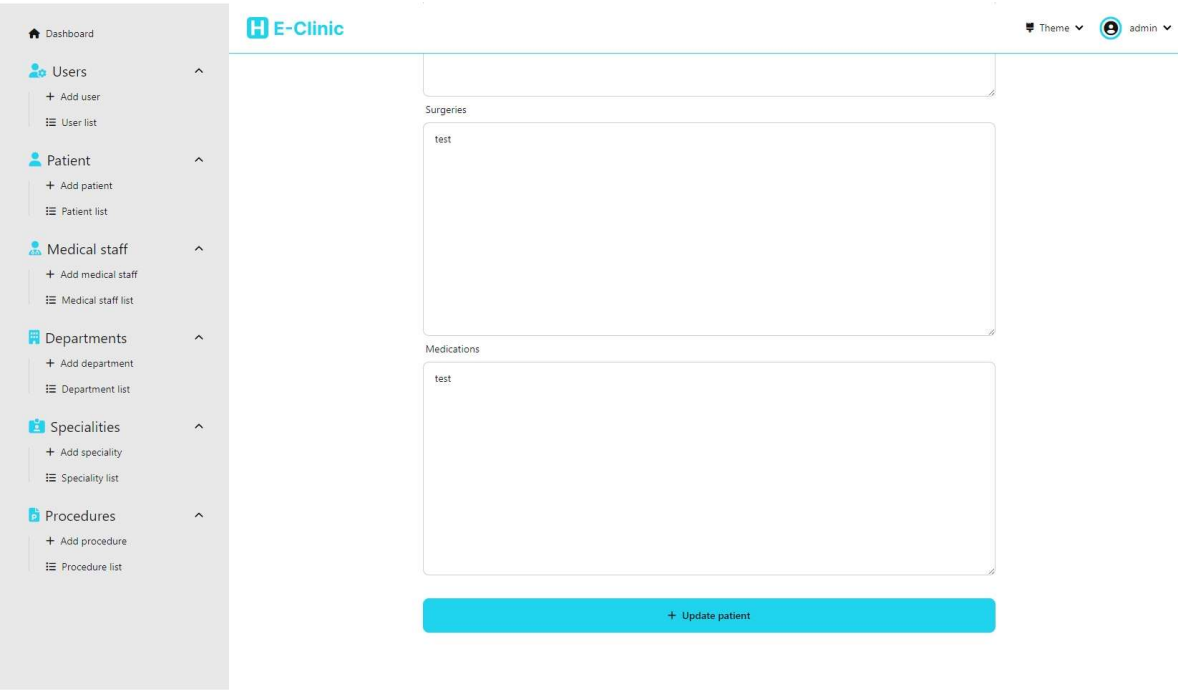


Medical history

(accessible from patient list)

This page allow us to edit the medical history of patient





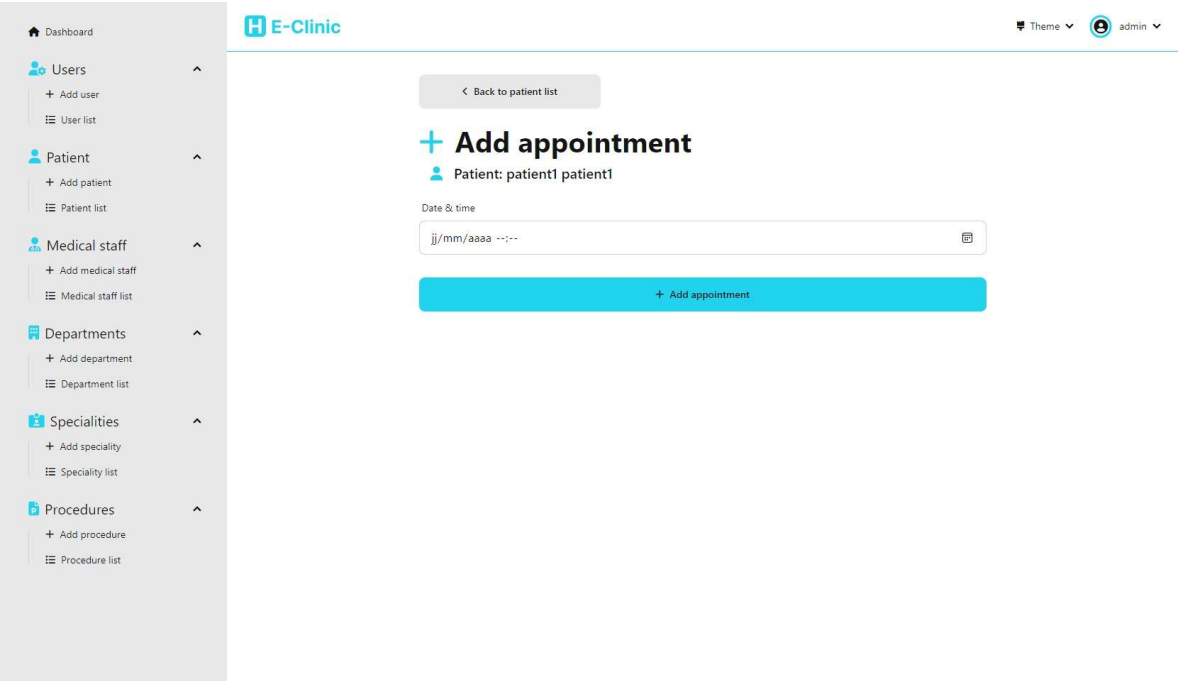
Appointment management

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

Add appointment

(accessible from patient list)

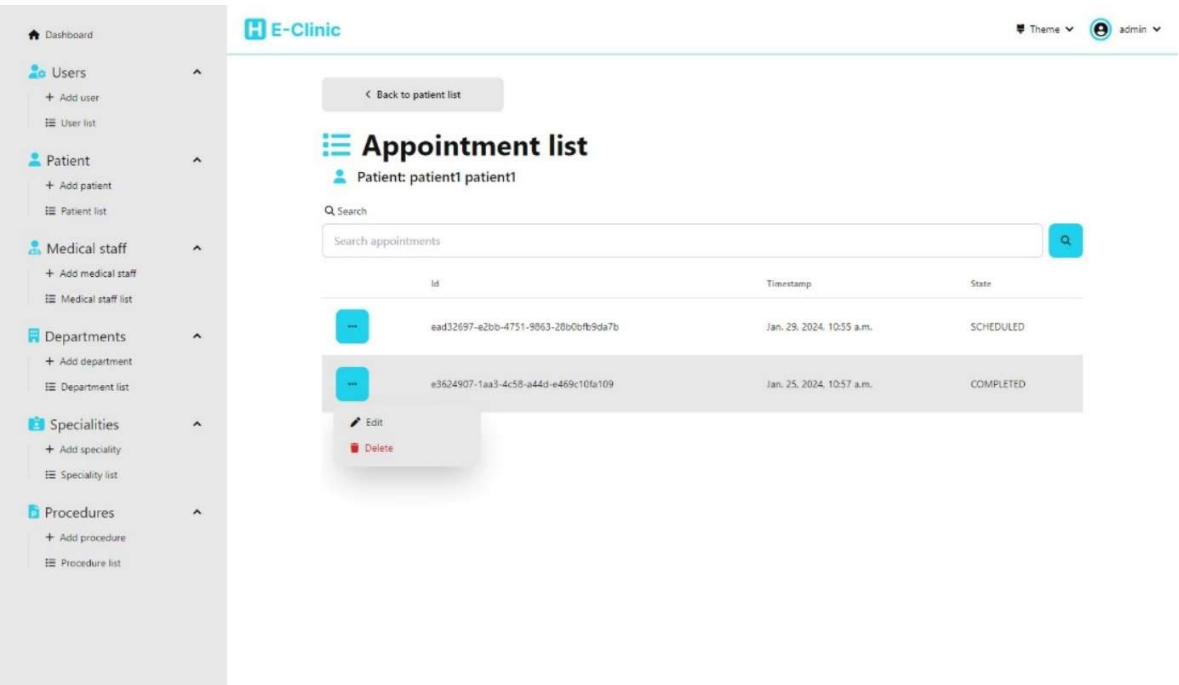
This page allow us to add an appointment for a patient (accessible via the menu shown before)



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



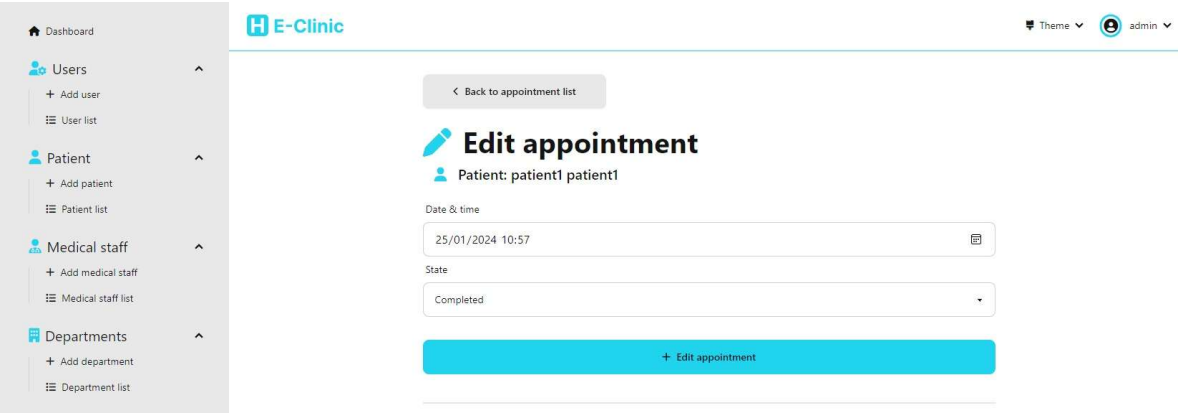
Edit appointment

(accessible from appointment list)

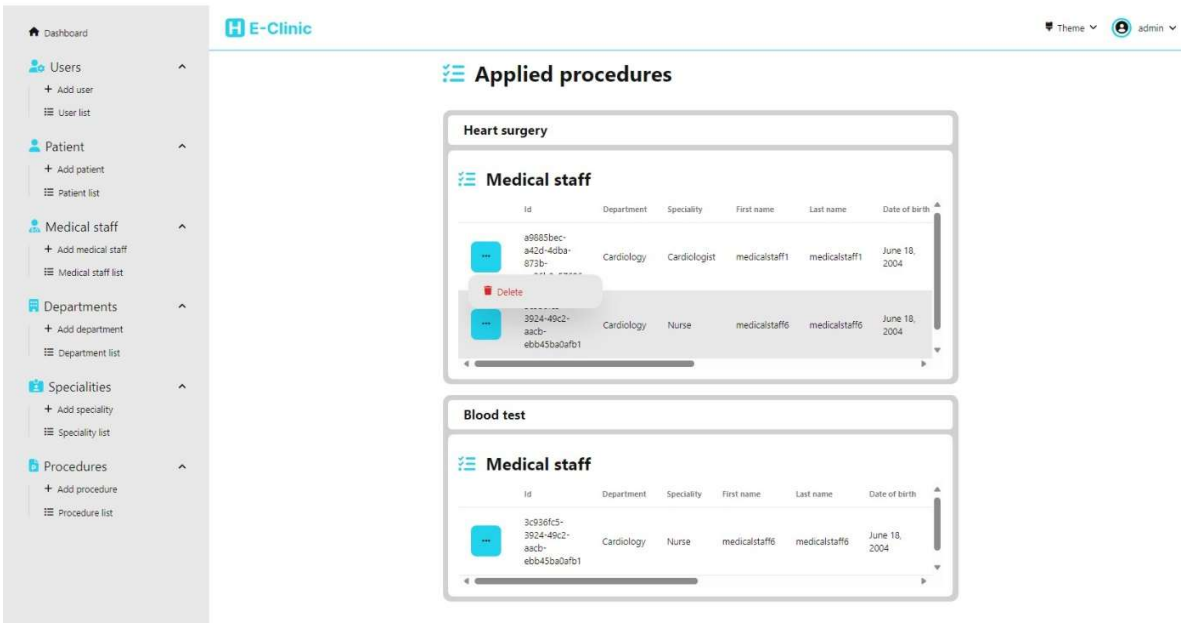
This page allows us to edit an appointment for a given patient

