

IZRADA WEB APLIKACIJE ZA UPRAVLJANJE PODATCIMA U PRIMARNOJ ZDRAVSTVENOJ ZAŠTITI

Mihanović, Ivana

Undergraduate thesis / Završni rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Split / Sveučilište u Splitu**

Permanent link / Trajna poveznica: <https://um.nsk.hr/um:nbn:hr:228:529665>

Rights / Prava: [In copyright](#) / [Zaštićeno autorskim pravom.](#)

Download date / Datum preuzimanja: **2024-07-16**



Repository / Repozitorij:

[Repository of University Department of Professional Studies](#)



SVEUČILIŠTE U SPLITU
SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Računarstvo

IVANA MIHANOVIĆ

ZAVRŠNI RAD

**IZRADA WEB APLIKACIJE ZA UPRAVLJANJE
PODATCIMA U PRIMARNOJ ZDRAVSTVENOJ
ZAŠTITI**

Split, rujan 2023.

SVEUČILIŠTE U SPLITU
SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Računarstvo

Predmet: Odabrani alati i naredbe u Linuxu

Z A V R Š N I R A D

Kandidat: Ivana Mihanović

Naslov rada: Izrada web aplikacije za upravljanje podacima u primarnoj zdravstvenoj zaštiti

Mentor: Nikola Grgić, viši predavač

Split, rujan 2023.

Sadržaj

Sažetak	1
1 Uvod	2
2 Korištene tehnologije	3
2.1 ASP.NET Core	3
2.1.1 C#	3
2.1.2 Upravitelj paketa NuGet	3
2.1.3 Entity Framework Core	4
2.1.4 Razor Pages	5
2.1.5 ASP.NET Core Identity	5
2.1.5.1 Scaffold Identity	6
2.2 Docker	7
2.3 PostgreSQL	11
3 Izvedba praktičnog rada	12
3.1 Postavljanje aplikacije	12
3.2 Struktura <i>PrimeCareMed</i> aplikacije	13
3.3 Baza podataka	14
3.3.1 Struktura relacijske baze podataka	14
3.3.2 Entiteti	15
3.3.3 Interakcija s bazom podataka	16
3.3.4 Migracije	18
3.3.5 Punjenje podacima	19
3.4 Repozitoriji	21
3.5 Servisi	23
3.6 RazorPages izvedba	26
3.6.1 .cshtml datoteke	26
3.6.2 .cshtml.cs datoteke	28
3.7 Autorizacija	30
3.8 Funkcionalnosti aplikacije	31

4 Zaključak	39
Literatura	40
Dodatci	42

Sažetak

Cilj ovog završnog rada jest razviti web aplikaciju za upravljanje podacima u primarnoj zdravstvenoj zaštiti. Aplikacija *PrimeCareMed* izrađena je koristeći Razor Pages koje su dio okvira ASP.NET Core. Aplikaciju upotrebljavaju korisnici koji mogu imati četiri različite uloge o kojima ovise funkcionalnosti koje pojedini korisnik može raditi. Najvažnije su funkcionalnosti aplikacije prijava u sustav te dodavanje novih pregleda pacijenata. Uređivanje pregleda ostvareno je unosom izvješća, cijepljenja i recepata za pojedini pregled. U ovom pisanom radu bit će opisan način na koji je aplikacija implementirana te će biti predstavljene tehnologije koje su korištene.

Ključne riječi: *ASP.NET Core, Docker, Identity, RazorPages*

Summary

Developing web application for primary healthcare data management

The aim of this final paper is to develop a web application for managing data in primary healthcare. The application *PrimeCareMed* is built using Razor Pages, which are part of the ASP.NET Core framework. The application is intended for users who can have one of four different roles, determining the functionalities available to each user. The most crucial features of the application include the system login and the addition of new patient records. Editing patient records is achieved through inputting reports, vaccinations, and prescriptions for each appointment. This written paper will describe the approach used to implement the application and present the technologies that were utilized.

Keywords: *ASP.NET Core, Docker, Identity, RazorPages*

1. Uvod

Uzevši u obzir napredovanje računarske tehnologije, sigurno je za reći da postoje aplikacije i sustavi koji su za današnje mogućnosti "zastarjele". Moderna poboljšana tehnologija trebala bi biti dostupna u svim područjima da bi olakšala svakodnevni rad. Prilikom izrade sustava za upravljanje podataka u zdravstvenim područjima potrebno je omogućiti sustav koji je pristupačan i jednostavan za upotrebu te uzeti u obzir one korisnike koji nisu računalno obrazovani. U svrhu toga izrađena je web aplikacija *PrimeCareMed*.

Pristup aplikaciji imaju četiri vrste korisnika o čijoj ulozi ovise funkcionalnosti koje mogu raditi: *SysAdministrator*, *Administrator*, *Doctor* i *Nurse*. Liječnik i medicinska sestra unose nove preglede u čekaonicu, imaju uvid u detalje pregleda te mogućnost dodavanja cijepljenja i recepata za lijek. Uz navedene zajedničke funkcionalnosti, liječnik ima mogućnost stvaranja narudžbe pacijenta na pregled te odabir termina za isti. Također, imaju uvid u popis pacijenata te dodavanje novog. *SysAdministrator* ima omogućene sve funkcionalnosti, dok sam administrator ima mogućnosti unosa novih liječnika, medicinskih sestara ili tehničara, lijekova, cjepiva te ureda bez uvida u popis pacijenata i pregleda.

U sljedećim poglavljima bit će predstavljene korištene tehnologije, a zatim će se detaljno i pregledno opisati način na koji je aplikacija implementirana.

2. Korištene tehnologije

2.1. ASP.NET Core

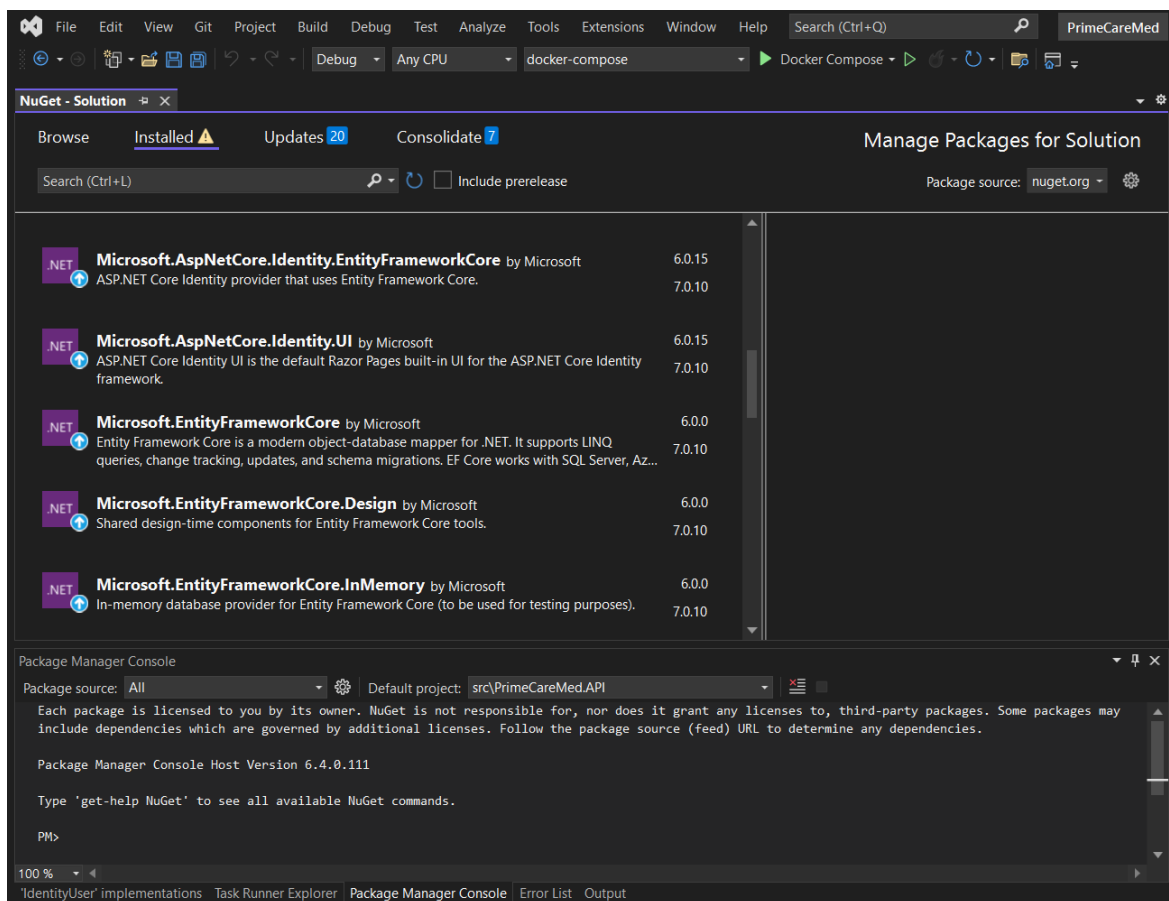
ASP.NET Core (engl. *Active Server Pages Network Enabled Technologies Core*) višepplatformski je okvir otvorenog kôda korišten za izradu aplikacija. Inačica je ASP.NET okvira opće namjene koja se može koristiti na operacijskim sustavima Windows, Linux, macOS te Dockeru. ASP.NET Core stvoren je uzevši najkorisnije značajke .NET-a i .NET Corea. Pomoću okvira ASP.NET Core omogućena je izrada aplikacija za bilo koji tehnološki uređaj kao što su iOS uređaji, Android uređaji, oblak i drugi. ASP.NET Core u *namespaceu* `Microsoft.AspNetCore.Authorization` sadrži tipove koji omogućuju autorizaciju [1].

2.1.1. C#

C# je objektno-orijentiran programski jezik čija se popularnost povećala pojavom okvira .NET. Svrha C#-a jest da bude jednostavan i moderan programski jezik dobiven kombinacijom elemenata jezika C i C++ s utjecajem jezika Java te time postaje robustan jezik za razvoj *softvera*. Uveden je i koncept alata LINQ (engl. *Language Integrated Query*) za postavljanje upita i manipuliranje podacima iz baze podataka i drugih izvora. Omogućeno je korištenje asinkronih metoda (pogledati poglavlje 3.4) uz pomoć ključnih riječi `async` i `await` kojima se omogućuje izvršavanje zadataka bez blokiranja glavne niti. Jezik C# podržava i automatsko upravljanje memorijom (engl. *auto-garbage collection*). Uz prethodno navedene, neke od glavnih karakteristika C#-a su višenitnost (engl. *multithreading*) i stroga tipizacija koja zahtijeva određivanje tipova podataka tijekom kompajliranja [2].

2.1.2. Upravitelj paketa NuGet

NuGet (engl. *NuGet Package Manager*) alat je za upravljanje paketa u ASP.NET okviru. Korišten je za instalaciju paketa (biblioteka) koji su potrebni pri izradi aplikacije. Pakete je moguće izraditi pomoću aplikacije NuGet i pohraniti ih u privatni ili javni repozitorij kao ZIP datoteke s ekstenzijom `.nupack` ili `.nupkg` [3].



