MEDICAL APPOINTMENTS

Students: Bucur Alexandra Ioana
Bogdan Maria
Bogdan Tudor Alexandru

Group: 30434

2021-2022

Contents

1 Introduction

The project presented in this document is about a web application that will let users create appointments for themselves at certain doctors. The users are allowed to navigate through several web pages not only in order to read useful information about health and lifestyle, but also find out their doctor's resume.
This app is designed in order to ease the process of making a new appointment at a certain doctor.

2 General view of actors and use cases

Actor	***Type***	***Description***
Administrator	*Human*	*The person administrating the system.*
Customer/User	*Human*	*Performs general operations over the system, such that visiting the Web-pages, making appointments, updating profile. Rate doctor.*
Doctor	*Human*	*Has a program with current appointments. Sees patient related info, uploads new information(scans, results) regarding a certain patient.*

3 Detailed description of actors and use cases

3.1 Actors

3.1.1 Administrator

Type: human
Detailed description:
This actor is the person administrating the system. The administrator is the one responsible for maintenance of the Web-Site. The administrator manages and updates the Web-Site.

3.1.2 Customer

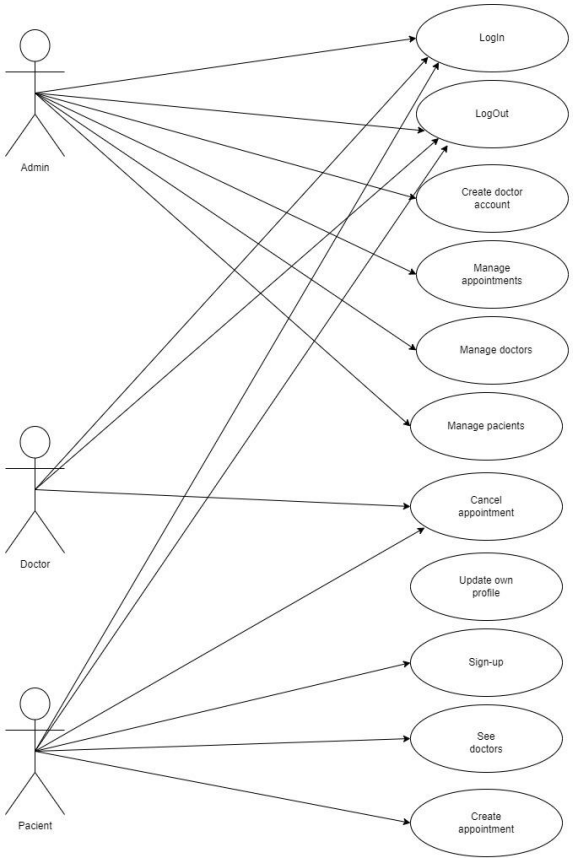
Type: human
Detailed description:
The customer is actually the patient that wants to go to the doctor. He or she will be able to create an account and the create certain appointments at several doctors.

3.1.3 Doctor

Type: human

Detailed description:
Doctor will be able to honor/cancel appointments and update not only patients status (health advice/ medical results), but also the CV.

3.2 Use case diagram



3.3 Use cases

Use Case for *Actor1*

Actor: Admin - human

Description: The admin will be in charge of maintaining all the web application. He will make sure that the app is running accordingly and will seek improvements all the time. He will be able to edit/delete/create doctors and patients.