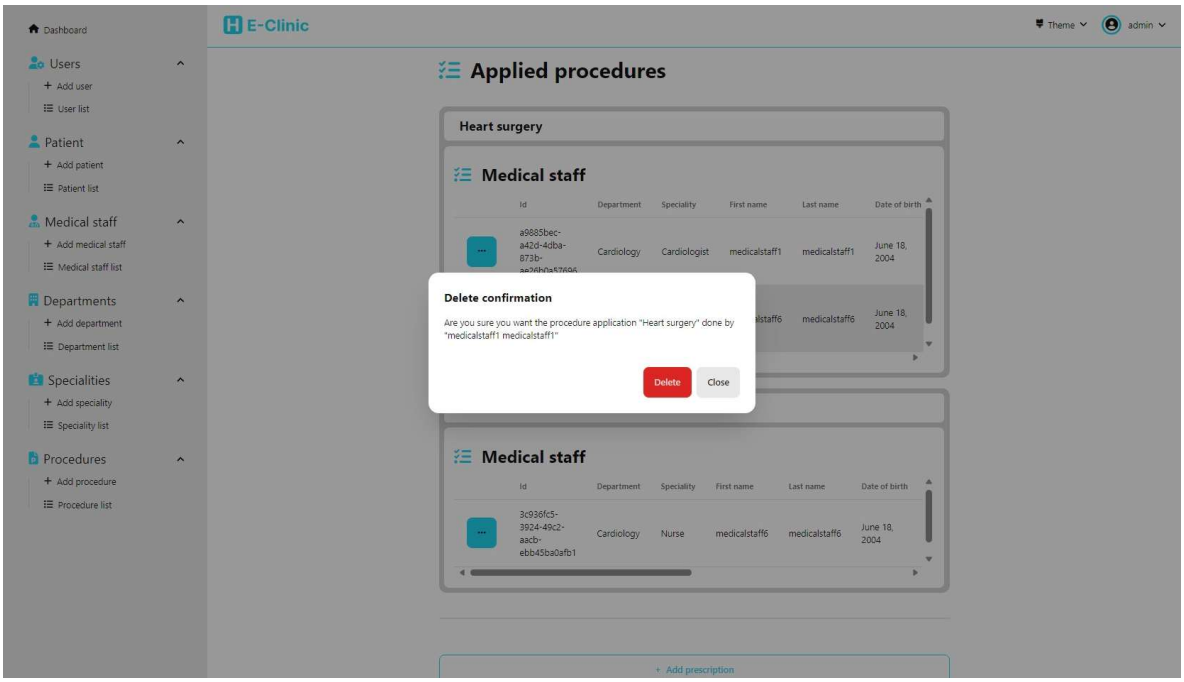
It is composed of 3 part:

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



- The 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


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
E-Clinic
Theme
admin
Applied procedures
Heart surgery
Medical staff
Id
Department
Speciality
First name
Last name
Date of birth
a9885bec-a42d-4dba-873b-
Cardiology
Cardiologist
medicalstaff1
medicalstaff1
June 18, 2004
Delete
3924-49c2-aacb-ebb45ba0afb1
Cardiology
Nurse
medicalstaff6
medicalstaff6
June 18, 2004
Blood test
Medical staff
3c936fc5-3924-49c2-aacb-ebb45ba0afb1
Cardiology
Nurse
medicalstaff6
medicalstaff6
June 18, 2004


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
E-Clinic
Theme
admin
Applied procedures
Heart surgery
Medical staff
Id
Department
Speciality
First name
Last name
Date of birth
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
Cardiology
Nurse
medicalstaff6
medicalstaff6
June 18, 2004
+ Add prescription

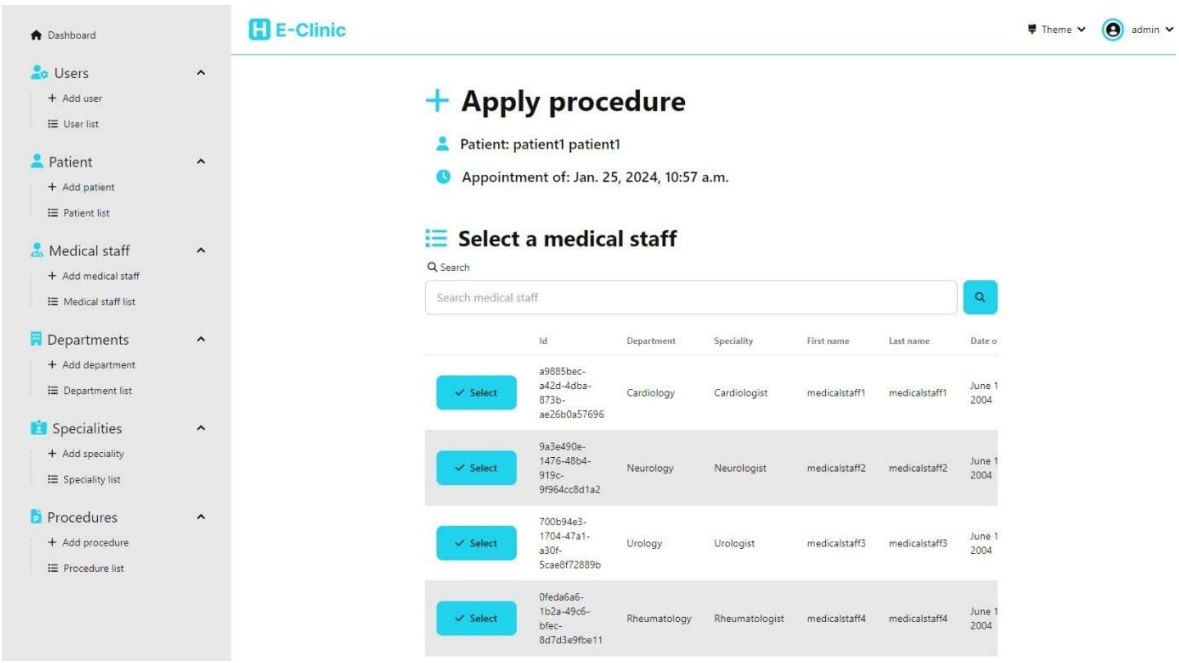
Procedure application management

Add procedure application

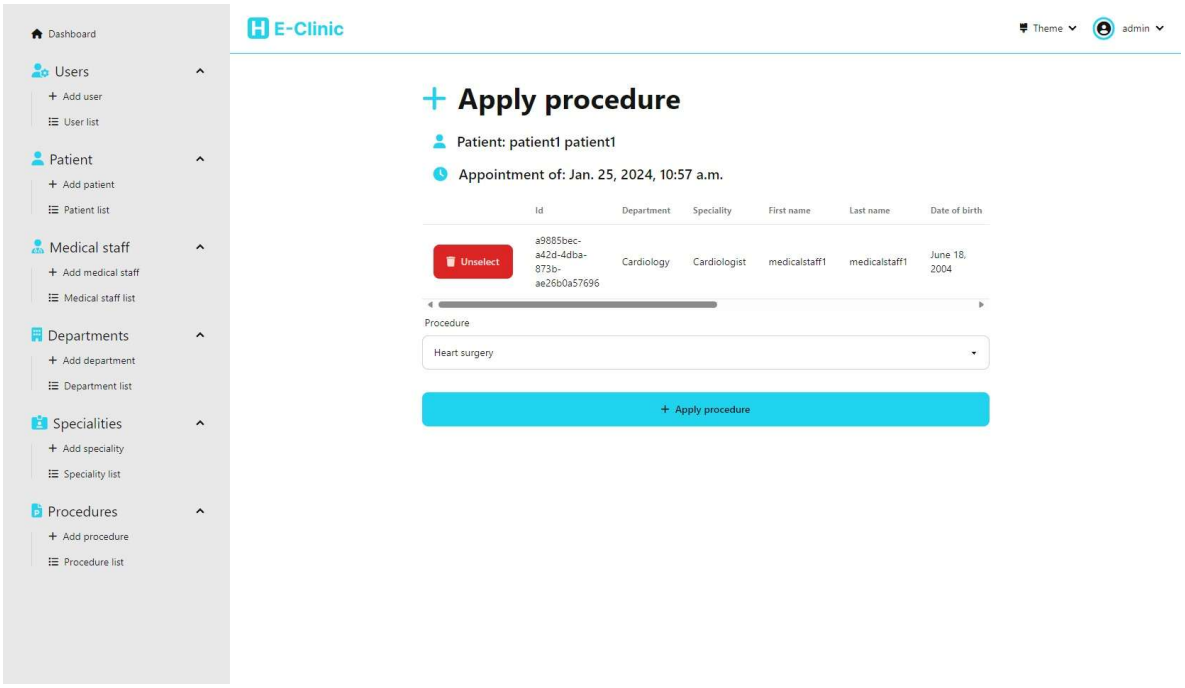
(accessible from edit appointment)

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

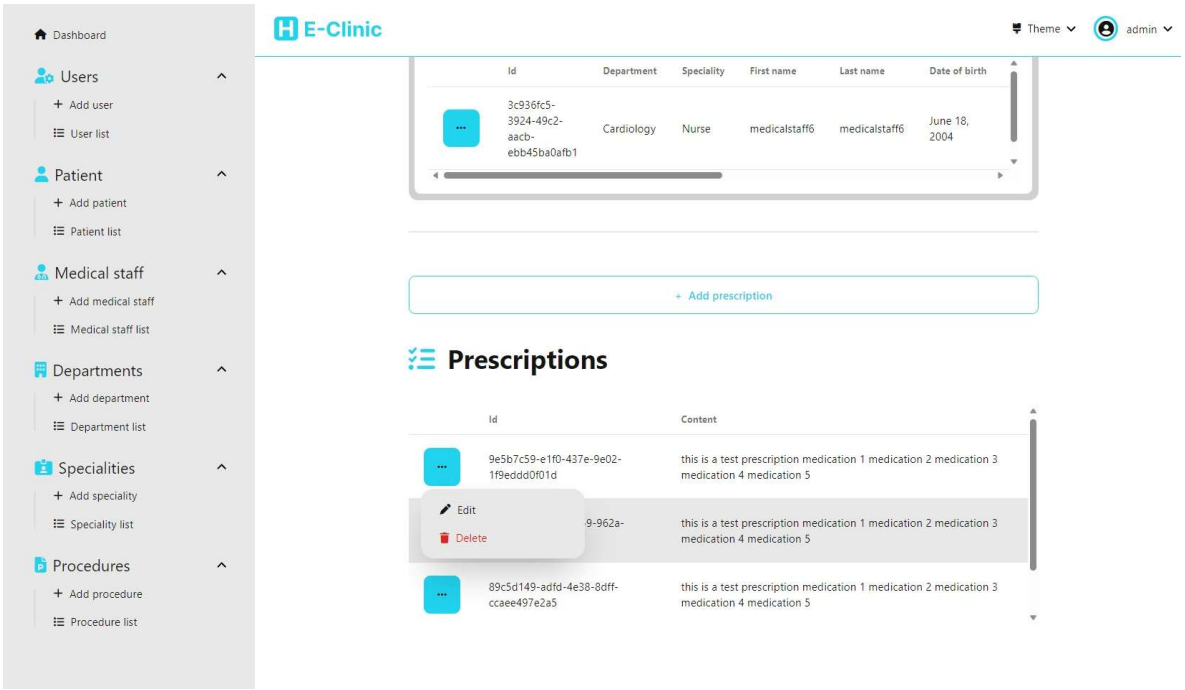
- First we have the list of available medical staff, we can select only one at a time, We can also search for medical staff

