Web aplikacija za rezervaciju dijeljenih radnih mjesta

Gotal, Valentina

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEK
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

WEB APLIKACIJA ZA REZERVACIJU DIJELJENIH RADNIH MJESTA

Diplomski rad

Valentina Gotal

Osijek, 2021.

SADRŽAJ

1. UVOD

Godina 2020. je zbog COVID-19 pandemije obilježena velikim promjenama u društvenom i poslovnom aspektu u Hrvatskoj i svijetu. Firme su se našle pod velikim pritiskom kako održati svoje poslovanje bez otpuštanja zaposlenika. Firme u IT području su se zbog svoje fleksibilnosti i načina rada mogle lakše prebaciti posao da se odvija od kuće (engl. *remote*). Rad od kuće je donio mnoge prednosti u vrijeme pandemije, a prije svega je to bila sigurnost. S vremenom se takav način rada pokazao lošijim za pojedine zaposlenike zbog manjka adekvatnog prostora i slabije povezanosti s timom. U Hrvatskoj se analizom rezultata i zadovoljstva zaposlenika radom od kuće pokazalo da svaki peti zaposlenik jedva čeka povratak na posao, a tek manji dio je iskazao želju nastavka rada od kuće nakon pandemije [1]. Najveći postotak ispitanika je izjasnio želju za kombiniranim načinom rada, tj. mogućnosti rada djela vremena od kuće, a drugog djela u uredu. Upravo je to potaknulo povećani razvoj aplikacija koje omogućuju takvu vrstu rada zaposlenika. Posljedica takvog načina rada su dijeljena radna mjesta jer više nema potrebe da svaki zaposlenik ima svoje radno mjesto nego se po potrebi dijele i rezerviraju.

Ovaj rad je podijeljen na pet glavnih poglavlja. U prvom poglavlju će biti opisan zadatak diplomskog rada i trenutno stanje na tržištu s obzirom na problematiku koja se nameće u ovom radu. Drugo poglavlje će objasniti nužnu teorijsku osnovu o korištenim tehnologijama, a njihova implementacija će biti obrađena u trećem poglavlju u kojem će biti opisana funkcionalnost aplikacije s razine administratora i zaposlenika. Testiranje uspješnosti realiziranog rješenja će biti detaljno obrađeno u četvrtom poglavlju i u zadnjem poglavlju će biti opisani sveukupni rezultati i potencijalna buduća poboljšanja.

1.1. Zadatak diplomskog rada

Zadatak diplomskog rada je napraviti web aplikaciju za dijeljena radna mjesta koja bi omogućila zaposlenicima kombiniran način rada i mogućnost rezervacije radnih mjesta ovisno o određenim kriterijima. Pri rezervaciji radnih mjesta vodit će se briga i o njihovoj korištenosti kako bi se postigla maksimalna iskoristivost svakog radnog mjesta. Napravit će se dvije razine pristupa, zaposlenik i administrator.

2. TRENUTNO STANJE NA TRŽIŠTU

Veliki broj firmi je pokazao interes za kupnju ili razvoj vlastitih aplikacija za dijeljena radna mjesta što je rezultiralo velikim i raznolikim brojem dostupnih aplikacija na tržištu. Sve te aplikacije imaju zajedničke karakteristike zbog istog cilja, a to je omogućiti što jednostavnije i fleksibilnije rezerviranje radnih mjesta.

Za usporedbu su izdvojene sljedeće aplikacije: *Spacewell* [2], *Envoy Desks* [3], *Condeco* [4], *Skedda* [5] i *CXApp* [6]. Svaka od tih aplikacija nudi organizaciju radnog prostora. U sklopu toga nude i rezervaciju radnih mjesta s hibridnim načinom rada s naglaskom na maksimalno iskorištavanje radnog prostora i povećanje zadovoljstva zaposlenika. Također, nude mogućnost rezervacije radnih mjesta s mogućnošću odabira radnih mjesta ovisno o značajkama radnog mjesta i susjedstvu u kojem zaposlenik želi raditi, tj. kolega pored kojih bi htio rezervirati radno mjesto. Uz to još omogućuju odabir i rezervaciju prostorija za sastanke. Aplikacija prati sve zakazane aktivnosti zaposlenika i na vrijeme ih obavještava o njima. Velika prednost tih aplikacija je što povezuju sve zaposlenike jer se putem aplikacija mogu pronaći lokacije drugih članova tima. Kako bi se očuvala privatnost članova drugih timova, oni imaju mogućnost odabira žele li dopustiti vidljivost svoje lokacije. Zaposlenik će u slučaju pojave kvara u uredu, poput prestanka rada klima uređaja, prijaviti kvar nadležnoj grupi za održavanje ureda. Najveća prednost za firmu je to što uz evidenciju zaposlenika i njihovo zadovoljstvo, mogu analizirati iskorištenost prostora i radnih mjesta te tako ostvariti uštede u vidu najma i/ili regulacije broja radnih mjesta.

Aplikacije se međusobno razlikuju po nekim dodatnim implementiranim mogućnostima pa tako, *Spacewell* nudi mogućnost trajnog rezerviranja jednog radnog stola zaposlenicima koji će raditi isključivo u firmi. Osim toga, ostalim zaposlenicima nudi i mogućnost pregleda radnih uvjeta (temperatura prostorije, sunčana strana, razina vlage i slično) svake prostorije firme. Aplikacija *Condeco* na svaki stol implementira čitač kartica koji služi za prijavljivanje i odjavljivanje zaposlenika s radnog mjesta putem vlastitih kartica i prema tome se ažurira aplikacija. Uz to, nude dodatne mogućnosti rezervacije poput ormarića, mjesta u kuhinji, parkirnog mjesta i slično. U odnosu na *Condeco*, *Envoy Desks* je izbacio u potpunosti kartice i ključeve te je omogućio zaposlenicima prijavljivanje i otključavanje vrata prostorije putem mobilnih uređaja ovisno o dodijeljenim pravima pristupa. *CXApp* i *Envoy Desks* pružaju zaposlenicima putem mobilne platforme mogućnost prikaza putanje do rezervirane prostorije ili do mjesta sastanka, tako da se prati njihova lokacija i tako ih se usmjerava.

Mnoge firme su prepoznale važnost dobre organizacije zaposlenika i stvaranja kvalitetne radne okoline. U nastavku će biti navedene aplikacije i neke od firmi koje ih koriste:

- *Skedda* – Harvard University, Siemens, Mercedes-Benz, Calvin Klein, Tommy Hilfiger, …
- *Envoy* – Lionsgate, Minnesota Twins, Meggit, Manhattan School of Music, BuzzFeed, Zoom, Pandora, …
- *Spacewell* – Cushman & Wakefield, Garmin, Danone, …
- *CXApp* – Adobe, Amazon Web Services, Avaya, NetApp, Lenovo, Nvidia, Visa, …
- *Condeco* – Comcast, Diageo, Johnson & Johnson, Loreal Paris, Nestle, …

Kada se uzme u obzir samo dio vezan za rezervaciju radnih mjesta i napravi usporedba s web aplikacijom koju je cilj napraviti u ovom diplomskom radu, postoji ključna razlika: aplikacija ovog rada je prvenstveno bazirana na kriteriju korištenosti svakog radnog mjesta tako da se automatski dodjeljuje ono najmanje korišteno radno mjesto. Razlog leži u tome što se aplikacija bazira na maksimalnoj iskorištenost radnih mjesta, dok su sve gore navedene aplikacije bazirane na organizaciji cijelog poslovnog sustava što sa sobom nosi veliki niz dodatnih mogućnosti i stavlja se naglasak na zadovoljstvo zaposlenika, ali ne i na podjednaku iskorištenost radnih mjesta.

3. KORIŠTENE TEHNOLOGIJE

Potrebno je izraditi ASP.NET Core MVC web aplikaciju koja će biti napravljena u Visual Studiju 2019 koji je integriran s Microsoft SQL Serverom, što pruža mogućnost povezivanja web aplikacije s bazom podataka. Baza podataka će biti napravljena u Visual Studiju putem *Code First* pristupa, nakon čega će se putem migracije kreirati baza podataka u SQL Server Management Studiju. Koristit će se Microsoft-ova najnovija .NET 5 platforma koja pruža mogućnost izrade i pokretanja raznih vrsta aplikacija na raznim operacijskim sustavima. Pri izradi aplikacije koristit će se HTML, CSS, C# i LINQ.

3.1. .NET 5

Microsoft zbog internih potreba kreće s razvojem .NET platforme i u veljači 2002. godine javno objavljuje prvu verziju .NET Framework-a 1.0. Od tada je .NET Framework prošao kroz razne promjene i poboljšanja u odnosu na prvu verziju. Trenutna verzija .NET Framework-a je .NET Framework 4.8. pušten u travnju 2019. godine. On je namijenjen isključivo za razvoj Windows OS aplikacija. Kako bi se omogućilo pokretanje .NET platforme i na drugim operacijskim sustavima, kao što su Linux ili MAC OS, razvija se .NET Core koji je višeplatformski (engl. *cross-platform*). Prva verzija .NET Core 1.0. je puštena u lipnju 2016. godine, a zadnja dostupna verzija .NET Core-a je 3.1 [7].

Nakon razvoja .NET Core-a, Microsoft je krenuo s razvojem nove .NET platforme pod nazivom One .NET. Početak razvoja One .NET-a predstavlja ključni trenutak u budućnosti cjelokupnog .NET-a, jer se njime nastoje ujediniti svi alati, vremena izvođenja (engl. *runtimes*) i *framework*-i[1] (engl.) koji su razvijeni u sklopu .NET-a, kao što su na primjer: .NET Framework, .NET Core, Xamarin, Mono, .NET Standard i drugi. Ujedinjavanje se radi na principu da se iz svakog navedenog *framework*-a izdvoje i usvoje najbolji dijelovi kako bi se od njih sastojala platforma koju će svi moći koristiti. .NET 5 predstavlja upravo prvi korak prema ostvarivanju tog cilja, točnije on predstavlja prvu verziju One .NET-a. Prikaz prethodnih i trenutnih verzija .NET platformi moguće je vidjeti na slici 3.1.

[1] Softverski okvir koji pruža softversko okruženje s gotovim komponentama za lakši razvoj aplikacija.

Slika 3.1. Prikaz .NET platformi[2]

Na slici 3.2. prikazana je .NET 5 platforma. Na njoj se jasno vidi kako je cilj Microsoft-a korištenje jedne platforme za razvoj bilo koje vrste aplikacija neovisno o tome je li riječ o desktop, web, cloud, mobilnoj aplikaciji, igrama, *Internet of Things*, strojnom učenju (engl. *machine learning*), umjetnoj inteligenciji (engl. *artificial inteligence*) i mnogim drugima [8]. Također, cilj je i omogućavanje korištenja bilo kojeg alata za kodiranje, kao što su Visual Studio, Visual Studio for MAC, Visual Studio Code ili Command Line Interface.

Slika 3.2. Prikaz strukture .NET 5 platforme[3]

[2] https://www.youtube.com/watch?v=t3zj0c244UE

[3] https://devblogs.microsoft.com/dotnet/introducing-net-5/

3.2. ASP.NET Core MVC

ASP.NET proširuje .NET platformu s alatima i bibliotekama specifičnim za razvoj web aplikacija. ASP.NET Core MVC je Microsoft-ov razvojni *framework* za web aplikacije koji integrira urednost i efektivnost MVC (engl. *Model-View-Controller*) predloška, ideje i tehnike agilnog razvoja i najbolje dijelove .NET platforme. ASP.NET Core MVC je predstavljen 2015. godine i temeljen je na .NET Core-u [9].

Karakteristike ASP.NET Core MVC-a su sljedeće: [10], [11]

- *Open-source* (engl.) – besplatan je i može se izmijeniti po vlastitim potrebama.
- *Cross-platform* (engl.) – može se koristiti na različitim platformama za razvoj i implementaciju. Za razvoj aplikacije za Windows i MAC OS koristi se Visual Studio, a za Linux Visual Studio Code.
- Potpuna kontrola nad HTML-om i HTTP-om
    - Može se kreirati jednostavan ili složeni tip HTML-a oblikovan u CSS-u za prikaz na pregledniku.
    - Potpuna kontrola nad HTTP zahtjevima koji se prosljeđuju između preglednika i poslužitelja.
    - Dostupne biblioteke (engl. *libraries*): jQuery, Angular, React i Bootstrap.
- Nadogradivi *framework* – označava mogućnost lake nadogradnje aplikacije u budućnosti. Ključne značajke koje omogućuju veliku moć nadogradnje su: komponenta pogleda (engl. *view component*), pomagači za oznake (engl. *tag helpers*) i usmjeravanje (engl. *routing*) .
- Jednostavno testiranje i održavanje – zahvaljujući mogućnosti razdvajanja različitih područja aplikacije u neovisne dijelove omogućeno je njihovo lako i neovisno testiranje. Ovime se, također, olakšava i održavanje većih aplikacija.
- Usmjeravanje – uklanja potrebu za ručnim pisanjem URL-a, tako što se usmjeravanje ovisno o odabranoj strukturi automatski obavlja.
- API (engl. *Application Programming Interface*) – ASP.NET Core MVC API-ji omogućavaju potpuno iskorištenje inovacija u programskom jeziku i vremenu izvođenja, odnosno korištenje, npr. lambda izraza, LINQ-a, anonimnih i dinamičkih tipova.