Use Case for *Actor2*

Actor: User- human

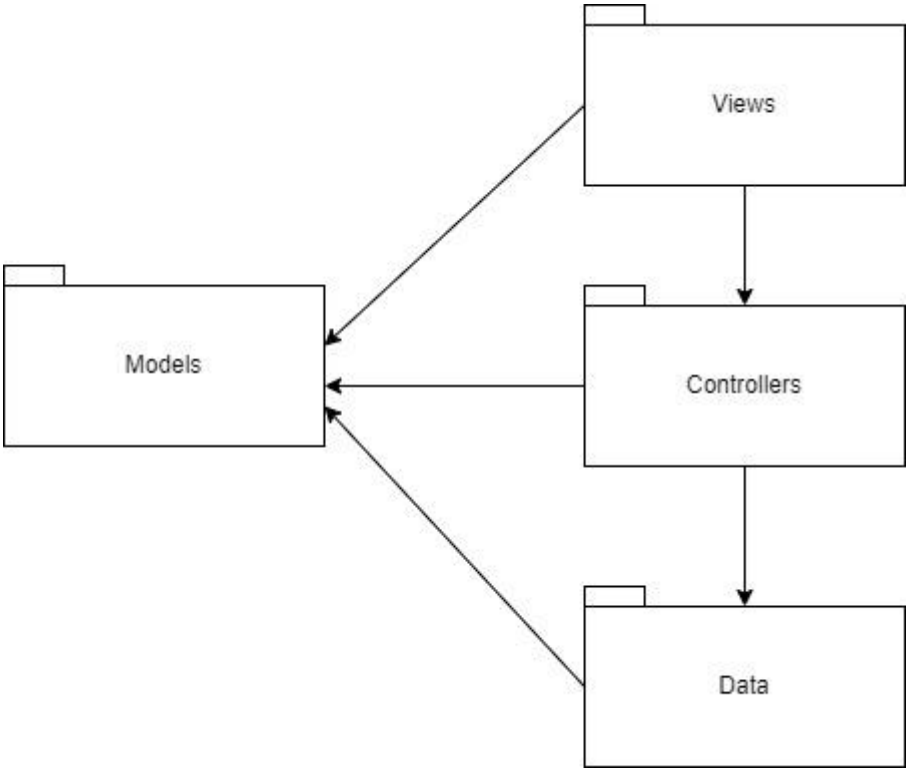
Description: Each user will have a personal account, where the information will be stored. The client will be able to update the information and view all current appointments. The user will also be able to create new appointments at a certain doctor within their working hours. After one appointment was completed, the patient will be able to rate the doctor.