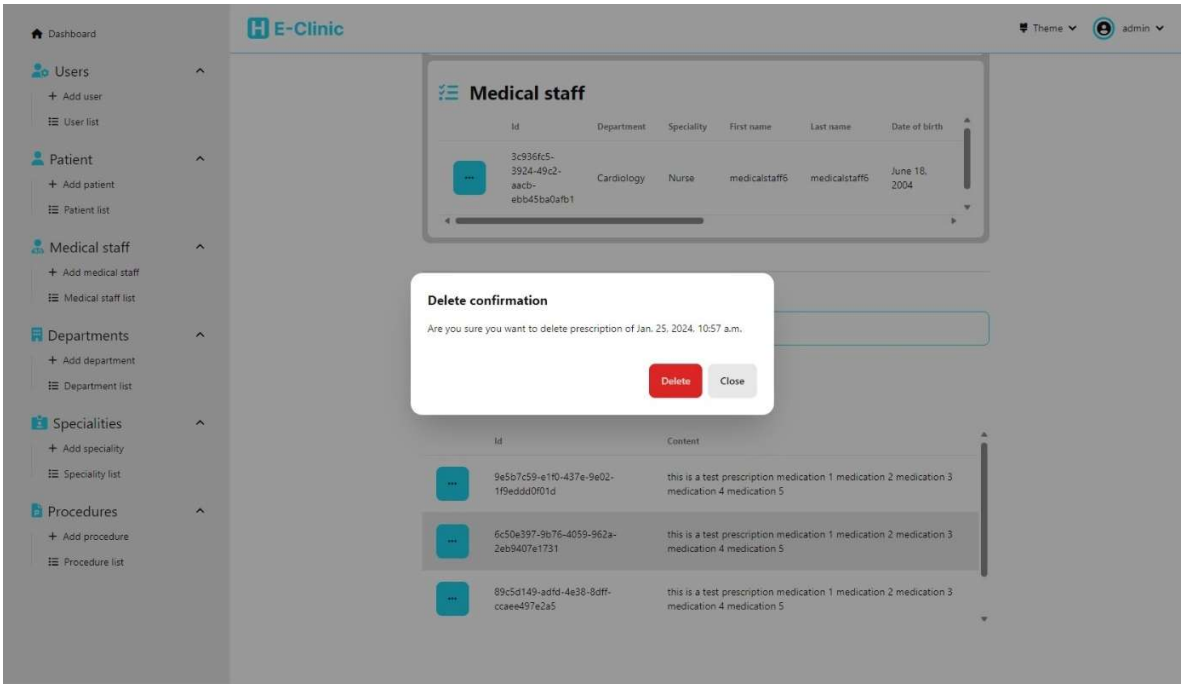


Once we select a medical staff, we need to chose the procedure to apply



- 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



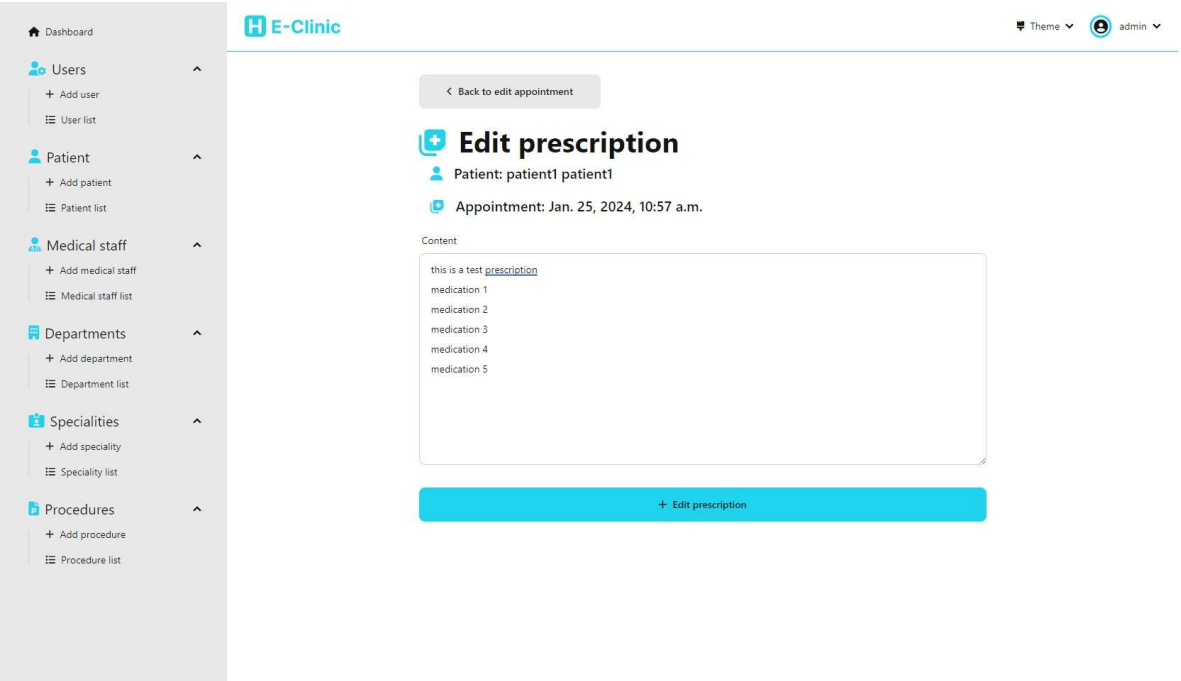


Prescription management

Edit prescription

(accessible from edit appointment)

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

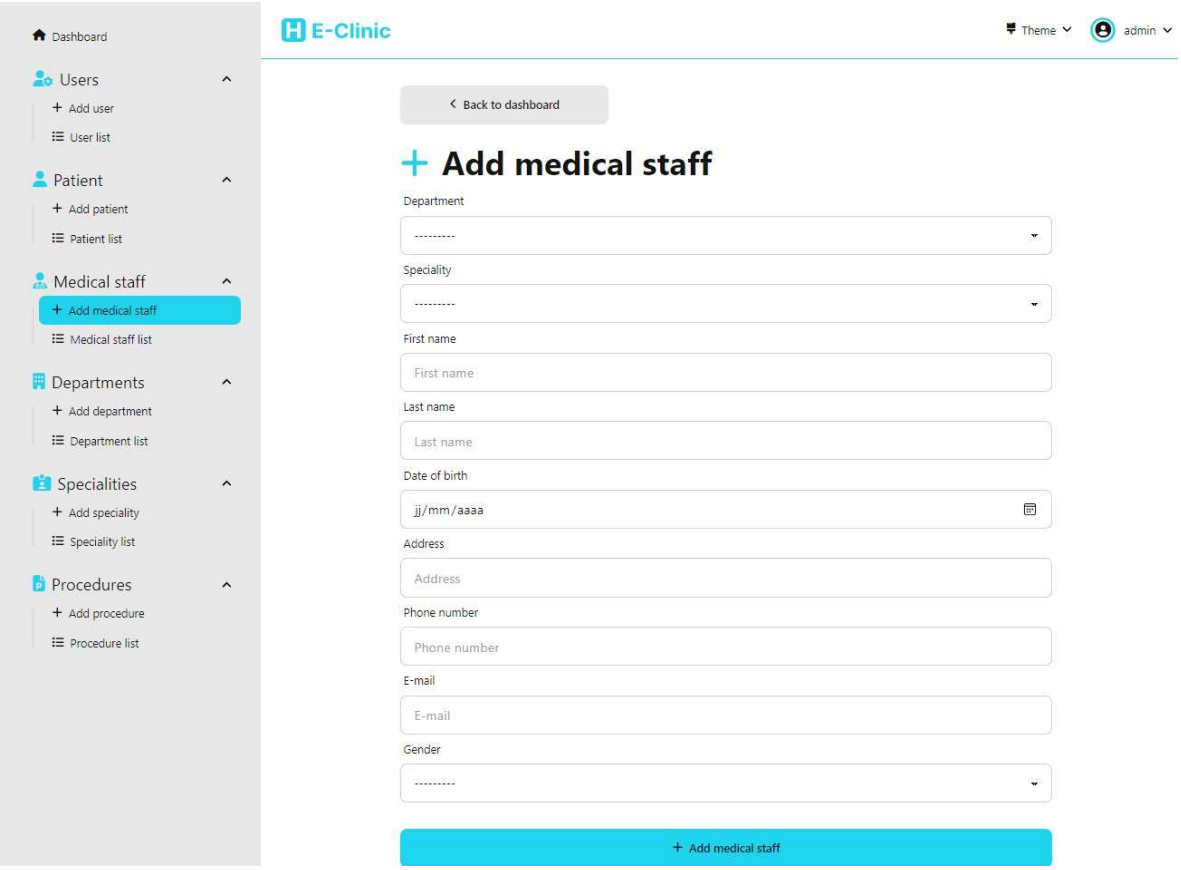


Medical staff management

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

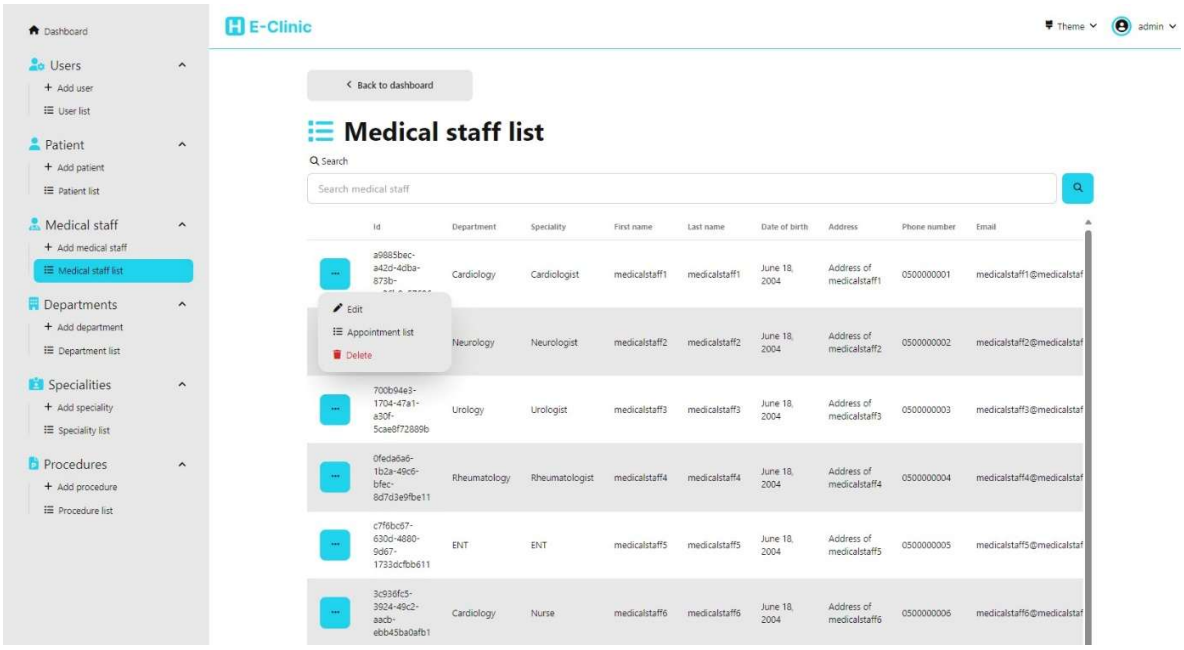
Add medical staff

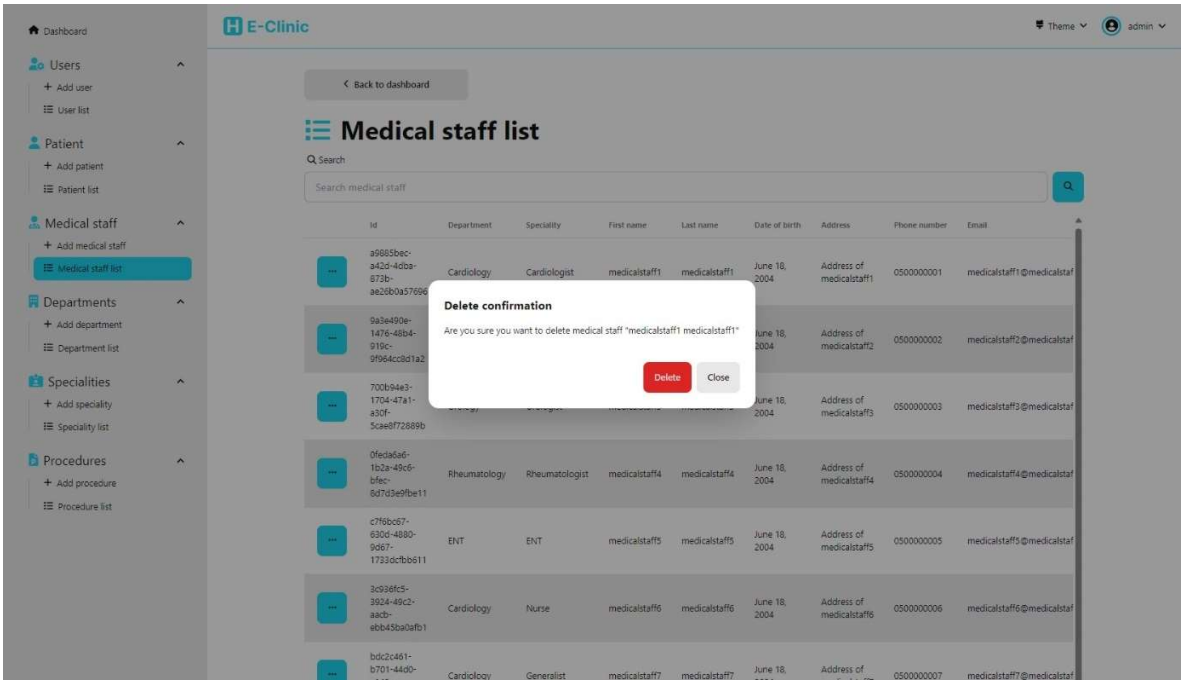
This page allows us to add a new medical staff to the system



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

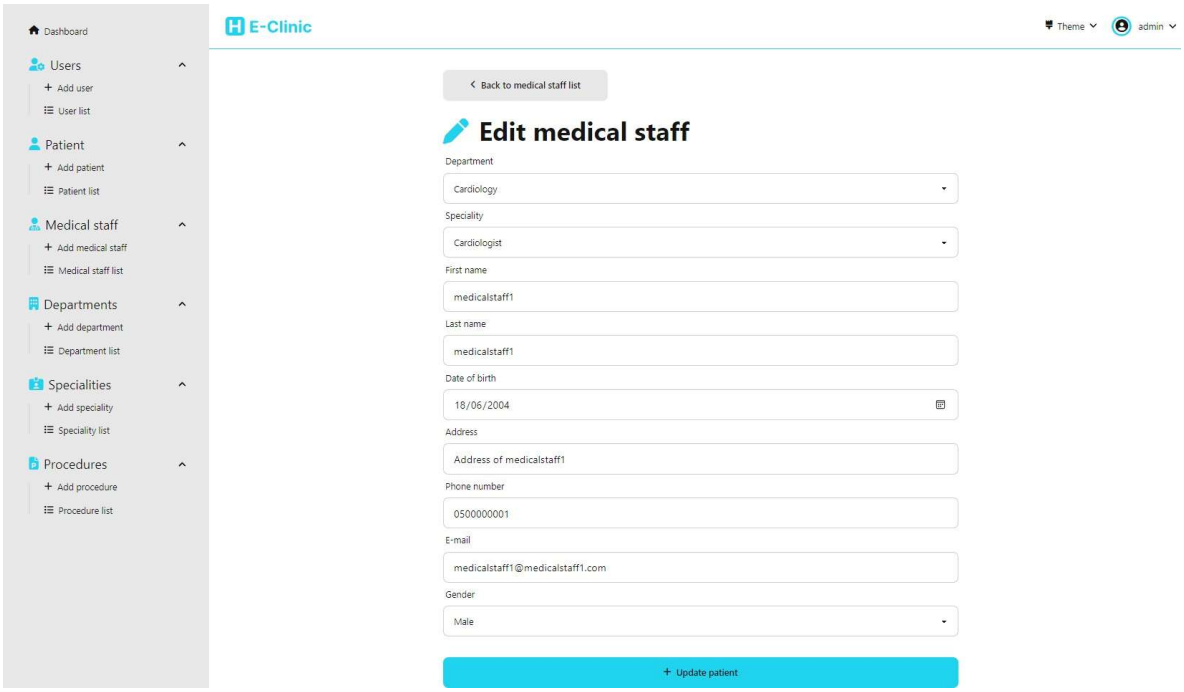




Edit medical staff

(accessible from medical staff list)