Slika 1: NuGet Package Manager i Package Manager Console

2.1.3. Entity Framework Core

Entity Framework Core višeploatformska je biblioteka otvorenog kôda koja omogućuje pristup bazi podataka preko izvornog kôda pomoću ORM-a (*Object-relational mapping*) [4]. Entity Framework Core radi na operacijskim sustavima Windows, MacOS i Linux a omogućen je instalacijom paketa NuGet u aplikaciji pomoću upravitelja paketa (engl. *Package Manager*) ili .NET sučelja naredbenog retka (engl. *Command Line Interface*) naredbom prikazanom u ispisu 1.

```

1 dotnet add [<PROJECT>] package <PACKAGE_NAME>
2   [-f|--framework <FRAMEWORK>] [--interactive]
3   [-n|--no-restore] [--package-directory <PACKAGE_DIRECTORY>]
4   [--prerelease] [-s|--source <SOURCE>] [-v|--version <VERSION>]

```

Ispis 1: Naredba za dodavanje NuGet paketa

2.1.4. Razor Pages

Razor Pages model je za programiranje web aplikacija koji omogućuje jednostavno učitavanje podataka. Sintaksno je i funkcionalno sličan modelu MVC (*Model-View-Controller*). Najvažnija značajka kojom se model Razor Pages razlikuje od modela MVC su datoteka `.cshtml` te datoteka `.cshtml.cs` takozvanog kôda iza (engl. *code-behind*) koje su omogućile neupotrebu odvojenih datoteka za modele i upravljače (engl. *controller*). Stranica upravlja vlastitim modelom koji se definira u code-behind datoteci [5]. Primjer upravljanja modelom bit će prikazan u poglavlju 3.6.

2.1.5. ASP.NET Core Identity

ASP.NET Core Identity dostupan je kao biblioteka Razor Class omogućena instalacijom paketa NuGet. ASP.NET Core Identity podržava funkcionalnosti za upravljanje korisničkog sučelja, lozinki, uloga korisnika, potvrdu mail računa i drugih značajki korisnika aplikacije. Izvorni kôd dostupan je na platformi GitHub [6]. U aplikaciji *PrimeCareMed*, instalirani su paketi NuGet:

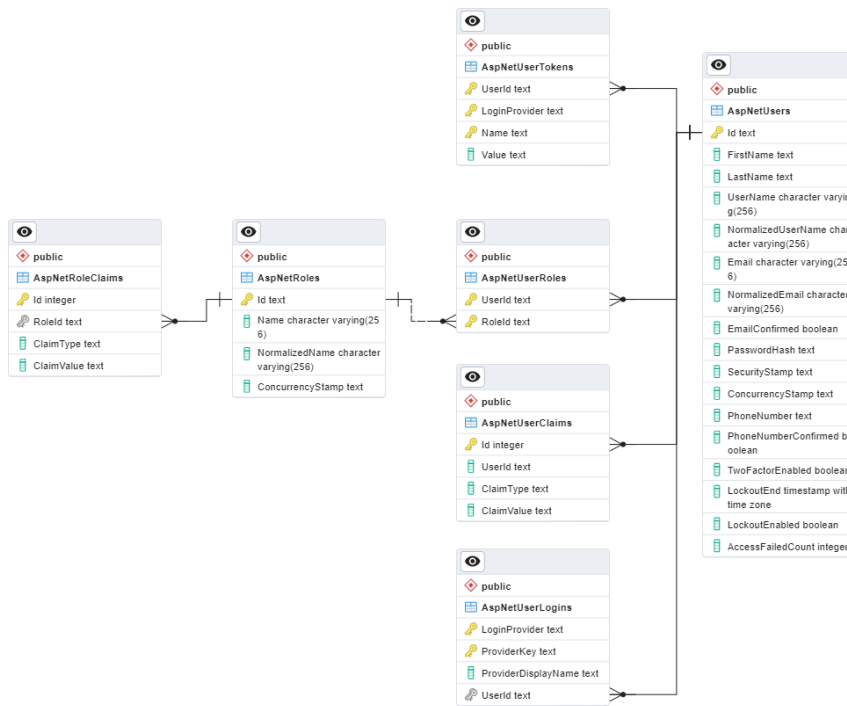
```
Microsoft.AspNetCore.Identity.EntityFrameworkCore
```

```
Microsoft.AspNetCore.Identity.UI
```

čime je omogućeno korištenje klase `IdentityUser` u svrhu naslijeđa od strane klase `ApplicationUser`. Uz nasljeđivanje klase `IdentityUser`, potrebno je naslijediti `IdentityDbContext` tipa `ApplicationUser` prikazano na ispisu 2.

```
1 public class DatabaseContext : IdentityDbContext<ApplicationUser>
```

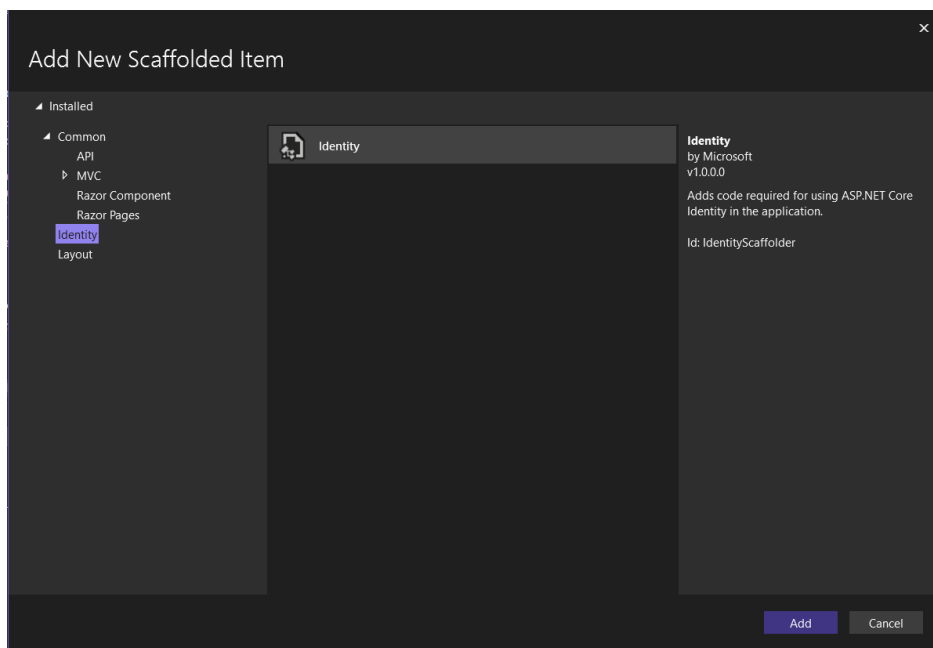
Ispis 2: Nasljeđivanje klase `IdentityDbContext`



Slika 2: Baza podataka nakon nasljeđivanja klase IdentityUser

2.1.5.1. Scaffold Identity

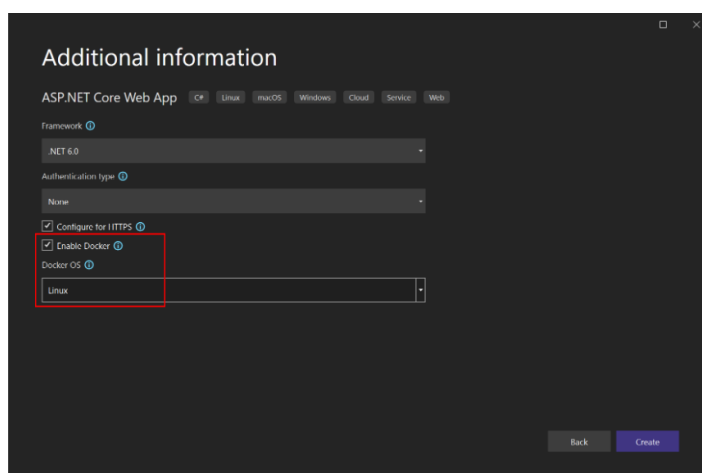
Desnim klikom na `PrimeCareMed.Frontend > Add > New Scaffolded Item` otvara se prozor prikazan na slici 3 te je omogućeno dodavanje opcije `Identity`. Dodavanjem `Identity Scaffoldera` na projekt `PrimeCareMed.Frontend` dobivene su unaprijed kreirane stranice Razor koje omogućuju funkcionalnosti povezane s upravljanjem korisničkih računa [7].



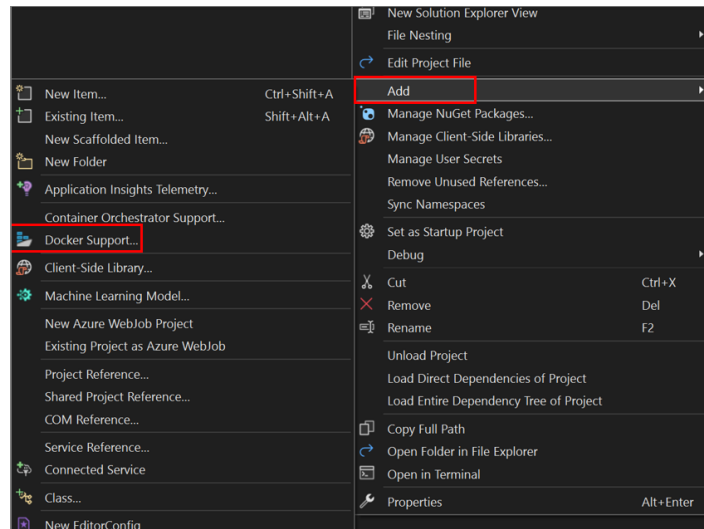
Slika 3: Identity Scaffolded Item

2.2. Docker

Kao razvojno okruženje, u svrhu izvođenja web aplikacije na različitim operativnim sustavima korišteni su Docker spremnici (engl. *containers*). Prilikom stvaranja projekta `PrimeCareMed.Frontend`, odabrana je opcija `Enable Docker` prikazana na slici 4. Na postojeći projekt `PrimeCareMed.API` Docker podrška omogućena je dodavanjem opcije `Docker Support` prikazane na slici 5 [8] [9].

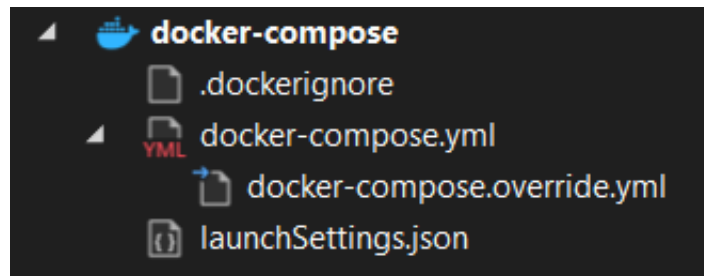


Slika 4: Uključivanje Docker podrške prilikom kreiranja projekta



Slika 5: Uključivanje Docker podrške na postojeći projekt

Odabirom opcije *Container Orchestrator Support* sa slike 5 stvara se zaseban projekt unutar *solutiona* naziva `docker-compose` koji sadrži datoteke `.dockerignore`, `docker-compose.yml` i `launchSettings.json` vidljive na slici 6 dok se datoteke naziva `Dockerfile` nalaze u pripadajućim projektima: `PrimeCareMed.Frontend` i `PrimeCareMed.API`.



