

**SVEUČILIŠTE U SPLITU**  
**SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE**

Preddiplomski stručni studij Računarstvo

**IVANA MIHANOVIĆ**

**Z A V R Š N I   R A D**

**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 podatcima u primarnoj zdravstvenoj zaštiti

**Mentor:** Nikola Grgić, viši predavač

Split, rujan 2023.

# Sadržaj

<b>Sažetak</b>	<b>1</b>
<b>1 Uvod</b>	<b>2</b>
<b>2 Korištene tehnologije</b>	<b>3</b>
2.1 ASP.NET Core . . . . .	3
2.1.1 C . . . . .	3
2.1.2 NuGet upravitelj paketa . . . . .	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
<b>3 Izvedba praktičnog rada</b>	<b>13</b>
3.1 Postavljanje aplikacije . . . . .	13
3.2 Struktura <i>PrimeCareMed</i> aplikacije . . . . .	13
3.3 Baza podataka . . . . .	15
3.3.1 Struktura relacijske baze podataka . . . . .	15
3.3.2 Entiteti . . . . .	15
3.3.3 Interakcija s bazom podataka . . . . .	17
3.3.4 Migracije . . . . .	18
3.3.5 Punjenje podatcima . . . . .	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

<b>4 Zaključak</b>	<b>37</b>
<b>Literatura</b>	<b>38</b>
<b>Dodatci</b>	<b>39</b>

## Sažetak

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

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

## Summary

### Developing web application for primary healthcare data management

The goal of this final thesis is to develop a web application for data management in primary health care. The *PrimeCareMed* application is built using Razor Pages which are part of the ASP.NET Core framework. The application is used by a user who can have one of four different roles on which depend functionalities that a particular user can do. The most important functionalities of the application are logging into the system and adding new patient appointments. Editing of appointments is achieved by inputting medical report, vaccinations and prescriptions for each examination. This paper will describe the way in which the application was implemented and the presented technologies that were used.

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

# 1. Uvod

Uzveši 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 korištenje te uzeti u obzir korištenje aplikacije od strane onih korisnika koji nisu računalno obrazovani. U svrhu toga izrađena je web aplikacija *PrimCareMed*.

Pristup aplikaciji imaju četiri vrste korisnika o čijoj ulozi ovise funkcionalnosti koje mogu raditi: SysAdministrator, administrator, liječnik ili liječnica i medicinska sestra ili tehničar. Liječnik i medicinska sestra unose nove preglede u čekaonicu, imaju uvid u detalje samog 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 uz uvjet da su matični broj osigurane osobe i osobni identifikacijski broj jedinstveni. SysAdministrator ima omogućene sve funkcionalnosti, dok sam administrator ima mogućnosti unosa novih liječnika, medicinskih sestara ili tehničara, lijekova, cijepiva te samih 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 sama aplikacija implementirana.

## 2. Korištene tehnologije

### 2.1. ASP.NET Core

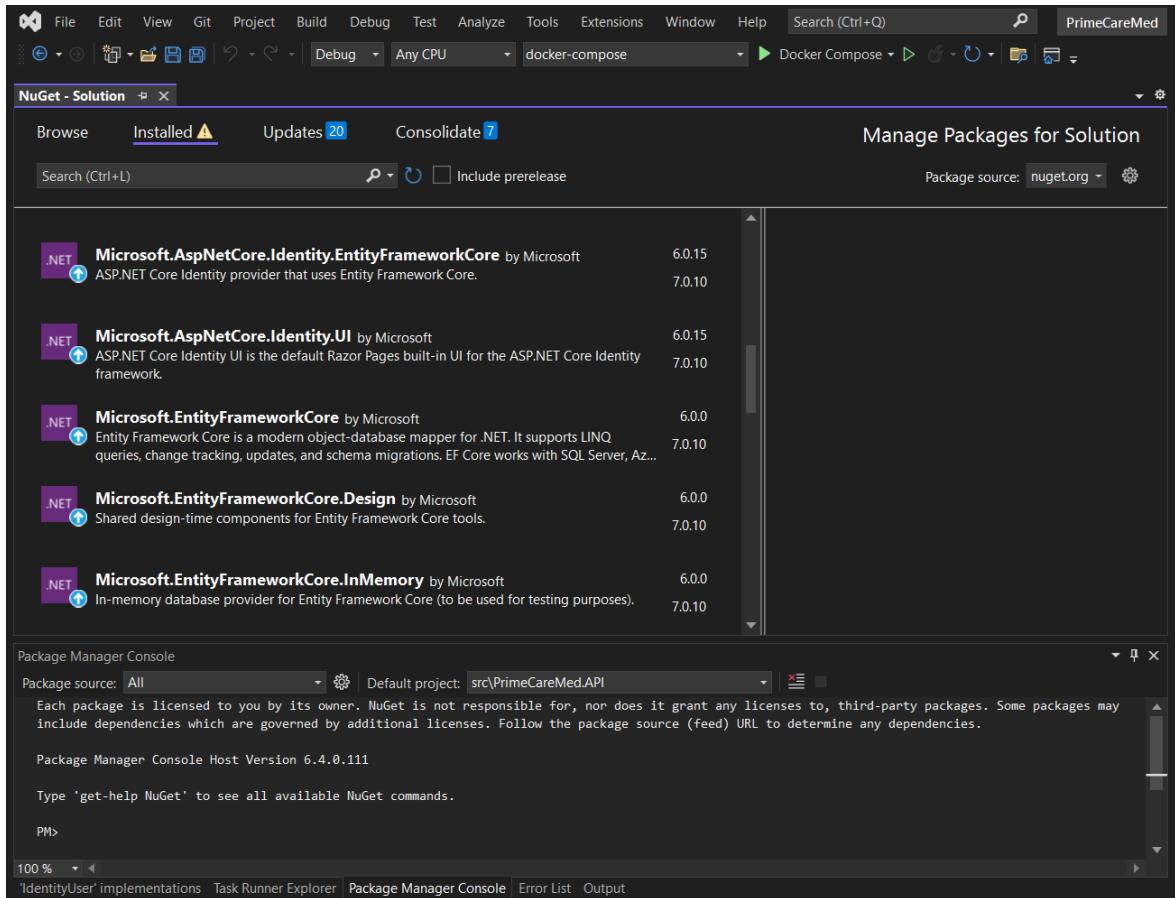
ASP.NET Core (engl. *Active Server Pages Network Enabled Technologies Core*) višeplatformski je okvir otvorenog kôda korišten za izradu aplikacija. Inačica je ASP.NET okvira opće namjene koja se može koristiti na Windows, Linux, macOS operacijskim sustavima te Dockeru. ASP.NET Core stvoren je uzevši najkorisnije značajke .NET-a i .NET Corea. Pomoću ASP.NET Core okvira 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.

#### 2.1.1. C

C# je objektno-orientiran programski jezik stvoren za rad s .NET okvirom. Svrha C#-a jest da bude jednostavan i moderan programski jezik dobiven kombinacijom elemenata C i C++ jezika s utjecajem Java jezika te time postaje robustan jezik za razvoj softvera. Uveden je i koncept LINQ (engl. *Language Integrated Query*) alata za postavljanje upita i manipuliranje podatcima 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. C# jezik podržava i automatsko upravljanje memorijom (engl. *auto-garbage collection*). Uz prethodno navedene, neke od glavnih karakteristika C#-a su višenitost (engl. *multithreading*) i stroga tipizacija koja zahtijeva određivanje tipova podataka tijekom kompajliranja.