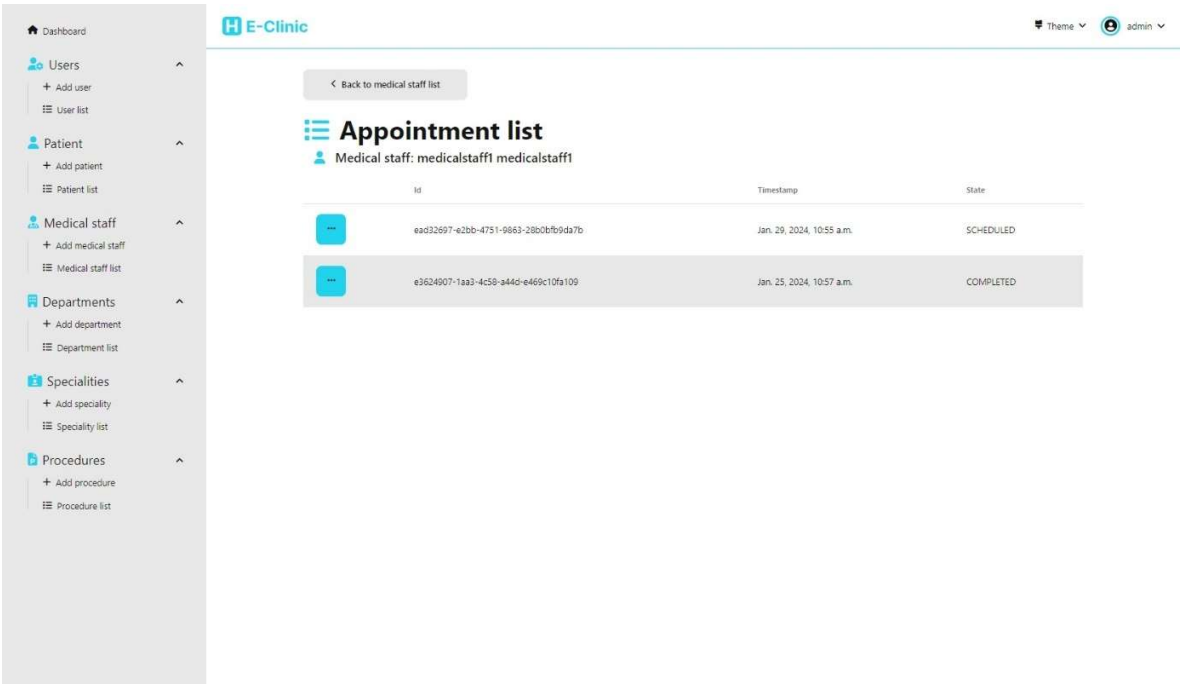
This page allows us to edit a given medical staff



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

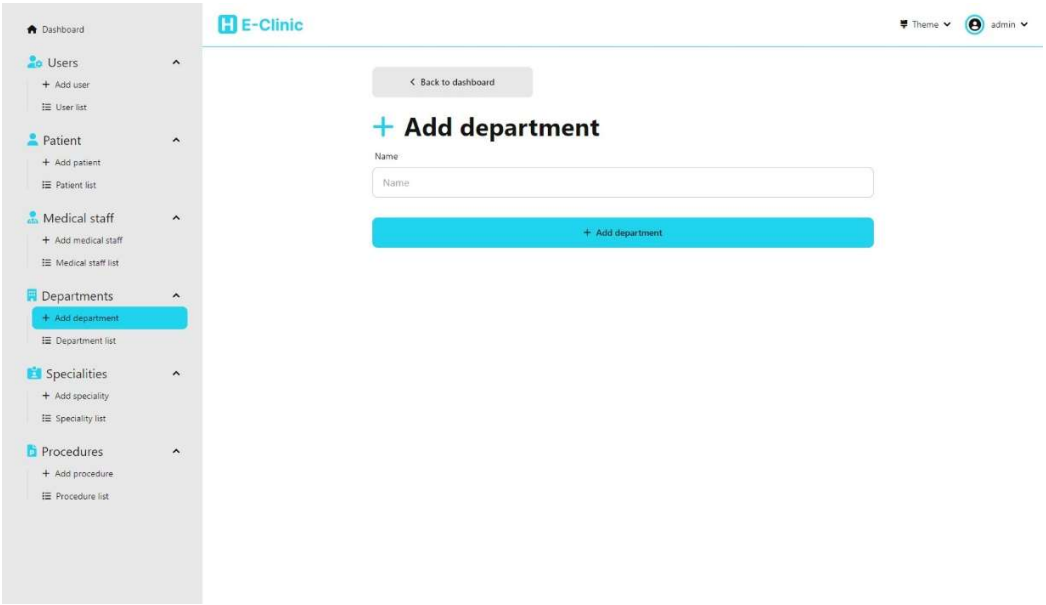


Department management

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

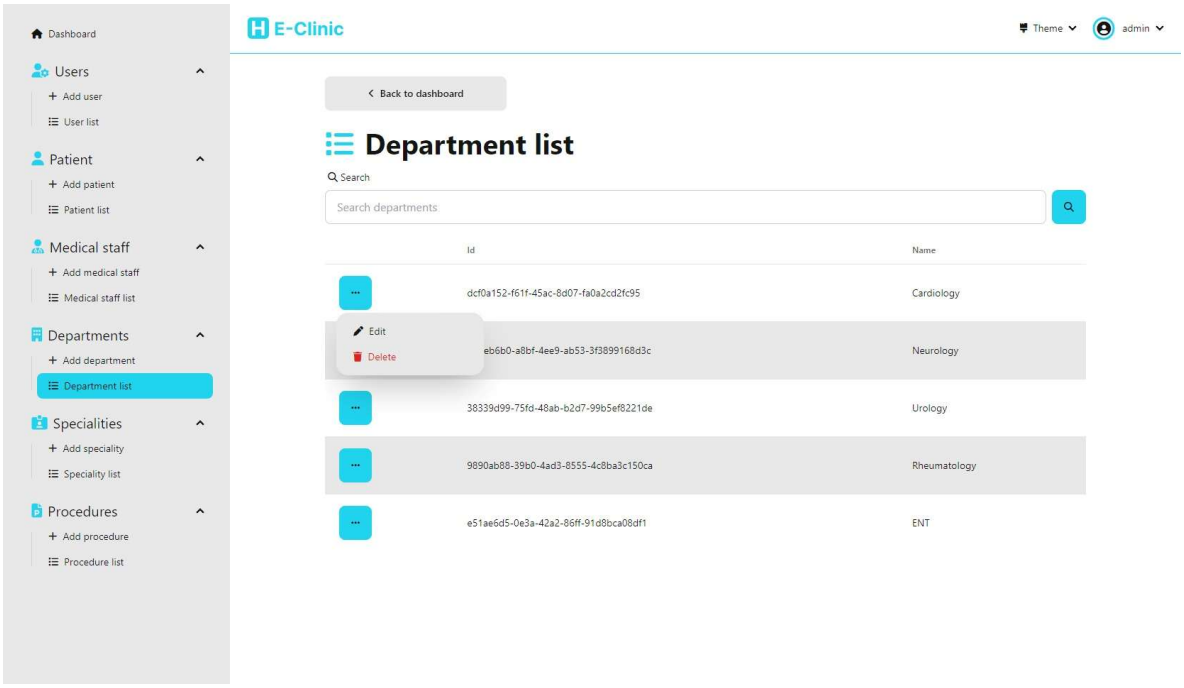
Add department

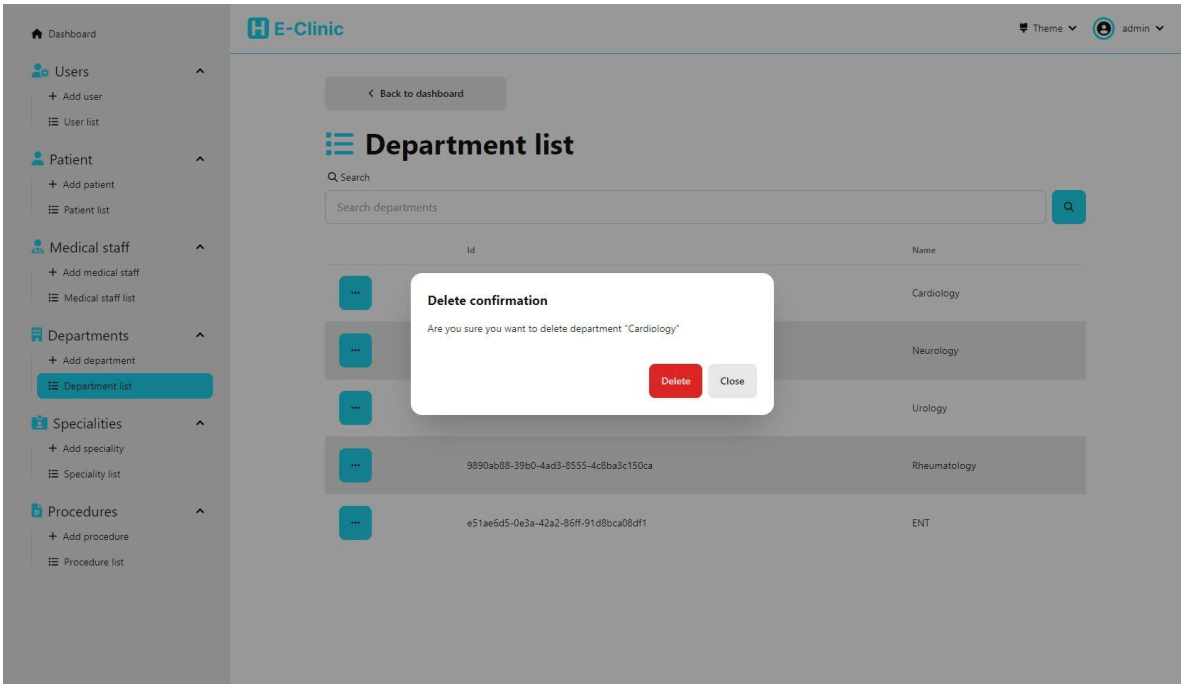
This page allows us to add a new medical department to the system



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.

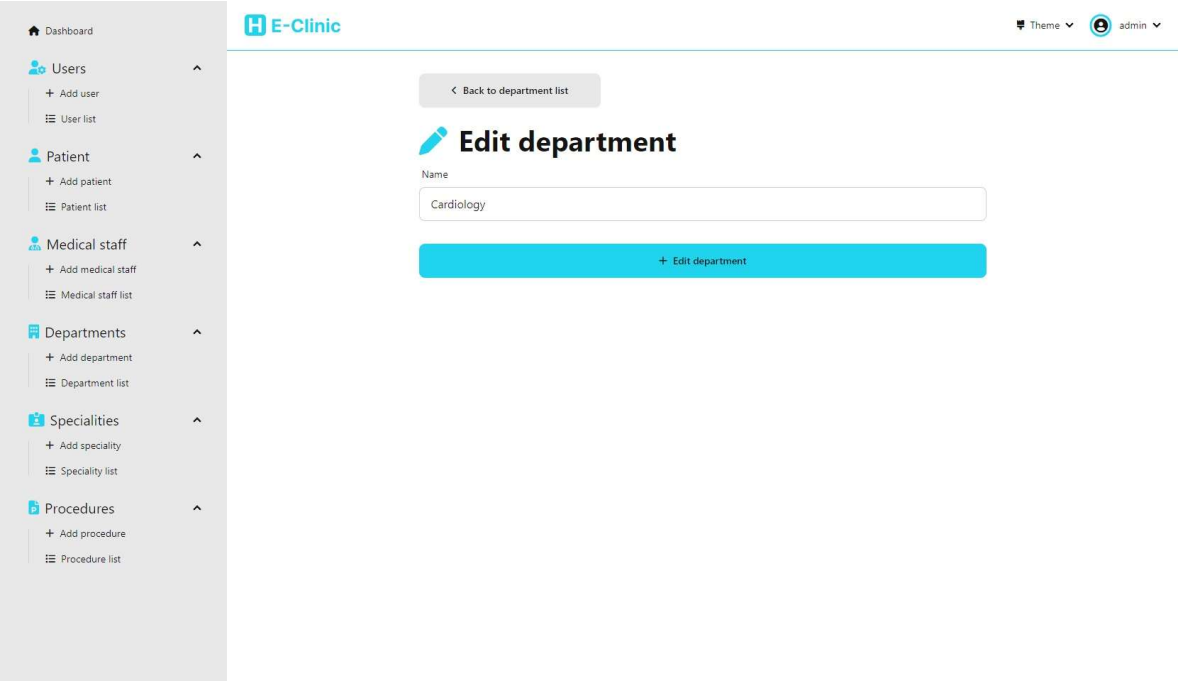

Dashboard
Users
Add user
User list
Patient
Add patient
Patient list
Medical staff
Add medical staff
Medical staff list
Departments
Add department
Department list
Specialities
Add speciality
Speciality list
Procedures
Add procedure
Procedure list
E-Clinic
Theme
admin
Back to dashboard
Department list
Search
Search departments
Id
Name
dcf0a152-f61f-45ac-8d07-fa0a2cd2fc95
Cardiology
Edit
Delete
eb6b0-a8bf-4ee9-ab53-3f3899168d3c
Neurology
38339d99-75fd-48ab-b2d7-99b5ef8221de
Urology
9890ab88-39b0-4ad3-8555-4c8ba3c150ca
Rheumatology
e51ae6d5-0e3a-42a2-86ff-91d8bca08df1
ENT