Slika 6: Projekt `docker-compose`

U ispisu 3 prikazani su docker servisi u datoteci `docker-compose.yml` koji predstavljaju spremnike u aplikaciji Docker Desktop koji će biti pokrenuti prilikom pokretanja aplikacije (pogledati sliku 7).

```

1 version: '3.4'
2
3 services:
4
5   primecaremed.frontend:

```

```

6   image: ${DOCKER_REGISTRY-}primecaremedfrontend
7   build:
8     context: .
9     dockerfile: src/PrimeCareMed.Frontend/Dockerfile
10  environment:
11    CONNECTION_STRING: "Host=postgres;Port=5432;Database=postgres;
    Username=admin;Password=root;Integrated Security=true;Pooling=true;"
12
13  primecaremed.api:
14    image: ${DOCKER_REGISTRY-}primecaremedapi
15    build:
16      context: .
17      dockerfile: src/PrimeCareMed.API/Dockerfile
18    environment:
19      CONNECTION_STRING: "Host=postgres;Port=5432;Database=postgres;
    Username=admin;Password=root;Integrated Security=true;Pooling=true;"
20
21  postgres:
22    image: postgres:alpine
23    environment:
24      POSTGRES_DB: postgres
25      POSTGRES_USER: admin
26      POSTGRES_PASSWORD: root
27    ports:
28      - 5432:5432
29    volumes:
30      - postgres-data:/var/lib/postgresql/data
31    restart: unless-stopped
32
33  pgadmin4:
34    image: dcagatay/pwless-pgadmin4:latest
35    depends_on:
36      - postgres
37    ports:
38      - 15432:80
39    environment:
40      POSTGRES_USER: admin
41      POSTGRES_PASSWORD: root
42    restart: unless-stopped

```

```

43
44 mailhog:
45   image: mailhog/mailhog
46   ports:
47     - '1025:1025' # smtp server
48     - '8025:8025' # web ui
49
50 volumes:
51   postgres-data:

```

Ispis 3: Datoteka docker-compose.yml

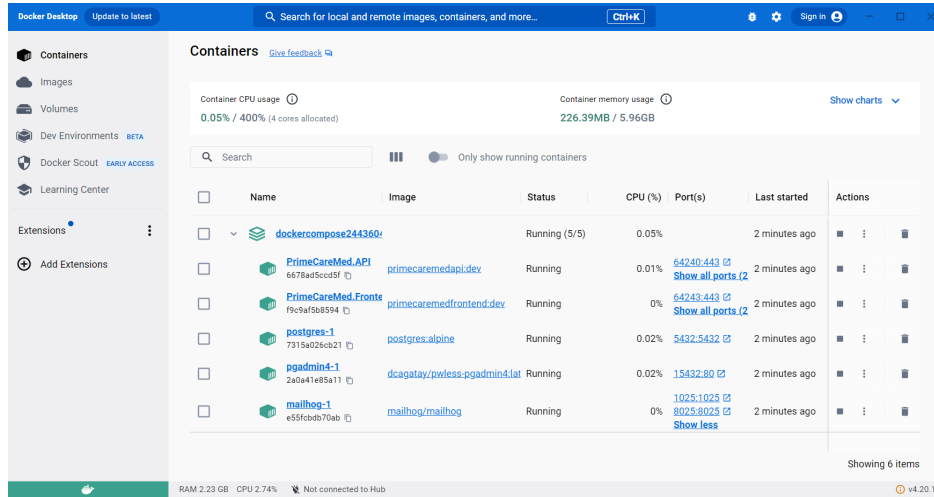
U ispisu 4 prikazana je datoteka Dockerfile u kojoj se stvara datoteka PrimeCareMed.Frontend.dll (*DLL - Dynamic-link library*) koja sadrži izvršni kôd.

```

1 FROM mcr.microsoft.com/dotnet/aspnet:6.0 AS base
2 WORKDIR /app
3 EXPOSE 80
4 EXPOSE 443
5
6 FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build
7 WORKDIR /src
8 COPY ["src/PrimeCareMed.Frontend/PrimeCareMed.Frontend.csproj", "src/
   PrimeCareMed.Frontend/"]
9 RUN dotnet restore "src/PrimeCareMed.Frontend/PrimeCareMed.Frontend.
   csproj"
10 COPY . .
11 WORKDIR "/src/src/PrimeCareMed.Frontend"
12 RUN dotnet build "PrimeCareMed.Frontend.csproj" -c Release -o /app/build
13
14 FROM build AS publish
15 RUN dotnet publish "PrimeCareMed.Frontend.csproj" -c Release -o /app/
   publish /p:UseAppHost=false
16
17 FROM base AS final
18 WORKDIR /app
19 COPY --from=publish /app/publish .
20 ENTRYPOINT ["dotnet", "PrimeCareMed.Frontend.dll"]

```

Ispis 4: Datoteka Dockerfile u projektu PrimeCareMed.Frontend



Slika 7: Pokrenuti spremnici u aplikaciji Docker Desktop

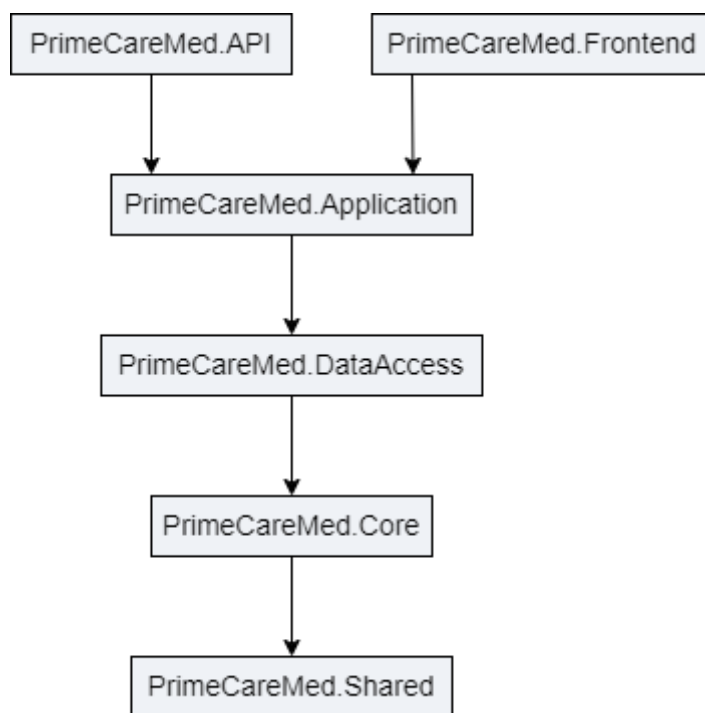
2.3. PostgreSQL

PostgreSQL besplatan je program otvorenog kôda za upravljanje objektno-relacijskim bazama podataka (engl. *ORDBMS - Object-Relational Database Management System*) koristeći jezik SQL (*Structured Query Language*). Dozvoljena su proširenja mogućnosti PostgreSQLa kao što su dodavanje novih tipova podataka, operatora ili funkcija [10]. PostgreSQL trenutno je jedna od najnaprednijih baza podataka otvorenog kôda. Prikaz sheme baze podataka, strukture svih entiteta i druge mogućnosti PostgreSQLa dostupne su u sučelju pgAdmin.

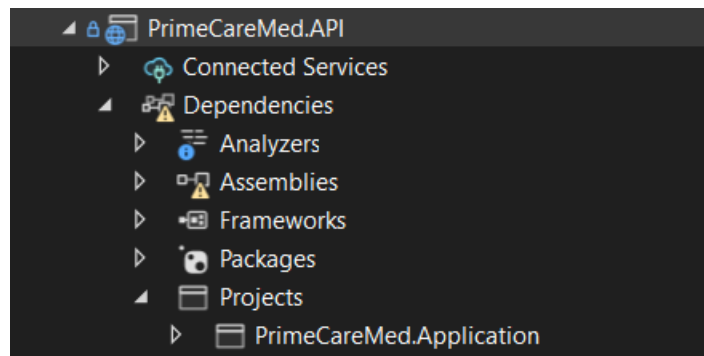
3. Izvedba praktičnog rada

3.1. Postavljanje aplikacije

Za postavljanje aplikacije korišten je predložak **N-Tier** arhitekture s GitHub platforme dostupan na poveznici [11]. N-Tier arhitektura višeslojna je klijent-server arhitektura čija je glavna značajka odvajanje aplikacije u fizičke razine i logičke slojeve. U korištenom predlošku, koristi se jedna fizička razina i pet logičkih slojeva kojima je dodan još jedan logički sloj naziva `PrimeCareMed.Frontend` kao projekt ASP.NET Core Web App u kojem je korišten okvir Razor Pages za izradu aplikacija. Pojedini sloj može koristiti usluge nižeg sloja što je omogućeno dodavanjem ovisnosti na svaki sloj [12] [13]. Na slici 8 prikazane su ovisnosti projekata u korištenoj arhitekturi. Naprimjer, sloj `PrimeCareMed.API` može koristiti usluge sloja `PrimeCareMed.Application` zbog ovisnosti prikazane na slici 9.



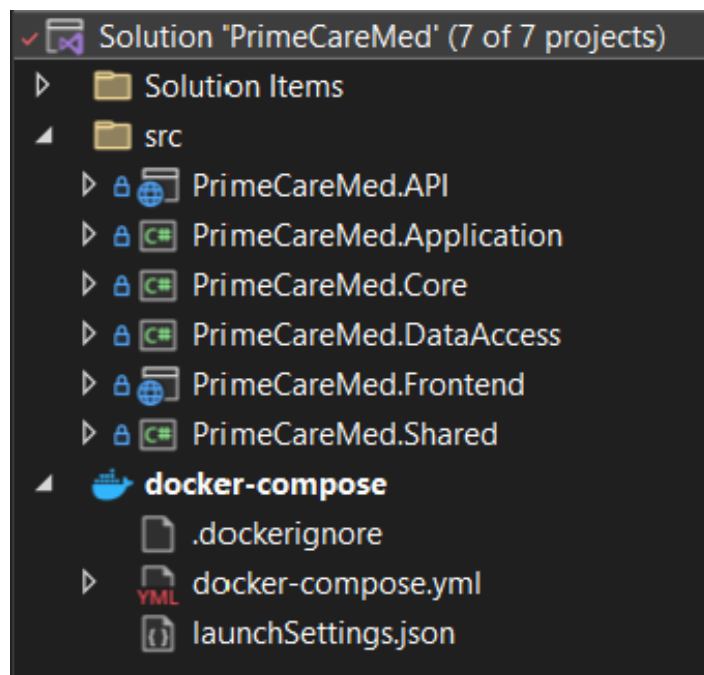
Slika 8: Ovisnosti slojeva



Slika 9: Ovisnosti projekta `PrimeCareMed.API`

3.2. Struktura *PrimeCareMed* aplikacije

Na slici 10 prikazana je struktura aplikacije.



Slika 10: Struktura aplikacije *PrimeCareMed*

Struktura direktorija `src` sastoji se od šest projekata:

- `PrimeCareMed.API` - korišten je primarno za pokretanje ostalih nižih slojeva tj. projekata. Također, sadrži direktorij `Seed` u kojem se nalaze datoteke *JSON* (*JSON - JavaScript Object Notation*) s podacima korištenima za stvaranje elemenata entiteta.
- `PrimeCareMed.Application` - sadrži direktorij `Models` koji sadrži modele za

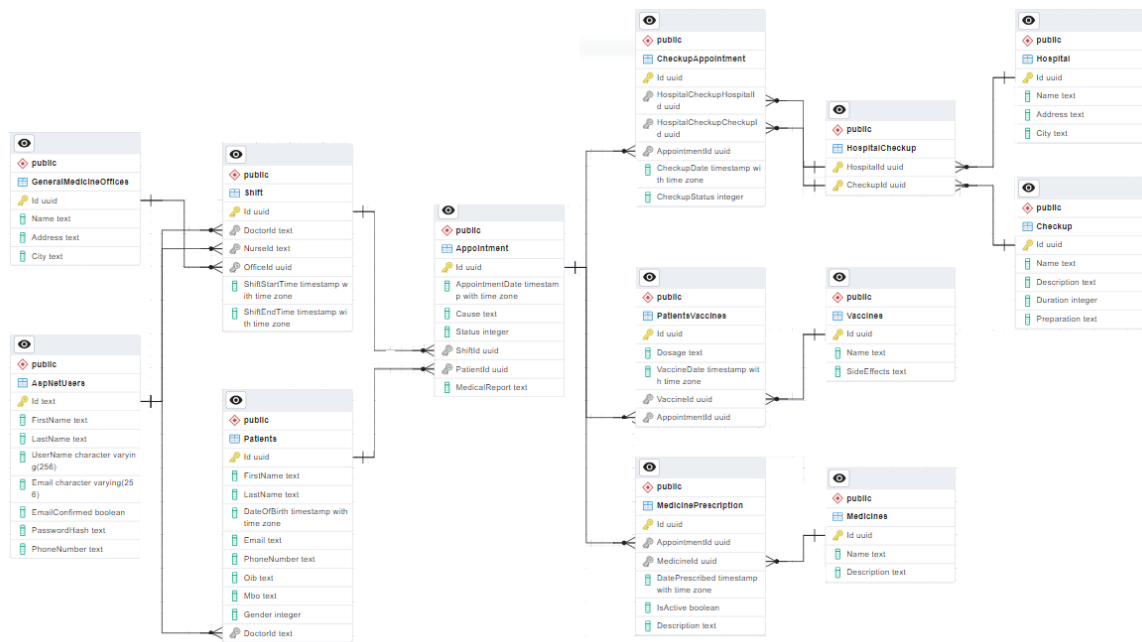
stvaranje, ažuriranje i prikaz pojedinog entiteta te direktorij `Service`