#### 2.1.2. NuGet upravitelj paketa

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. Paketi su ZIP datoteke s ekstenzijom `.nupack` ili `.nupkg` koje je također moguće i izraditi pomoću aplikacije NuGet i pohraniti ih u privatni ili javni repozitorij. [1]



**Slika 1:** NuGet Package Manager i Package Manager Console

### 2.1.3. Entity Framework Core

Entity Framework Core višeplatformska je biblioteka otvorenog kôda koja omogućuje pristup bazi podataka preko izvornog kôda pomoću ORM-a (*Object-relational mapping*). Entity Framework Core radi na Windows, MacOS i Linux operacijskim sustavima a omogućen je instalacijom NuGet paketa 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 MVC (*Model-View-Controller*) modelu. Najvažnija značajka kojom se Razor Pages model razlikuje od MVC modela jest .cshtml datoteka te .cshtml.cs datoteka 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. Primjer upravljanja modelom bit će prikazan u poglavljju 3.6

#### **2.1.5. ASP.NET Core Identity**

ASP.NET Core Identity dostupan je kao Razor Class biblioteka omogućena instalacijom NuGet paketa. 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 GitHub platformi [2]. U aplikaciji *PrimeCareMed*, instalirani su:

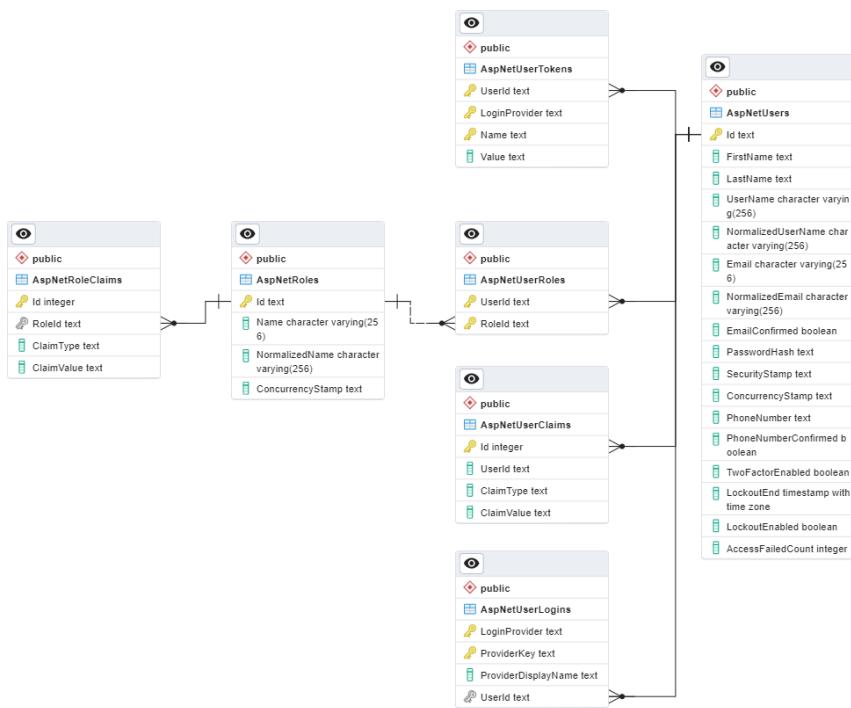
```
Microsoft.AspNetCore.Identity.EntityFrameworkCore
```

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

NuGet paketi čime je omogućeno korištenje `IdentityUser` klase u svrhu naslijeda od strane `ApplicationUser` klase. Uz naslijedivanje `IdentityUser` klase, potrebno je naslijediti `IdentityDbContext` tipa  `ApplicationUser` prikazano na ispisu 2

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

**Ispis 2:** Naslijedivanje `IdentityDbContext` klase

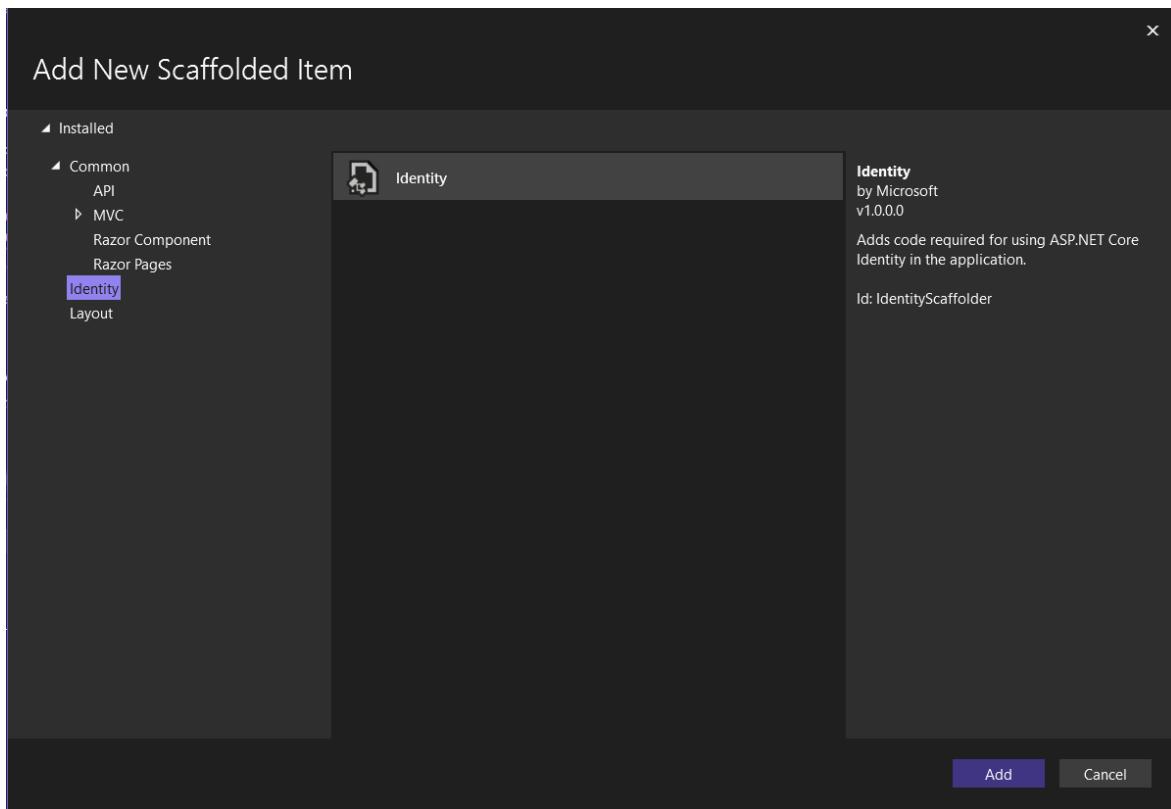


**Slika 2:** Baza podataka nakon naslijeda IdentityUser klase

### 2.1.5.1. Scaffold Identity

Desnim klikom na PrimeCareMed.Frontend > Add >

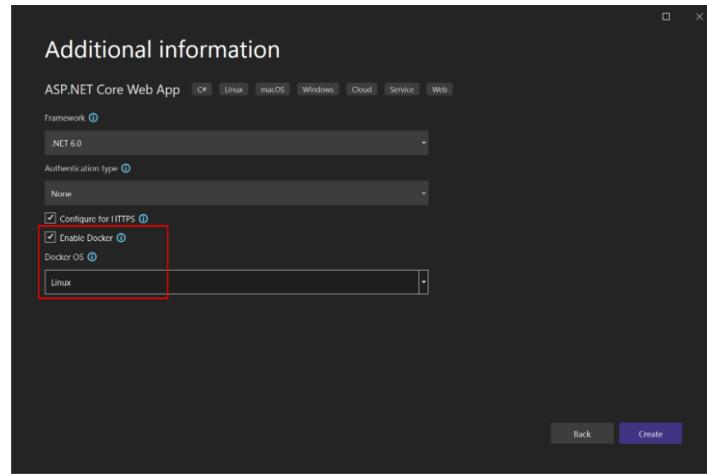
New Scaffolded Item otvara se prozor prikazan na slici 3 te je omogućeno dodavanje opcije Identity. Dodavanjem Identity Scaffoldera na PrimeCareMed.Frontend projekt dobivene su unaprijed kreirane Razor stranice koje omogućuju funkcionalnosti povezane s upravljanjem korisničkih računa. [3]