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
E-Clinic
Theme
admin
Back to dashboard
Department list
Search
Search departments
Id
Name
Delete confirmation
Are you sure you want to delete department "Cardiology"
Delete
Close
Cardiology
Neurology
Urology
9890ab88-39b0-4ad3-8555-4c8ba3c150ca
Rheumatology
e51ae6d5-0e3a-42a2-86ff-91d8bca08df1
ENT

Edit department

(accessible from department list)

This page allows us to edit a given department

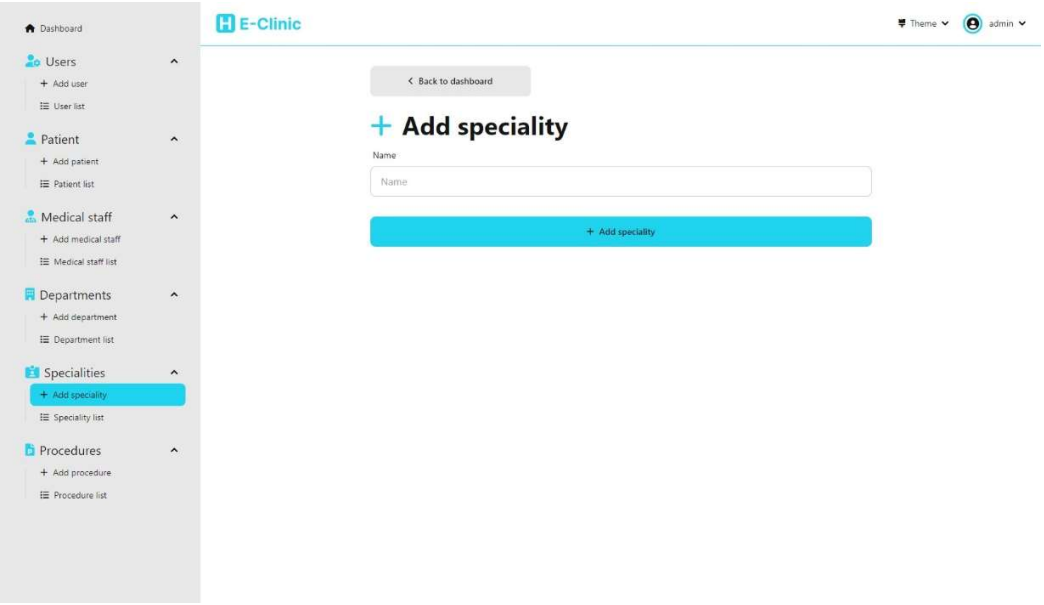


Speciality management

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

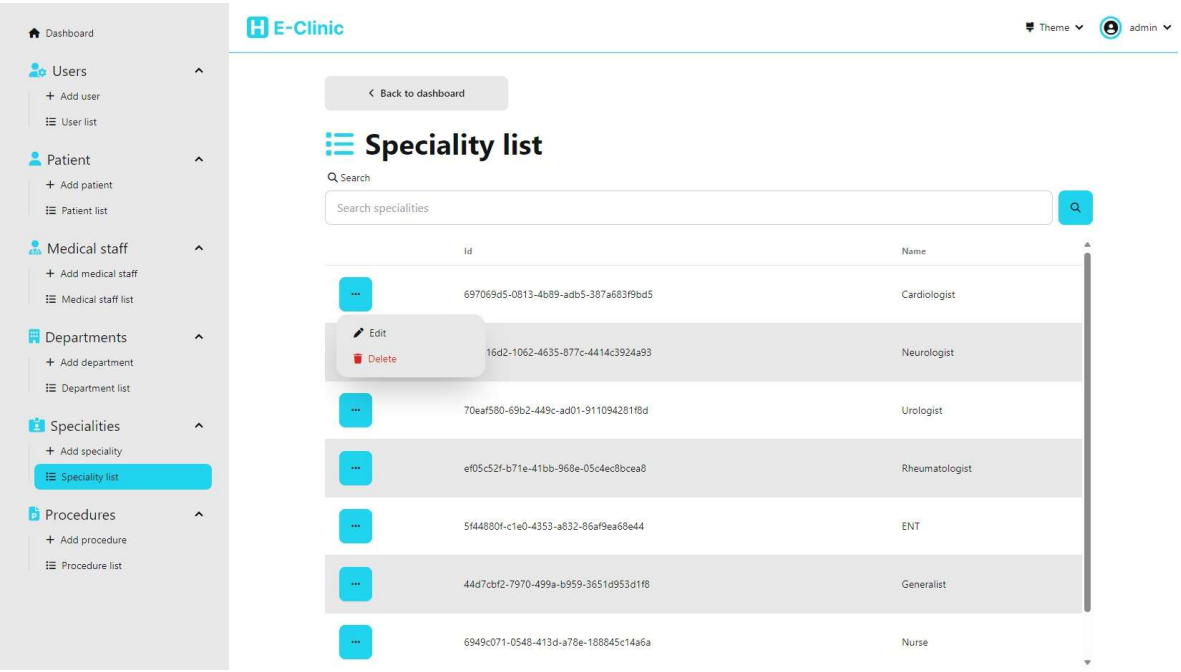
Add specialty

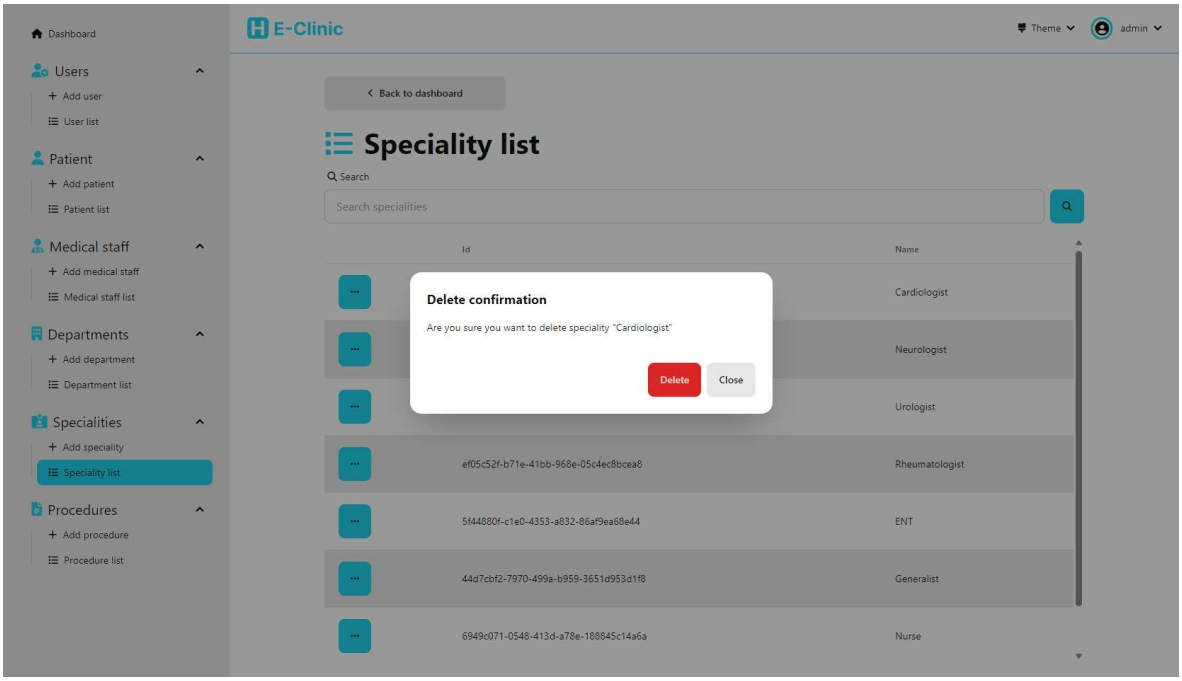
This page allows us to add a new speciality to the system



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.

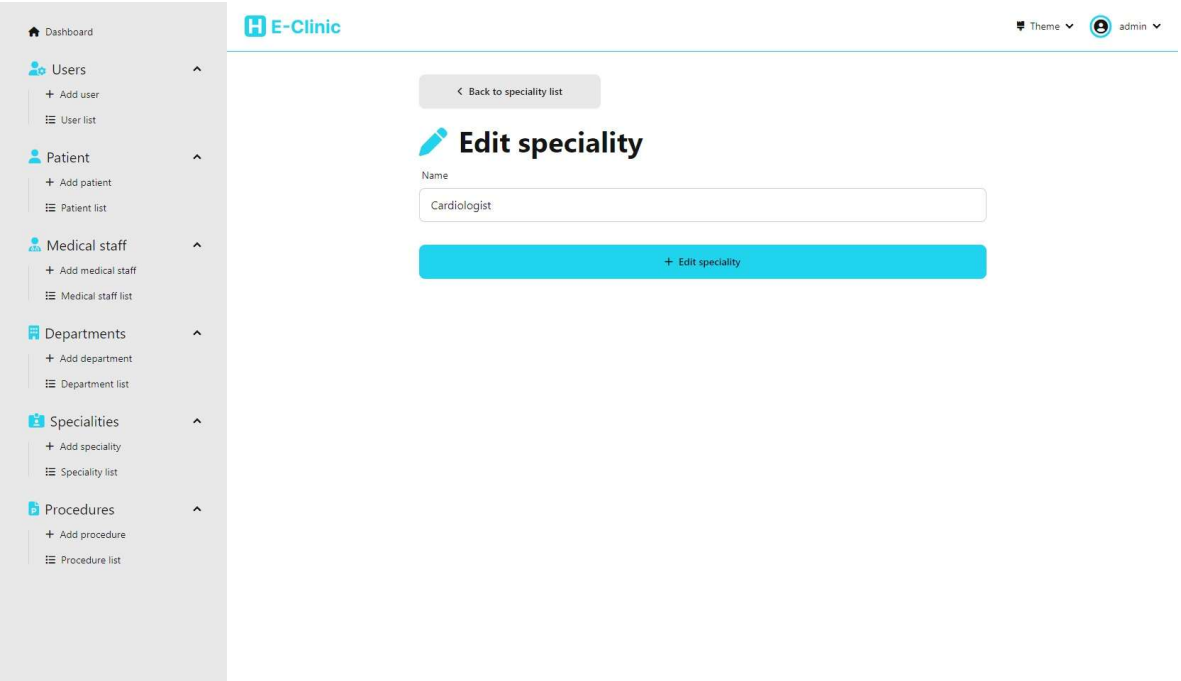




Edit speciality

(accessible from speciality list)