- `PrimeCareMed.Core` - sadrži klase entiteta i podatke tipa `enum`
- `PrimeCareMed.DataAccess` - korišten je za komunikaciju s bazom podataka. Sadrži klasu `DatabaseContext` u kojoj su definirani entiteti i veze među entitetima te stvorenu migraciju. Također, sadrži direktorij `Repositories` u kojem se nalaze repozitoriji koji manipuliraju podacima u bazi podataka.
- `PrimeCareMed.Frontend` - sadrži direktorij `wwwroot` unutar kojeg se nalaze datoteke CSS (*CSS - Cascading Style Sheets*) i JavaScript te poddirektorij `images` za skladištenje slika korištenih na korisničkom sučelju. Sadrži i poddirektorije za pojedine entitete u kojima se nalaze Razor stranice s pripadajućim datotekama *code-behind* za usmjeravanje.
- `PrimeCareMed.Shared` - sadrži zajedničke datoteke ostalih slojeva

3.3. Baza podataka

3.3.1. Struktura relacijske baze podataka

Na slici 11 prikazan je ER (engl. *Entity Relationship*) dijagram relacijske baze podataka u pgAdmin sučelju. Sa slike su izostavljene tablice dobivene korištenjem klase `IdentityUser` prikazane na slici 2.



Slika 11: Dijagram relacijske baze podataka

3.3.2. Entiteti

Entiteti su definirani u projektu `PrimeCareMed.Core`. Na ispisu 5 prikazana je klasa entiteta `Appointment` koja u sebi, uz druge podatkovne članove, sadrži referencu `Patient Patient` koja predstavlja objekt entiteta `Patient`, a `EntityFrameworkCore` stvara *shadow key property* `PatientId` onog tipa koji je primarni ključ u klasi `Patient`.

```

1 using PrimeCareMed.Core.Common;
2 using PrimeCareMed.Core.Enums;
3
4 namespace PrimeCareMed.Core.Entities
5 {
6     public class Appointment : BaseEntity
7     {
8         public DateTime AppointmentDate { get; set; }
9         public string Cause { get; set; }
10        public AppointmentStatus Status { get; set; }
11        public Shift Shift { get; set; }
12        public Patient Patient { get; set; }
13 #nullable enable
14        public string? MedicalReport { get; set; }

```

```

15     public ICollection<PatientsVaccine>? PatientsVaccines { get; set;
        } = new List<PatientsVaccine>();
16     public ICollection<MedicinePrescription>? MedicinePrescriptions {
        get; set; } = new List<MedicinePrescription>();
17     public ICollection<CheckupAppointment>? CheckupAppointments { get
        ; set; } = new List<CheckupAppointment>();
18 #nullable disable
19     }
20 }

```

Ispis 5: Entitet Appointment

Relacija jedan-na-više [14] omogućena je navedenom referencom i definiranom kolekcijom tipa Appointment u klasi Patient prikazanom na ispisu 6.

```

1 public ICollection<Appointment>? Appointments { get; set; } = new List<
    Appointment>();

```

Ispis 6: Kolekcija objekata Appointment u klasi Patient

3.3.3. Interakcija s bazom podataka

Interakcija aplikacije s bazom podataka započinje od klase DbContext koja je korištena za dohvaćanje podataka pomoću upita te za spremanje novih i ažuriranje postojećih elemenata entiteta. Nakon definiranih entiteta, u datoteci DatabaseContext.cs definirane su kolekcije svih entiteta koje su kasnije korištene za pristup podacima u bazi podataka. EntityFrameworkCore pomoću klase modelBuilder automatski stvara jednostavne veze između entiteta. U metodi OnModelCreating korištena je klasa modelBuilder odgovorna za stvaranje samih modela. Koristi definirane kolekcije entiteta DbSet za stvaranje entiteta i veza iz poznatih podataka. Dodatne konfiguracije mogu biti posebno zapisane u metodi.

```

1 public DbSet<Medicine> Medicines { get; set; }
2 public DbSet<GeneralMedicineOffice> GeneralMedicineOffices { get; set; }
3 public DbSet<Vaccine> Vaccines { get; set; }

```

```

4 public DbSet<Patient> Patients { get; set; }
5 public DbSet<Shift> Shift { get; set; }
6 public DbSet<PatientsVaccine> PatientsVaccines { get; set; }
7 public DbSet<MedicinePrescription> MedicinePrescription { get; set; }
8 public DbSet<Appointment> Appointment { get; set; }
9 public DbSet<Hospital> Hospital { get; set; }
10 public DbSet<Checkup> Checkup { get; set; }
11 public DbSet<HospitalCheckup> HospitalCheckup { get; set; }
12 public DbSet<CheckupAppointment> CheckupAppointment { get; set; }
13
14 protected override void OnModelCreating(ModelBuilder builder)
15 {
16     builder.ApplyConfigurationsFromAssembly(Assembly.GetExecutingAssembly()
17         );
18     base.OnModelCreating(builder);
19     builder.Entity<Patient>()
20         .HasIndex(u => u.Oib)
21         .IsUnique();
22
23     builder.Entity<Patient>()
24         .HasIndex(u => u.Mbo)
25         .IsUnique();
26
27     builder.Entity<ApplicationUser>()
28         .HasMany(r => r.Patients)
29         .WithOne(r => r.DoctorId)
30         .IsRequired(false);
31
32     builder.Entity<Shift>()
33         .HasOne(e => e.Nurse)
34         .WithMany(e => e.NursesShifts)
35         .IsRequired();
36
37     builder.Entity<Shift>()
38         .HasOne(e => e.Doctor)
39         .WithMany(e => e.DoctorsShifts)
40         .IsRequired();
41

```

```

42     builder.Entity<GeneralMedicineOffice>()
43         .HasMany(e => e.Shifts)
44         .WithOne(e => e.Office)
45         .IsRequired();
46
47     builder.Entity<Shift>()
48         .HasMany(e => e.Appointments)
49         .WithOne(e => e.Shift)
50         .IsRequired();
51
52     builder.Entity<HospitalCheckup>().HasKey(gu => new
53         {gu.HospitalId, gu.CheckupId });
54
55     builder.Entity<HospitalCheckup>().HasOne(ub => ub.Hospital)
56         .WithMany(x => x.HospitalCheckups).HasForeignKey(h => h.HospitalId);
57
58     builder.Entity<HospitalCheckup>().HasOne(ub => ub.Checkup)
59         .WithMany(x => x.HospitalCheckups).HasForeignKey(h => h.CheckupId);
60 }

```

Ispis 7: Datoteka DatabaseContext.cs

3.3.4. Migracije

Migracije podataka kreirane su naredbom prikazanom u ispisu 8 u konzoli upravitelja paketa (engl. *Package Manager Console*).

```

1 Add-Migration InitialCreate -Project PrimeCareMed.DataAccess -
   StartupProject PrimeCareMed.API -OutputDir "Persistence/Migrations"

```

Ispis 8: Naredba za kreiranje migracije

Nakon izvođenja naredbe, kreirana je migracija naziva `InitialCreate` koja se nalazi u projektu `PrimeCareMed.DataAccess` u direktoriju `Persistence/Migrations`. S obzirom na to da je korištena višeslojna arhitektura, potrebno je navesti `-StartupProject PrimeCareMed.API` koji predstavlja projekt najvišeg sloja koji je postavljen kao zadani pri izvršavanju migracije.

```
1 Update-Database -Project PrimeCareMed.DataAccess -StartupProject
   PrimeCareMed.API -Connection "Host=localhost;Port=5432;Database=
   postgres;Username=admin;Password=root;Integrated Security=true;Pooling
   =true;"
```

Ispis 9: Primjena migracije na bazu podataka

3.3.5. Punjenje podacima

Ovisno o entitetu, punjenje baze podataka podacima (engl. *seeding*) obavljeno je datotekama *JSON* ili generatorom Bogus za lažne podatke specificiranim za .NET jezike. Elementi entiteta *ApplicationUser* i *Patient* zapisani su pomoću generatora Bogus u klasi *DatabaseContextSeed* koji je instaliran kao paket NuGet. U ispisu 10 vidljiva je metoda za punjenje podataka za entitet *Patient* koja prima broj elemenata entiteta koji će biti stvoreni [15].

```
1     public static List<Patient> PatientInit(int count)
2     {
3         var patientFaker = new Faker<Patient>()
4             .RuleFor(p => p.FirstName, f => f.Person.FirstName)
5             .RuleFor(p => p.LastName, f => f.Person.LastName)
6             .RuleFor(p => p.DateOfBirth, f => f.Person.DateOfBirth.Date.
7             ToUniversalTime())
8             .RuleFor(p => p.Email, f => f.Person.Email)
9             .RuleFor(p => p.PhoneNumber, f => f.Person.Phone)
10            .RuleFor(p => p.Oib, f => string.Join("", f.Random.Digits(11)))
11            .RuleFor(p => p.Mbo, f => string.Join("", f.Random.Digits(9)))
12            .RuleFor(p => p.Gender, f => f.PickRandom<Gender>());
13
14            return patientFaker.Generate(count);
15    }
```

Ispis 10: Metoda *PatientInit* za punjenje lažnim podacima

Metoda za spremanje generiranih podataka u bazu podataka iz metode *PatientInit* pozvana je u asinkronoj metodi *SeedDatabaseAsync* koja se izvršava

prilikom pokretanja aplikacije (pogledati ispis 11). Više o asinkronim metodama bit će rečeno u poglavlju 3.4.

```
1 if (!context.Patients.Any())
2     {
3         await context.Patients.AddRangeAsync(PatientsSeed);
4         await context.SaveChangesAsync();
5     }
```

Ispis 11: Metoda AddRangeAsync za spremanje generiranih podataka

Punjenje podacima pomoću datoteka JSON korišteno je za entitete *Medicine*, *Vaccine* i *GeneralMedicineOffice* zbog njihovih specifičnih atributa kao što su imena lijekova ili cjepiva. U ispisu 12 prikazan je dio datoteke `medicines.json` koji će predstavljati jedan element entiteta *Medicine* u bazi podataka dok je u ispisu 13 prikazan dio kôda za deserijalizaciju u listu entiteta tipa *Medicine*. Navedene podatke moguće je generirati uz pomoć paketa `NuGet Bogus.Healthcare` za koji je potrebna plaćena licenca.

```
1 {
2     "Name": "Voltaren",
3     "Description": "Reduces substances in the body that cause pain and
4     inflammation."
5 }
```

Ispis 12: Dio datoteke `medicines.json` koji će predstavljati jedan element entiteta *Medicine* u bazi podataka

```
1     if (!context.Medicines.Any())
2     {
3         var medicinesJson = File.ReadAllText(path + Path.
DirectorySeparatorChar + "medicines.json");
4         var medicines = JsonConvert.DeserializeObject<List<Medicine
>>(medicinesJson);
5         await context.Medicines.AddRangeAsync(medicines);
6         await context.SaveChangesAsync();
```