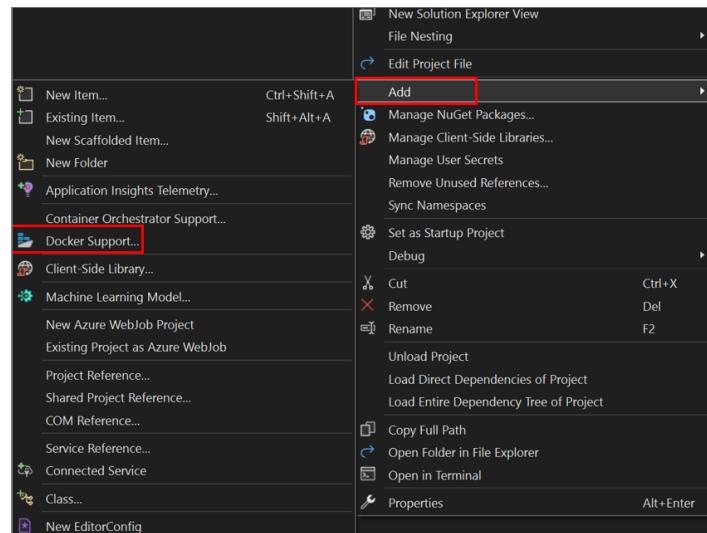
**Slika 3:** Identity Scaffolded Item

## 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 Enable Docker opcija prikazana na slici 4. Na postojeći projekt PrimeCareMed.API Docker podrška omogućena je dodavanjem *Docker Support* opcijom prikazanim na slici 5. [4] [5]

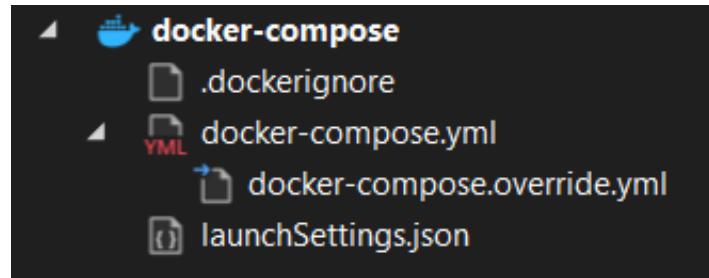


Slika 4: Uključivanje Docker podrške prilikom kreiranja projekta



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 `.dockerignore`, `docker-compose.yml` i `launchSettings.json` datoteke vidljive na slici 6 dok se `Dockerfile` datoteke nalaze u pripadajućim projektima: `PrimeCareMed.Frontend` i `PrimeCareMed.API`.