This page allows us to edit a given speciality

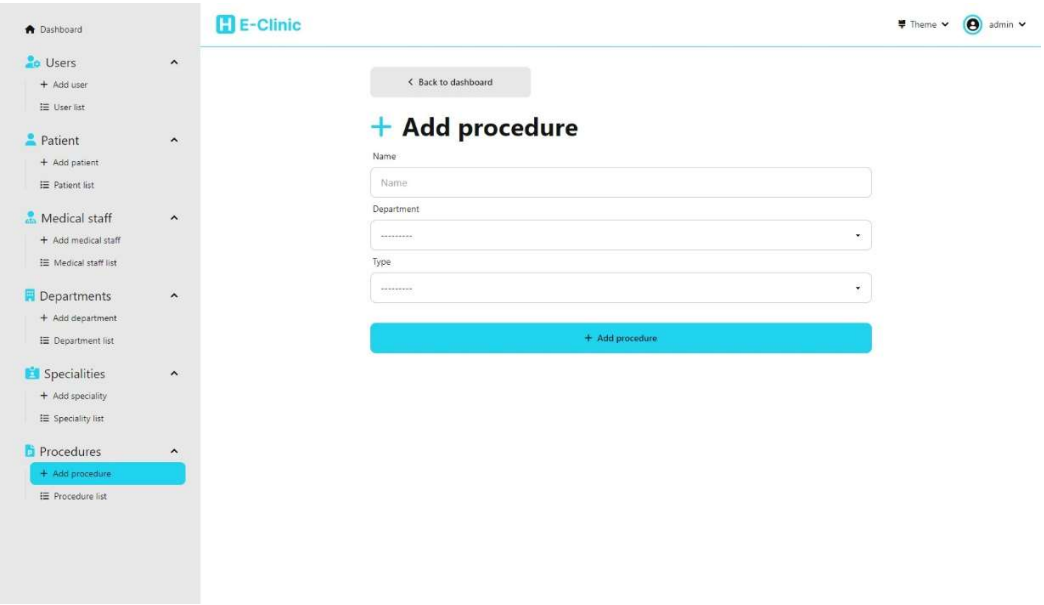


Procedure management

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

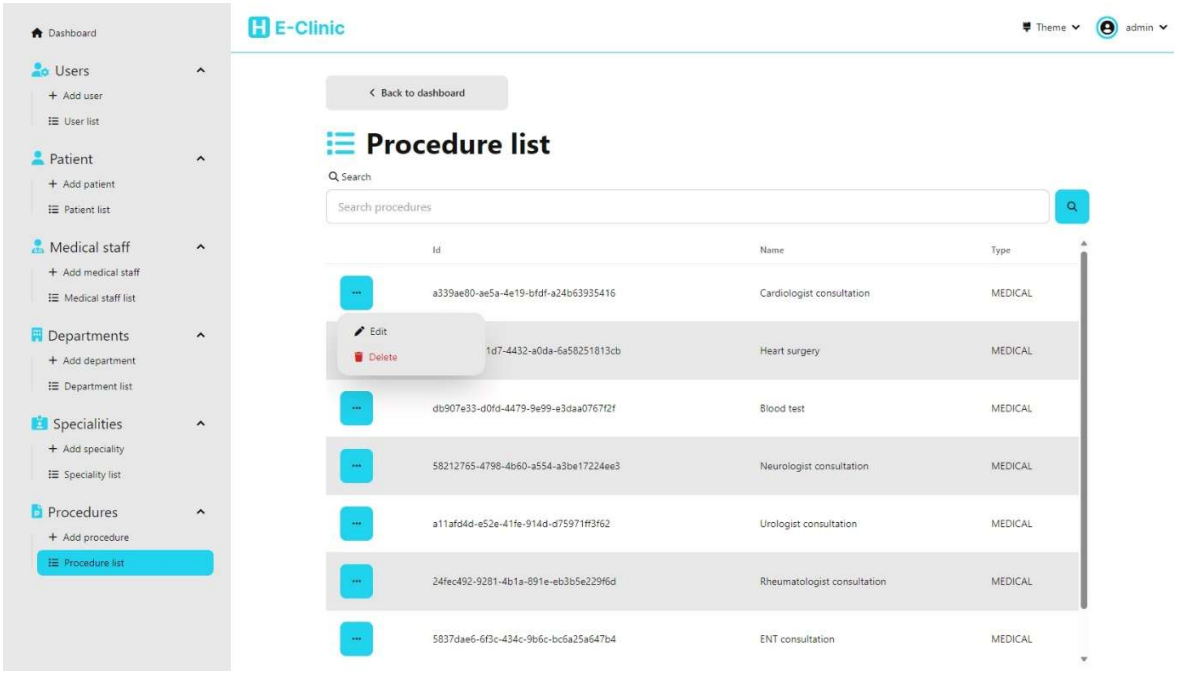
Add procedure

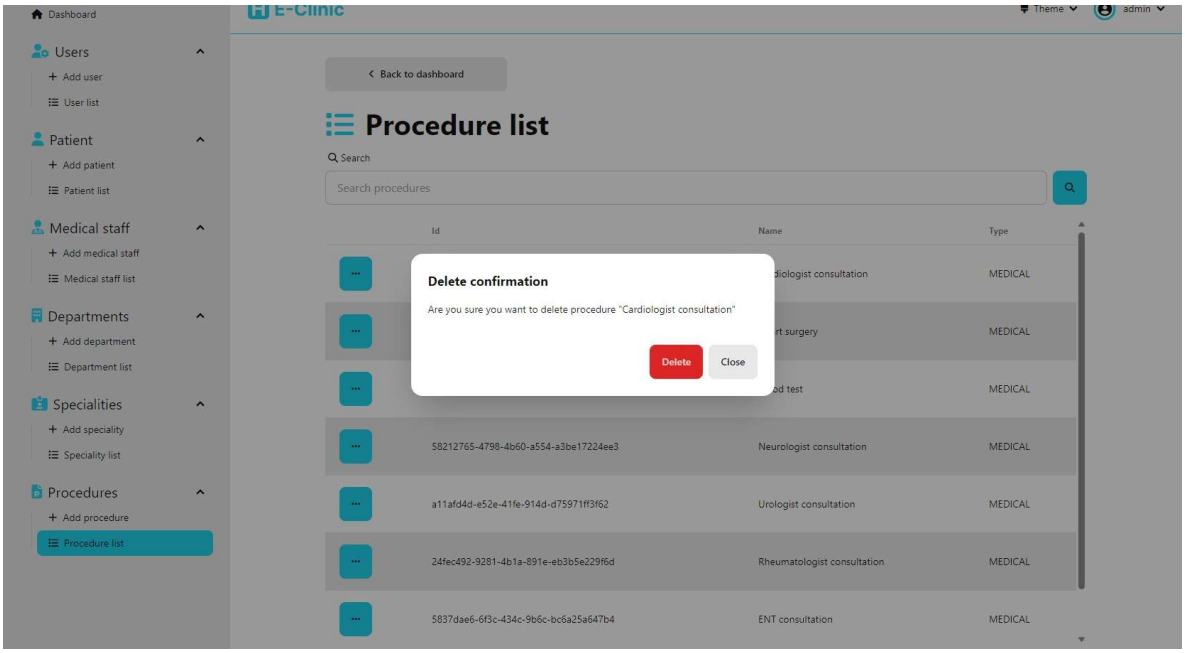
This page allows us to add a new procedure to the system



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.

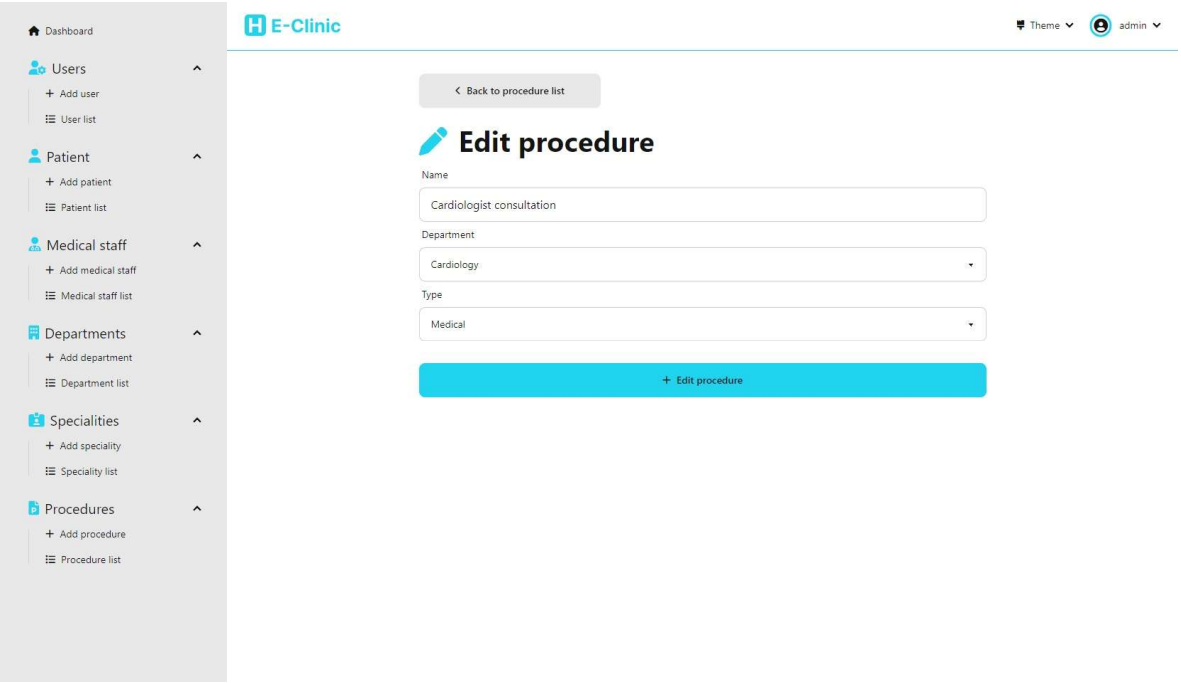




Edit procedure

(accessible from procedure list)

This page allows us to edit a given procedure



Code structure

Folders tree

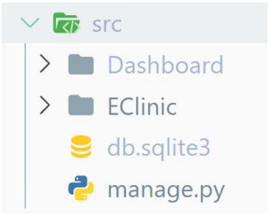
Root folder

- 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 dtabase with test data

- migrations: contains db migrations

- statis: 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

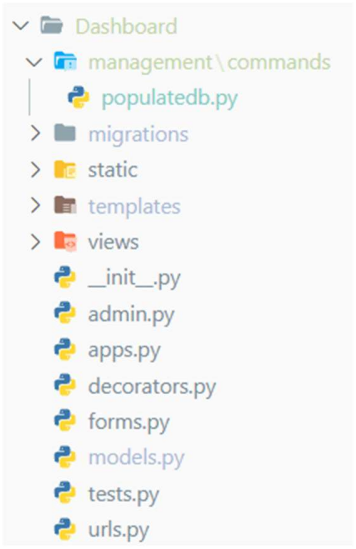
- decorators.py: Contains utility python decorators that are used in the app

- forms: Contains Django form for the app

- models: Contains Django model for the app

- test.py Contains test code

- urls.py: Contains URLs configuration for the app

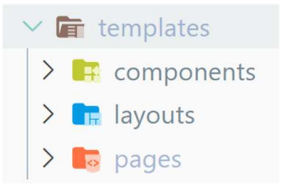


Templates

- 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

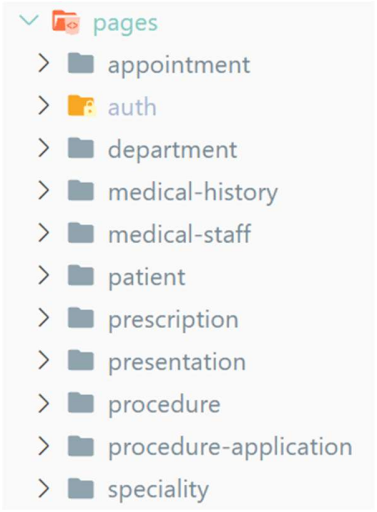
- prescription: Contains prescription related pages

- 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

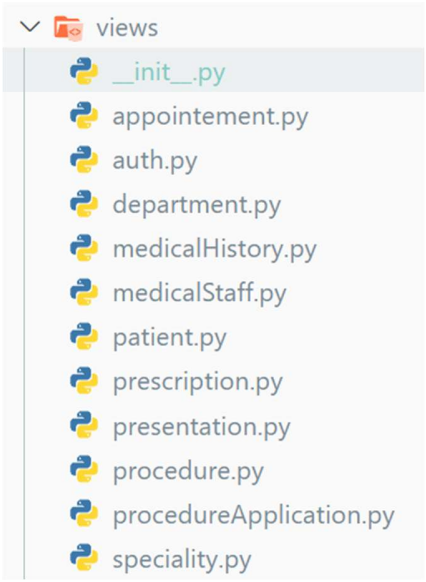
- prescription.py: Contains prescription related views

- presentation.py: Contains presentation related views

- procedureApplication.py: Contains procedure applications related views

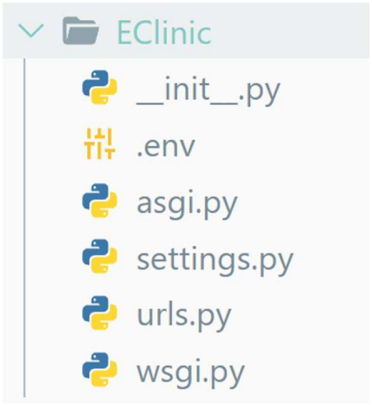
- procedures.py: Contains procedure related views

- 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

```python
from django.urls import path
from .views import medicalStaff, presentation, auth, patient, medicalHistory, department, procedure, speciality, appointement, 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/', appointement.addAppointement, name='add-appointment'),
  path('dashboard/appointments/edit/<str:appointmentId>/', appointement.editAppointment, name='edit-appointment'),
  path('dashboard/appointments/patient/<str:patientId>/', appointement.appointmentListPatient, name='appointment-list-patient'),
  path('dashboard/appointments/medical-staff/<str:medicalStaffId>/', appointement.appointmentListMedicalStaff, name='appointment-list-medical-staff'),
  path('dashboard/appointments/delete/<str:appointmentId>/', appointement.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

```python
class SignInForm(forms.Form):
  username = forms.CharField()
  password = forms.CharField()

class SearchForm(forms.Form):
  search  = forms.CharField(required=False)

class PatientForm(forms.ModelForm):
  class Meta:
    model = Patient
    fields = '__all__'

class MedicalHistoryForm(forms.ModelForm):
  class Meta:
    model = MedicalHistory
    fields = '__all__'

class MedicalStaffForm(forms.ModelForm):
  class Meta:
    model = MedicalStaff
    fields = '__all__'

class DepartmentForm(forms.ModelForm):
  class Meta:
    model = Department
    fields = '__all__'

class SpecialityForm(forms.ModelForm):
  class Meta:
    model = Speciality
    fields = '__all__'

class ProcedureForm(forms.ModelForm):
  class Meta:
    model = Procedure
    fields = '__all__'

class ProcedureApplicationForm(forms.ModelForm):
  class Meta:
    model = ProcedureApplication
    fields = '__all__'

class AssignMedicalStaffForm(forms.Form):
  medicalStaffId = forms.CharField()
  remove = forms.BooleanField(required=False)

class AppointmentForm(forms.ModelForm):
  class Meta:
    model = Appointment
    fields = '__all__'

class PrescriptionForm(forms.ModelForm):
  class Meta:
    model = Prescription
    fields = '__all__'
```

Dashboard models