Ispis 13: Punjenje tablice `Medicine` podacima iz JSON datoteke

3.4. Repozitoriji

Sloj aplikacije najbliži bazi podataka jest repozitorij. Repozitoriji u aplikaciji smješteni su u projektu `PrimeCareMed.DataAccess`, a njihova je uloga dohvaćanje podataka iz baze podataka pomoću upita LINQ te spremanje, ažuriranje i brisanje podataka iz iste [16]. Asinkrone metode prepoznate su po ključnim riječima `await` i `async` koje omogućuju izvršavanje više različitih zahtjeva na bazu podataka istovremeno. U ispisu 14 u metodi `AddAsync` ključnom riječju `await` sprema se novi entitet te se daljnji kôd u metodi ne izvršava dok se asinkrona operacija ne izvrši. Asinkrone metode u svom nazivu sadrže riječ `Async` da bi bile prepoznate.

```
1 namespace PrimeCareMed.DataAccess.Repositories.Impl
2 {
3     public class CheckupAppointmentRepository :
4         ICheckupAppointmentRepository
5     {
6         private readonly DatabaseContext _context;
7         public CheckupAppointmentRepository(DatabaseContext context)
8         {
9             _context = context ?? throw new ArgumentNullException(nameof(
10                context));
11         }
12         public async Task<CheckupAppointment> AddAsync(CheckupAppointment
13            checkupAppointment)
14         {
15             await _context.CheckupAppointment.AddAsync(checkupAppointment
16                );
17             await _context.SaveChangesAsync();
18             return checkupAppointment;
19         }
20         public async Task<IEnumerable<CheckupAppointment>>
21            GetAllCheckupAppointmentsForPatientAsync(Guid PatientId)
```

```

17     {
18         return await _context.CheckupAppointment.OrderByDescending(r
=> r.CheckupDate).Include(r => r.HospitalCheckup).ThenInclude(r => r.
Hospital).Include(r => r.HospitalCheckup).ThenInclude(r => r.Checkup).
Include(r=>r.Appointment).ThenInclude(r=>r.Patient).Where(r => r.
Appointment.Patient.Id == PatientId).ToListAsync();
19     }
20     public async Task DeleteCheckupAppointmentAsync(Guid id)
21     {
22         var deleteItem = _context.CheckupAppointment.FirstOrDefault(r
=> r.Id == id);
23         _context.CheckupAppointment.Remove(deleteItem);
24         await _context.SaveChangesAsync();
25     }
26     public async Task<CheckupAppointment>
GetCheckupAppointmentByIdAsync(string id)
27     {
28         return await _context.CheckupAppointment.FirstOrDefaultAsync(
t => t.Id.ToString() == id);
29     }
30 }
31 }

```

Ispis 14: Repozitorij CheckupAppointment

U ispisu 14 prikazan je dio repozitorija CheckupAppointmentRepository u kojem su korištene različite metode LINQa:

- `OrderByDescending` - silazno sortira skup podataka na temelju zadanog podatka.
- `Include` - učitava podatke povezanih entiteta, u konkretnom slučaju za pojedini element entiteta `CheckupAppointment` bit će dobiveni i podatci entiteta `HospitalCheckup`.
- `ThenInclude` - učitava podatke povezanih entiteta s entitetom dobivenim koristeći metodu `Include`. Uz prethodno navedene podatke, bit će dobiveni podatci entiteta `Hospital` i `Checkup`.
- `Where` - filtrira skup podataka na temelju određenog uvjeta. U konkretnom primjeru,

filtriraju se oni elementi entiteta `CheckupAppointment` kojima je `Appointment.Patient.Id` jednak primljenom argumentu funkcije `PatientId`. S obzirom na to da `Id` u aplikaciji označava primarni ključ, rezultat će biti isti kao da je korišten `FirstOrDefault`.

- `FirstOrDefault` - dohvaća prvu vrijednost elemenata koja zadovoljava uvjet ili vraća zadanu vrijednost ako se ne pronađe nijedan element, najčešće je to vrijednost *null*.

3.5. Servisi

Servisi u aplikaciji predstavljaju sloj zadužen za oblikovanje i obradu podataka te omogućuju njihov prikaz na korisničkom sučelju s odabranim atributima. Servisi primaju podatke iz repozitorija ili iz stranica Razor, a upotrebljavajući *namespace* `AutoMapper` oblikuju podatke po unaprijed definiranom modelu te ih prosljeđuju.

```
1 namespace PrimeCareMed.Application.Services.Impl
2 {
3     public class PatientService : IPatientService
4     {
5         private readonly IMapper _mapper;
6         private readonly IPatientRepository _patientRepository;
7         private readonly IAppointmentService _appointmentService;
8
9         public PatientService(IMapper mapper,
10             IPatientRepository patientRepository,
11             IAppointmentService appointmentService
12         )
13         {
14             _mapper = mapper;
15             _patientRepository = patientRepository;
16             _appointmentService = appointmentService;
17         }
18
19         public async Task<PatientModel> AddAsync(PatientModelForCreate
20             createPatientModel)
```

```

21         var config = new MapperConfiguration(cfg => {
22             cfg.CreateMap<PatientModelForCreate, Patient>();
23         });
24         var patient = config.CreateMapper().Map<Patient>(
createPatientModel);
25         await _patientRepository.AddAsync(patient);
26         return _mapper.Map<PatientModel>(patient);
27     }
28     public Patient EditPatientAsync(PatientModelForCreate
patientModel)
29     {
30         var patient = _mapper.Map<Patient>(patientModel);
31         return _patientRepository.UpdateAsync(patient).Result;
32     }
33 }
34 }

```

Ispis 15: Servis PatientService

U ispisu 15 prikazan je dio servisa PatientService s metodama koje koriste modele i sučelje IMapper. U ispisu 16 vidi se model korišten u formi za kreiranje novog pacijenta, a u ispisu 17 model i podatci korišteni kod prikaza entiteta Patient.

```

1 using PrimeCareMed.Core.Enums;
2 using System.ComponentModel.DataAnnotations;
3
4 namespace PrimeCareMed.Application.Models.Patient
5 {
6     public class PatientModelForCreate
7     {
8         public string Id { get; set; }
9         [Required]
10        [DataType(DataType.Text)]
11        [Display(Name = "First Name")]
12        public string FirstName { get; set; }
13        [Required]
14        [DataType(DataType.Text)]
15        [Display(Name = "Last Name")]

```

```

16     public string LastName { get; set; }
17     [Required]
18     [DataType(DataType.DateTime)]
19     [Display(Name = "Date of birth")]
20     public DateTime DateOfBirth { get; set; }
21     [Required]
22     [DataType(DataType.EmailAddress)]
23     [Display(Name = "Email")]
24     public string Email { get; set; }
25     [Required]
26     [DataType(DataType.Text)]
27     [Display(Name = "Phone number")]
28     public string PhoneNumber { get; set; }
29     [Required]
30     [DataType(DataType.Text)]
31     [Display(Name = "Oib")]
32     public string Oib { get; set; }
33     [Required]
34     [DataType(DataType.Text)]
35     [Display(Name = "Mbo")]
36     public string Mbo { get; set; }
37     [Required]
38     [DataType(DataType.Text)]
39     [Display(Name = "Gender")]
40     public Gender Gender { get; set; }
41 }
42 }

```

Ispis 16: Model PatientModelForCreate

```

1 using PrimeCareMed.Core.Enums;
2
3 namespace PrimeCareMed.Application.Models.Patient
4 {
5     public class PatientModel : BaseResponseModel
6     {
7         public string Mbo { get; set; }
8         public string Oib { get; set; }

```

```

9     public string FirstName { get; set; }
10    public string LastName { get; set; }
11    public DateTime DateOfBirth { get; set; }
12    public Gender Gender { get; set; }
13    public string Email { get; set; }
14    public string PhoneNumber { get; set; }
15    }
16 }

```

Ispis 17: Model PatientModel

3.6. RazorPages izvedba

Razor Pages koristi datoteke `.cshtml` i `.cshtml.cs` za obradu podataka dobivenih iz servisa, njihovu dodatnu obradu, prosljeđivanje te prikaz podataka na korisničkom sučelju. Podacima se u predlošku pristupa pomoću Razor anotacija. Datoteka `.cshtml` sadrži HTML strukturu i sintaksu Razor za generiranje dinamičkog sadržaja, dok datoteka `.cshtml.cs` sadrži metode, svojstva i druge komponente C#-a koje podržavaju funkcionalnost pripadajuće stranice.

3.6.1. `.cshtml` datoteke

Datoteke s ekstenzijom `.cshtml` nazivaju se i Razor predlošcima korištenima za izradu dinamičkih web stranica kombinacijom programskog jezika C#-a i HTML-a (*HTML - HyperText Markup Language*). Mogu sadržavati i razne kontrolne strukture, petlje i uvjete. U ispisu 18 prikazana je datoteka `CreateCheckup.cshtml` koja je korištena pri kreiranju novog pregleda. Za unos se koristi atribut `asp-for` koji povezuje elemente HTML-a s odgovarajućim svojstvima modela, u ovom slučaju modela `CheckupModelForCreate`.

```

1 @page
2 @model PrimeCareMed.Frontend.Pages.Checkup.CreateCheckupModel
3 @{
4     ViewData["Title"] = "New checkup";
5 }
6 <div class="container text-center d-flex align-items-center justify-
    content-center">

```

```

7     <div>
8         <div>
9             <h1>@ViewData["Title"]</h1>
10        </div>
11        <div>
12            <form id="createCheckupForm" method="post">
13                <hr />
14                <div asp-validation-summary="ModelOnly" class="text-
15danger"></div>
16                <div class="form-floating">
17                    <input asp-for="NewCheckup.Name" class="form-control"
18aria-required="true" />
19                    <label asp-for="NewCheckup.Name"></label>
20                    <span asp-validation-for="NewCheckup.Name" class="
21text-danger"></span>
22                </div>
23                <div class="form-floating mt-2">
24                    <b style="color: #002133">
25                        Description <br />
26                    </b>
27                    <textarea rows="8" cols="35" id="Description" name="
28Description"></textarea>
29                </div>
30                <div class="form-floating mt-2">
31                    <input asp-for="NewCheckup.Duration" class="form-
32control" aria-required="true" />
33                    <label asp-for="NewCheckup.Duration"></label>
34                    <span asp-validation-for="NewCheckup.Duration" class=
35"text-danger"></span>
36                </div>
37                <div class="form-floating mt-2">
38                    <b style="color: #002133">
39                        Preparation <br />
40                    </b>
41                    <textarea rows="8" cols="35" id="Preparation" name="
42Preparation"></textarea>
43                </div>
44                <button id="createCheckupSubmit" type="submit" class="w
45-100 btn btn-lg text-white" style="background-color: #006622">Confirm

```



```

    </button>
38         </form>
39     </div>
40 </div>
41 </div>

```

Ispis 18: Datoteka CreateCheckup.cshtml

U formama koje koriste padajući izbornik, korišten je padajući izbornik `Select2` koji omogućuje pretraživanje za koji je napisana Javascript funkcija na dnu datoteke `.cshtml`. Na ispisu 19 moguće je vidjeti primjer Javascript funkcije za padajući izbornik `Select2`.

```

1 <script>
2     $( '#select-patient' ).select2( {
3     theme: "bootstrap-5",
4     width: $( this ).data( 'width' ) ? $( this ).data( 'width' ) : $(
5     this ).hasClass( 'w-100' ) ? '100%' : 'style',
6     placeholder: $( this ).data( 'placeholder' ),
7     } );
8 </script>

```

Ispis 19: Javascript funkcija za padajući izbornik `Select2`

3.6.2. `.cshtml.cs` datoteke

Svaka datoteka `.cshtml` ima svoju datoteku `.cshtml.cs` poznatiju i kao *Code-Behind* koja sadrži kôd C#-a koji podržava logiku i funkcionalnosti povezane s odgovarajućim Razor predloškom. Prilikom poziva stranice, datoteka `.cshtml.cs` prikazuje stranicu ili izvršava metodu `OnGet`. S obzirom na to da je navedena stranica korištena isključivo za unos podataka, dovoljno je navesti model ili atributa entiteta ispod atributa `[BindProperty]` s kojim će se unos sa stranice povezati. Korištenje atributa `[BindProperty]` olakšava manipulaciju podatcima između korisničkog sučelja i server-ske logike.

```

1 using Microsoft.AspNetCore.Authorization;