Slika 6: docker-compose projekt

U ispisu 3 prikazani su docker servisi u docker-compose.yml datoteci koji predstavljaju spremnike u Docker Desktop aplikaciji koji će biti pokrenuti prilikom pokretanja same 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;
12        Username=admin;Password=root;Integrated Security=true;Pooling=true;""
13
14   primecaremed.api:
15     image: ${DOCKER_REGISTRY}-primecaremedapi
16     build:
17       context: .
18       dockerfile: src/PrimeCareMed.API/Dockerfile
19     environment:
20       CONNECTION_STRING: "Host=postgres;Port=5432;Database=postgres;
21         Username=admin;Password=root;Integrated Security=true;Pooling=true;""
22
23   postgres:
24     image: postgres:alpine
25     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 volumes:
45   postgres-data:

```

### Ispis 3: docker-compose.yml datoteka

U ispisu 4 prikazana je Dockerfile datoteka u kojoj se stvara PrimeCareMed.Frontend.dll (*DLL - Dynamic-link library*) datoteka 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/"]

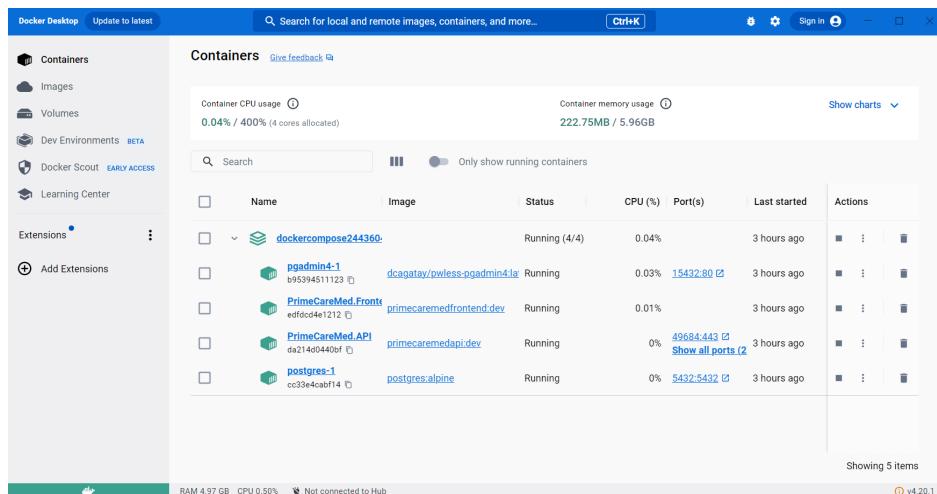
```

```

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:** Dockerfile datoteka u PrimeCareMed.Frontend projektu



**Slika 7:** Pokrenuti spremnici u Docker Desktop aplikaciji

## 2.3. PostgreSQL

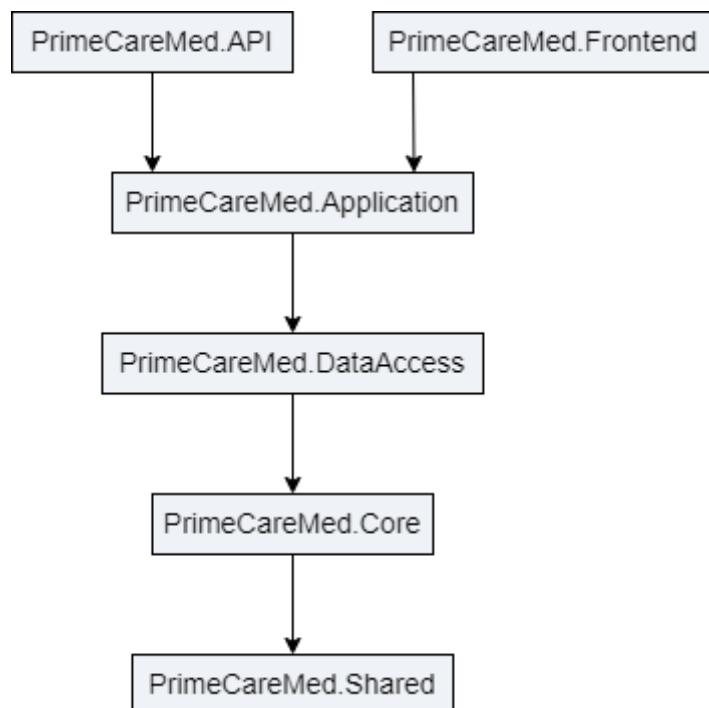
PostgreSQL besplatan je program otvorenog kôda za upravljanje objektno-relacijskim bazama podataka .(engl. *ORDBMS - Object-Relational Database Management System*) koristeći SQL (*Structured Query Language*) jezik. Dozvoljena su proširenja PostgreSQL mogućnosti kao što je dodavanje novih tipova podataka, funkcija, operatora ili agregatnih funkcija. PostgreSQL trenutno je jedna od najnaprednijih baza podataka otvorenog kôda. Prikaz

sheme baze podataka, strukture svih entiteta i druge mogućnosti PostgreSQLa dostupne su u pgAdmin sučelju.

### 3. Izvedba praktičnog rada

#### 3.1. Postavljanje aplikacije

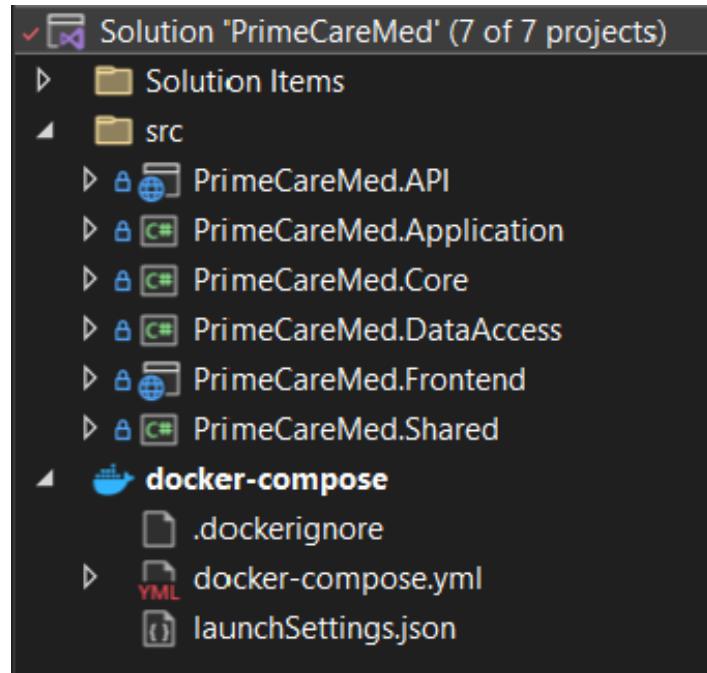
Za postavljanje aplikacije korišten je predložak **N-Tier** arhitekture s GitHub platforme dostupan na poveznici [6]. N-Tier arhitektura je višeslojna klijent-server arhitektura čija je glavna značajka odvajanje aplikacije u fizičke razine i logičke slojeve. U korištenom predlošku, koristi se jedna fizička razina i pet logičkih slojeva kojima je dodan još jedan logički sloj naziva `PrimeCareMed.Frontend` kao ASP.NET Core Web App projekt u kojem je korišten Razor Pages okvir za izradu aplikacija. Pojedini sloj može koristiti usluge nižeg sloja što je omogućeno dodavanjem ovisnosti na svaki sloj. [7] [8] Na slici 8 prikazane su ovisnosti projekata u korištenoj arhitekturi. Naprimjer, sloj `PrimeCareMed.API` može koristiti usluge sloja `PrimeCareMed.Application`.



Slika 8: Ovisnosti slojeva

#### 3.2. Struktura **PrimeCareMed** aplikacije

Na slici 9 prikazana je struktura aplikacije.



Slika 9: Struktura *PrimeCareMed* aplikacije

Struktura `src` direktorija 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 *JSON (JSON - JavaScript Object Notation)* datoteke s podatcima 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 *enumerated* tipove podataka
- `PrimeCareMed.DataAccess` - korišten je za komunikaciju s bazom podataka. Sadrži `DbContext` klasu u kojoj su definirani entiteti i veze među entitetima te stvorenu migraciju. Također, sadrži direktorij `Repositories` u kojem se nalaze repozitoriji koji manipuliraju podatcima u bazi podataka.
- `PrimeCareMed.Frontend` - sadrži `wwwroot` direktorij unutar kojeg se nalaze *CSS (CSS - Cascading Style Sheets)* i *JavaScript* datoteke te poddirektorij `images` za skladištenje slika korištenih na korisničkom sučelju. Sadrži i poddirektorije za pojedine entitete u kojima se nalaze *Razor* stranice sa pripadajućim *code-behind* datotekama.

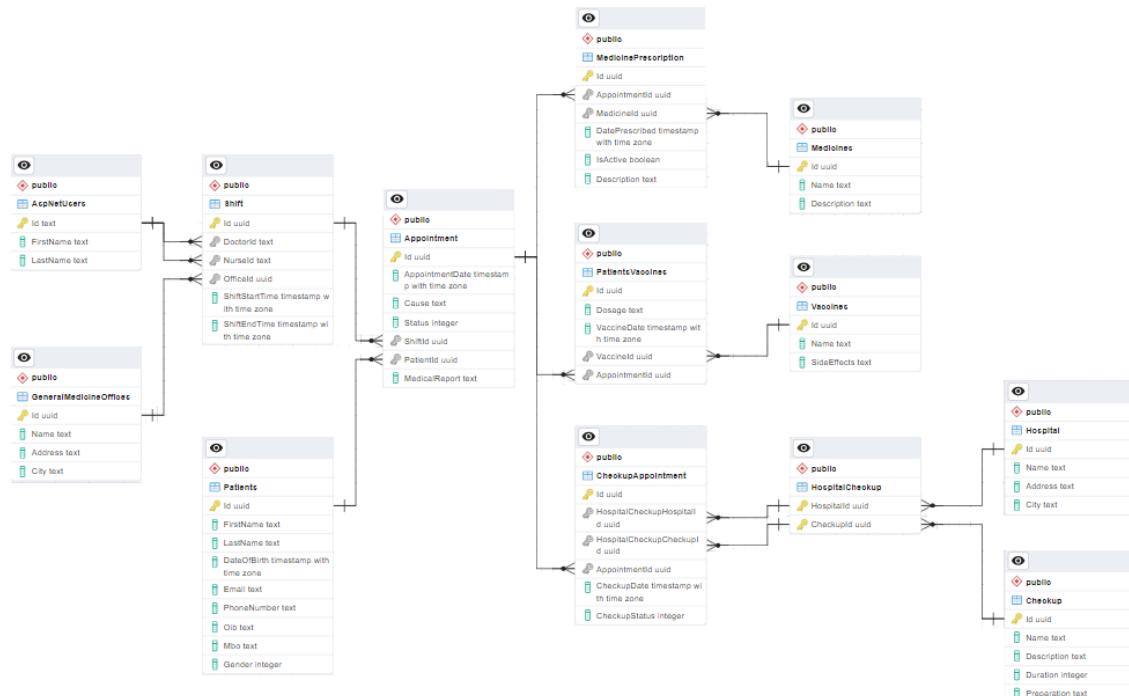
kama za usmjeravanje.

- PrimeCareMed.Shared - sadrži zajedničke datoteke ostalih slojeva

### 3.3. Baza podataka

#### 3.3.1. Struktura relacijske baze podataka

Na slici 10 prikazan je ER (engl. *Entity Relationship*) dijagram relacijske baze podataka u pgAdmin sučelju. Sa slike su izostavljene tablice dobivene korištenjem IdentityUser klase prikazane na slici 2.



Slika 10: Dijagram relacijske baze podataka

#### 3.3.2. Entiteti

Entiteti su definirani u PrimeCareMed.Core projektu. 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 kojeg je njegov 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; }
16         } = new List<PatientsVaccine>();
17         public ICollection<MedicinePrescription>? MedicinePrescriptions { get;
18             set; } = new List<MedicinePrescription>();
19         public ICollection<CheckupAppointment>? CheckupAppointments { get;
20             set; } = new List<CheckupAppointment>();
21 #nullable disable
22     }
23 }
```