```python
from django.db import models
import uuid
from django.core.validators import EmailValidator, RegexValidator

# Create your models here.
class Genders(models.TextChoices):
  MALE = 'MALE'
  FEMALE = 'FEMALE'

class ProcedureType(models.TextChoices):
  MEDICAL = 'MEDICAL',
  ADMINISTRATIVE = 'ADMINISTRATIVE'

class AppointmentState(models.TextChoices):
  SCHEDULED = 'SCHEDULED'
  COMPLETED = 'COMPLETED'
  CANCELED = 'CANCELED'

class Patient(models.Model):
  id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
  firstName = models.CharField(max_length=256, null=False)
  lastName = models.CharField(max_length=256, null=False)
  dateOfBirth = models.DateField(null=False)
  address = models.CharField(max_length=256, null=False)
  phoneNumber = models.CharField(max_length=20, null=False, unique=True, validators=[RegexValidator(r'^(00213|\+213|0)(5|6|7)\d{8}$')])
  email = models.CharField(max_length=254, null=False, unique=True, validators=[EmailValidator()])
  gender = models.CharField(max_length=10, choices=Genders.choices, null=False)
  createdAt = models.DateTimeField(auto_now_add=True)
  updatedAt = models.DateTimeField(auto_now=True)

  def __str__(self):
     return f"{self.lastName} {self.firstName}"

class MedicalHistory(models.Model):
  id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
  patient = models.OneToOneField('Patient', on_delete=models.CASCADE)
  notes = models.TextField(null=True, blank=True)
  allergies = models.TextField(null=True, blank=True)
  surgeries = models.TextField(null=True, blank=True)
  medications = models.TextField(null=True, blank=True)
  updatedAt = models.DateTimeField(auto_now=True, blank=True)

class MedicalStaff(models.Model):
  id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
  department = models.ForeignKey('Department', on_delete=models.CASCADE)
  speciality = models.ForeignKey('Speciality', on_delete=models.CASCADE)
  firstName = models.CharField(max_length=256, null=False)
  lastName = models.CharField(max_length=256, null=False)
  dateOfBirth = models.DateField(null=False)
  address = models.CharField(max_length=256, null=False)
  phoneNumber = models.CharField(max_length=20, null=False, unique=True, validators=[RegexValidator(r'^(00213|\+213|0)(5|6|7)\d{8}$')])
  email = models.CharField(max_length=254, null=False, unique=True, validators=[EmailValidator()])
  gender = models.CharField(max_length=10, choices=Genders.choices, null=False)
  createdAt = models.DateTimeField(auto_now_add=True)
  updatedAt = models.DateTimeField(auto_now=True)

  def __str__(self):
     return f"{self.lastName} {self.firstName}"
```

```python
class Department(models.Model):
  id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
  name = models.CharField(max_length=256, null=False, unique=True)

  def __str__(self):
     return f"{self.name}"

class Speciality(models.Model):
  id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
  name = models.CharField(max_length=256, null=False, unique=True)

  def __str__(self):
     return f"{self.name}"

class Procedure(models.Model):
  id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
  name = models.CharField(max_length=256, null=False, unique=True)
  department = models.ForeignKey('Department', on_delete=models.CASCADE, null=True, blank=True)
  type = models.CharField(max_length=20, choices=ProcedureType.choices, null=False)

  def __str__(self):
     return f"{self.name}"

class ProcedureApplication(models.Model):
  id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
  appointment = models.ForeignKey('Appointment', on_delete=models.CASCADE)
  medicalStaff = models.ForeignKey('MedicalStaff', on_delete=models.CASCADE)
  procedure = models.ForeignKey('Procedure', on_delete=models.CASCADE)
  report = models.TextField(null=True, blank=True)

  class Meta:
    unique_together = ('appointment', 'medicalStaff', 'procedure')

class Appointment(models.Model):
  id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
  patient = models.ForeignKey('Patient', on_delete=models.CASCADE)
  timestamp = models.DateTimeField(null=False)
  state = models.CharField(max_length=15, choices=AppointmentState.choices, default=AppointmentState.SCHEDULED, blank=True)
  createdAt = models.DateTimeField(auto_now_add=True)
  updatedAt = models.DateTimeField(auto_now=True)

class Prescription(models.Model):
  id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
  appointment = models.ForeignKey('Appointment', on_delete=models.CASCADE)
  content = models.TextField()
```

A utility decorator

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

```python
from django.shortcuts import redirect

def login_not_required(view_func):
  def wrapper(request, *args, **kwargs):
    if request.user.is_authenticated:
      return redirect('dashboard')
    else:
      return view_func(request, *args, **kwargs)

  return wrapper

```

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

```python
@login_required(login_url='sign-in')
def addAppointement(request: HttpRequest, patientId: str):
  patient = get_object_or_404(Patient, id=patientId)
  form = AppointmentForm(initial={'patient': patient})
  if request.method == 'POST':
    data = request.POST.copy()
    data.appendlist('patient', patient)
    form = AppointmentForm(data=data)
    if form.is_valid():
      appointment = form.save()
      messages.success(request=request, message='Appointment created successfully')
      return redirect('edit-appointment', appointmentId=appointment.id)
  context = { 'form': form, 'patient': patient }
  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

```python
@login_required(login_url='sign-in')
def editAppointment(request: HttpRequest, appointmentId: str):
  appointment = get_object_or_404(Appointment, id=appointmentId)
  patient = get_object_or_404(Patient, id=appointment.patient.id)
  procedureApplications = ProcedureApplication.objects.filter(appointment=appointment.id).order_by('procedure')
  prescriptions = Prescription.objects.filter(appointment=appointment)

  form = AppointmentForm(instance=appointment, initial={'patient': patient})
  if request.method == 'POST':
    data = request.POST.copy()
    data.appendlist('patient', patient)
    form = AppointmentForm(data=data, instance=appointment)
    if form.is_valid():
      form.save()
      messages.success(request=request, message='Appointment updated successfully')
      return redirect('appointment-list-patient', patientId=patient.id)
  context = { 'form': form, 'patient': patient, 'procedureApplications': procedureApplications, 'prescriptions': prescriptions, 'appointment': appointment }
  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

```python
@login_required(login_url='sign-in')
def deleteAppointment(request: HttpRequest, appointmentId: str):
  appointment = get_object_or_404(Appointment, id=appointmentId)
  patient = appointment.patient
  appointment.delete()
  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

```python
@login_required(login_url='sign-in')
def appointmentListMedicalStaff(request: HttpRequest, medicalStaffId: str):
  medicalStaff = get_object_or_404(MedicalStaff, id=medicalStaffId)

  procedure_applications_for_medical_staff = ProcedureApplication.objects.filter(medicalStaff=medicalStaff)
  appointments_for_medical_staff = [pa.appointment for pa in procedure_applications_for_medical_staff]

  length = len(appointments_for_medical_staff)

  context = { 'appointments': appointments_for_medical_staff, 'medicalStaff': medicalStaff, 'length': length }
  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

```python
@login_required(login_url='sign-in')
def appointmentListPatient(request: HttpRequest, patientId: str):
  patient = get_object_or_404(Patient, id=patientId)
  form = SearchForm(request.GET)
  appointments = []
  if form.is_valid():
    search = form.cleaned_data['search']
    appointments = Appointment.objects.filter(
      Q(id__icontains=search) |
      Q(timestamp__icontains=search) |
      Q(state__icontains=search),
      patient=patient,
    ).order_by('-timestamp')
  else:
    appointments = Appointment.objects.filter(
      patient=patient,
    ).order_by('-timestamp')
  context = { 'appointments': appointments, 'form': form, 'patient': patient }
  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 “PotgresSQL”