```

```

2 using Microsoft.AspNetCore.Mvc;
3 using Microsoft.AspNetCore.Mvc.RazorPages;
4 using PrimeCareMed.Application.Models.Checkup;
5 using PrimeCareMed.Application.Services;
6
7 namespace PrimeCareMed.Frontend.Pages.Checkup
8 {
9     [Authorize(Roles = "Administrator, SysAdministrator")]
10    public class CreateCheckupModel : PageModel
11    {
12        private readonly ICheckupService _checkupService;
13        public CreateCheckupModel(
14            ICheckupService checkupService)
15        {
16            _checkupService = checkupService;
17        }
18        [BindProperty]
19        public CheckupModelForCreate NewCheckup { get; set; }
20
21        public async Task<IActionResult> OnPostAsync(string Description,
22            string Preparation)
23        {
24            NewCheckup.Description = Description;
25            NewCheckup.Preparation = Preparation;
26            try
27            {
28                await _checkupService.AddAsync(NewCheckup);
29                return RedirectToPage("ViewAllCheckups");
30            }
31            catch (Exception ex)
32            {
33                Console.WriteLine(ex.Message);
34                return Page();
35            }
36        }
37    }

```

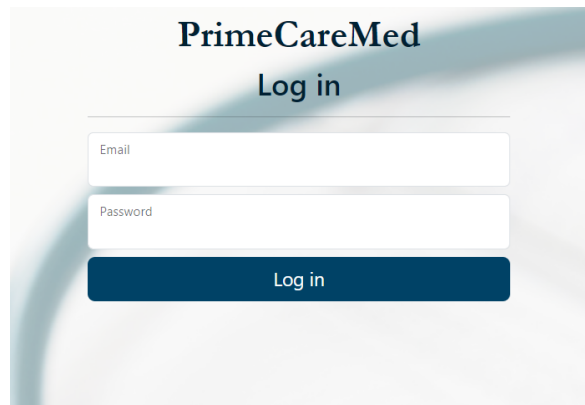
Ispis 20: Datoteka CreateCheckup.cshtml.cs

Datoteka `.cshtml` sadrži formu za unos s metodom `post` koja se izvršava u metodi `OnPostAsync`. Metoda prima parametre `Description` i `Preparation` pomoću `HTML` atributa `name`. Spremaju se odvojeno u model, dok su atributi modela `Name` i `Duration` povezani pomoću navedenog atributa `asp-for`. Poziva se asinkrona metoda `AddAsync` iz servisa koja dalje model mapira u pripadajući entitet `Checkup` te ga šalje u repozitorij koji ga zatim sprema u bazu podataka. Metoda `OnPostAsync` ima povratnu vrijednost `ActionResult` koja omogućava preusmjerenje na željenu stranicu. U slučaju uspješnog spremanja elementa entiteta, `ActionResult` preusmjerava korisnika na stranicu na kojoj su prikazani svi elementi entiteta `Checkup` to jest na stranicu `ViewAllCheckups`.

3.7. Autorizacija

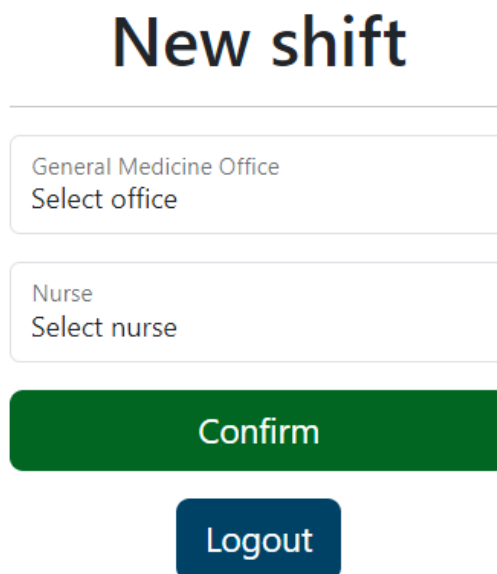
U ispisu 20 prikazan je primjer korištenja atributa `[Authorize]` s parametrima koji dopušta izvršavanje metode `OnGet` ili prikaz stranice autoriziranim korisnicima s ulogom `Administrator` ili `SysAdministrator`. Korištenje atributa `[Authorize]` bez parametara bila bi sama autentikacija i dopustilo bi se izvršavanje samo autenticiranim korisnicima aplikacije [17].

3.8. Funkcionalnosti aplikacije



Slika 12: Forma za prijavu

Na slici 12 prikazana je forma za prijavu korisnika dobivena pomoću Scaffold Identityja. Nakon uspješnog unosa korisničkog imena i lozinke, korisnik se autentificira u sustav ili mu se odbija pristup u slučaju pogrešno unesenih podataka. Korisnici s ulogom `Administrator` i `SysAdministrator`, nakon uspješne autentifikacije, preusmjeravaju se na glavni aplikacijski izbornik. Autenticirani korisnici s ulogom `Doctor` ili `Nurse` preusmjeravaju se na formu za odabir smjene te nemaju pravo korištenja aplikacije dok ne ispune taj međukorak. Provjera poveznice doktora, medicinske sestre i pripadajuće smjene implementirana je kolačićima (engl. *cookies*).



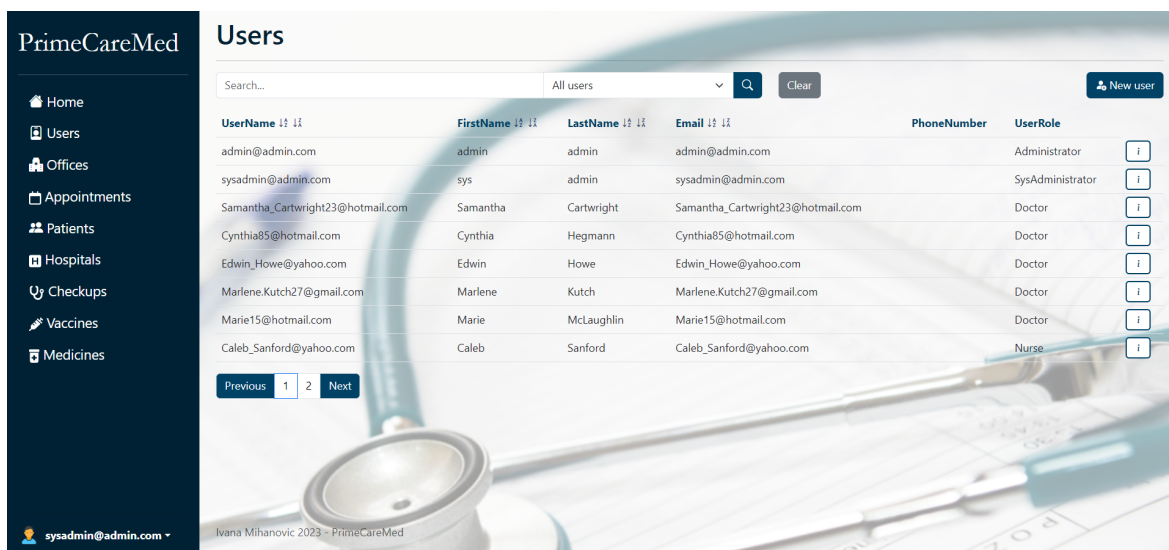
Slika 13: Forma za odabir smjene

Nakon prijave u sustav, korisnici s ulogom Doctor ili Nurse preusmjereni su na stranicu za odabir smjene. Ako je za trenutnog korisnika već odabrana smjena, bit će preusmjeren na glavni izbornik. Nakon odabrane smjene, podatci o novoj smjeni spremljeni su u kolačiće i korišteni za prikaz na izborniku što je vidljivo na slici 20.



Slika 14: Početna stranica aplikacije

Na slici 14 prikazana je početna stranica aplikacije za korisnika s ulogom SysAdministrator s izbornikom s lijeve strane. Broj izbora na spomenutom izborniku ovisi o ulozi korisnika.



Slika 15: Prikaz svih korisnika aplikacije

Korisnicima s ulogom Administrator ili SysAdministrator omogućen je

prikaz svih korisnika aplikacije. Na stranicu su implementirane mogućnosti pretrage, filtriranja i sortiranja zapisa. Za svakog korisnika postoji i botun s kojim je omogućeno uređivanje podataka i brisanje pacijenta iz sustava.

Slika 16: Unos novog pacijenta

Slika 16 prikazuje formu za unos novog pacijenta u sustav. Prilikom kreiranja novog pacijenta u sustav unose se svi potrebni podatci. Najvažniji su podatci OIB i MBO koji jedinstveno identificiraju pacijenta. OIB i MBO tekstualna su *string* polja kako bi se omogućilo pravilno spremanje vrijednosti u bazu podataka ako polje počinje s nulom. U svrhu zaštite integriteta podataka ograničen je unos samo znamenki te se OIB sastoji od jedanaest, a MBO od devet znamenki. Sva su polja za unos obavezna osim odabira određenog liječnika.

Slika 17: Informacije o pacijentu

Slika 17 prikazuje informacije o pacijentu. Klikom na poveznice moguće je vidjeti sve preglede, recepte, cjepiva i dolaske pacijenta. Omogućeno je i uređivanje podataka pacijenta.

The screenshot shows the 'New checkup' form in the PrimeCareMed system. The form is titled 'New checkup' and is set against a background image of a stethoscope and medical charts. The form contains the following fields and sections:

- Name:** A text input field containing 'Abdominal Ultrasound'.
- Description:** A text area containing the text: 'A transducer will be placed over various locations on your abdomen. Sound waves will bounce off certain organs and tissue in your body. This creates "echoes" that are reflected back to the transducer, which converts them to electronic signals. A computer then processes the signals into pictures and shows them on a television monitor.'
- Duration:** A text input field containing '30'.
- Preparation:** A text area containing the text: 'You must not eat or drink for eight hours before your exam. Water and taking medication is okay. If ultrasound pelvis is also being done, for female patients, please drink 32 ounces of water one hour before the scan. You can go to the bathroom to relieve yourself, as long as you keep drinking water. Male patients do not need to have a full bladder.'
- Confirm:** A green button at the bottom of the form.

The left sidebar of the PrimeCareMed system is visible, showing navigation options: Home, Users, Offices, Appointments, Patients, Hospitals, Checkups, Vaccines, and Medicines. The user is logged in as 'sysadmin@admin.com'.