### Ispis 5: Entitet Appointment

Relacija jedan-na-više 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<
2 Appointment>();
```

### Ispis 6: Kolekcija Appointment objekata u klasi Patient

### 3.3.3. Interakcija s bazom podataka

Interakcija aplikacije s bazom podataka započinje od `DbContext` instance koja je korištena za dohvaćanje podataka pomoću upita te spremanja novih i ažuriranja postojećih instanci entiteta. Nakon definiranih entiteta, u `DatabaseContext.cs` datoteci definirane su kolekcije svih entiteta koje su kasnije korištene za pristup podacima u bazi podataka. `EntityFrameworkCore` pomoću `ModelBuilder` klase automatski stvara jednostavne veze između entiteta. U `OnModelCreating` metodi korištena je `ModelBuilder` klasa odgovorna za stvaranje samih modela. Koristi definirane `DbSet` kolekcije entiteta 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
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<Shift>()
28     .HasOne(e => e.Nurse)
29     .WithMany(e => e.NursesShifts)
30     .IsRequired();
31
32     builder.Entity<Shift>()
33     .HasOne(e => e.Doctor)
34     .WithMany(e => e.DoctorsShifts)
35     .IsRequired();
36
37     builder.Entity<GeneralMedicineOffice>()
38     .HasMany(e => e.Shifts)
39     .WithOne(e => e.Office)
40     .IsRequired();
41
42     builder.Entity<Shift>()
43     .HasMany(e => e.Appointments)
44     .WithOne(e => e.Shift)
45     .IsRequired();
46
47     builder.Entity<HospitalCheckup>().HasKey(gu => new
48     {gu.HospitalId, gu.CheckupId });
49
50     builder.Entity<HospitalCheckup>().HasOne(ub => ub.Hospital)
51     .WithMany(x => x.HospitalCheckups).HasForeignKey(h => h.HospitalId);
52
53     builder.Entity<HospitalCheckup>().HasOne(ub => ub.Checkup)
54     .WithMany(x => x.HospitalCheckups).HasForeignKey(h => h.CheckupId);
55 }
```

### Ispis 7: DatabaseContext.cs datoteka

#### 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 PrimeCareMed.DataAccess projektu u Persistence/Migrations direktoriju. Obzirom 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 podatcima

Ovisno o entitetu, punjenje baze podataka podatcima (engl. *seeding*) obavljeno je JSON datotekama ili Bogus generatorom za lažne podatke specificiran za .NET jezike. Elementi entiteta ApplicationUser i Patient zapisani su pomoću Bogus generatora u DatabaseContextSeed klasi koji je instaliran kao NuGet paket. U ispisu 10 vidljiva je metoda za punjenje podataka za entitet Patient koja prima broj elemenata entiteta koji će biti stvorenici. [9]

```
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.  
ToUniversalTime())  
7             .RuleFor(p=>p.Email, f=>f.Person.Email)
```

```

8     .RuleFor(p=>p.PhoneNumber, f=>f.Person.Phone)
9     .RuleFor(p=>p.Oib, f=>string.Join("", f.Random.Digits(11)))
10    .RuleFor(p=>p.Mbo, f => string.Join("", f.Random.Digits(9)))
11    .RuleFor(p=>p.Gender, f=>f.PickRandom<Gender>());
12
13    return patientFaker.Generate(count);
14}

```

**Ispis 10:** PatientInit metoda za punjenje lažnih podataka

Metoda za spremanje generiranih podataka u bazu podataka iz metode PatientSeeder 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:** AddRangeAsync metoda za spremanje generiranih podataka

Punjenje podatcima pomoću JSON datoteka korišteno je za entitete Medicine, Vaccine i GeneralMedicineOffice zbog njihovih specifičnih atributa kao što su imena lijekova ili cjepliva. U ispisu 12 prikazan je dio JSON datoteke 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 Bogus.NuGet paketa za koji je potrebna plaćena licenca.

```

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

```

**Ispis 12:** Dio JSON datoteke koji će predstavljati jedan element entiteta Medicine u bazi podataka

```
1     if (!context.Medicines.Any())
2     {
3         var medicinesJson = File.ReadAllText(path + Path.
4             DirectorySeparatorChar + "medicines.json");
5         var medicines = JsonConvert.DeserializeObject<List<Medicine>>(medicinesJson);
6         await context.Medicines.AddRangeAsync(medicines);
7         await context.SaveChangesAsync();
8     }
```

**Ispis 13:** Punjenje tablice Medicine podatcima iz JSON datoteke

### 3.4. Repozitoriji

Sloj aplikacije najbliži bazi podataka je repozitorij. Repozitoriji u aplikaciji smješteni su u PrimeCareMed.DataAccess projektu, a njihova uloga je dohvaćanje podataka iz baze podataka pomoću LINQ upita te spremanje, ažuriranje i brisanje podataka iz iste. Asinkrone metode prepoznate su po ključnim riječima await i async. One omogućavaju 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 izvrše. Asinkrone metode u svom nazivu sadrže riječ Async da bi bile prepoznate. [10]

```
1 namespace PrimeCareMed.DataAccess.Repositories.Impl
2 {
3     public class CheckupAppointmentRepository :
4         ICheckupAppointmentRepository
5     {
6         private readonly DatabaseContext _context;
7         public CheckupAppointmentRepository(DatabaseContext context)
```

```

7         {
8             _context = context ?? throw new ArgumentNullException(nameof(
9             context));
10            }
11           public async Task<CheckupAppointment> AddAsync(CheckupAppointment
12             checkupAppointment)
13           {
14               await _context.CheckupAppointment.AddAsync(checkupAppointment
15             );
16               await _context.SaveChangesAsync();
17               return checkupAppointment;
18           }
19           public async Task<IEnumerable<CheckupAppointment>>
20             GetAllCheckupAppointmentsForPatientAsync(Guid PatientId)
21           {
22               return await _context.CheckupAppointment.OrderByDescending(r
23             => r.CheckupDate).Include(r => r.HospitalCheckup).ThenInclude(r => r.
24             Hospital).Include(r => r.HospitalCheckup).ThenInclude(r => r.Checkup).
25             Include(r=>r.Appointment).ThenInclude(r=>r.Patient).Where(r => r.
26             Appointment.Patient.Id == PatientId).ToListAsync();
27           }
28           public async Task DeleteCheckupAppointmentAsync(Guid id)
29           {
30               var deleteItem = _context.CheckupAppointment.FirstOrDefault(r
31             => r.Id == id);
32               _context.CheckupAppointment.Remove(deleteItem);
33               await _context.SaveChangesAsync();
34           }
35           public async Task<CheckupAppointment>
36             GetCheckupAppointmentByIdAsync(string id)
37           {
38               return await _context.CheckupAppointment.FirstOrDefaultAsync(
39                 t => t.Id.ToString() == id);
40           }
41       }