3.2.1. Komponenta pogleda

Aplikacije često imaju sadržaj koji nije povezan s njihovom glavnom svrhom, kao npr. traka za navigaciju. Glavni problem takvog slučaja je prikazivanje sadržaja koji nije dio podataka koje model prosljeđuje pogledu putem akcijske metode (engl. *action method*) kontrolera. Djelomični pogledi (engl. *partial views*) se koriste za stvaranje ponovno iskoristivih dijelova koda potrebnih pogledima bez potrebe za dupliciranjem istog sadržaja na više mjesta u aplikaciji [12]. Korisna su značajka, sadrže samo dijelove HTML-a i Razor-a i podatci s kojima rade su dobiveni iz roditeljskog pogleda. Kod njih dolazi do problema u slučajevima kada je potrebno prikazati podatke koje se ne nalaze u roditeljskom pogledu. Moguće je pristupiti izravno podatcima iz djelomičnog prikaza, ali time se narušava MVC predložak što rezultira ugradnjom logike za dohvaćanje i obradu podataka u datoteku pogleda. Alternativa je proširiti modele pogleda tako da oni uključuju potrebne podatke, ali time je teško izdvojiti funkcionalnosti akcijskih metoda za efektivno testiranje. Kao uspješno rješenje, u ASP.NET Core MVC-u se uključuju komponente pogleda [9].

Komponenta pogleda je C# klasa koja pruža djelomičan pogled s potrebnim podatcima, neovisno o roditeljskom pogledu i akcijskoj metodi koja ju prikazuje. Ne može primati HTTP zahtjeve, a primljeni sadržaj će uvijek biti uključen u roditeljskom pogledu [13].

3.2.2. Pomagači za oznake

Velika promjena koja je omogućila transformaciju standardnih HTML elemenata je uvođenje pomagača za oznake u ASP.NET Core MVC. Oni su komponente poslužitelja (engl. *server-side components*) i karakteristični su po korištenju prefiksa *asp-* prije svog naziva. Potrebno ih je uključiti u projekt kako bi se mogli koristiti. Ako se želi omogućiti njihovo korištenje u cijelom projektu, potrebno je uključiti njihovu biblioteku u datoteku *_ViewImports.cshtml* kao što je prikazano na slici 3.3. U suprotnom se uključuju samo kod onih pogleda gdje se planiraju koristiti.

```
@using AplikacijaDijeljenihRadnihMjesta
@using AplikacijaDijeljenihRadnihMjesta.Models
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

Slika 3.3. Uključivanje biblioteke pomagača za oznake unutar _ViewImports.cshtml

Pomagači za oznake omogućuju web programerima korištenje starih konvencionalnih HTML oznaka (engl. *tags*) u web aplikacijama za dizajniranje prezentacijskog sloja, a da pri tome mogu pisati i poslovnu logiku u C# kodu na strani poslužitelja [14].

Postoje dvije vrste pomagača za oznake:

- Ugrađeni pomagači za oznake (engl. *Built-In Tag Helpers*) – dolaze ugrađeni u *Core framework* i izvode uobičajene zadatke kao što su kreiranje forme, prikaz validacijskih poruka i slično.
- Prilagođeni pomagači za oznake (engl. *Custom Tag Helpers*) - predstavljaju pomagače za oznake koje programer sam piše kako bi izvršio željenu transformaciju na HTML elementu [9].

Pomagači za oznake generiraju vezu (engl. *link*) na temelju predloška usmjeravanja aplikacije što znači da će kreirana veza raditi ispravno i u slučaju promjena unutar predloška usmjeravanja u budućnosti [15]. Ovo predstavlja veliku prednost u odnosu na ručno pisanje putanje veze HTML kodom jer bi se u tom slučaju svaka linija putanje veze morala ručno izmijeniti. Usporedba između pisanja veze putanje s pomagačima za oznake i HTML kodom je prikazana na slici 3.4.

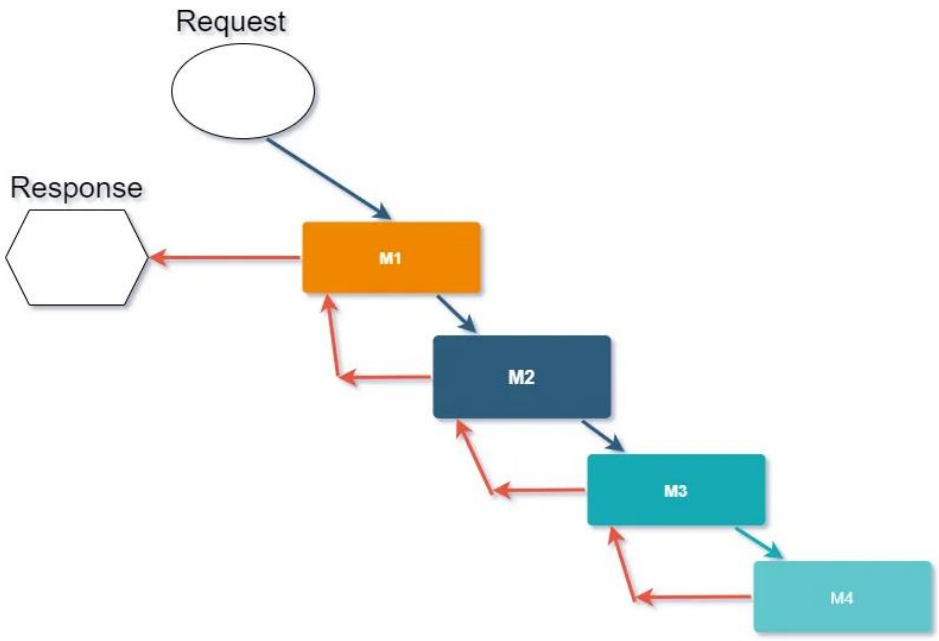
```
Ručno generiran kod
<a href="/Index/PregledZahtjeva">Pregled zahtjeva</a>
Kod pisan pomoću pomagača za oznake
<a asp-controller="PregledZahtjeva" asp-action="Index">Pregled zahtjeva</a>
```

Slika 3.4. Usporedba ručno pisanog HTML koda i pomagača za oznake

Pomagači za oznake imaju slične mogućnosti kao HTML pomagači (engl. *HTML Helpers*). Jedna od važnijih razlika je to što pomagači za oznake web dizajnerima pružaju mogućnost uređivanja pogleda bez potrebe za učenjem C# i Razor sintakse, dok je za korištenje HTML pomagača potrebno to predznanje [16]. Osim toga, Visual Studio pruža bogatu podršku pomagačima za oznake, što nije slučaj za HTML pomagače jer oni kao parametar najčešće primaju jednostavne znakovne nizove (engl. *string*). Važno je naglasiti kako pomagači za oznake nisu razvijeni kako bi zamijenili HTML pomagače i mogu se koristiti istovremeno.

3.2.3. Usmjeravanje

Usmjeravanje je proces koji pregledava primljene zahtjeve, tj. URL-ove i zatim mapira njihov zahtjev na kontroler i njegovu akcijsku metodu. Mapiranje se vrši po definiranim pravilima usmjeravanja za aplikaciju. Usmjeravanje uključuje dodavanje posrednika za usmjeravanje (engl. *routing middleware)* u cjevovod (engl. *pipeline*) za obradu zahtjeva, jer svi razmijenjeni zahtjevi između poslužitelja i aplikacije prolaze kroz cjevovod koji se sastoji od niza posrednika. Svaki posrednik unutar cjevovoda može izmijeniti i odlučiti hoće li neki zahtjev propustiti do idućeg posrednika. Slučaj kada posrednik odluči ne propustiti zahtjev do idućeg posrednika se naziva kratki spoj zahtjeva cjevovoda (engl. *short-circuiting the request pipeline*) [17]. Uobičajena praksa je propuštanje zahtjeva do kraja cjevovoda, tj. do posljednjeg uključenog posrednika, nakon čega se povratno u suprotnom smjeru šalje konačan odgovor poslužitelju. Potrebno je napomenuti važnost redoslijeda uključivanja posrednika u cjevovod jer s obzirom na njihov poredak će primati i po potrebi obrađivati zahtjev. Opisani princip rada moguće je vidjeti na slici 3.5.



Slika 3.5. Princip rada usmjeravanja [4]

U ASP.NET Core MVC aplikaciji postoje dva načina za definiranje usmjeravanja: konvencionalno usmjeravanje (engl. *convention based routing*) i usmjeravanje temeljeno na atributima (engl. *atribute-based routing*). Kod konvencionalnog usmjeravanja, ruta se određuje

[4] https://www.yogihosting.com/aspnet-core-configurations/#csproj

na temelju definiranih konvencija u predlošcima rute koje mapiraju dolazne zahtjeve na odgovarajuće kontrolere i njihove akcijske metode. Općenito, ova vrsta ruta definirana je unutar metode *Configure()* koja se nalazi u klasi *Startup*. Klasa *Startup* detaljno je objašnjena u potpoglavlju 4.1. U slučaju korištenja usmjeravanja temeljenog na atributima, ruta se određuje na temelju atributa, tj. u klasi *Startup* se dodaje metoda, odnosno posrednik, npr. *AddMvc()*, koji ne prima kao parametar predefinirani predložak rute, već se ruta zasebno definira na razini kontrolera ili na razini akcijske metode. U jednoj aplikaciji je moguće koristiti obje vrste usmjeravanja [17]. Na slici 3.6. i 3.7. prikazana je razlika između ovih vrsta usmjeravanja.

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id?}");
});
```

Slika 3.6. Primjer konvencionalnog pristupa usmjeravanja [5]

```
public class HomeController : Controller
{
    [Route("")]
    [Route("Home")]
    [Route("Home/Index")]
    [Route("Home/Index/{id?}")]
    public IActionResult Index(int? id)
    {
        return ControllerContext.MyDisplayRouteInfo(id);
    }

    [Route("Home/About")]
    [Route("Home/About/{id?}")]
    public IActionResult About(int? id)
    {
        return ControllerContext.MyDisplayRouteInfo(id);
    }
}
```

Slika 3.7. Primjer dodavanja atributa od usmjeravanja na temelju atributa [6]

[5] https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/routing?view=aspnetcore-5.0

[6] https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/routing?view=aspnetcore-5.0

3.3. HTML, CSS i Bootstrap

HTML (engl. *HyperText Markup Language*) je često korišten opisni jezik za definiranje izgleda web aplikacije. Uz njega se koristi i stilski jezik CSS (*engl. Cascading Style Sheets*) koji ga nadopunjuje. Kroz godine je razvijen najpopularniji HTML, CSS i JavaScript *framework* Boostrap, koji uvelike olakšava razvoj prilagodljive (engl. *responsive*) web aplikacije koja se može pokrenuti na bilo kojoj vrsti uređaja bez utjecaja na promjene u dizajnu aplikacije. Bootstrap je u suštini velika kolekcija ponovno iskoristivih dijelova koda napisanih u HTML, CSS-u i JavaScript-u koji programerima omogućuje brži razvoj web stranica [18].

3.4. SQL Server Management Studio

SQL Server Management Studio, SSMS je napredno razvojno okruženje koje omogućuje konfiguriranje, upravljanje i administriranje mehanizma baze podataka SQL servera (engl. *SQL server database engine*) [19]. SSMS je vrlo popularan i jedan od najčešće korištenih razvojnih okruženja za upravljanje bilo kojom SQL infrastrukturom. Glavni razlozi njegove široke primjene su pružanje naprednog korisničkog iskustva, raznih dodatnih opcija, jednostavnosti korištenja i instaliranja te dostupnost svima zbog svoje besplatne licence. SSMS se može koristiti za izvršavanje raznih zadataka baze podataka, a neki od njih su: brza izrada ili izmjena baze podataka, dodavanje objekata uključujući tablice, poglede i pohranjene procedure (engl. *stored procedures*), testiranje objekata baze podataka koristeći vanjski alat za testiranje, izvršavanje upita nad bazom podataka, vraćanje baze podataka na prethodno stanje (engl. *restore database*), kreiranje sigurnosne kopije (engl. *backup*) baze podataka i mnoge druge. S druge strane, postoji i niz zadataka poslužitelja koje može obaviti kao što su: povezivanje ili registracija na licu mjesta ili na udaljenim poslužiteljima, praćenje aktivnosti poslužitelja, registriranje poslužitelja za upravljanje administracijom, pružanje alata za pisanje skripti i pokretanje istih za izvršavanje administratorskih zadataka i slično.

3.5. LINQ

LINQ (engl. *Language-Integrated Query*) je skraćenica za jezično integrirani upit koji predstavlja API za upite nad bazom podataka neovisno o samoj strukturi podataka. Prije nego je razvijen LINQ, za svaku je vrstu izvora podataka bilo potrebno naučiti drugačiji jezik upita: SQL baze podataka, ADO tehnologija pristupa podatcima, XML dokumenti, razne web usluge i slično, no s razvojem LINQ-a je riješen taj problem. LINQ omogućuje istu sintaksu upita za različite izvore podataka što je ujedno i njegova najveća prednost.

3.6. MVC

MVC (engl. *Model-View-Controller*) predložak je prvi put predstavljen 1970-tih godina, ali nije bio prihvaćen niti se na njega obraćala prevelika pažnja idućih tridesetak godina. Veliki pomak se dogodio 2004. godine kada je nastao Rails *framework* za razvoj web-a temeljen na Ruby jeziku kojem je velika inspiracija bio MVC predložak. Nakon toga, MVC predložak je postao izrazito popularan i Microsoft ga je odlučio implementirati u svoj .NET *framework* [20].