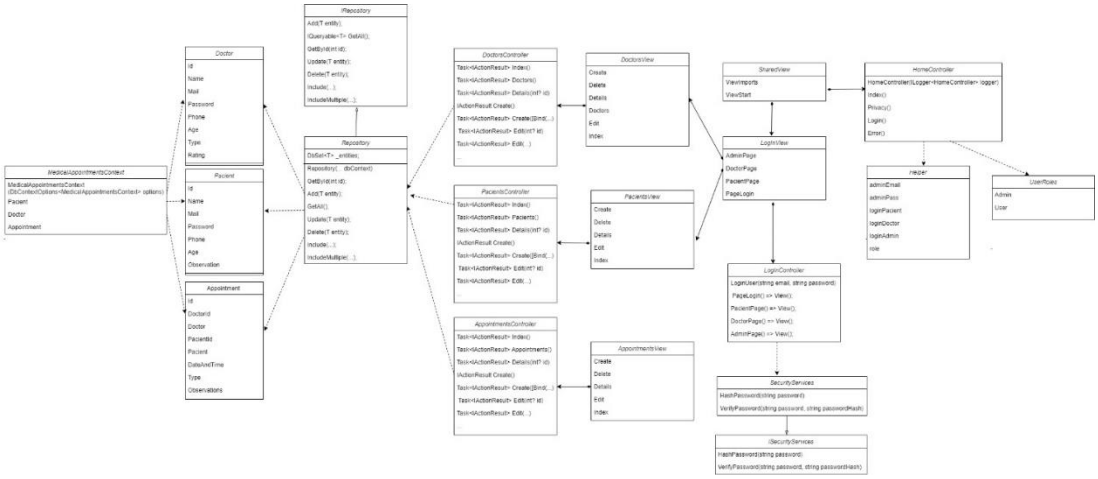
Use Case for *Actor3*

Actor: Doctor - human

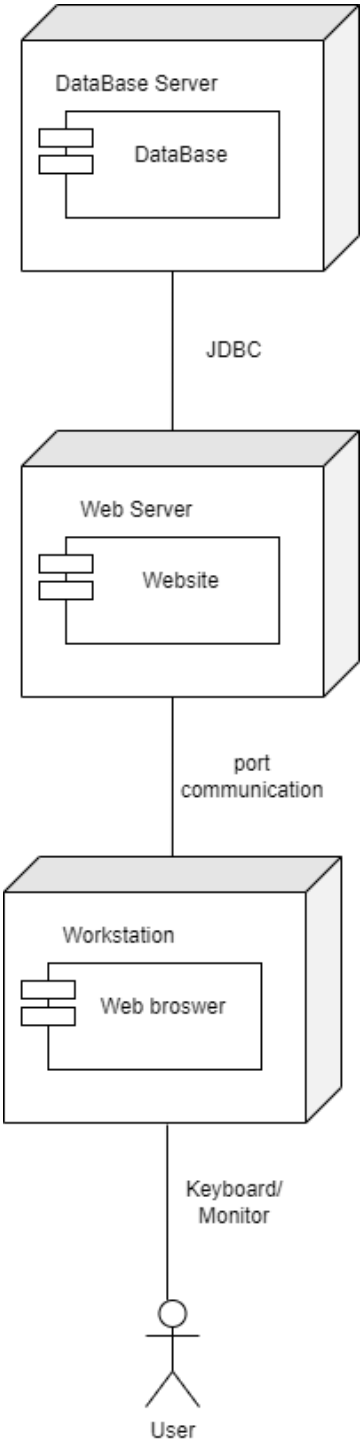
Description: Doctors will be able to see patients' health records in order to honor an appointment. They will also be able to upload new information related to patient's health such as X-rays or results after medical tests, or even medical prescriptions or advice related to some health problems. The doctors will not be ables to create a new account for them, they will receive the account from the administrator and will only be able to modify some data, like password, name, age etc. The mail will remain the same.