```

#### Ispis 14: CheckupAppointment repozitorij

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

- `OrderByDescending` - silazno sortira skup podataka na temelju zadanoj podatka.
- `Include` - koristi se za učitavanje podataka povezanih entiteta, u konkretnom slučaju za pojedini `CheckupAppointment` element bit će dobiveni i podatci `HospitalCheckup` entiteta.
- `ThenInclude` - učitava podatke povezanih entiteta s entitetom dobivenim koristeći `Include` metodu. 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 `CheckupAppointment` entiteta kojima je `Appointment.Patient.Id` jednak primljenom argumentu funkcije `PatientId`.
- `FirstOrDefault` - dohvaca prvu vrijednost elemenata koji zadovoljavaju uvjet ili vraća zadatu vrijednost ako se ne pronađe nijedan element, najčešće je to *null* vrijednost

### 3.5. Servisi

Servisi u aplikaciji predstavljaju sloj zadužen za oblikovanje i obradu podataka te omogućavanje istih za prikaz na korisničkom sučelju. Servisi primaju podatke iz repozitorija ili iz Razor stranica, upotrebljavajući AutoMapper *namespace* oblikuju podatke po unaprijed definiranom modelu te ih proslijeđuju.

```
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
createPatientModel)
20     {
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: PatientService servis

U ispisu 15 prikazan je dio PatientService servisa s metodama koje koriste modele i IMapper sučelje. 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: PatientModelForCreate model

```
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: PatientModel model

## 3.6. RazorPages izvedba

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

### 3.6.1. .cshtml datoteke

Datoteke s .cshtml ekstenzijom nazivaju se i Razor predlošcima korištenima za izradu dinamičkih web stranica kombinacijom C# programskog jezika i HTML-a(HTML -

*HyperText Markup Language*). Mogu sadržavati i razne kontrolne strukture, petlje i uvjete. U ispisu 18 prikazana je CreateCheckup.cshtml datoteka koja je korištena pri kreaciji novog pregleda. Za unos se koristi `asp-for` atribut koji povezuje HTML elemnata s odgovarajućim svojstvima modela, u ovom slučaju `CheckupModelForCreate` modela.

```
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-
            danger"></div>
15                    <div class="form-floating">
16                        <input asp-for="NewCheckup.Name" class="form-control" 
            aria-required="true" />
17                        <label asp-for="NewCheckup.Name"></label>
18                        <span asp-validation-for="NewCheckup.Name" class="text-
            danger"></span>
19                    </div>
20                    <div class="form-floating mt-2">
21                        <b style="color: #002133">
22                            Description <br />
23                        </b>
24                        <textarea rows="8" cols="35" id="Description" name="
            Description"></textarea>
25                    </div>
26                    <div class="form-floating mt-2">
27                        <input asp-for="NewCheckup.Duration" class="form-
            control" aria-required="true" />
```

```

28             <label asp-for="NewCheckup.Duration"></label>
29             <span asp-validation-for="NewCheckup.Duration" class=
30                 "text-danger"></span>
31         </div>
32         <div class="form-floating mt-2">
33             <b style="color: #002133">
34                 Preparation <br />
35             </b>
36             <textarea rows="8" cols="35" id="Preparation" name="
37                 Preparation"></textarea>
38         </div>
39         <button id="createCheckupSubmit" type="submit" class="w
40                 -100 btn btn-lg text-white" style="background-color: #006622">Confirm
41     </button>
42     </form>
43 </div>
44 </div>
45 </div>

```

**Ispis 18:** CreateCheckup.cshtml datoteka

### 3.6.2. .cshtml.cs datoteke

Svaka .cshtml datoteka ima svoju .cshtml.cs datoteku poznatiju i kao *Code-Behind* koja sadrži C# kôd koji podržava logiku i funkcionalnosti povezane s odgovarajućim Razor predloškom. Prilikom poziva stranice, .cshtml.cs datoteka prikazuje samu stranicu ili izvršava OnGet metodu. Obzirom da je navedena stranica korištena isključivo za unos podataka, dovoljno je navesti model ili atribute entiteta ispod [BindProperty] atributa s kojim će se unos sa stranice povezati. Korištenje [BindProperty] atributa olakšava manipulaciju podacima između korisničkog sučelja i serverske 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 19: CreateCheckup.cshtml.cs datoteka

.cshtml datoteka sadrži formu za unos s post metodom koja se izvršava u OnPostAsync metodi. Metoda prima parametre Description i Preparation pomoću name HTML atributa. Spremaju se odvojeno u model, dok su atributi modela Name i Duration po-

vezani pomoću navedenog `asp-for` atributa. Poziva se asinkrona metoda `AddAsync` iz servisa koja dalje model mapira u pripadajući entitet `Checkup` te ga šalje u rezervorij koji ga zatim sprema u bazu podataka. `OnPostAsync` metoda ima povratnu vrijednost `IActionResult` koja omogućava povratak na željenu stranicu. U slučaju uspješnog spremanja elementa entiteta, `IActionResult` preusmjerava korisnika na stranicu na kojoj su prikazani svi elementi `Checkup` entiteta to jest na stranicu `ViewAllCheckups`.

### 3.7. Autorizacija

U ispisu 19 prikazan je primjer korištenja `[Authorize]` atributa s parametrima koji dopušta izvršavanje `OnGet` metode ili prikaz stranice autoriziranim korisnicima s ulogom `Administrator` ili `SysAdministrator`. Korištenje `[Authorize]` atributa bez parametara označavalo bi autentikaciju i dopustilo izvršavanje samo autenticiranim korisnicima aplikacije.

### 3.8. Funkcionalnosti aplikacije



**Slika 11:** Login forma

Na slici 11 prikazana je forma za prijavu korisnika dobivena pomoću Scaffold Identitya. Nakon uspješnog unosa korisničkog imena i lozinke, korisnik se autenticira u sustav ili mu se odbija pristup u slučaju krivo unesenih podataka. Uspješnom prijavom autenticirani korisnik ima pristup resursima aplikacije za koje njegova korisnička uloga ima postavljena autorizacijska prava. Korisnici s ulogom Administrator i SysAdministrator odmah nakon uspješne autentikacije preusmjeravaju se na glavni aplikacijski izbornik. Autenticirani korisnici s ulogom Doctor i 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.

# New shift

General Medicine Office  
Select office

Nurse  
Select nurse

**Confirm**

**Logout**

**Slika 12:** Forma za odabir smjene

Nakon prijave u sustav, korisnici su ulogom Doctor ili Nurse preusmjereni su na stranicu za odabir smjene. Ako je za trenutnog korisnika već odabrana smjena, bit će preusmjerjen na glavni izbornik. Nakon odabrane smjene,



**Slika 13:** Početna stranica aplikacije

Na slici 13 prikazana je početna stranica aplikacije s izbornikom s lijeve strane za korisnika s ulogom SysAdministrator. Broj izbora na spomenutom izborniku ovisi o ulozi korisnika.