Glavna ideja MVC predloška je podjela aplikacije na tri potpuno funkcionalna djela. MVC se sastoji od modela, pogleda i kontrolera. U nastavku će biti objašnjen MVC općenito i u sklopu ASP.NET Core MVC-a.

Model je kolekcija objekata koji sadrže podatke s kojima radi aplikacija. Postoji tri vrste modela ovisno o tome kako i gdje su korišteni: [21]

- Model domene (engl. *domain model*) – predstavlja entitet u bazi podataka. Ovaj model se još naziva model entiteta (engl. *entity model*) ili podatkovni model (*engl. data model*) i on sadrži atribute entiteta.
- Model za pogled (engl. *view model*) – sadrži podatke koji se prikazuju na prezentacijskom sloju aplikacije. Za razliku od modela domene, on ne mora sadržavati sve atribute iz entiteta jer se ne žele uvijek u prezentacijskom sloju prikazati svi podatci dohvaćeni iz baze podataka.
- Model za uređivanje (engl. *edit model*) – sličan modelu za pogled, ali za razliku od njega sadrži podatke koji se koriste za izmjenu postojećeg ili dodavanje novog podatka.

Pogled kao dio MVC predloška služi za prikaz sadržaja korisniku. U ASP.NET Core MVC aplikaciji, pogled je dokument koji sadrži HTML elemente i C# kod za obradu i generiranje odgovora. Pogled omogućava razdvajanje podataka za prikaz od logike koja obrađuje zahtjeve. Također, jedan pogled se može koristiti u više kontrolera [9]. Kod pogleda je važno spomenuti mehanizam za pogled (engl. *view engine*) jer je on odgovoran za isporuku HTML odgovora na poziv akcijske metode kontrolera. Akcijske metode vraćaju razne vrste odgovora koji se zajedno nazivaju akcijski rezultati (engl. *action results*). Rezultat pogleda (engl. *view result*) je akcijski rezultat koji pruža HTML odgovor, a kreira ga mehanizam za pogled. Predefinirani mehanizam za pogled u ASP.NET Core MVC-u je *Razor* mehanizam pogleda (engl. *Razor view engine*). Kontroler prosljeđuje podatke pogledu. Pogledi zahvaljujući *Razor*-u imaju sposobnost obrade tih podataka i generiranja odgovora. *Razor* omogućuje uporabu C# koda u

HTML datoteci. Datoteke koje sadrže Razor oznaku „@“ obično imaju „*.cshtml*“ proširenje [22]. Njegovo korištenje olakšava postojanje Visual Studio podrške za *Razor*. Na slici 3.8. prikazan je primjer korištenja Razor sintakse.

```
@if ((string)TempData["Uspješno"] != null)
{
    <div class="alert alert-success">
        <strong>@TempData["Uspješno"]</strong>
    </div>
}
@if ((string)TempData["Neuspješno"] != null)
{
    <div class="alert alert-danger">
        <strong>@TempData["Neuspješno"]</strong>
    </div>
}
 @if ((string)TempData["Info"] != null)
{
    <div class="alert alert-warning">
        <strong>@TempData["Info"]</strong>
    </div>
}
```

Slika 3.8. Primjer korištenja *Razor* sintakse

Kontroleri su temelj MVC projekta jer sadrže glavnu logiku web aplikacije. Sadrže logiku za primanje zahtjeva, ažuriranje stanja aplikacije ili modela i odabir odgovora koji će biti poslan klijentu [9]. U ASP.NET Core MVC-u kontroleri su C# klase čije se javne metode (engl. *public methods*) pozivaju za obradu HTTP zahtjeva. Metode mogu preuzeti odgovornost za davanje odgovora izravno klijentu, ali češći pristup je vraćanje rezultata metodi koja govori MVC-u kako bi odgovor trebao biti pripremljen.

Ukratko, MVC predložak radi na principu da svaka aktivnost na pogledu rezultira akcijskim metodama koje se pozivaju na kontroleru. U ASP.NET Core MVC-u taj princip rada se implementira putem HTTP zahtjeva koji se usmjeravaju na odgovarajući kontroler pomoću podsustava za usmjeravanje zahtjeva (engl.*request routing subsystem*) [13]. Svaki jedinstveni URL mapira se na posebnu metodu u kontroleru. Unutar te akcijske metode, podatci pogleda se obrađuju i ažurira se model. MVC kontroleri imaju i dodatnu odgovornost određivanja koji pogled se treba prikazati.

Osim definiranja slojeva, MVC predložak pruža pravila o dopuštenoj komunikaciji između tih slojeva. Za poštivanje MVC predloška važno je slijediti iduće dopuštene predloške komunikacije (engl. *allowed communication patterns*) [13]:

- Korisnici mogu komunicirati s pogledom.
- Pogledi mogu komunicirati s kontrolerom.
- Kontroleri mogu komunicirati s pogledima.
- Kontroleri mogu komunicirati s drugim kontrolerima.
- Kontroleri mogu komunicirati s modelom.

Ograničeni obrasci komunikacije (engl. *restricted communication patterns*) uključuju sljedeća pravila [13]:

- Korisnici ne smiju izravno komunicirati s kontrolerima.
- Korisnici ne smiju izravno komunicirati s modelom.
- Pogledi ne smiju izravno komunicirati s drugim pogledima.
- Pogledi ne smiju izravno mijenjati model.
- Modeli ne smiju mijenjati druge modele.

Postoje mnoge prednosti MVC predloška, a jedna od njih je to što jedan pogled nije vezan za točno jedan model što znači da se može koristiti više pogleda koji su povezani s jednim modelom, npr. s jednim modelom može se napraviti pogled za svaku CRUD (engl. *Create-Read-Update-Delete*) operaciju zasebno (pogled za stvaranje novih zapisa, čitanje, izmjenu ili brisanje postojećih zapisa). Visual Studio ima ugrađene značajke koje pojednostavljuju stvaranje aplikacije koja slijedi MVC predložak, poznate kao *scaffolding* (engl.) značajke. *Scaffolding* značajke omogućuju generiranje pogleda na temelju klase modela [13]. Druga prednost je razdvajanje pogleda i kontrolera, što omogućuje promjenu reakcije aplikacije na unos korisnika bez promjene pogleda i obrnuto. Također, održavanje logike kontrolera odvojeno od logike pogleda omogućava preuređivanje rasporeda stranica onoliko često koliko to korisnik zahtjeva bez slučajnog narušavanja poslovne logike. Još jedna velika prednost MVC-a je omogućavanje raspodjele posla na članove tima ovisno o njihovim sposobnostima jer vrlo malo ljudi posjeduje sposobnost za kreiranje korisničkog sučelja koristeći HTML i CSS i da pri tome znaju C#. Uz to omogućuje članovima tima istovremeni rad na različitim dijelovima stranice i omogućava korištenje automatiziranih jediničnih (engl. *unit*) testova.

4. IMPLEMENTACIJA PROGRAMSKOG RJEŠENJA

Nakon upoznavanja s potrebnom tehnologijom, cilj je implementirati programsko rješenje koristeći navedene tehnologije. Na početku kreiranja projekta potrebno je postaviti strukturu kako bi se postigla preglednost i praćenje MVC predloška. Zatim je potrebno osmisliti bazu podataka. Za njezino kreiranje korišten je *Code First* pristup u kojem se pišu modeli u Visual Studiju pomoću kojih će se kasnije putem migracije kreirati entiteti u bazi podataka u SQL Server Management Studiju. Programsko rješenje sadrži dvije vrste pristupa: korisnička i administratorska strana i kao takvo će biti objašnjeno u nastavku.

4.1. Struktura aplikacije

Kreiranjem projekta s predloškom ASP.NET Core MVC aplikacije pojavljuju se predefinirani elementi strukture [9], [13], [23], [24]:

- *Connected Services* – namijenjen je za povezivanje projekta s vanjskim uslugama kao što su Azure Storage ili Application Insights.
- *Dependencies* – sadrži sve pakete ili druge projekte o kojima će ovisiti ovaj projekt.
- *Startup.cs* – sadrži logiku za konfiguriranje ASP.NET Core MVC aplikacije i dodavanje komponenti posrednika u ASP.NET Core cjevovod. Općenito, klasa izvršava sve zadatke inicijalizacije (migracija, popunjavanje baze). Sastoji se od dvije glavne metode:
    - *ConfigureServices()* – metoda u kojoj se registriraju ovisne klase[7] pomoću ugrađenog IoC spremnika (engl. *Inversion of Control container*). IoC spremnik je dio predloška ubrizgavanja ovisnosti (engl. *dependency injection pattern*). Nakon registracije ovisne klase, ona se može koristiti bilo gdje u aplikaciji uključivanjem kroz parametar konstruktora željene klase pri čemu će ju IoC spremnik automatski proslijediti.
    - *Configure()* – koristi se za određivanje načina na koji će aplikacija odgovoriti na HTTP zahtjeve. Cjevovod je postavljen dodavanjem komponenta posrednika. Komponente posrednika se dodaju redoslijedom kojim će i primati i po potrebi izmijeniti zahtjev. Na početku metode se pomoću objekta tipa *IWebHostEnviroment* radi provjera vrste korištenog okruženja. Okruženje može biti okarakterizirano s tri metode: *isDevelopment()*, *IsStaging()* i *IsProduction()*.

[7] ASP.NET Core naziva ovisnu klasu uslugom (engl. *services*)

Kako bi se spriječio prekid rada aplikacije, u *else* grani se specificira što bi aplikacija trebala napraviti u slučaju pojave neke greške.

- *launchSettings.json* – sadrži profile pokretanja. Svaki profil definira način pokretanja projekta kada se pritisne gumb *Pokreni*.
- *wwwroot* – sastoji se statičnih mapa: CSS, JS, biblioteke i slika. Njihovo korištenje je omogućeno zbog uključivanja komponente posrednika *UseStaticFiles()* u *Startup.cs*.
- *appsettings.json* – sastoji se od konekcijskih znakovnih nizova i aplikacijskih specifičnih postavki pohranjenih u JSON formatu.
- *Program.cs* – glavna klasa koja se prva pokreće i ona poziva *Startup* klasu za dovršavanje postavki aplikacije.
- *Models* – mapa koja sadržava sve klase koje predstavljaju entitete u bazi podataka.
- *Controllers*- mapa koja sadrži poslovnu logiku implementiranu putem klasa.
- *Views* – mapa koja sadrži poglede svakog kontrolera, *_ViewImports.cshmtl*[8], *_ViewStart.cshtml*[9] i mapa *Shared* koja sadrži sljedeće:
    - *_Layout.cshtml* – predefinirani pogled za aplikaciju
    - *_ValidationScriptsPartial.cshtml* – upiti za validaciju
    - *Error.cshtml* – predefinirani pogled za prikaz pogreške

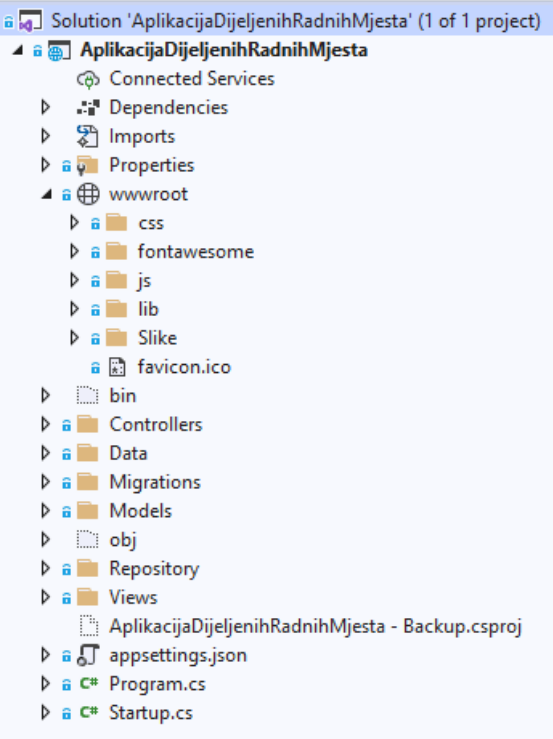
Zbog potreba projekta i same preglednosti rješenja, posebno su kreirane sljedeće mape:

- *Data* – mapa koja sadrži klasu *AppDbContext* koja sadrži popis svih entiteta nad kojima će se raditi CRUD operacije.
- *Migrations* – mapa koja sadrži sve napravljene migracije s bazom podataka i on se automatski generira dodavanjem prve migracije.
- *Repository* – sadrži različite klase od kojih svaka sadrži CRUD operacije nad jednim entitetom i dodatne potrebne metode za rad s bazom.
- *ViewModel* – mapa koja se nalazi u mapi *Models* i sadrži modele za prikaz kako bi se uvijek preko kontrolera prosljeđivali nužni podatci pogledu.

Na slici 4.1. prikazana je struktura projekta.