3.4 Class diagram

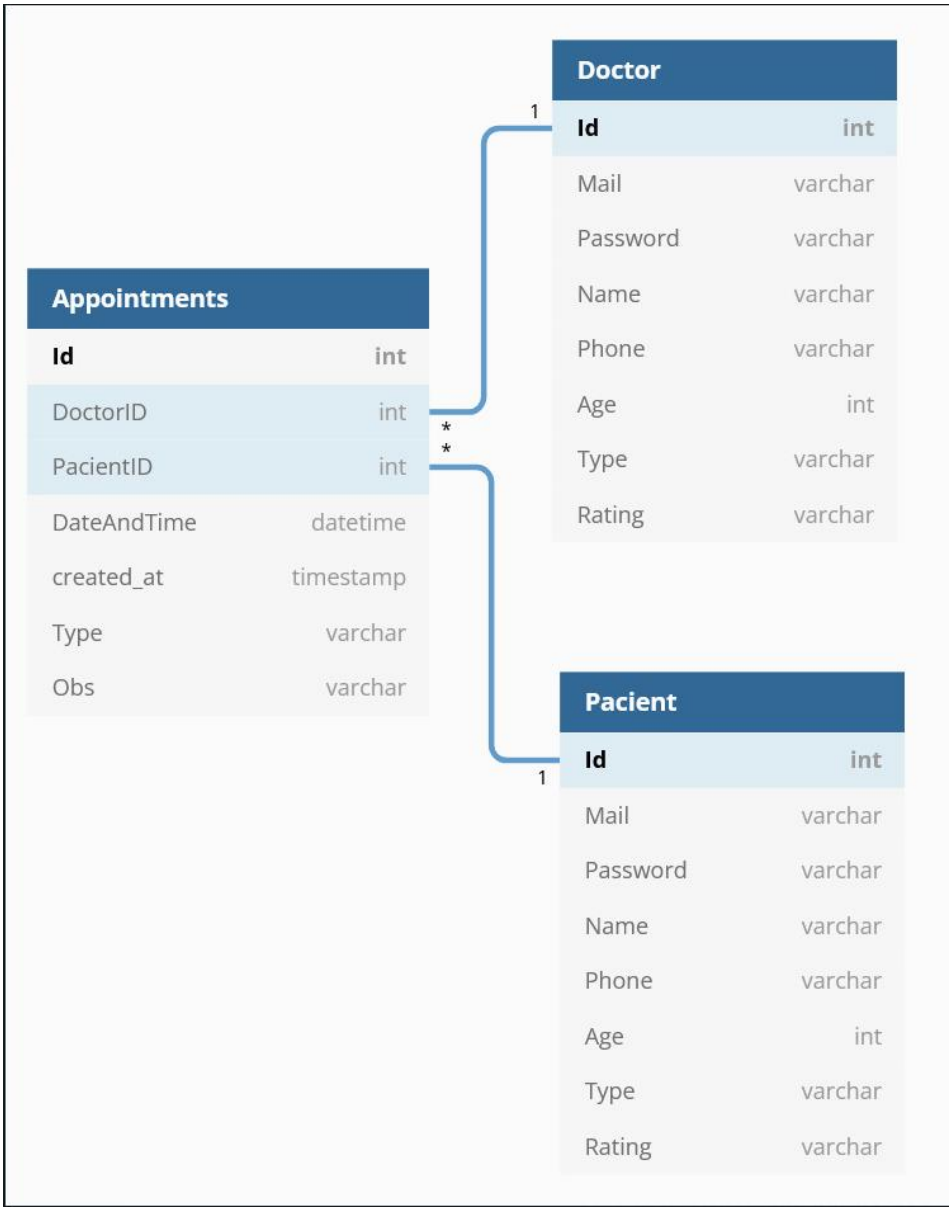



IRepository
AddT entity();
IQueryable<T> GetAll();
GetById(int id);
UpdateT entity();
DeleteT entity();
Include(...);
IncludeMultiple(...);
Doctor
Id
Name
Mail
Password
Phone
Age
Type
Rating
Repository
DbSet<T> _entities;
Repository(... dbContext)
GetById(int id);
AddT entity();
GetAll();
UpdateT entity();
DeleteT entity();
Include(...);
IncludeMultiple(...);
DoctorsController
Task<IActionResult> Index()
Task<IActionResult> Doctors()
Task<IActionResult> Details(int? id)
IActionResult Create()
Task<IActionResult> Create([Bind(...)
Task<IActionResult> Edit(int? id)
Task<IActionResult> Edit(...)
DoctorsView
Create
Delete
Details
Doctors
Edit
Index
SharedView
ViewImports
ViewStart
HomeController
HomeController(ILogger<HomeController> logger)
Index()
Privacy()
Login()
Error()
MedicalAppointmentsContext
MedicalAppointmentsContext
(DbContextOptions<MedicalAppointmentsContext> options)
Pacient
Doctor
Appointment
Pacient
Id
Name
Mail
Password
Phone
Age
Observation
PacientsController
Task<IActionResult> Index()
Task<IActionResult> Pacients()
Task<IActionResult> Details(int? id)
IActionResult Create()
Task<IActionResult> Create([Bind(...)
Task<IActionResult> Edit(int? id)
Task<IActionResult> Edit(...)
PacientsView
Create
Delete
Details
Edit
Index
LoginView
AdminPage
DoctorPage
PacientPage
PageLogin
Helper
adminEmail
adminPass
loginPacient
loginDoctor
loginAdmin
role
UserRoles
Admin
User
Appointment
Id
DoctorId
Doctor
PacientId
Pacient
DateAndTime
Type
Observations
AppointmentsController
Task<IActionResult> Index()
Task<IActionResult> Appointments()
Task<IActionResult> Details(int? id)
IActionResult Create()
Task<IActionResult> Create([Bind(...)
Task<IActionResult> Edit(int? id)
Task<IActionResult> Edit(...)
AppointmentsView
Create
Delete
Details
Edit
Index
LoginController
LoginUser(string email, string password)
PageLogin() => View();
PacientPage() => View();
DoctorPage() => View();
AdminPage() => View();
SecurityServices
HashPassword(string password)
VerifyPassword(string password, string passwordHash)
ISecurityServices
HashPassword(string password);
VerifyPassword(string password, string passwordHash)

3.5 Deployment diagram



4 Database



When designing the database, the main idea was to store general information of our users into 2 tables: the doctors and the pacients.

As for the Appointment table, both doctors and patients can have more than one appointment, so the relationship is 1 to many, based on the id.

5 Code

My work is centered around the backend of our application, more exactly the database. Together with Alexandra Bucur we developed the Controllers and the Views for Appointments, Pacients and Doctors.