UserName	FirstName	LastName	Email	PhoneNumber	UserRole
admin@admin.com	admin	admin	admin@admin.com		Administrator
sysadmin@admin.com	sys	admin	sysadmin@admin.com		SysAdministrator
Samantha_Cartwright23@hotmail.com	Samantha	Cartwright	Samantha_Cartwright23@hotmail.com		Doctor
Cynthia85@hotmail.com	Cynthia	Hegmann	Cynthia85@hotmail.com		Doctor
Edwin_Howe@yahoo.com	Edwin	Howe	Edwin_Howe@yahoo.com		Doctor
Marlene.Kutch27@gmail.com	Marlene	Kutch	Marlene.Kutch27@gmail.com		Doctor
Marie15@hotmail.com	Marie	McLaughlin	Marie15@hotmail.com		Doctor
Caleb_Sanford@yahoo.com	Caleb	Sanford	Caleb_Sanford@yahoo.com		Nurse

**Slika 14:** 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 15:** Unos novog pacijenta

Slika 15 prikazuje formu za unos novog pacijenta u sustav primarne zdravstvene zaštite. Prilikom kreiranja novog pacijenta u sustav, unose se svi potrebni podatci. Najvažniji podatci su OIB i MBO koji jedinstveno identificiraju pacijenta. OIB i MBO su tekstualna 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 polja za unos su obavezna osim odabira određenog doktora.

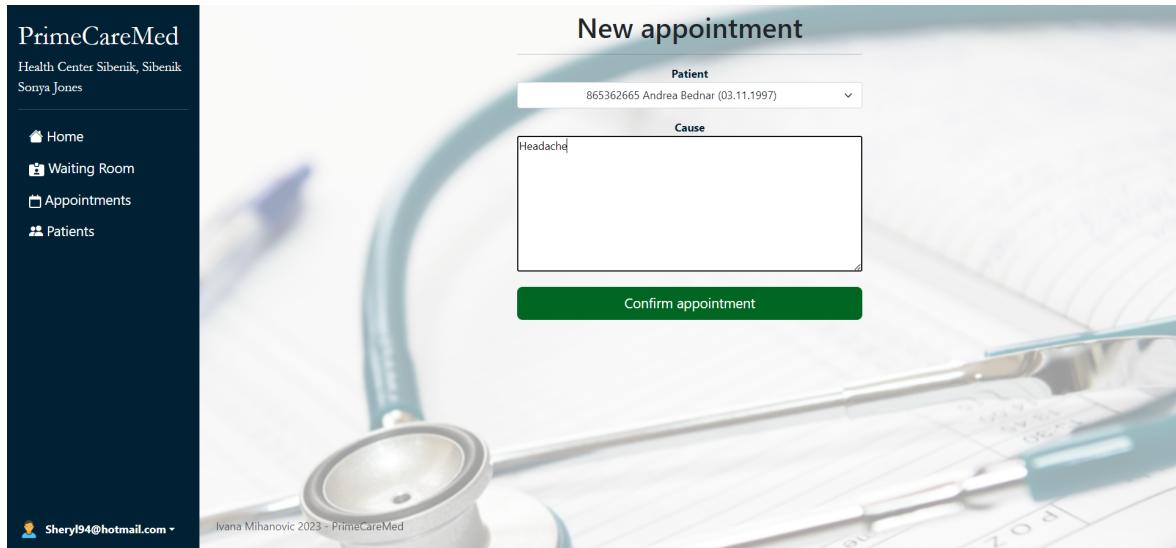
The screenshot shows the PrimeCareMed application's interface. On the left is a dark sidebar with navigation links: Home, Users, Offices, Appointments, Patients, Hospitals, Checkups, Vaccines, and Medicines. At the bottom of the sidebar is an email address: sysadmin@admin.com. The main content area has a title 'New checkup'. It contains a 'Name' field with 'Abdominal Ultrasound', a 'Description' field with a detailed text about abdominal ultrasound, a 'Duration' field set to '30', and a 'Preparation' field with instructions for patients. A large stethoscope image is visible in the background of the form.

**Slika 16:** Forma za unos novog Checkup entiteta

Na slici 16 prikazana je forma za unos. Za navedenu stranicu prikazani su ispisi 3.6.1 za .cshtml i ispis 3.6.2 za .cshtml.cs datoteke. Nakon spremanja, korisnik je preusmjeren na stranicu prikazanu na slici 17.

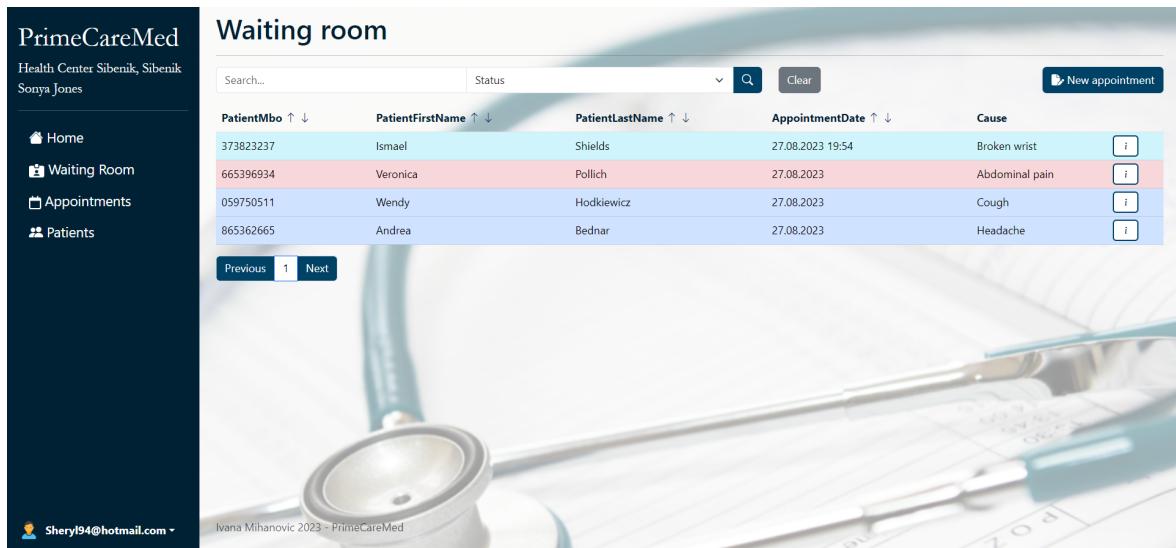
The screenshot shows the 'Checkups' list page. The sidebar on the left is identical to Screenshot 16. The main content area has a title 'Checkups'. It lists a single item: 'Abdominal Ultrasound' with its description, duration (30 minutes), and preparation instructions. Navigation buttons for 'Previous', '1', and 'Next' are at the bottom. A large stethoscope image is visible in the background.