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

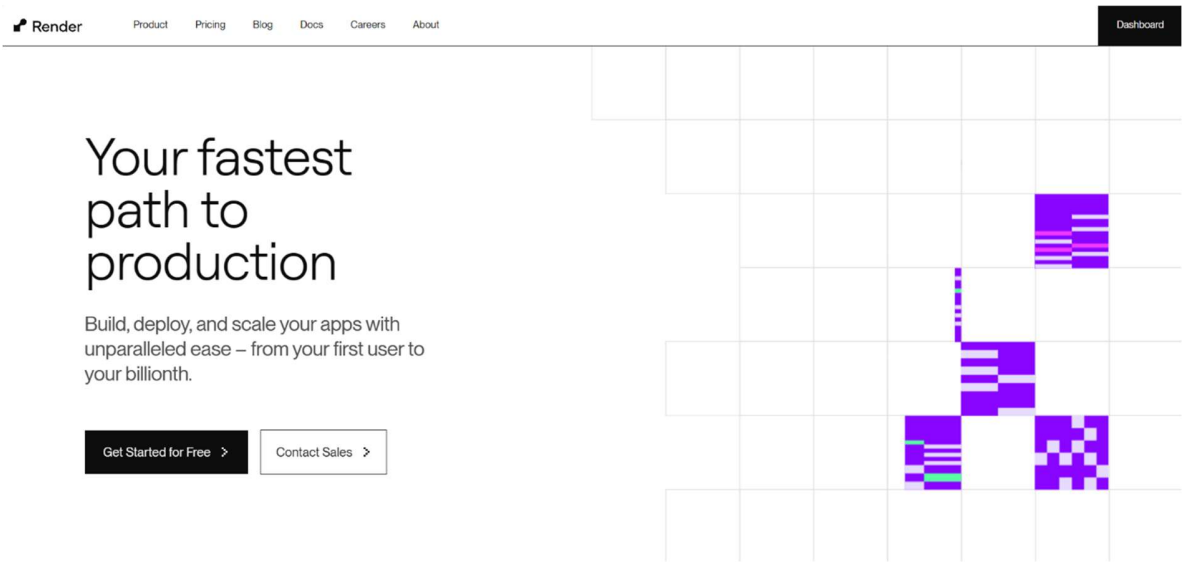
```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': env('DB_NAME'),
        'USER': env('DB_USER'),
        'PASSWORD': env('DB_PASSWORD'),
        'HOST': env('DB_HOST'),
        'PORT': env('DB_PORT'),
    } if env('DEBUG') == 'False' else {
```

```
    '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

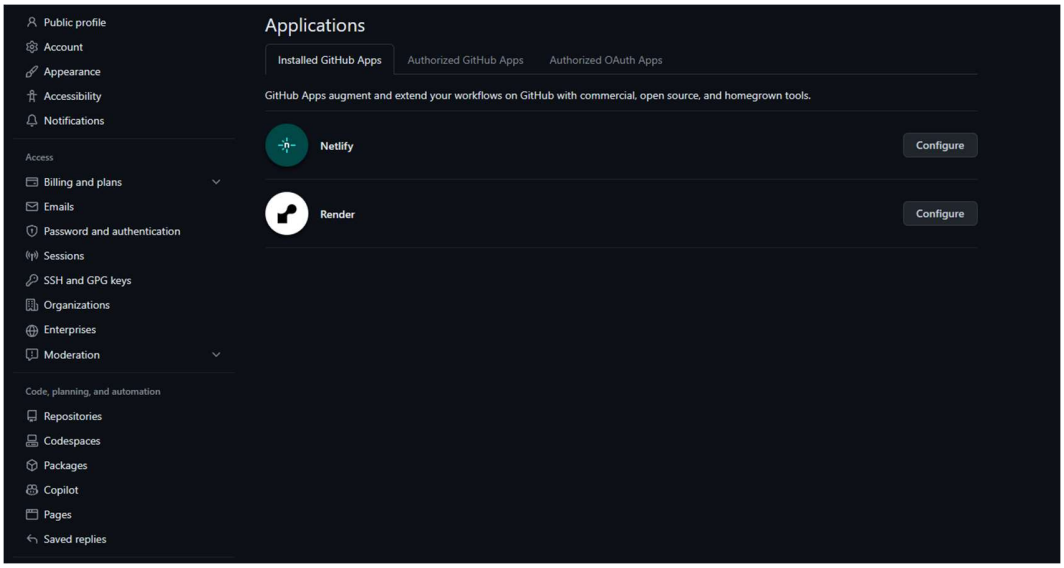


Render emerges as a cloud application hosting platform aimed at simplifying the deployment and management of web apps, APIs, and background jobs for developers. Utilizing a serverless architecture, Render facilitates a code-centric workflow, relieving developers of infrastructure concerns. Renowned for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

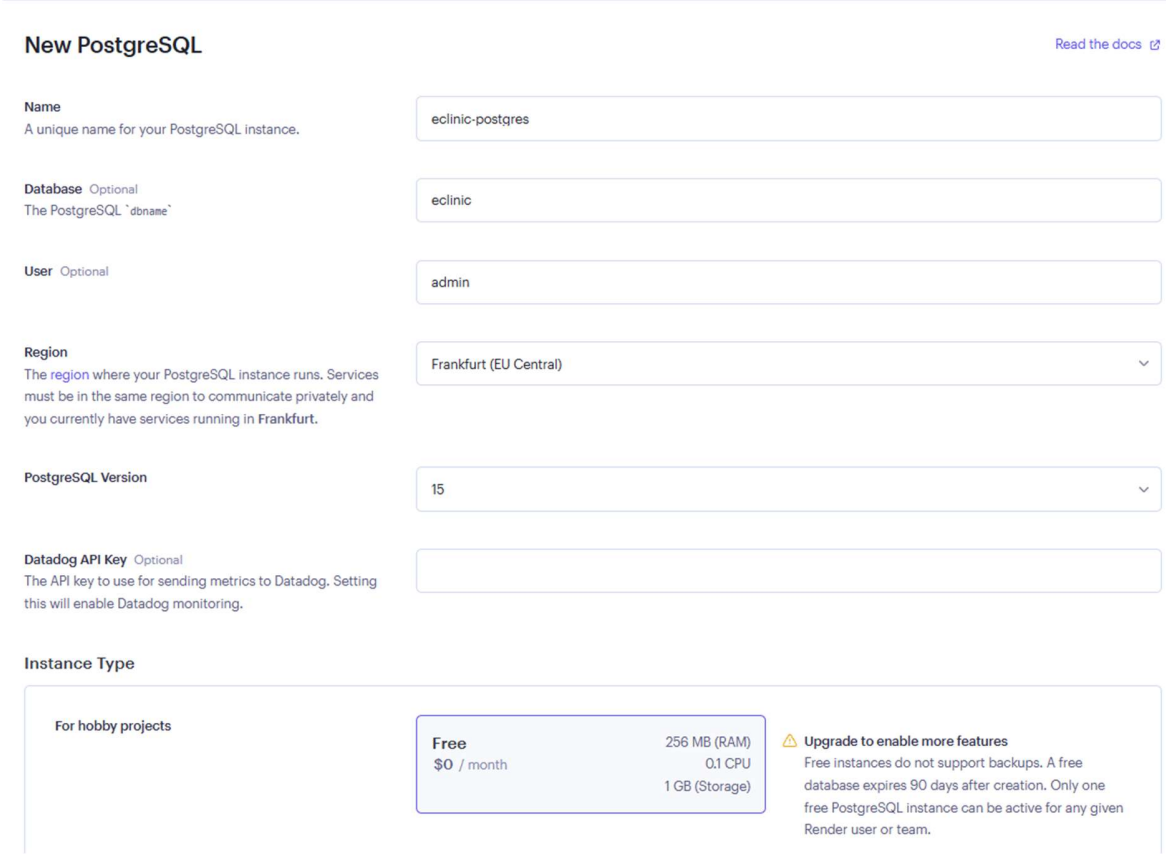
Publishing to github

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”

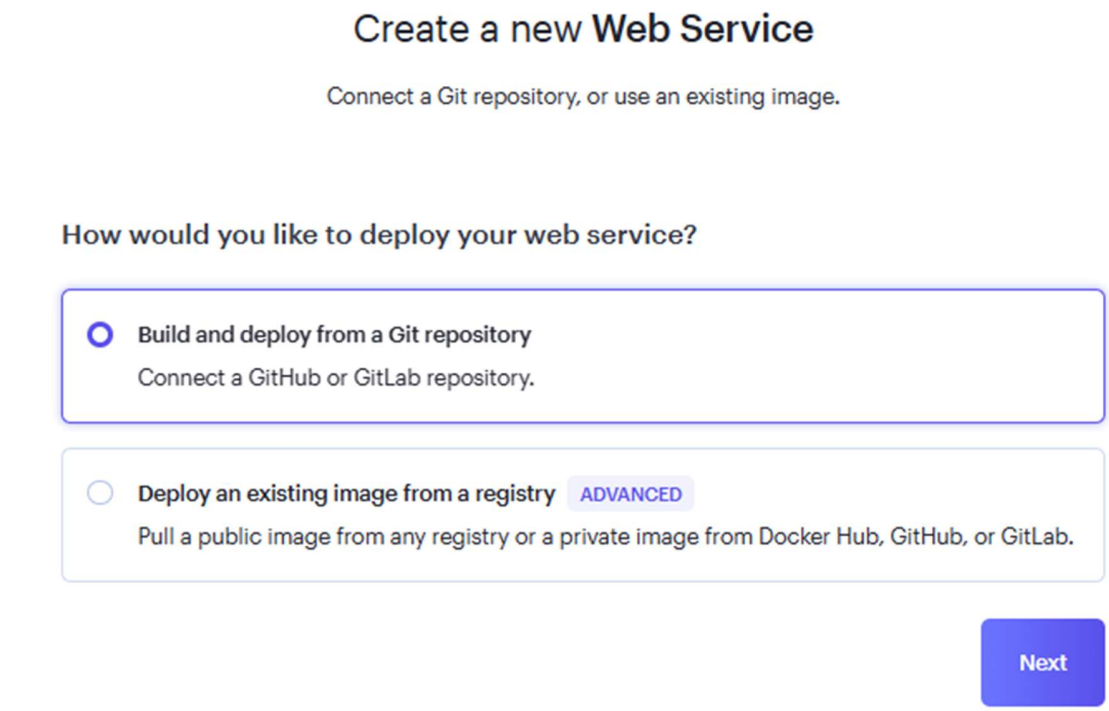
We also need to allow access to the repository of our project for the hosting provider

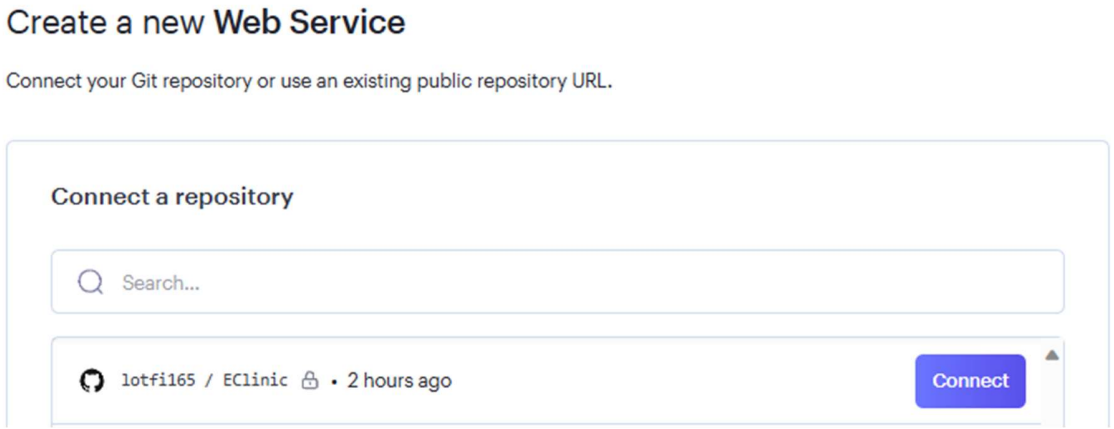


Creating the postgres database



Deploying the application





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
A unique name for your web service.

EClinic

Region
The region where your web service runs. Services must be in the same region to communicate privately and you currently have services running in **Frankfurt**.

Frankfurt (EU Central)

Branch
The repository branch used for your web service.

main

Root Directory Optional
Defaults to repository root. When you specify a root directory that is different from your repository root, Render runs all your commands in the specified directory and ignores changes outside the directory.

e.g. src

Runtime
The runtime for your web service.

Python 3

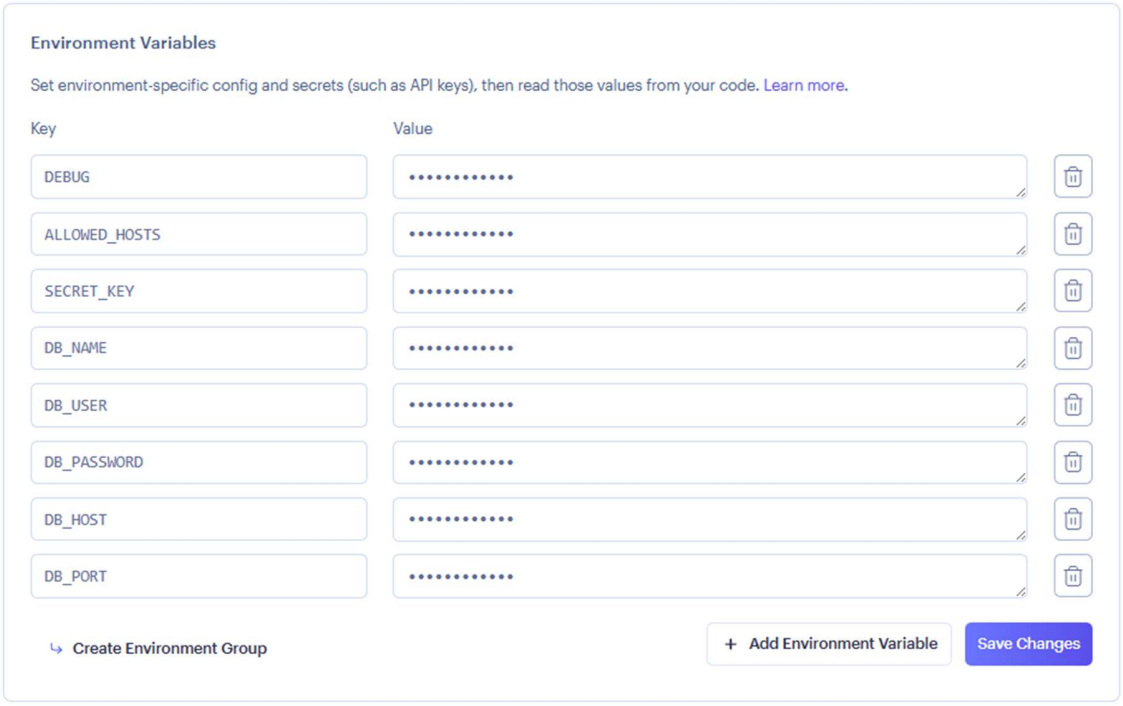
Build Command
This command runs in the root directory of your repository when a new version of your code is pushed, or when you deploy manually. It is typically a script that installs libraries, runs migrations, or compiles resources needed by your app.

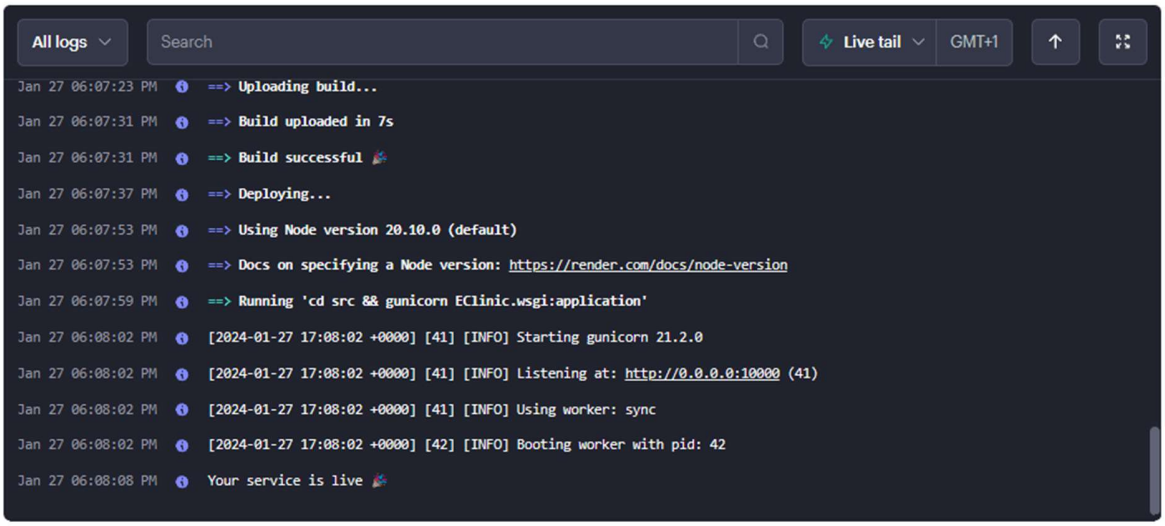
```
$ pip install -r requirements.txt && cd src && python manage.py collectstatic
```

Start Command
This command runs in the root directory of your app and is responsible for starting its processes. It is typically used

```
$ cd src && gunicorn EClinic.wsgi:application
```

Configuring environment variables

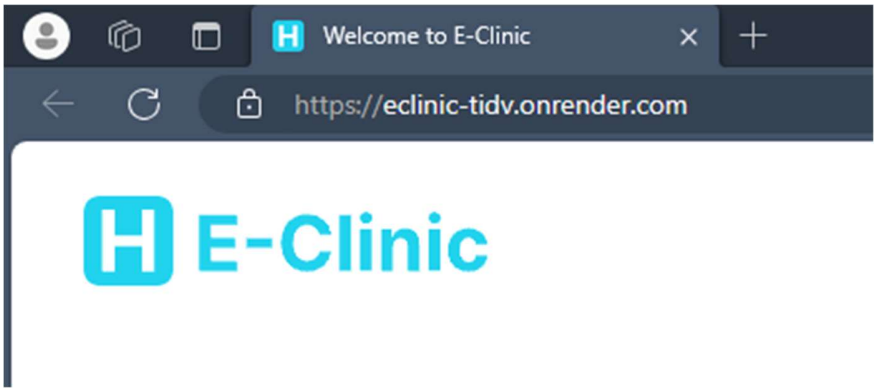




The app is now deployed and can be accessed via an url provided by the hosting service

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.