5.1 DataBase Models:

The DataBase of our application is composed of 3 parts: Appointment, Doctor and Pacient:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.ComponentModel.DataAnnotations;

namespace MedicalAppointments.Models
{
    public class Appointment
    {
        [Required]
        public int Id { get; set; }


        public int DoctorID { get; set; }


        public Doctor Doctor { get; set; }


        public int PacientID { get; set; }


        public Pacient Pacient { get; set; }


        public DateTime DateAndTime { get; set; }


        public String Type { get; set; }


        public string Observations { get; set; }

    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.ComponentModel.DataAnnotations;

namespace MedicalAppointments.Models
{
    public class Doctor
```

```
    {
        public int Id { get; set; }
        public string Mail { get; set; }
        public string Password { get; set; }
        public string Name { get; set; }
        public string Phone { get; set; }
        public int Age { get; set; }
        public string Type { get; set; }
        public float Rating { get; set; }

    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.ComponentModel.DataAnnotations;
using Microsoft.AspNetCore.Identity;

namespace MedicalAppointments.Models
{
    public class Pacient
    {
        public int Id { get; set; }
        public string Mail { get; set; }
        public string Password { get; set; }
        public string Name { get; set; }
        public string Phone { get; set; }
        public string Age { get; set; }
        public string Observation { get; set; }
    }
}
```

5.2 DataBase Controllers and Views(designed together with Alexandra Bucur):

The appointments doctors and pacients have all 4 types of pages.
The pages are similar, so the views and code will be presented only for one of the 3 possible tables.

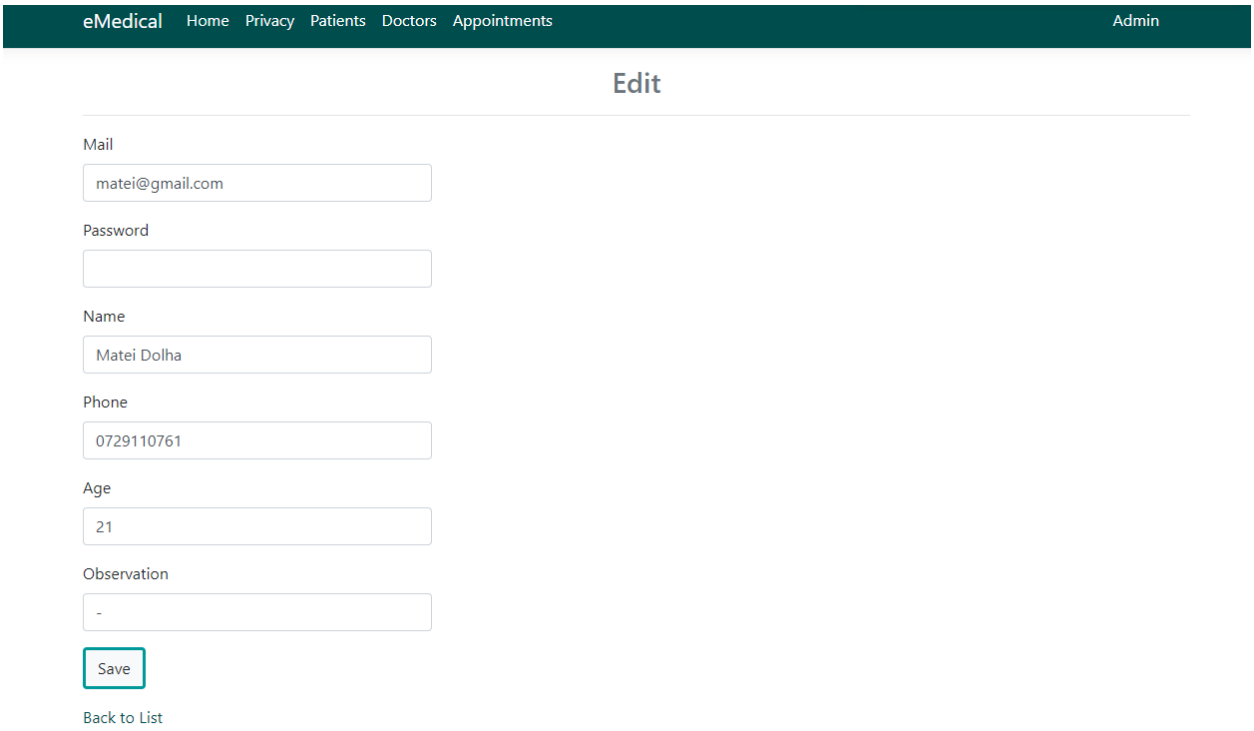
5.2.1 Controller

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using MedicalAppointments.Data;
using MedicalAppointments.Models;

using BC = BCrypt.Net.BCrypt;

namespace MedicalAppointments.Controllers
{
    public class PacientsController : Controller
    {
        private readonly MedicalAppointmentsContext _context;

        public PacientsController(MedicalAppointmentsContext context)
        {
            _context = context;
        }

        // GET: Pacients
        public async Task<IActionResult> Index()
```

```
{
    return View(await _context.Pacient.ToListAsync());