**Slika 17:** Prikaz svih elemenata entiteta Checkup



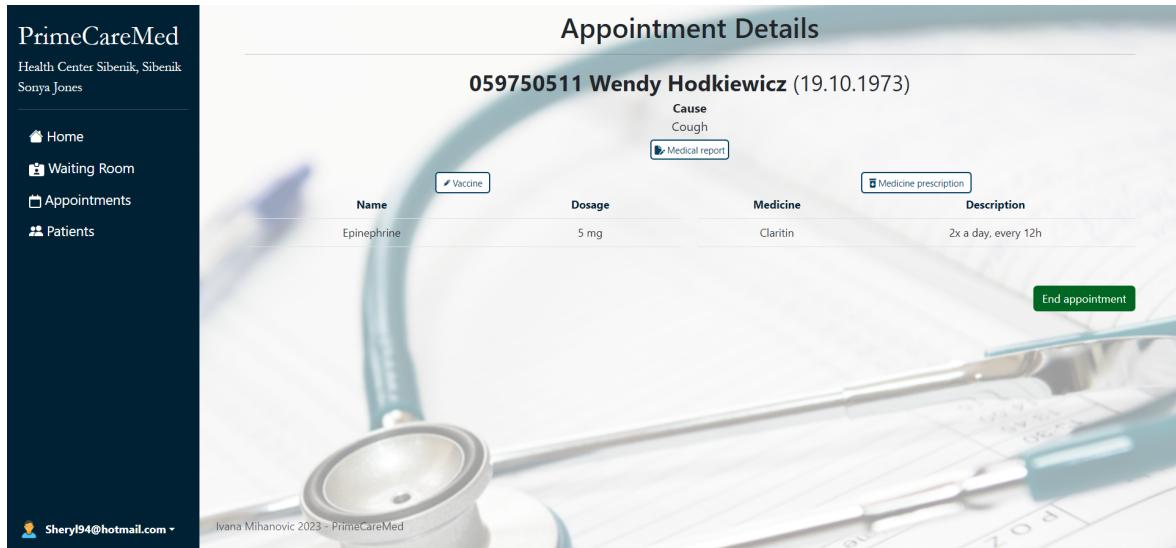
**Slika 18:** Unos novog pregleda u čekaonicu

Na slici 18 prikazan je unos novog pregleda za koji se odabire pacijent koristeći select2 i razlog dolaska. Nakon unosa, korisnik je preusmjeren na stranicu koja predstavlja čekaonicu WaitingRoom ili na prikaz svih pregleda ovisno o ulozi.



**Slika 19:** Čekaonica

Slika 19 prikazuje čekaonicu za korisnike s ulogama Doctor i Nurse. Pojedini status pregleda, označen je bojom.



Slika 20: Detalji pregleda

Pritiskom na botun za detalje pregleda, otvara se stranica prikazana na slici 20 na kojoj je omogućen unos novih recepata za lijek, cjepiva i izvješća. Pritiskom na botun End appointment, status pregleda mijenja se na Done i tada su izmjene dostupne samo korisniku s ulogom SysAdministrator.

## 4. Zaključak

Rezultat ovog završnog rada je web aplikacija za upravljanje podatcima 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 cjepiva, 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 cjepiva, 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. ASP.NET Core okvir 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.

Dodavanjem NuGet paketa ASP.NET Core Identity omogućeno je korištenje kôda za autorizaciju te IdentityScaffolder alata pomoću kojeg su generirane gotove Razor stranice za funkcionalnosti korisničkog računa čime je uvelike olakšana izrada same aplikacije.

Obzirom da primarna zdravstvena zaštita obuhvaća velik spektar raznolikih podataka o pacijentima, *PrimeCareMed* web aplikacija ima potencijal za daljnje unapređenje i proširenje. Zdravstvo je širok pojam pa postoje mnoge funkcionalnosti koje mogu biti dodane da se osigura maksimalna iskoristivost sustava. Samu aplikaciju korisno bi bilo nadograditi statističkim podatcima 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 - NuGet introduction,” <https://learn.microsoft.com/hr-hr/nuget/what-is-nuget>, posjećeno 14.8.2023.
- [2] <https://github.com/dotnet/aspnetcore>, posjećeno 14.8.2023.
- [3] 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.
- [4] ——, “Microsoft Documentation - Container Tools For Docker,” <https://learn.microsoft.com/en-us/visualstudio/containers/overview>, posjećeno 12.6.2023.
- [5] ——, “Microsoft Documentation - Multi-container app with Docker Compose,” <https://learn.microsoft.com/en-us/visualstudio/containers/tutorial-multicontainer>, posjećeno 12.6.2023.
- [6] Grigoras Alexandru, “N-Tier-Architecture,” <https://github.com/nuyonu/N-Tier-Architecture>, posjećeno 28.3.2023.
- [7] Microsoft Corporation, “Microsoft Documentation - N-Tier Architecture,” <https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/n-tier>, posjećeno 5.4.2023.
- [8] Tarnum Java SRL, “Baeldung - N-Tier Architecture,” <https://www.baeldung.com/cs/n-tier-architecture>, posjećeno 5.4.2023.
- [9] Brian Chavez, “GitHub Repository - Bogus,” <https://github.com/bchavez/Bogus>, posjećeno 12.6.2023.
- [10] Microsoft Corporation, “Microsoft Documentation - LINQ,” <https://learn.microsoft.com/en-us/dotnet/csharp/linq/>, posjećeno 12.6.2023.

# Dodatci

## Popis slika

1	NuGet Package Manager i Package Manager Console . . . . .	4
2	Baza podataka nakon naslijeda <code>IdentityUser</code> klase . . . . .	6
3	Identity Scaffolded Item . . . . .	7
4	Uključivanje Docker podrške prilikom kreiranja projekta . . . . .	8
5	Uključivanje Docker podrške na postojeći projekt . . . . .	8
6	<code>docker-compose</code> projekt . . . . .	9
7	Pokrenuti spremnici u Docker Desktop aplikaciji . . . . .	11
8	Ovisnosti slojeva . . . . .	13
9	Struktura <i>PrimeCareMed</i> aplikacije . . . . .	14
10	Dijagram relacijske baze podataka . . . . .	15
11	Login forma . . . . .	31
12	Forma za odabir smjene . . . . .	32
13	Početna stranica aplikacije . . . . .	32
14	Prikaz svih korisnika aplikacije . . . . .	33
15	Unos novog pacijenta . . . . .	33
16	Forma za unos novog <code>Checkup</code> entiteta . . . . .	34
17	Prikaz svih elemenata entiteta <code>Checkup</code> . . . . .	34
18	Unos novog pregleda u čekaonicu . . . . .	35
19	Čekaonica . . . . .	35
20	Detalji pregleda . . . . .	36

## Popis tablica

## Popis ispisa kôda

1	Naredba za dodavanje NuGet paketa . . . . .	4
2	Naslijedivanje <code>IdentityDbContext</code> klase . . . . .	5
3	<code>docker-compose.yml</code> datoteka . . . . .	9

4	Dockerfile datoteka u PrimeCareMed.Frontend projektu . . . . .	10
5	Entitet Appointment . . . . .	15
6	Kolekcija Appointment objekata u klasi Patient . . . . .	16
7	DatabaseContext.cs datoteka . . . . .	17
8	Naredba za kreiranje migracije . . . . .	19
9	Primjena migracije na bazu podataka . . . . .	19
10	PatientInit metoda za punjenje lažnih podataka . . . . .	19
11	AddRangeAsync metoda za spremanje generiranih podataka . . . . .	20
12	Dio JSON datoteke koji će predstavljati jedan element entiteta Medicine u bazi podataka . . . . .	20
13	Punjene tablice Medicine podatcima iz JSON datoteke . . . . .	21
14	CheckupAppointment repozitorij . . . . .	21
15	PatientService servis . . . . .	23
16	PatientModelForCreate model . . . . .	24
17	PatientModel model . . . . .	26
18	CreateCheckup.cshtml datoteka . . . . .	27
19	CreateCheckup.cshtml.cs datoteka . . . . .	28