[8] Služi za uključivanje raznih biblioteka kao npr. pomagači za oznake

[9] Postavljanje vrijednosti koje moraju koristiti svi pogledi

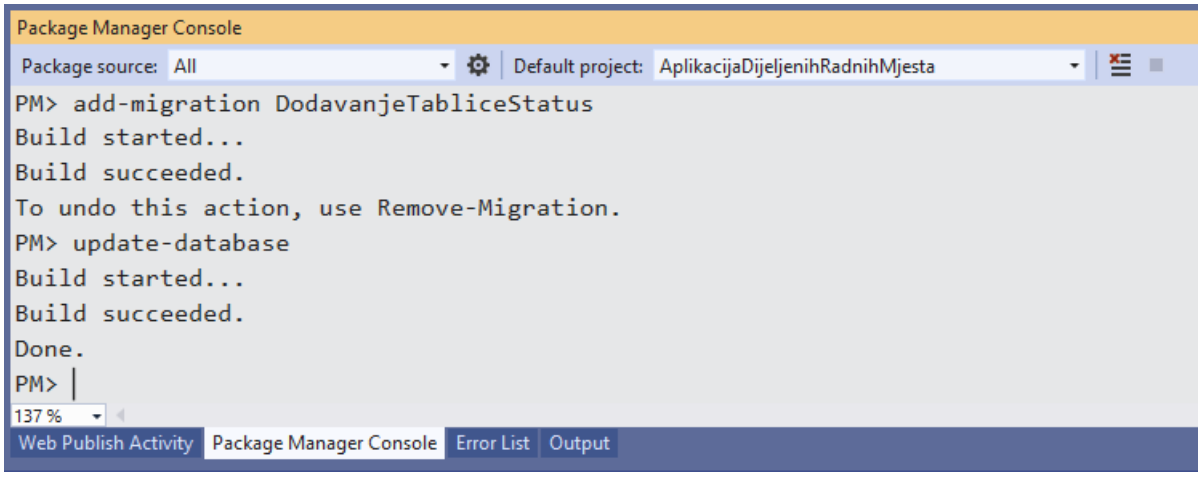


Slika 4.1. Prikaz strukture aplikacije

4.2. Baza podataka – Code First

Code First pristup omogućava definiranje entiteta i njegovih atributa pomoću koda. Svaki entitet je predstavljen modelom koji se sastoji od atributa entiteta i atributa koji predstavljaju relacije s drugim entitetima. Nakon što se kreiraju svi potrebni modeli, izvodi se migracija. Kod ovog pristupa nije potrebno imati postojeću bazu, ona se u tom slučaju kreira pri prvom izvođenju migracije. Putem migracije se svaka promjena u modelima pohranjuje u bazu podataka pomoću dvije linije koda koje su prikazane na slici 4.2.:

- *add-migration nazivMigracije* - služi za kreiranje migracije i prikaz promjena koje će se dogoditi na bazi podataka.
- *update-database* – izvodi i pohranjuje sve potrebne izmjene direktno na bazi podataka
- *remove-migration* – omogućava brisanje napravljenih promjena u prethodnoj migraciji, tj. povratak na prethodnu verziju



Slika 4.2. Prikaz naredbi za izvođenje migracije

Kod kreiranja modela važno je voditi brigu o relacijama između entiteta. Postoje tri vrste relacija i različite konvencije za ostvarivanje istih. U nastavku je na primjeru objašnjena svaka relacija s odabranom konvencijom:

- Jedan naprema više – primjer takve relacije je model *Zahtjev* i *TipZahtjeva*. Jedan zahtjev može imati jedan tip zahtjeva, dok isti tip zahtjeva može imati više zahtjeva. Relacija se postiže definiranjem odnosa na oba kraja. Na slici 4.3. prikazan je model *TipZahtjeva* koji sadrži atribut s kolekcijom zahtjeva, dok je na slici 4.4. prikazan model *Zahtjev* koji sadrži dva atributa, strani ključ i atribut tipa *TipZahtjeva*. Moguće je koristiti kolekciju *IEnumerable* ili *ICollection*. *IEnumerable* je osnovni tip kolekcije koja sadrži samo listu elemenata, dok *ICollection* proširuje njegovu funkcionalnosti omogućavajući dodavanje, brisanje i ažuriranje podataka. Također, omogućava dohvaćanje broja elementa u listi bez prolaska kroz nju [25].

```
namespace AplikacijaDijeljenihRadnihMjesta.Models
{

    public class TipZahtjeva
    {

        public int Id{get; set;}


        public string Tip{get; set;}

        0 references
        public IEnumerable<Zahtjev> Zahtjevi{get; set;}
    }
}
```

Slika 4.3. Odnos jedan naprema više u modelu *TipZahtjeva*

```
namespace AplikacijaDijeljenihRadnihMjesta.Models
{
    4 references
    public class Zahtjev
    {
        15 references
        public int Id{get; set;}

        12 references
        public DateTime Datum{get; set;}

        22 references
        public int TipZahtjevaId{get; set;}

        6 references
        public TipZahtjeva TipZahtjeva{get; set;}

        21 references
        public int DjelatnikId{get; set;}

        0 references
        public Djelatnik Djelatnik{get; set;}

        0 references
        public IEnumerable<RezervacijaOtkazivanje> RezervacijaOtkazivanje{get; set;}
    }
}
```

Slika 4.4. Odnos jedan naprema više u modelu *Zahtjev*

- Jedan naprema jedan – u projektu nema relacija ovog tipa. Relacija se postiže dodavanjem atributa tipa jednog modela u drugi model i obrnuto. U jedan od modela se dodaje i strani ključ drugog modela.
- Više naprema više – primjer takve relacije je model *OrganizacijskaJedinica* i *Lokacija*. Jedna organizacijska jedinica se može nalaziti na više različitih lokacija, a na jednoj lokaciji može biti više organizacijskih jedinica. Relacija se ostvaruje dodavanjem kolekcija jednog modela u drugi i obrnuto (Slika 4.5 i 4.6). Kod ove relacije kreira se međutablica koja se sastoji od dva strana ključa koji zajedno čine primarni ključ.

```
namespace AplikacijaDijeljenihRadnihMjesta.Models
{
    8 references
    public class OrganizacijskaJedinica
    {
        27 references
        public int Id{get; set;}
        18 references
        public string Naziv{get; set;}
        0 references
        public IEnumerable<Lokacija> Lokacije{get; set;}
        0 references
        public IEnumerable<Djelatnik> Djelatnici{get; set;}
        1 reference
        public IEnumerable<LokacijaOrganizacijskaJedinica> LokacijaOrganizacijskaJedinica{get; set;}
    }
}
```

Slika 4.5. Odnos više naprema više u modelu *OrganizacijskaJedinica*

```
namespace AplikacijaDijeljenihRadnihMjesta.Models
{
    9 references
    public class Lokacija
    {
        50 references
        public int Id{get; set;}
        29 references
        public string Adresa{get; set;}
        0 references
        public IEnumerable<OrganizacijskaJedinica> OrganizacijskeJedinice{get; set;}
        1 reference
        public IEnumerable<LokacijaOrganizacijskaJedinica> LokacijaOrganizacijskaJedinica {get; set;}
        8 references
        public Grad Grad{get; set;}
        25 references
        public int GradId{get; set;}
        0 references
        public IEnumerable<RadnoMjesto> RadnaMjesta{get; set;}
    }
}
```

Slika 4.6. Odnos više naprema više u modelu *Lokacija*

Kako bi se ostvarilo povezivanje Visual Studija i SQL Server Management Studija potrebno je definirati znakovni niz za povezivanje u datoteci *appsetting.json*. Potrebno je odrediti server i naziv baze podataka koja će se kreirati i/ili ažurirati pomoću migracije (Slika 4.7.).

```
"ConnectionStrings": {
  "DefaultConnection": "Data Source=DESKTOP-A3HFSQB\\VGSQLEXPRESS;Initial Catalog=DBDijeljenaRadnaMjesta;
},
```

Slika 4.7. Prikaz znakovnog niza za povezivanje u *appsetting.json* datoteci

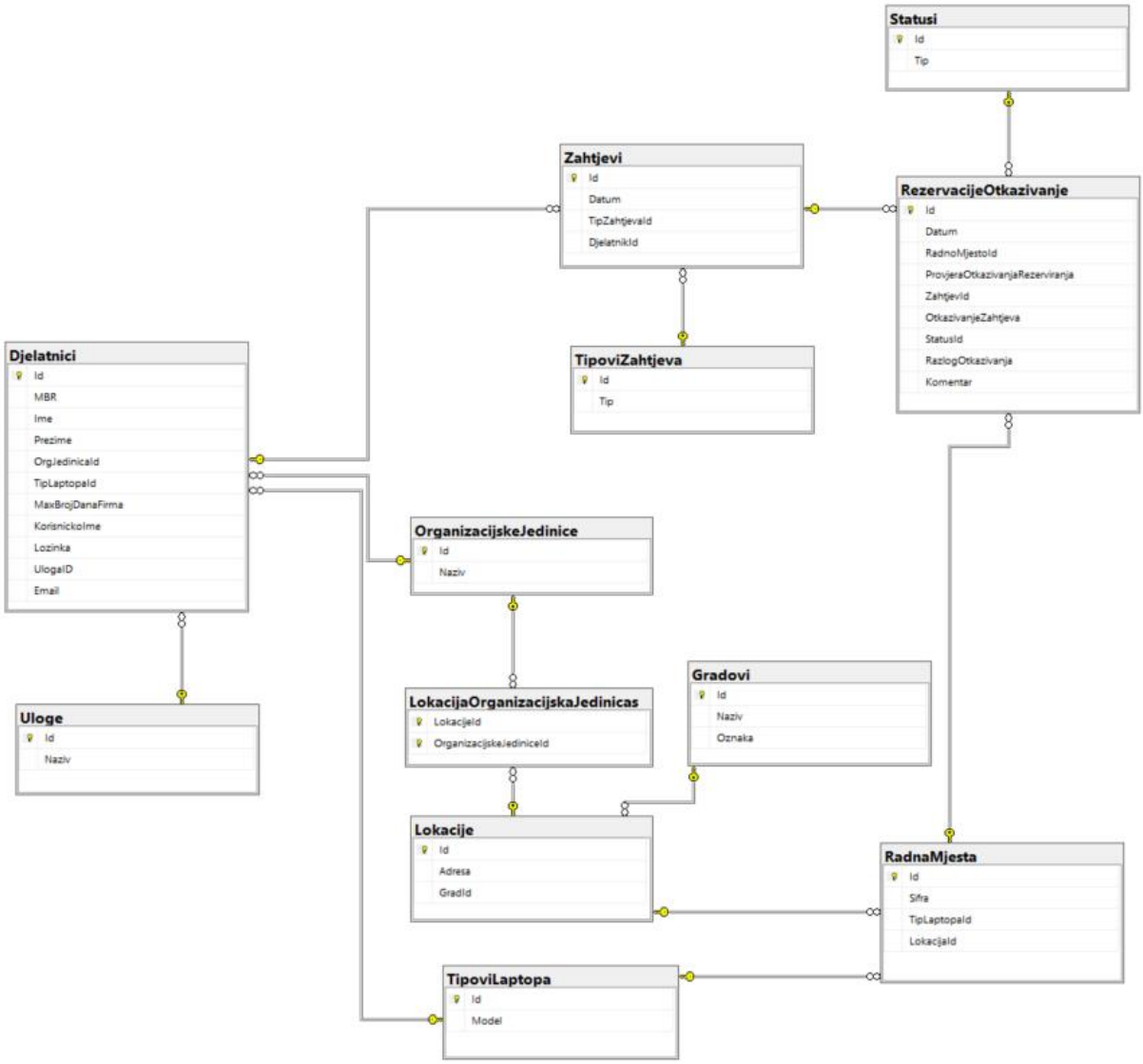
Nakon definiranja konekcijskog znakovnog niza, potrebno je definirati uslugu (engl. *services*) *DbContext* za komunikaciju s bazom podataka unutar metode *ConfigureServices()* u *Startup.cs* klasi. Usluge su zapravo vanjske klase koje sadrže metode i atribute za ostvarivanje nekih funkcionalnosti koje nisu predefinirano uključene u projekt. Registriraju se pomoću IoC spremnika. U projektu je napravljena posebna klasa *AppDbContext* unutar mape *Data* koja nasljeđuje klasu *DbContext* i sadrži *DbSet<TEntity>* za svaki model te omogućuje razne metode za rad s bazom podataka kao što je izvršavanje *CRUD* operacija nad podatcima u bazi podataka (*SaveChanges()*), pronalazak podatka u bazi pomoću primarnog ključa (*Find()*) i slično. Na slici 4.8. prikazana je metoda *ConfigureServices()* s korištenim uslugama u aplikaciji.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<AppDbContext>(
        options => options.UseSqlServer(
            Configuration.GetConnectionString("DefaultConnection")
            ));