}

// GET: Pacients/Details/5
public async Task<IActionResult> Details(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var pacient = await _context.Pacient
        .FirstOrDefaultAsync(m => m.Id == id);
    if (pacient == null)
    {
        return NotFound();
    }

    return View(pacient);
}

// GET: Pacients/Create
public IActionResult Create()
{
    return View();
}

// POST: Pacients/Create
// To protect from overposting attacks, enable the specific properties you want to bind to, for
// more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("Id,Mail,Password,Name,Phone,Age,Observation")] Pacient pacient)
{
    if (ModelState.IsValid)
    {
        pacient.Password = BC.HashPassword(pacient.Password);
        _context.Add(pacient);
        await _context.SaveChangesAsync();
        return RedirectToAction("Index", "Home");
    }
    return View(pacient);
}

// GET: Pacients/Edit/5
public async Task<IActionResult> Edit(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var pacient = await _context.Pacient.FindAsync(id);
    if (pacient == null)
    {
        return NotFound();
    }
    return View(pacient);
}

// POST: Pacients/Edit/5
// To protect from overposting attacks, enable the specific properties you want to bind to, for
// more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id, [Bind("Id,Mail,Password,Name,Phone,Age,Observation")] Pacient pacient)
{
    if (id != pacient.Id)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            pacient.Password = BC.HashPassword(pacient.Password);
            _context.Update(pacient);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!PacientExists(pacient.Id))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction("Index", "Home");
    }
    return View(pacient);
}

// GET: Pacients/Delete/5
public async Task<IActionResult> Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var pacient = await _context.Pacient
        .FirstOrDefaultAsync(m => m.Id == id);
    if (pacient == null)
    {
        return NotFound();
```

```
            }

            return View(pacient);
        }

        // POST: Pacients/Delete/5
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
        {
            var pacient = await _context.Pacient.FindAsync(id);
            _context.Pacient.Remove(pacient);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }

        private bool PacientExists(int id)
        {
            return _context.Pacient.Any(e => e.Id == id);
        }
    }
}
```