Slika 18: Forma za unos novog elementa entiteta Checkup

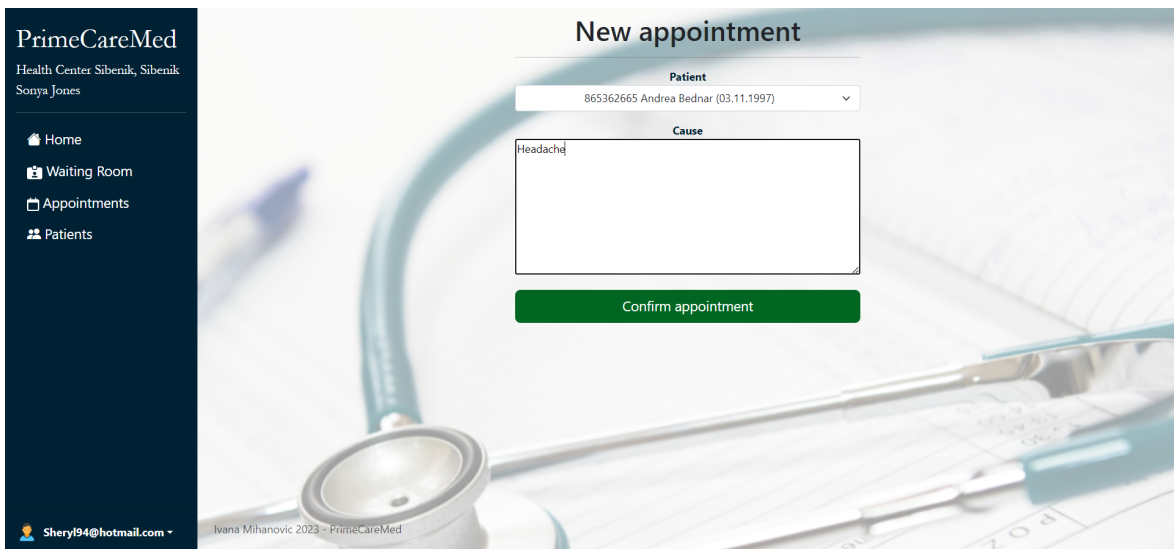
Na slici 18 prikazana je forma za unos. Za navedenu stranicu prikazani su ispisi 3.6.1 za datoteku .cshtml i ispis 3.6.2 za datoteku .cshtml.cs. Nakon spremanja, korisnik je preusmjeren na stranicu prikazanu na slici 19.

The screenshot shows the 'Checkups' list view in the PrimeCareMed system. The page title is 'Checkups'. There is a search bar at the top with a search icon and a 'Clear' button. A 'New checkup' button is located in the top right corner. The table below displays the following data:

Name	Description	Duration	Preparation
Abdominal Ultrasound	A transducer will be placed over various locations on your abdomen. Sound waves will bounce off certain organs and tissue in your body. This creates "echoes" that are reflected back to the transducer, which converts them to electronic signals. A computer then processes the signals into pictures and shows them on a television monitor. These moving images may be viewed immediately or photographed for further study.	30 minutes	You must not eat or drink for eight hours before your exam. Water and taking medication is okay. If ultrasound pelvis is also being done, for female patients, please drink 32 ounces of water one hour before the scan. You can go to the bathroom to relieve yourself, as long as you keep drinking water. Male patients do not need to have a full bladder.

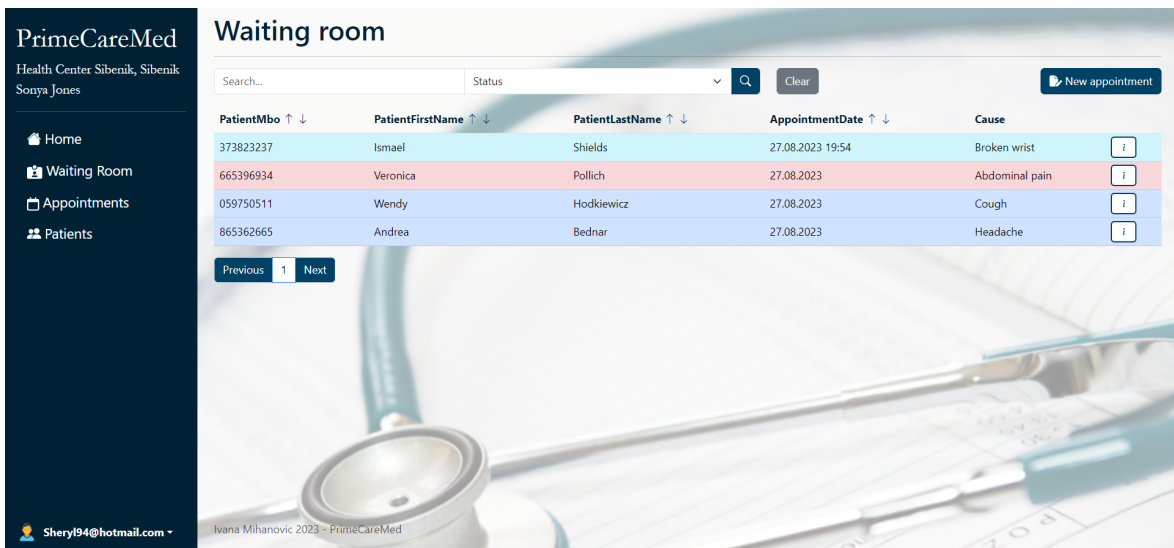
At the bottom of the table, there are navigation buttons: 'Previous', '1', and 'Next'. The left sidebar of the PrimeCareMed system is visible, showing navigation options: Home, Users, Offices, Appointments, Patients, Hospitals, Checkups, Vaccines, and Medicines. The user is logged in as 'sysadmin@admin.com'.

Slika 19: Prikaz svih elemenata entiteta Checkup



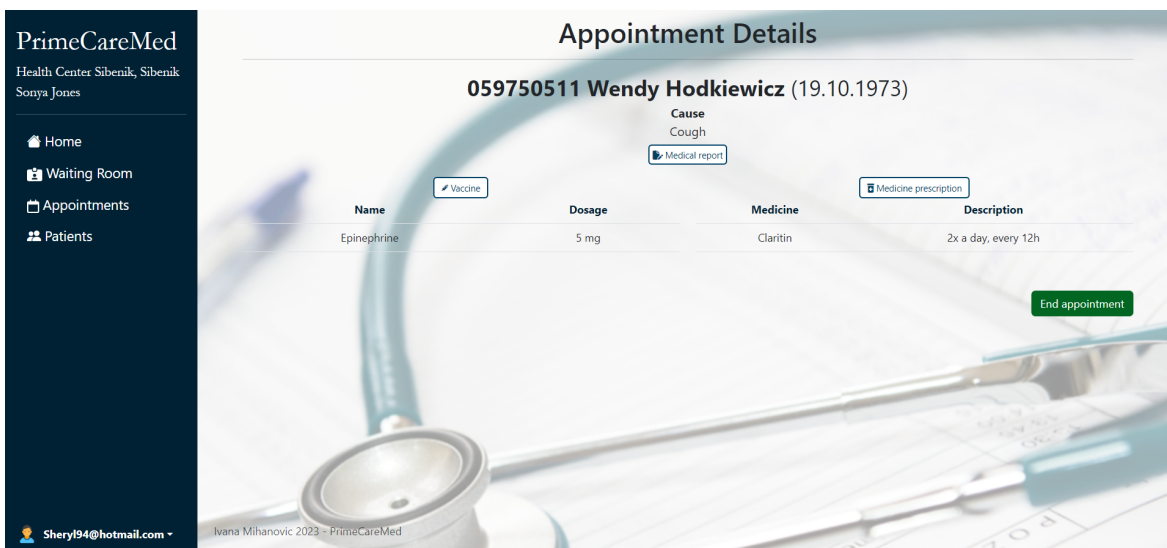
Slika 20: Unos novog pregleda u čekaonicu

Na slici 20 prikazan je unos novog pregleda u čekaonicu za koji se odabire pacijent koristeći padajući izbornik `Select2` i unosi se razlog dolaska. Nakon unosa, korisnik je preusmjeren na stranicu koja predstavlja čekaonicu `WaitingRoom` ili na prikaz svih pregleda (ovisno o ulozi).



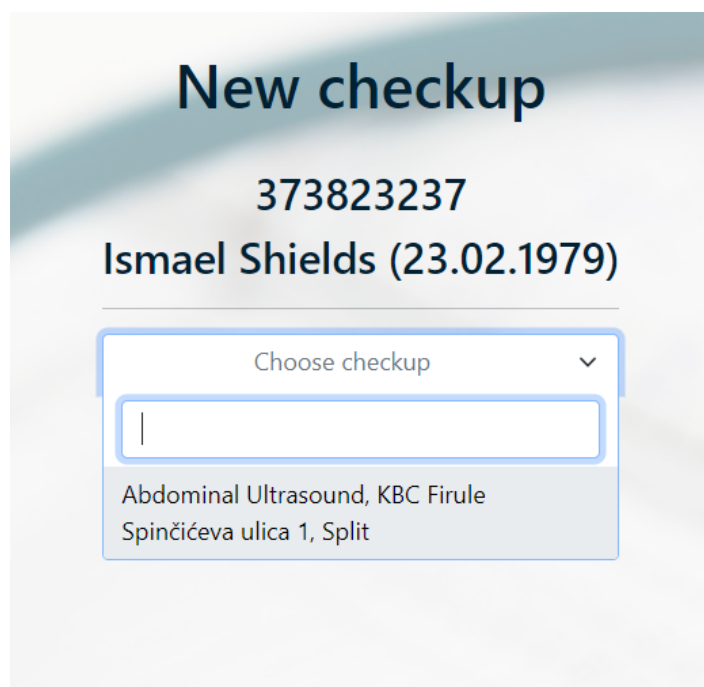
Slika 21: Čekaonica

Slika 21 prikazuje čekaonicu za korisnike s ulogama `Doctor` i `Nurse`. Pojedini status pregleda, označen je zasebnom bojom.



Slika 22: Detalji pregleda

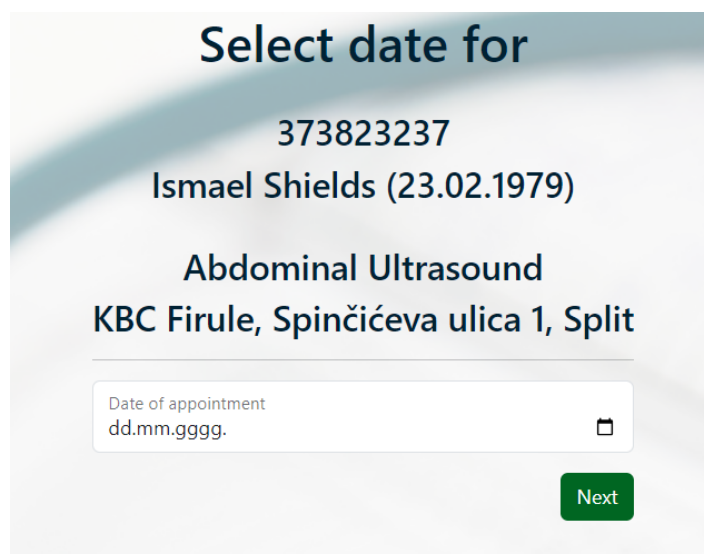
Pritiskom na botun za detalje pregleda, otvara se stranica prikazana na slici 22 na kojoj je omogućen unos novih recepata za lijek, cjeviva i izvješća. Pritiskom na botun End appointment, status pregleda mijenja se na Done i tada su izmjene dostupne samo korisniku s ulogom SysAdministrator. Na slikama 23, 24 i 25 prikazan je postupak narudžbe pacijenta na pregled u bolnici.



Slika 23: Odabir pregleda

Na slici 23 prikazan je početak narudžbe pacijenta na pregled u kojem liječnik odabire

pregled pomoću padajućeg izbornika `Select`2.