    services.AddSession(so =>
    {
        so.IdleTimeout = TimeSpan.FromSeconds(30000);
    });
    services.Configure<MailSettings>(Configuration.GetSection("MailSettings"));
    services.AddTransient<IMailService, MailService>();
    services.AddControllersWithViews();
    services.AddCloudscribePagination();
}
```

Slika 4.8. Metoda *ConfigureServices()* u *Startup.cs* klasi

Nakon postavljanja okruženja i kreiranja modela, potrebno je napraviti migraciju. Rezultat migracije je prikazan na slici 4.9.



Slika 4.9. Kreirana baza podataka za projekt

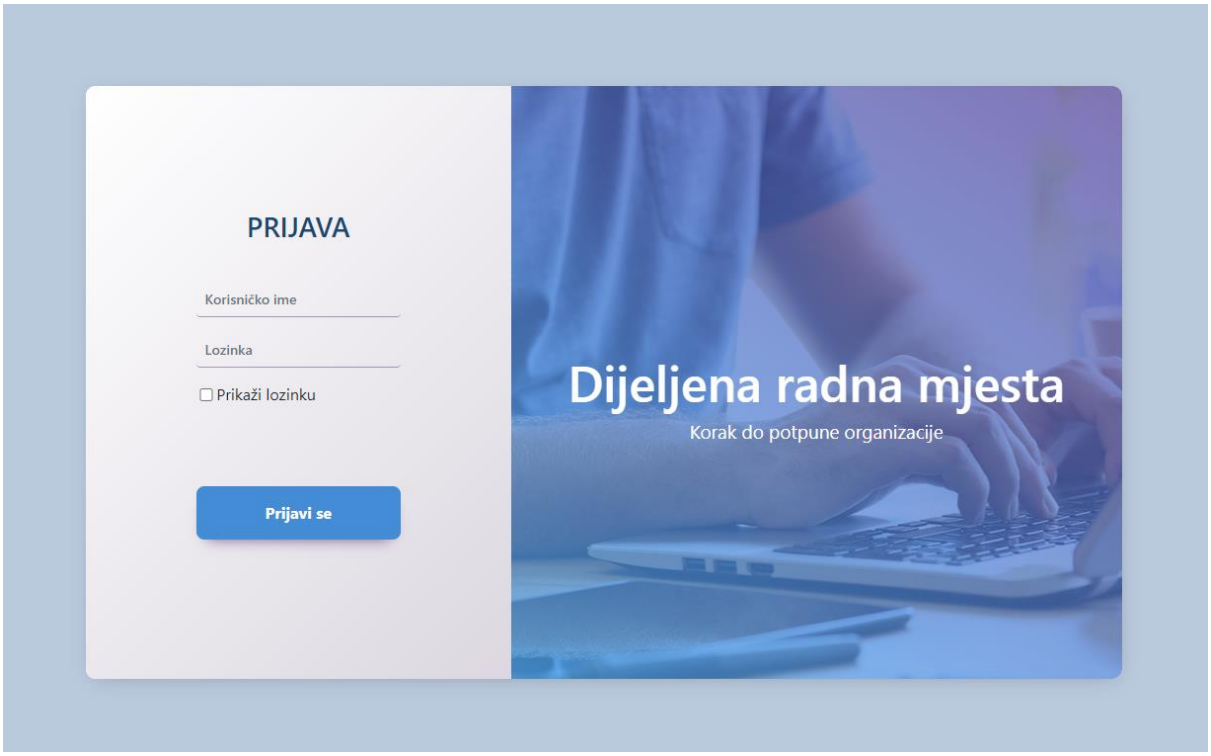
4.3. Korisnička strana programskog rješenja

Aplikacija je podijeljena na dvije razine pristupa: djelatnik i administrator. Djelatnik je korisnik koji ima nižu razinu pristupa i nema uvid u sve funkcionalnosti aplikacije. On ima pristup četiri funkcionalnosti aplikacije: prijava u sustav, pregled zahtjeva, rezervacija i otkazivanje zahtjeva. Djelatniku aplikacija služi za rezerviranje i otkazivanje radnih dana u tvrtki.

4.3.1. Prijava u aplikaciju

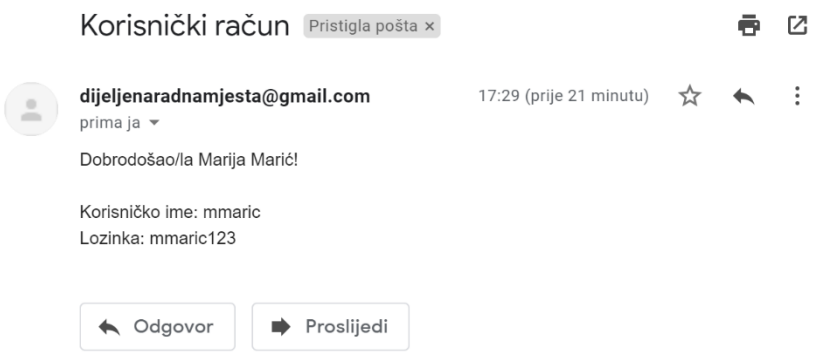
Pri pokretanju aplikacije prvo se pojavljuje zaslon za prijavu korisnika prikazan na slici 4.10. gdje korisnik unosi korisničko ime i lozinku. Logika je implementirana pomoću

AutentifikacijaController-a koji sadrži sve potrebne metode i provjere ispravnosti unesenih podataka.



Slika 4.10. Izgled zaslona za prijavu u sustav

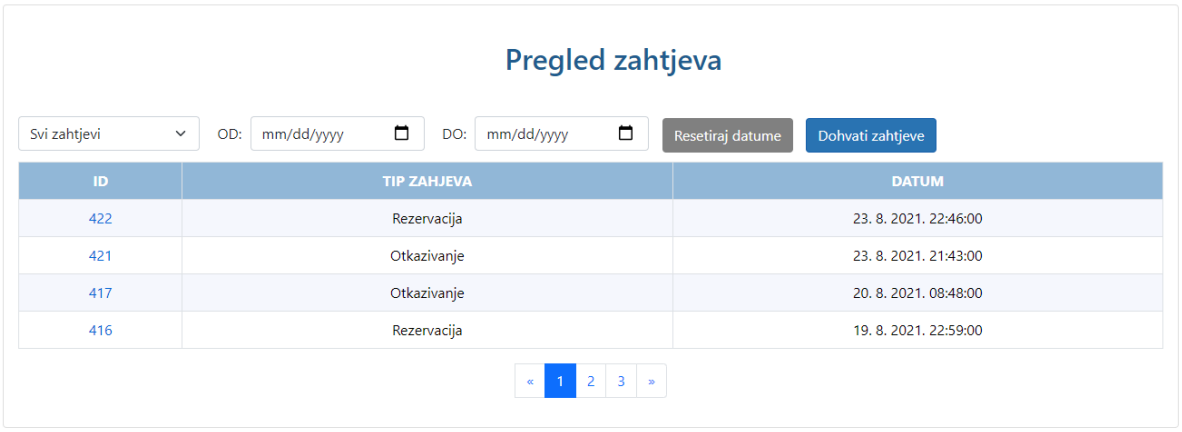
Kako bi aplikacija razlikovala djelatnika i administratora, dodjeljuje im se uloga prilikom njihovog kreiranja. Također, pri dodavanju korisnika u sustav, on dobiva e-mail s informacijama o svom korisničkom računu (Slika 4.11.). U e-mail-u korisnik dobiva lozinku u izvornom obliku, dok se ona u bazu pohranjuje u kodiranom obliku pomoću MD5 enkripcije. MD5 enkripcija je jednosmjerna enkripcija koja onemogućuje dekodiranje jednom kodirane vrijednosti čime se postiže očuvanje korisnikove privatnosti. Prilikom prijave u sustav, upisana lozinka se kodira i uspoređuje s pohranjenom kodiranom vrijednošću u bazi podataka. Za slanje e-mailova korištena je *MailKit* biblioteka koja je ugrađena u .NET Standard, besplatna je i izrazito proširiva [26]. Prije korištenja biblioteka, potrebno je dodati uslugu slanja e-mailova u *ConfigureServices()* u Startup klasi i napraviti konfiguraciju u appsetting.json-a.



Slika 4.11. Primjer e-mail-a prilikom dodavanja djelatnika u sustav

4.3.2. Pregled zahtjeva

Nakon uspješne prijave u sustav, pojavljuje se zaslon za pregled zahtjeva koji sadrži sve korisnikove podnesene zahtjeve sortirane po datumu (Slika 4.12.). Zahtjeve je moguće filtrirati prema tipu zahtjeva i vremenskom rasponu.



Slika 4.12. Dizajn zaslona za pregled zahtjeva

Kako bi se povećala preglednost podataka, koristi se paginacija koja ograničava broj prikazanih podataka. Paginacija je napravljena pomoću ASP.NET Core ugrađenih pomagača za oznake za paginaciju, tj. *cloudscribe.Web.Pagination*. Kako bi se izbjeglo dupliciranje istog koda za paginaciju kod svakog pogleda, korišten je djelomičan pogled (Slika 4.13.)

```
<partial name="_Pagination" view-data="ViewData" model="Model" />
```

Slika 4.13. Dodavanje djelomičnog pogleda na pogled za pregled zahtjeva

Radi poboljšanja korisničkog iskustva (engl. *User Experience, UX*) napravljen je prikaz poruke prilikom prelaska miša po ID-ju zahtjeva. Pritiskom na ID zahtjeva, moguće je vidjeti detalje o samom zahtjevu i u kojoj je fazi odobravanja (Slika 4.14.).

Slika 4.14. Prikaz zaslona za detalje zahtjeva

Kako bi se osigurala razmjena i pohrana privremenih podataka za vrijeme trajanja korisnikove sesije, koristi se ASP.NET Core biblioteka *Session*. Biblioteka se dodaje kao usluga u metodu *ConfigureServices()* i u cjevovod. Dva najčešća načina skladištenja podataka su kolačići (engl. *cookies*) i sesije. Kolačići dugotrajno čuvaju podatke i pohranjuju se na korisnikovom uređaja, dok je sesija privremena i predefinirano čuva podatke dvadeset minuta, što je moguće promijeniti na željeni period. Također, podatci sesije se pohranjuju na poslužiteljskoj strani [27]. Jedan poslužitelj koriste mnogobrojni korisnici što znači da on sadrži mnoštvo sesija. Kako bi poslužitelj razlikovao sesije, svakoj sesiji se pridaje ID koji je poznat kao sesija-kolačić (engl. *session-cookies*). On se pohranjuje na korisnikovom uređaju i pomoću njih se pruža usluga višekorisničkog okruženja s pravilnim pristupom podatcima. Na slici 4.15. prikazan je primjer pohranjivanja vrijednosti u sesiju u slučaja odabranih vrijednosti iz filtera(tip zahtjeva i raspon datuma), a na slici 4.16. se dohvaćaju ti podatci iz sesije.

```
HttpContext.Session.SetInt32(SessionTipZahtjeva, (int)zahtjeviSPaginacijom.pregledZahtjeva.TipZahtjeva);
HttpContext.Session.SetString(SessionPocetniDatum, zahtjeviSPaginacijom.pregledZahtjeva.PocetniDatum.ToString());
HttpContext.Session.SetString(SessionKrajnjiDatum, zahtjeviSPaginacijom.pregledZahtjeva.KrajnjiDatum.ToString());
```

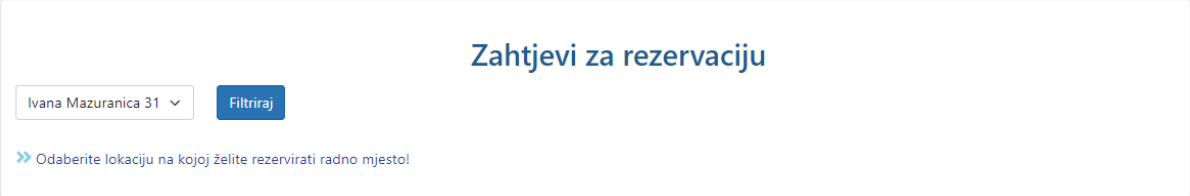
Slika 4.15. Primjeri pohranjivanja podataka u sesije

```
var pocetniDatum = HttpContext.Session.GetString(SessionPocetniDatum);
var krajnjiDatum = HttpContext.Session.GetString(SessionKrajnjiDatum);
var tipZahtjeva = HttpContext.Session.GetInt32(SessionTipZahtjeva);
```

Slika 4.16. Primjeri dohvaćanja podataka iz sesija

4.3.3. Zahtjevi za rezervaciju

Prilikom odabira taba *Rezervacija* s navigacijske trake, korisnik prvo odabire jednu od ponuđenih lokacija ovisno na kojoj želi rezervirati radno mjesto. Padajući izbornik sadrži lokacije koje pripadaju korisnikovoj organizacijskoj jedinici (Slika 4.17.). Također, korisniku je onemogućeno podnošenje zahtjeva za rezervaciju za isti dan na više različitih lokacija.



Slika 4.17. Prikaz odabira zaslona za rezervacije

Pritiskom na gumb *Filtriraj* pokreće se akcijska metoda *Rezervacije* kontrolera. Glavni dio akcijske metode je dohvaćanje rezervacija. Svi jednostavniji upiti nad bazom podataka su rađeni pomoću LINQ-a. Na slici 4.18. moguće je vidjeti metodu *DohvatiRezervacije()* koja vraća listu rezervacija za narednih pet radnih dana. Prvo se dohvaćaju rezervacije iz tablice *RezervacijeOtkazivanje* u bazi podataka i postavlja se *odgovorCheckBox*-a na *true* kako bi se onemogućilo ponovno označavanje checkbox-a na pogledu. Nakon dohvaćanja postojećih rezervacija, kreiraju se potencijalne rezervacije za dane kada korisnik nije podnio zahtjev za rezervaciju.