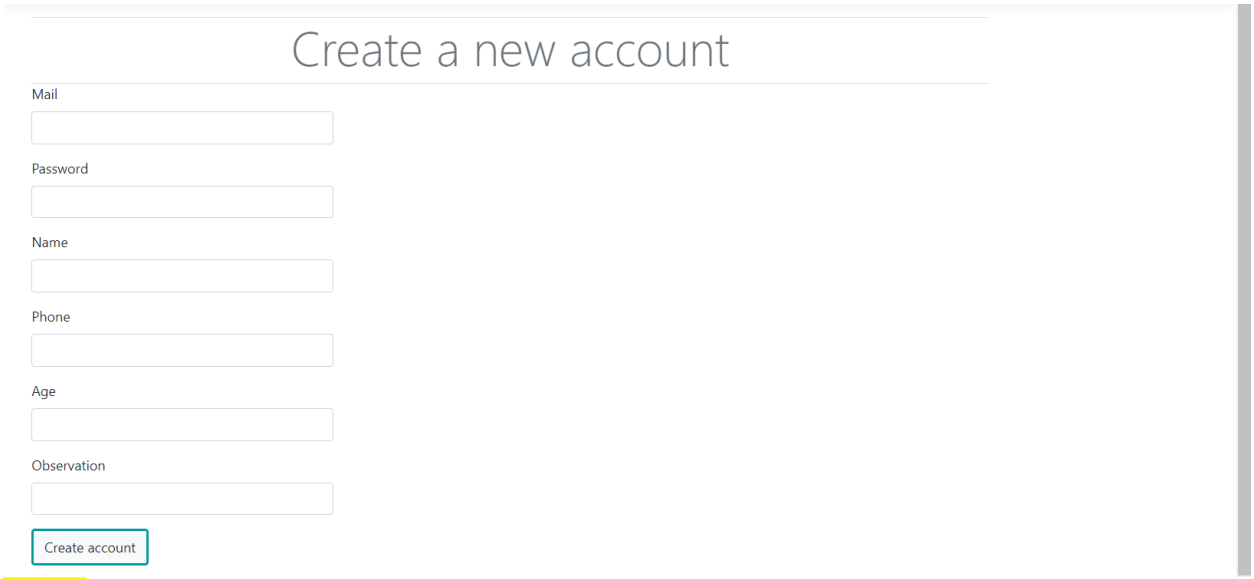
5.2.2 Edit view




```
@model MedicalAppointments.Models.Pacient

@{
    ViewData["Title"] = "Edit";
}

<h3 class="text-muted  text-center">Edit</h3>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Edit">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <input type="hidden" asp-for="Id" />
            <div class="form-group">
                <label asp-for="Mail" class="control-label"></label>
```

```
                <input asp-for="Mail" class="form-control" />
                <span asp-validation-for="Mail" class="text-danger"></span>
            </div>
            <div class="form-group">
                @if (MedicalAppointments.Data.Static.Helper.role == 2) //pacient
                {
                    <label asp-for="Password" class="control-label">Confirm
password</label>
                }
                else
                {
                    <label asp-for="Password" class="control-label"></label>
                }
                <input type="password" asp-for="Password" class="form-control" />
                <span asp-validation-for="Password" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Name" class="control-label"></label>
                <input asp-for="Name" class="form-control" />
                <span asp-validation-for="Name" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Phone" class="control-label"></label>
                <input asp-for="Phone" class="form-control" />
                <span asp-validation-for="Phone" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Age" class="control-label"></label>
                <input asp-for="Age" class="form-control" />
                <span asp-validation-for="Age" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Observation" class="control-label"></label>
                <input asp-for="Observation" class="form-control" />
                <span asp-validation-for="Observation" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input style="border: 3px solid #009999" type="submit" value="Save"
class="btn btn-light" />
            </div>
        </form>
    </div>
</div>

@if (MedicalAppointments.Data.Static.Helper.role == 1)
{
    <div>
        <a style="color: #004d4d" asp-action="Index">Back to List</a>
    </div>
}

@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

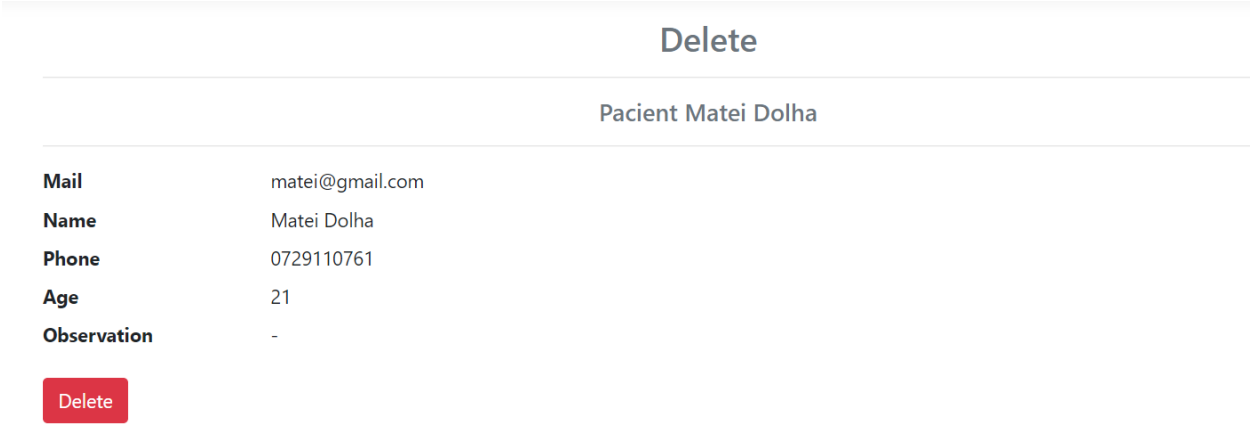
5.2.3 Create View