Select date for

373823237
Ismael Shields (23.02.1979)

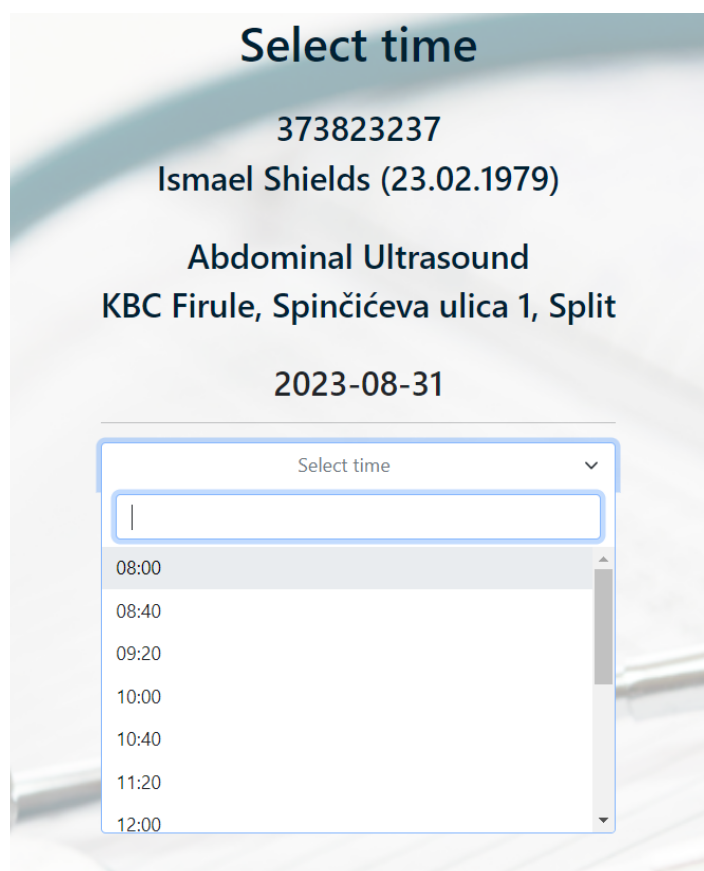
Abdominal Ultrasound
KBC Firule, Spinčićeva ulica 1, Split

Date of appointment
dd.mm.gggg.

Next

Slika 24: Odabir datuma pregleda

Nakon odabira pregleda, liječnik odabire datum pregleda te je preusmjeren na stranicu prikazanu na slici 25 za odabir termina pregleda.



Select time

373823237
Ismael Shields (23.02.1979)

Abdominal Ultrasound
KBC Firule, Spinčićeva ulica 1, Split

2023-08-31

Select time

08:00
08:40
09:20
10:00
10:40
11:20
12:00

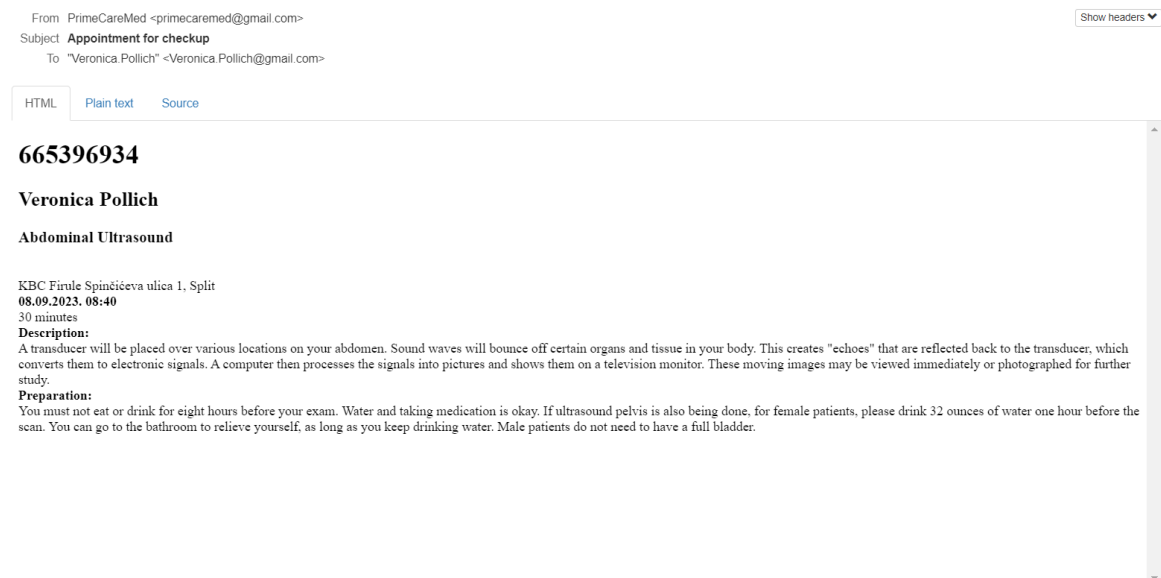
Slika 25: Odabir termina pregleda

Prikazani su samo dostupni termini pregleda koji su generirani za pojedini pregled na temelju njegova trajanja. Za svaki termin, moguće je napraviti tri narudžbe. Kada su tri narudžbe za isti pregled u isto vrijeme kreirane sa statusom `Active`, navedeni termin ne prikazuje se u padajućem izborniku. Otkazivanje pregleda prikazano je na slici 26



Slika 26: Otkazivanje pregleda

Na slici 27 prikazana je poruka poslana pomoću alata MailHog koji omogućava pregled i simulaciju slanja elektroničke pošte. Poruka se kreira prilikom spremanja nove narudžbe na pregled.



Slika 27: Poruka kreirana pomoću alata MailHog

4. Zaključak

Rezultat ovog završnog rada je web aplikacija za upravljanje podacima u primarnoj zdravstvenoj zaštiti s naglaskom na jednostavnost i pristupačnost korisnicima s manje računalnog obrazovanja. Aplikacija omogućuje liječnicima i medicinskim sestrama unos pregleda, pacijenata, recepata za lijek i cjeviva, uz dodatne funkcionalnosti za doktora kao što je unos izvješća o pregledu i narudžba na pregled u bolnici. Korisnik s ulogom Administrator zadužen je za unos novih cjeviva, lijekova, pregleda u bolnici i ureda ali nema pristup popisu pacijenata dok su korisniku s ulogom SysAdministrator dostupne sve funkcionalnosti.

U radu je korišten velik broj različitih tehnologija u svrhu učenja istih. Svaka od opisanih tehnologija pruža jednostavnost i različite mogućnosti prilikom izrade web aplikacije. Okvir ASP.NET Core zbog svojih značajki kao što su ugrađene zaštite od sigurnosnih prijetnji, rad aplikacija na različitim operacijskim sustavima, brža obrada zahtjeva i snažna podrška za asinkrono programiranje postaje jedan od najkorištenijih okvira u svijetu programiranja.

S obzirom na to da primarna zdravstvena zaštita obuhvaća velik spektar raznolikih podataka o pacijentima, web aplikacija *PrimeCareMed* ima potencijal za daljnje unapređenje i proširenje. Zdravstvo je širok pojam pa postoje mnoge funkcionalnosti koje mogu biti dane da se osigura maksimalna iskoristivost sustava. Aplikaciju bi bilo korisno nadograditi statističkim podacima koji bi se iskoristili za bolji rad sustava, naprimjer broj otkazanih pregleda ili onih na koji pacijent nije došao. Također, potrebna je integracija s drugim sustavima da bi aplikacija bila u potpunosti upotrebljiva.

Literatura

- [1] Microsoft Corporation, “Microsoft Documentation - ASP.NET Core overview,” <https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-6.0>, posjećeno 14.8.2023.
- [2] —, “Microsoft Documentation - Csharp introduction,” <https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/>, posjećeno 12.6.2023.
- [3] —, “Microsoft Documentation - NuGet introduction,” <https://learn.microsoft.com/hr-hr/nuget/what-is-nuget>, posjećeno 14.8.2023.
- [4] —, “Microsoft Documentation - Entity Framework Core ORM,” <https://learn.microsoft.com/en-us/ef/core/>, posjećeno 13.8.2023.
- [5] —, “Microsoft Documentation - Razor Pages introduction,” <https://learn.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-7.0&tabs=visual-studio>, posjećeno 14.8.2023.
- [6] <https://github.com/dotnet/aspnetcore>, posjećeno 14.8.2023.
- [7] Microsoft Corporation, “Microsoft Documentation - Scaffold Identity,” <https://learn.microsoft.com/en-us/aspnet/core/security/authentication/scaffold-identity?view=aspnetcore-6.0&tabs=visual-studio>, posjećeno 12.6.2023.
- [8] —, “Microsoft Documentation - Container Tools For Docker,” <https://learn.microsoft.com/en-us/visualstudio/containers/overview>, posjećeno 12.6.2023.
- [9] —, “Microsoft Documentation - Multi-container app with Docker Compose,” <https://learn.microsoft.com/en-us/visualstudio/containers/tutorial-multicontainer>, posjećeno 12.6.2023.
- [10] PostgreSQL Global Development Group, “PostgreSQL - RDBMS,” <https://www.postgresql.org/>, posjećeno 13.8.2023.
- [11] Grigoras Alexandru, “N-Tier-Architecture,” <https://github.com/nuyonu/N-Tier-Architecture>, posjećeno 28.3.2023.

- [12] Microsoft Corporation, “Microsoft Documentation - N-Tier Architecture,” <https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/n-tier>, posjećeno 5.4.2023.
- [13] Tarnum Java SRL, “Baeldung - N-Tier Architecture,” <https://www.baeldung.com/cs/n-tier-architecture>, posjećeno 5.4.2023.
- [14] Microsoft Corporation, “Microsoft Documentation - One-to-many relationships,” <https://learn.microsoft.com/en-us/ef/core/modeling/relationships/one-to-many>, posjećeno 12.6.2023.
- [15] Brian Chavez, “GitHub Repository - Bogus,” <https://github.com/bchavez/Bogus>, posjećeno 12.6.2023.
- [16] Microsoft Corporation, “Microsoft Documentation - LINQ,” <https://learn.microsoft.com/en-us/dotnet/csharp/linq/>, posjećeno 12.6.2023.
- [17] —, “Microsoft Documentation - ASP.NET Core Authentication,” <https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.authentication?view=aspnetcore-6.0>, posjećeno 14.8.2023.

Dodatci

Popis slika

1	NuGet Package Manager i Package Manager Console	4
2	Baza podataka nakon nasljeđivanja klase <code>IdentityUser</code>	6
3	Identity Scaffolded Item	7
4	Uključivanje Docker podrške prilikom kreiranja projekta	7
5	Uključivanje Docker podrške na postojeći projekt	8
6	Projekt <code>docker-compose</code>	8
7	Pokrenuti spremnici u aplikaciji Docker Desktop	11
8	Ovisnosti slojeva	12
9	Ovisnosti projekta <code>PrimeCareMed.API</code>	13
10	Struktura aplikacije <i>PrimeCareMed</i>	13
11	Dijagram relacijske baze podataka	15
12	Forma za prijavu	31
13	Forma za odabir smjene	31
14	Početna stranica aplikacije	32
15	Prikaz svih korisnika aplikacije	32
16	Unos novog pacijenta	33
17	Informacije o pacijentu	33
18	Forma za unos novog elementa entiteta <code>Checkup</code>	34
19	Prikaz svih elemenata entiteta <code>Checkup</code>	34
20	Unos novog pregleda u čekaonicu	35
21	Čekaonica	35
22	Detalji pregleda	36
23	Odabir pregleda	36
24	Odabir datuma pregleda	37
25	Odabir termina pregleda	37
26	Otkazivanje pregleda	38
27	Poruka kreirana pomoću alata <code>MailHog</code>	38

Popis ispisa kôda

1	Naredba za dodavanje NuGet paketa	4
2	Nasljeđivanje klase IdentityDbContext	5
3	Datoteka docker-compose.yml	8
4	Datoteka Dockerfile u projektu PrimeCareMed.Frontend	10
5	Entitet Appointment	15
6	Kolekcija objekata Appointment u klasi Patient	16
7	Datoteka DatabaseContext.cs	16
8	Naredba za kreiranje migracije	18
9	Primjena migracije na bazu podataka	19
10	Metoda PatientInit za punjenje lažnim podacima	19
11	Metoda AddRangeAsync za spremanje generiranih podataka	20
12	Dio datoteke medicines.json koji će predstavljati jedan element entiteta Medicine u bazi podataka	20
13	Punjenje tablice Medicine podacima iz JSON datoteke	20
14	Repozitorij CheckupAppointment	21
15	Servis PatientService	23
16	Model PatientModelForCreate	24
17	Model PatientModel	25
18	Datoteka CreateCheckup.cshtml	26
19	Javascript funkcija za padajući izbornik Select2	28
20	Datoteka CreateCheckup.cshtml.cs	28