```
4 references
public List<RezervacijaModel> DohvatiRezervacije(int djelatnikID, int lokacijaID)
{
    var pocetniDatum = DateTime.Now.DohvatiRadneDane(1).Date;
    var krajnjiDatum = DateTime.Now.DohvatiRadneDane(5).Date;
    var dohvaceneRezervacije = (from rezervacije in db.RezervacijeOtkazivanje
                                join radnoMjesto in db.RadnaMjesta on rezervacije.RadnoMjestoId equals radnoMjesto.Id
                                join lokacija in db.Lokacije on radnoMjesto.LokacijaId equals lokacija.Id
                                join zahtjev in db.Zahtjevi on rezervacije.ZahtjevId equals zahtjev.Id
                                join status in db.Statusi on rezervacije.StatusId equals status.Id
                                where (zahtjev.DjelatnikId == djelatnikID &&
                                rezervacije.ProvjeraOtkazivanjaRezerviranja == null &&
                                rezervacije.Datum >= pocetniDatum &&
                                rezervacije.Datum <= krajnjiDatum )
                                orderby rezervacije.Datum ascending
                                select new RezervacijaModel()
                                {
                                    Id = rezervacije.Id,
                                    SifraRadnogMjesta = radnoMjesto.Sifra,
                                    ZeljeniDatum = rezervacije.Datum,
                                    OdgovorCheckBox = true,
                                    Rezervirano = true,
                                    LokacijaID = radnoMjesto.LokacijaId,
                                    Adresa = lokacija.Adresa,
                                    Otkazano = false,
                                    Status=status.Tip,
                                    OtkazivanjeZahtjeva=rezervacije.OtkazivanjeZahtjeva
                                }).ToList();

    var adresa = db.Lokacije.Where(l => l.Id.Equals(lokacijaID)).Select(l => l.Adresa).FirstOrDefault().ToString();
    for (int i = 1;  i <= 5; i++)
    {
        var datum = DateTime.Now.DohvatiRadneDane(i).Date;
        if (dohvaceneRezervacije.FirstOrDefault(rez => rez.ZeljeniDatum.Date.Equals(datum)) == null)
        {
            dohvaceneRezervacije.Add(new RezervacijaModel()
            {
                ZeljeniDatum = datum,
                OdgovorCheckBox = false,
                Rezervirano = false,
                Adresa = adresa,
                Otkazano = false
            });
        }
    }
    return dohvaceneRezervacije.OrderBy(rez => rez.ZeljeniDatum).ToList();
}
```

Slika 4.18. Metoda *DohvatiRezervacije()*

Na slici 4.19. prikazan je rezultat metode *DohvatiRezervacije()* koji kontroler prosljeđuje na pogled. Idući korak, korisnik odabire dane za koje želi podnijeti zahtjev za rezervaciju, označava *checkbox* i pritišće gumb *Podnesi zahtjev* koji će vratiti najmanje slobodno korišteno radno mjesto.

Slika 4.19. Prikaz zaslona za rezervaciju zahtjeva

Aplikacija je namijenjena višekorisničkom okruženju s naglaskom na maksimalnu iskorištenost dijeljenih radnih mjesta. Radna mjesta se dohvaćaju pomoću algoritma koji će korisniku dodijeliti najmanje korišteno slobodno radno mjesto u tom trenutku, pri tome vodeći brigu o korisnikovoj lokaciji i tipu laptopa. Zbog složenosti algoritma, on je napisan u obliku procedure koja se izvršava na poziv iz programa. Algoritam se može podijeliti na četiri dijela:

1) Provjera je li djelatnik već podnio rezervaciju za taj datum (Slika 4.20.).

```
IF((SELECT COUNT(RezervacijeOtkazivanje.Id)
    FROM RezervacijeOtkazivanje
    LEFT JOIN Zahtjevi ON Zahtjevi.Id=RezervacijeOtkazivanje.ZahtjevId
    WHERE DjelatnikId=@DjelatnikID AND RezervacijeOtkazivanje.Datum=@Datum
    AND RezervacijeOtkazivanje.ProvjeraOtkazivanjaRezerviranja is null ) = 0)
```

Slika 4.20. Provjera postoji li rezervacija

2) Provjera je li već kreiran zahtjev i ako nije, kreira se. Ako se istovremeno podnese više rezervacija, one će se spremiti pod jedan zahtjev i kod pregledavanja detalja o zahtjevu, korisnik će dobiti informacije o svakom podzahtjevu zasebno. (Slika 4.21.).

```
IF((SELECT Zahtjevi.Id FROM Zahtjevi WHERE DjelatnikId=@DjelatnikID
    AND TipZahtjevaId=1 AND Datum=@datumZahtjeva) is not null)
    BEGIN
        SELECT @zahtjevID = Zahtjevi.Id FROM Zahtjevi WHERE DjelatnikId=@DjelatnikID
        AND TipZahtjevaId=1 AND Datum=@datumZahtjeva
    END
ELSE
BEGIN
    INSERT into Zahtjevi(DjelatnikId, TipZahtjevaId, Datum) VALUES (@DjelatnikID, 1, @datumZahtjeva)
    SELECT @zahtjevID = Id FROM Zahtjevi WHERE DjelatnikId=@DjelatnikID AND TipZahtjevaId=1
    AND Datum=@datumZahtjeva
END
```

Slika 4.21. Provjera postoji li već kreirani zahtjev i kreiranje zahtjeva

3) Pronalazak radnog mjesta koje je najmanje korišteno i zadovoljava tražene kriterije (Slika 4.22.).

```
SELECT TOP 1 @radnoMjestoID = RadnaMjesta.Id
FROM RezervacijeOtkazivanje
RIGHT JOIN RadnaMjesta ON RadnaMjesta.Id = RezervacijeOtkazivanje.RadnoMjestoId
where RadnaMjesta.TipLaptopaId=(SELECT TipLaptopaId FROM Djelatnici WHERE ID=@DjelatnikID)
AND RadnaMjesta.LokacijaId=@LokacijaID
AND RadnaMjesta.Id in
(SELECT RadnaMjesta.Id
FROM RadnaMjesta
LEFT JOIN Lokacije ON Lokacije.Id=RadnaMjesta.LokacijaId
LEFT JOIN LokacijaOrganizacijskaJedinicas ON LokacijaOrganizacijskaJedinicas.LokacijeId=Lokacije.Id
LEFT JOIN OrganizacijskeJedinice ON OrganizacijskeJedinice.Id=LokacijaOrganizacijskaJedinicas.OrganizacijskeJediniceId
LEFT JOIN RezervacijeOtkazivanje ON RezervacijeOtkazivanje.RadnoMjestoId=RadnaMjesta.Id
WHERE RadnaMjesta.TipLaptopaId=(SELECT TipLaptopaId FROM Djelatnici WHERE Id=@DjelatnikID)
AND OrganizacijskeJedinice.Id=(SELECT OrgJedinicaId FROM Djelatnici WHERE Id=@DjelatnikID )
AND Lokacije.Id=@LokacijaID
AND RadnaMjesta.Id not in
(SELECT RadnoMjestoId FROM RezervacijeOtkazivanje WHERE Datum = @Datum
AND RezervacijeOtkazivanje.ProvjeraOtkazivanjaRezerviranja is null)
GROUP BY RadnaMjesta.Id)
GROUP BY RadnaMjesta.Id
ORDER BY COUNT(RadnoMjestoId) ASC
```

Slika 4.22. Pronalazak najmanje korištenog radnog mjesta

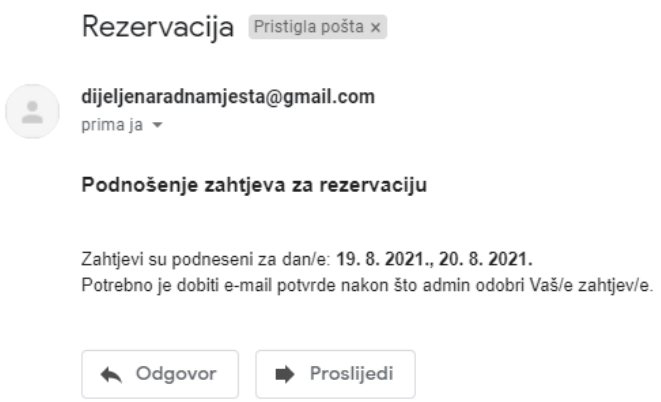
4) Ako je pronađeno radno mjesto, kreira se podzahtjev za rezervaciju. U suprotnom se briše napravljeni zahtjev. Kreirani podzahtjev ima Status *čekanje* dok administrator ne odobri ili odbije zahtjev (Slika 4.23.).