```
@model MedicalAppointments.Models.Pacient

@{
    ViewData["Title"] = "Create";
}

<div class="border-top border-bottom text-muted  text-center">
    <h1 class="display-4">Create a new account</h1>
</div>

<div class="row">
    <div class="col-md-4">
        <form asp-action="Create">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Mail" class="control-label"></label>
                <input asp-for="Mail" class="form-control" />
                <span asp-validation-for="Mail" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Password" class="control-label"></label>
                <input type="password" asp-for="Password" class="form-control" />
                <span asp-validation-for="Password" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Name" class="control-label"></label>
                <input asp-for="Name" class="form-control" />
                <span asp-validation-for="Name" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Phone" class="control-label"></label>
                <input asp-for="Phone" class="form-control" />
                <span asp-validation-for="Phone" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Age" class="control-label"></label>
                <input asp-for="Age" class="form-control" />
```

```
                <span asp-validation-for="Age" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Observation" class="control-label"></label>
                <input asp-for="Observation" class="form-control" />
                <span asp-validation-for="Observation" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input style="border: 3px solid #009999" type="submit" value="Create
account" class="btn btn-light" />
            </div>
        </form>
    </div>
</div>


@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

5.2.4 Delete view




```
@model MedicalAppointments.Models.Pacient

@{
    ViewData["Title"] = "Delete";
}

<h3 class="text-center text-muted" >Delete</h3>
<hr />

<div>
    <h5 class = "text-center text-muted" >Pacient  @Html.DisplayFor(model => model.Name)
</h5>
    <hr />
    <dl class="row">
        <dt class = "col-sm-2">
            @Html.DisplayNameFor(model => model.Mail)
```

```
        </dt>
        <dd class = "col-sm-10">
            @Html.DisplayFor(model => model.Mail)
        </dd>
        <dt class = "col-sm-2">
            @Html.DisplayNameFor(model => model.Name)
        </dt>
        <dd class = "col-sm-10">
            @Html.DisplayFor(model => model.Name)
        </dd>
        <dt class = "col-sm-2">
            @Html.DisplayNameFor(model => model.Phone)
        </dt>
        <dd class = "col-sm-10">
            @Html.DisplayFor(model => model.Phone)
        </dd>
        <dt class = "col-sm-2">
            @Html.DisplayNameFor(model => model.Age)
        </dt>
        <dd class = "col-sm-10">
            @Html.DisplayFor(model => model.Age)
        </dd>
        <dt class = "col-sm-2">
            @Html.DisplayNameFor(model => model.Observation)
        </dt>
        <dd class = "col-sm-10">
            @Html.DisplayFor(model => model.Observation)
        </dd>
    </dl>

    <form asp-action="Delete">
        <input type="hidden" asp-for="Id" />
        <input type="submit" value="Delete" class="btn btn-danger" />
    </form>
</div>
```

5.2.5 Index/see all view


```
@model IEnumerable<MedicalAppointments.Models.Pacient>

@{
    ViewData["Title"] = "Index";
}
```

```
<h1 class="text-center text-muted">All patients</h1>

<p>
    <a style="color: #004d4d" asp-action="Create">Create New</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Mail)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Name)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Phone)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Age)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Observation)
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
@foreach (var item in Model) {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Mail)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Name)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Phone)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Age)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Observation)
            </td>
            <td>
                <a style="border: 3px solid #009999" class="btn btn-light" asp-
action="Edit" asp-route-id="@item.Id"><i class="bi bi-pencil-square"></i>Edit</a> |
                <a style="border: 3px solid #009999" class="btn btn-light" asp-
action="Details" asp-route-id="@item.Id"><i class="bi bi-eye"></i>Details</a> |
                <a style="border: 3px solid #009999" class="btn btn-light" asp-
action="Delete" asp-route-id="@item.Id"><i class="bi bi-trash"></i>Delete</a>
            </td>
        </tr>
}
    </tbody>
</table>
```