```
IF (@radnoMjestoID IS NOT NULL)
    BEGIN
        INSERT INTO RezervacijeOtkazivanje(ZahtjevId, RadnoMjestoId, Datum, StatusId)
        VALUES ( @zahtjevID ,@radnoMjestoID , @Datum, 3);
    END
ELSE IF((SELECT COUNT(*) FROM RezervacijeOtkazivanje WHERE ZahtjevId = @zahtjevID) = 0)
    BEGIN
        DELETE FROM Zahtjevi WHERE Id = @zahtjevID
    END
```

Slika 4.23. Provjera uspješnosti pronalaska slobodnog radnog mjesta

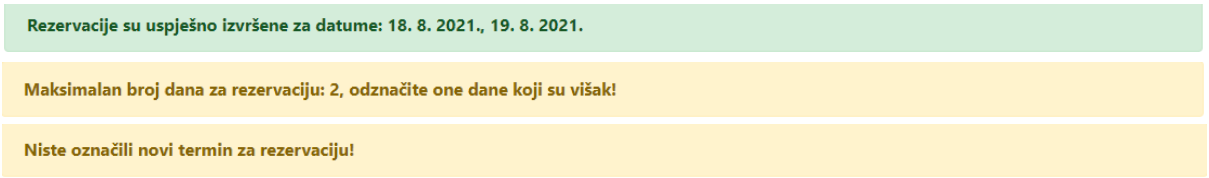
Osim provjera kroz algoritam, postavljeno je jedinstveno ograničenje (engl. *unique constraint*) na tablicu *RezervacijeOtkazivanje* koje osigurava da su sve upisane vrijednosti jedinstvene i tako se sprječava upisivanje rezervacija za isto radno mjesto na isti datum.

Nakon podnošenja zahtjeva za rezervaciju i u slučaju uspješnog podnošenja zahtjeva se na zaslonu pojavljuje šifra rezerviranog radnog mjesta i poruka o uspješnoj rezervaciji. U slučaju da algoritam ne pronađe radno mjesto pojavit će se poruka o neuspješnoj rezervaciji. Kako bi korisnik mogao doći raditi na taj dan u tvrtku, administrator treba odobriti zahtjev. U slučaju odbijanja zahtjeva od administratorske strane, pojavit će se poruka *Odbijeno* u stupcu *Radno mjesto* i korisnik više neće moći podnijeti zahtjev za taj datum, u suprotnom ostat će šifra radnog mjesta i korisnik će primiti e-mail o uspješnom podnošenju zahtjeva prikazan na slici 4.24.



Slika 4.24. Primjer e-mail-a kod podnošenja zahtjeva za rezervaciju

Radi postizanja boljeg korisničkog iskustva, nakon svake izvedene akcije u aplikaciji prikazuju se pripadajuće poruke na zaslonu, neke od njih su prikazane na slici 4.25.



Slika 4.25. Prikaz primjera poruka

4.3.4. Zahtjevi za otkazivanje

Korisnik može otkazati svaki odobreni zahtjev za rezervaciju, kako bi se izbjeglo učestalost otkazivanja i rezerviranja zahtjeva. Kod otkazivanja zahtjeva, postoji mogućnost unosa

objašnjenja zašto se zahtjev otkazuje kako bi se smanjio i olakšao posao administratora. Također, korisnik može podnijeti zahtjev za otkazivanje do jedan dan ranije. Izgled zaslona je prikazan na slici 4.26.

Slika 4.26. Zahtjevi za otkazivanje

Nakon što korisnik podnese zahtjev za otkazivanje, zahtjev promijeni svoju vrijednost atributa *OtkazivanjeZahtjeva*, ali pri tome ne otkaže zapravo zahtjev skroz dok administrator ne odobri otkazivanje zahtjeva, kako bi se sačuvalo radno mjesto u slučaju neodobravanja zahtjeva.

4.4. Administratorska strana programskog rješenja

Najčešća podjela jednostavnije hijerarhije unutar tvrtke je na korisnike i administratore. Administrator je korisnik koji ima veća prava i mogućnosti. U aplikaciji je administrator ujedno i djelatnik, tako da on ima djelatnikove funkcionalnosti uz dodatne funkcionalnosti kao što su kreiranje, promjena, brisanje i uređivanje (engl. *CRUD*):

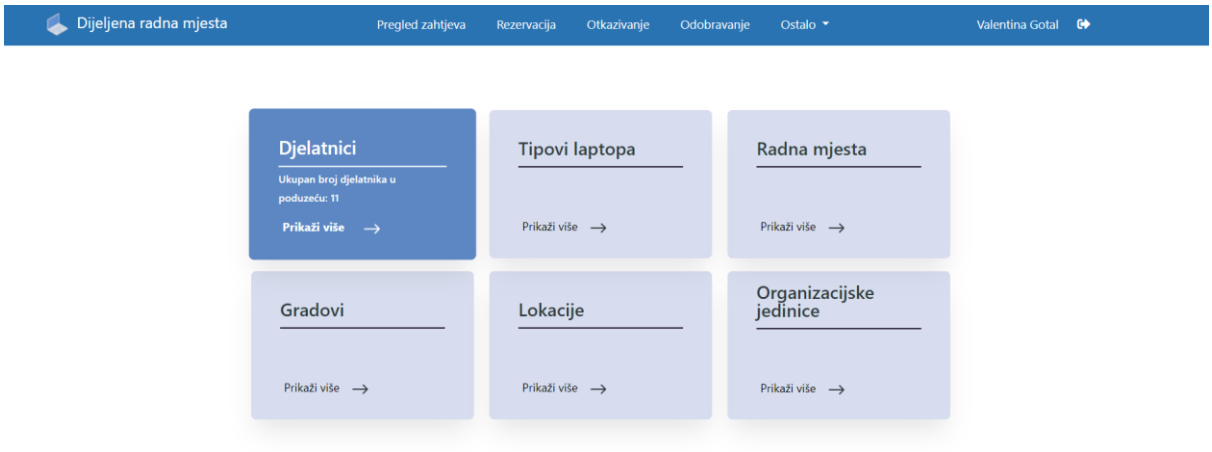
- Djelatnika
- Tipova laptopa
- Radnih mjesta
- Gradova
- Lokacija
- Organizacijskih jedinica

Osim navedenih zadataka, glavni zadatak administratora je odobravanje zahtjeva.

4.4.1. Početni zaslon aplikacije

Administrator i djelatnik imaju istu prijavu u sustav. Nakon prijave u sustav, djelatniku se prikazuje pregled zahtjeva, dok se administratoru prikazuje početni zaslon s karticama. Svaka kartica predstavlja jednu stavku koja je važna za pravilan rad aplikacije, tj. svaka kartica predstavlja jednu osnovu tablicu u bazi podataka i omogućava pristup i rukovanje pohranjenim podatcima u bazi podataka te dodavanje novih podataka preko korisničkog sučelja. Prelaskom

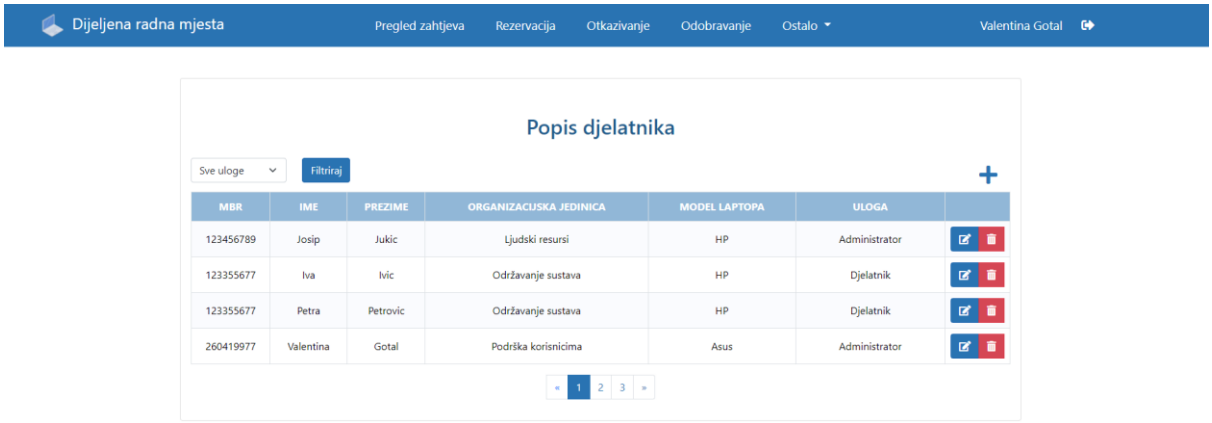
preko kartice, pojavljuje se informacija o broju postojećih podataka, npr. broj djelatnika unutar tvrtke kao što je prikazano na slici 4.27.



Slika 4.27. Izgled administratorskog početnog zaslona

4.4.2. Pregled podataka

Pregled podataka se dobiva pritiskom na jednu od kartica na početnom zaslonu. On omogućava pregled svih postojećih podataka u bazi podataka. Na svakom pregledu zahtjeva postoje određeni filteri koji olakšavaju pronalazak potrebnog podatka, paginacija i mogućnost uređivanja, dodavanja i brisanja podataka, kao što je vidljivo na slici 4.28. Kreiranje novog podatka, uređivanje i brisanje postojećeg podatka bit će objašnjeni na primjeru djelatnika.



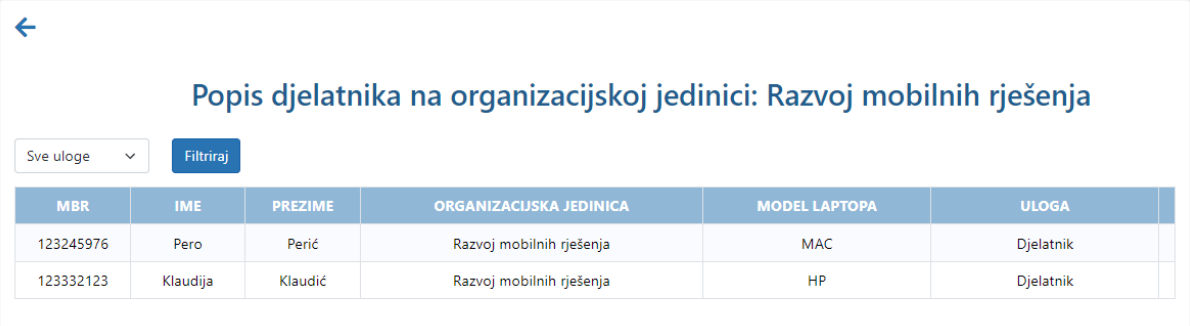
Slika 4.28. Izgled zaslona za pregled podataka o djelatniku

Radi postizanja lakšeg i organiziranijeg pristupa omogućeno je različito preusmjeravanje i pristup podatcima, npr. informacije o lokacijama i djelatnicima se može pristupiti putem njihovih kartica na početnom zaslonu, ali i preko drugih putanja. Odabirom kartice

Organizacijske jedinice, otvara se zaslon prikazan na slici 4.29. na kojem dobivamo mogućnost dodavanja, brisanja i uređivanja organizacijskih jedinica i pregled na koliko lokacija se nalazi organizacijska jedinica i koliko ima zaposlenih.

Slika 4.29. Izgled zaslona za pregled podataka o organizacijskim jedinicama

Također, pritiskom na informacije o djelatnicima, stranica nas preusmjeri na stranicu pregleda podataka o djelatnicima, ali za razliku od slike 4.28. na ovaj način se dobije pristup popisu djelatnika samo unutar te organizacijske jedinice kao što prikazano na slici 4.30. i to bez mogućnosti uređivanja, brisanja i dodavanja djelatnika. To se postiže preko istog pogleda jer ovisno o primljenim podatcima on prikazuje određene podatke na određeni način.



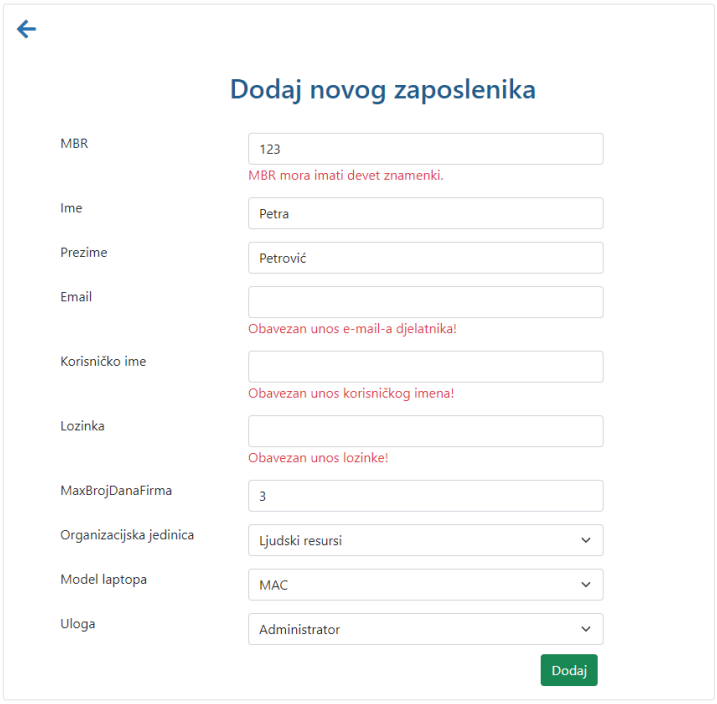
Slika 4.30. Izgled zaslona za pregled podataka o djelatnicima preko zaslona za organizacijske jedinice

Slična situacija je i kod pristupa lokacijama preko zaslona od organizacijskih jedinica, samo što se preko tog zaslona omogućava i naknadno dodavanje lokacije unutar organizacijske jedinice pritiskom na gumb s ikonom uredi. Izgled zaslona za lokacije kojem se pristupa preko zaslona organizacijske jedinice, vidljiv je na slici 4.31. Preko njega je moguće pristupiti i listi radnih mjesta na svakoj lokaciji pritiskom na gumb s uskličnikom.

Slika 4.31. Izgled zaslona za pregled podataka o lokacijama preko zaslona za organizacijske jedinice

4.4.3. Kreiranje novog podatka

Pritiskom na oznaku plus na zaslonu za pregled podataka, otvara se novi zaslon za dodavanje novog podatka, u ovom slučaju djelatnika. Postavljena su ograničenja na polja za unos kako bi se osiguralo pravilno spremanje podataka u bazu podataka. Na slici 4.32. moguće je vidjeti izgled zaslona za kreiranje novog djelatnika s prikazom upozorenja koja se pojavljuju nakon što administrator pritisne gumb *Dodaj* prije nego je popunio sve tražene podatke.



Slika 4.32. Izgled zaslona za dodavanje novog djelatnika

Nakon popunjavanja svih potrebnih polja, pritiskom na gumb *Dodaj* poziva se kontroler a akcijskom metodom *Create* s *post* HTTP (**engl.** *Hypertext Transfer Protocol*) metodom.

Funkcionalnost aplikacije je osmišljena pomoću dvije HTTP metode: *get* i *post*. Svaki zaslon ima dvije akcijske metode s istim nazivom, ali drugačijom HTTP metodom. HTTP metode su dizajnirane za omogućavanje komunikacije između klijenta, web preglednika i poslužitelja [28]. Na svaku akcijsku metodu se može primijeniti više HTTP metoda kako bi se specificiralo rukovanje različitih HTTP zahtjeva. U slučaju da se ne specificira HTTP metoda, sve akcijske metode će biti predefinirane, tj. *get*. *Get* metoda služi za slanje zahtjeva za dohvaćanje nekih podataka sa servera, dok *post* metoda služi za slanje zahtjeva s podatcima poslužitelju kako bi on kreirao ili ažurirao podatke.

Na slici 4.33. prikazana je *Create* akcijska metoda s *get* i *post* HTTP metodom. Kada se pozove *get* metoda ona kreira zahtjev koji usmjerava na akcijsku metodu *Create* kontrolera *Djelatnik* s parametrom *orgJedID*. Kontroler će kao odgovor vratiti pogled kojem predaje model. U slučaju pozivanja *post* metode, kontroler će primiti model s podatcima za kreiranje novog djelatnika. Akcijska metoda vraća kao odgovor model s podatcima za filtere bez podataka o novom djelatniku jer se tako omogućuje nastavak dodavanja novih djelatnika bez potrebe za ponovnim učitavanjem stranice.

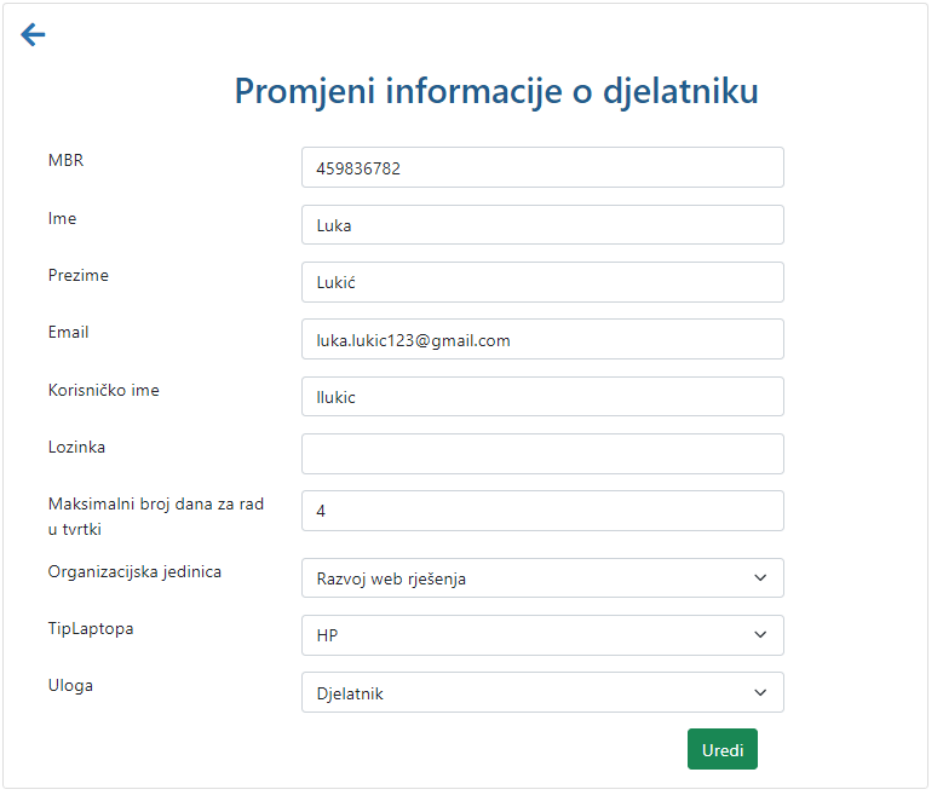
```
[HttpGet]
0 references
public IActionResult Create(int orgJedID)
{
    TempData["OrgJedID"] = orgJedID;
    var djelatnik = new DjelatnikVM();
    djelatnik = djelatnikRepository.PopuniFiltereSPodatcima(orgJedID, djelatnik);
    return View(djelatnik);
}

[HttpPost]
[ValidateAntiForgeryToken]
0 references
public IActionResult Create(DjelatnikVM djelatnik)
{
    var orgJedID = HttpContext.Session.GetInt32(SessionOrgJedID);
    if (ModelState.IsValid)
    {
        if (djelatnikRepository.DodajNovogDjelatnika(djelatnik))
        {
            HttpContext.Session.SetInt32(SessionDjelatnik, djelatnik.MBR);
            TempData["Uspješno"] = "Uspješno dodan novi djelatnik!";
        }
        else
        {
            TempData["Neuspješno"] = "Djelatnik s tim podatcima već postoji!";
        }
    }
    ModelState.Clear();
    var noviDjelatnik = new DjelatnikVM();
    noviDjelatnik = djelatnikRepository.PopuniFiltereSPodatcima(orgJedID, noviDjelatnik);
    return View(noviDjelatnik);
}
```

Slika 4.33. *Get* i *post* Create akcijske metode kontrolera Djelatnik.

4.4.4. Uređivanje podataka

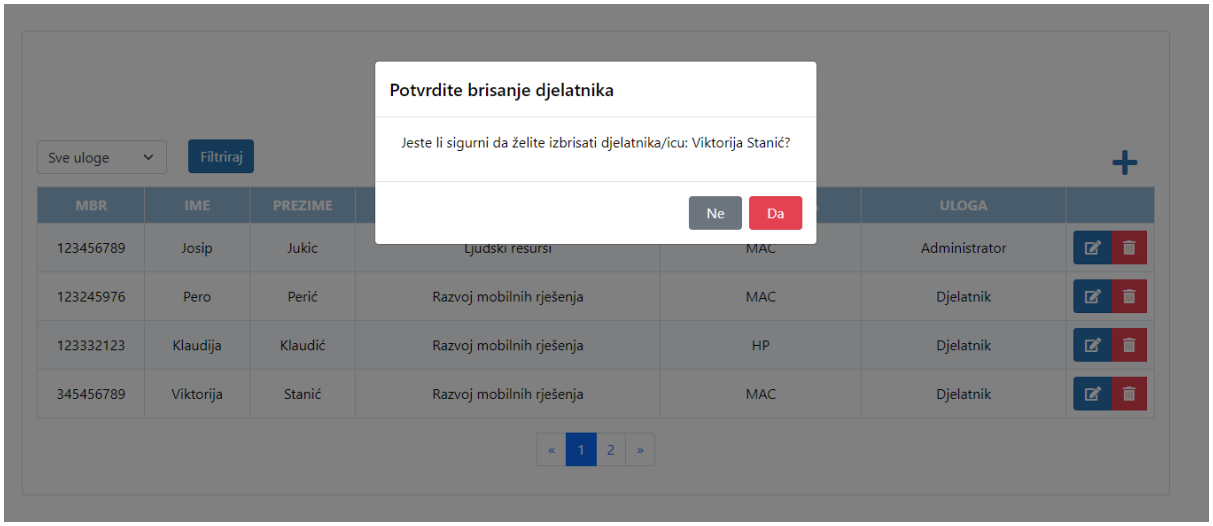
Na zaslonu za pregled podataka, u svakom retku postoji na kraju stupac s gumbom za uređivanje koji usmjerava na zaslon za uređivanje podataka. Na slici 4.34. prikazan je zaslon za uređivanje s podatcima jednog djelatnika. Moguće je primijetiti kako je polje lozinka prazno jer pri kreiranja djelatnika se unosi lozinka koja se putem e-mail-a šalje djelatniku i nakon toga nitko drugi nema pristup toj lozinki.



Slika 4.34. *Zaslon za uređivanje djelatnika*

4.4.5. Brisanje podataka

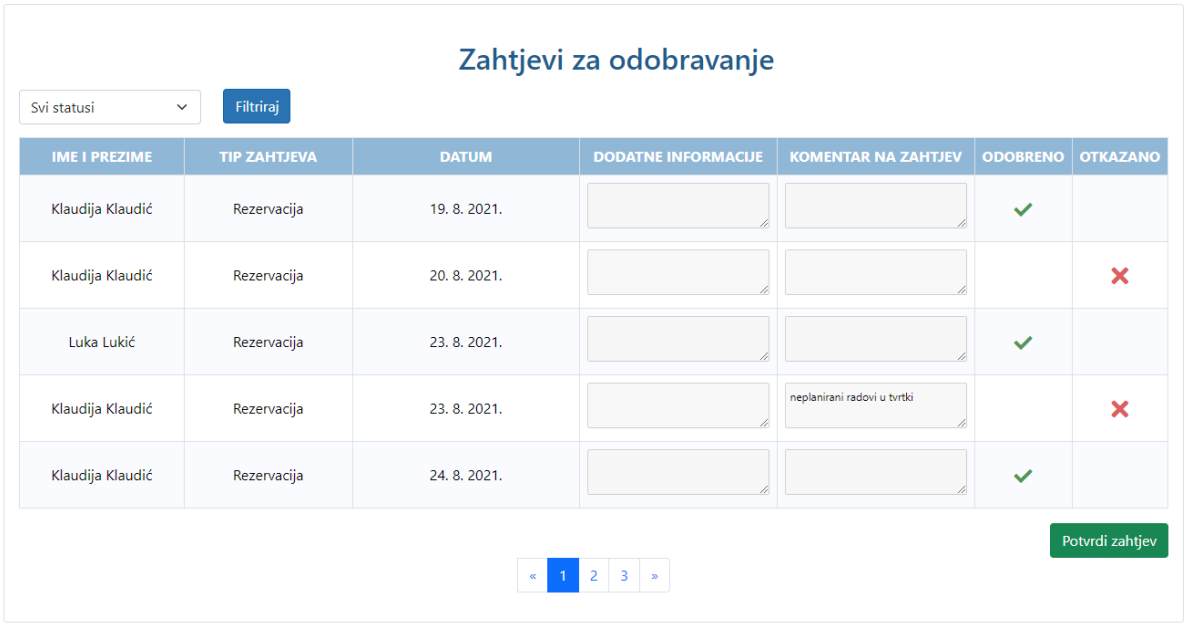
Brisanje podataka se omogućuje putem zaslona za pregled zahtjeva, pritiskom na gumb u zadnjem stupcu. Nije napravljen zaseban zaslon za brisanje podataka, nego se pojavljuje modal pomoću kojeg se potvrđuje brisanje, kao što je prikazano na slici 4.35. Nakon uspješnog brisanja, pojavljuje se poruka na zaslonu o uspješnosti brisanja djelatnika.



Slika 4.35. *Modal za brisanje djelatnika*

4.4.6. Odobravanje zahtjeva

Svaki djelatnik nakon što podnese zahtjev za rezervaciju ili zahtjev za otkazivanje, mora dobiti odobrenje od administratora. Kako je administrator ujedno i djelatnik, sustav je osmišljen tako da postoje minimalno dva administratora kako bi se izbjeglo samostalno potvrđivanje zahtjeva za rezervaciju. Izgled zaslona je prikazan na slici 4.36.



Slika 4.36. *Zaslon za odobravanje zahtjeva*

Administrator prilikom odbijanja može navesti razlog zašto je zahtjev odbijen, kako bi djelatnik dobio povratnu informaciju. Nakon odobravanja i/ili odbijanja zahtjeva, djelatnik prima e-mail o statusu zahtjeva kao što je prikazano na slici 4.37.

Slika 4.37. *Zaslon za odobravanje zahtjeva*

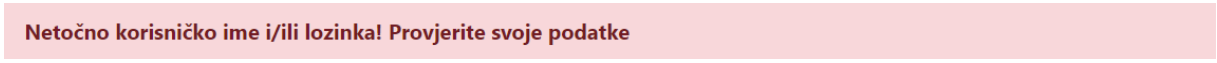
5. TESTIRANJE FUNKCIONALNOSTI APLIKACIJE

Aplikacija je napravljena za višekorisničko okruženja i kako bi se omogućilo testiranje aplikacije u takvom okruženju, postavljena je na Internet pomoću Microsoft Azure i GitHub-a. Aplikaciju je testirala u grupi od četvero ljudi kako bi se provjerila ispravnost rada aplikacije u višekorisničkom okruženju. Testiranje je podijeljeno u tri glavna djela:

- Prijava u sustav
- Podnošenje, pregled i odobrenje zahtjeva
- Rad s podatcima

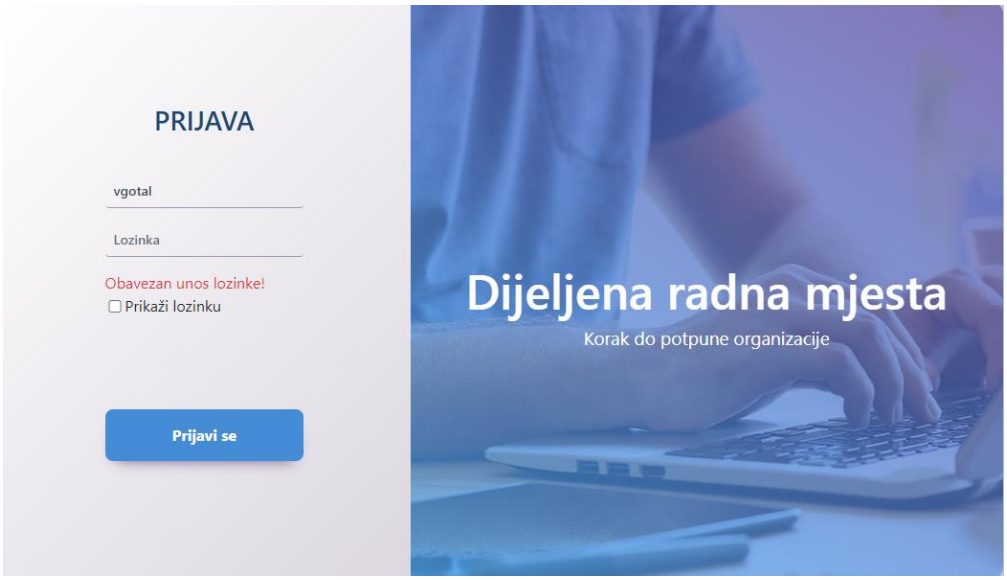
5.1. Prijava u sustav

Prvi zaslon koji se pojavi prilikom odlaska na *url* stranice je prijava u sustav. Testirana su ograničenja i kako aplikacija odgovara na pogrešne upite. U slučaju da korisnik unese pogrešne podatke, ispisat će se poruka na zaslonu prikazana na slici 5.1., a kako bi se osiguralo da korisnik može vidjeti unesenu lozinku u slučaju da nije siguran je li ispravna, omogućena je opcija „Prikaži lozinku“ (Slika 5.2.).



Slika 5.1. *Prikaz poruke kod krivog unosa korisničkog imena i/ili lozinke*

U slučaju da korisnik ne unese sve podatke, pojavit će se ispis poruke ispod prostora za unos kao što je prikazano na slici 5.2.



Slika 5.2. *Prikaz poruke kod krivog unosa korisničkog imena i/ili lozinke*

Kod neispravnog unosa, pojavit će se poruka o neispravnom unosu, dok će se kod ispravnog unosa, korisnik ulogirati. Također, testirana je ispravnost preusmjeravanja nakon prijave ovisno o tome je li korisnik djelatnik ili administrator.

5.2. Podnošenje, pregled i odobrenje zahtjeva

Glavna funkcionalnost aplikacije je pregled i podnošenje zahtjeva za rezervaciju i otkazivanje. Ovo testiranje je provedeno u grupi od četvero ljudi koji su pripadali istoj organizacijskoj jedinici i imali istu računalnu opremu. Cilj testiranja je bio provjeriti hoće li algoritam za dodjelu radnih mjesta raditi ispravno u slučaju kada njih četvero pokuša istovremeno s različitih uređaja rezervirati isto radno mjesto. Zbog potreba testa unutar te organizacijske jedinice je postojalo jedno radno mjesto. Rezultat testa je bio uspješan jer jedna od četiri osobe je dobila poruku o uspješnom rezerviranju i e-mail, dok su ostalih troje dobili poruku o nemogućnosti podnošenja rezervacije za taj dan. Test je ponovljen više puta s različitim brojem radnih mjesta i svaki put bi rezultat bio uspješan čime se potvrdila ispravnost napisanog algoritma za dodjelu najmanje korištenih slobodnih radnih mjesta.

Kako bi se provjerila ispravnost ostalih funkcionalnosti na zaslonima: pregled zahtjeva, rezervacija i otkazivanje, aplikaciju je testiralo troje ljudi koji nisu nikad prije vidjeli aplikaciju kako bi se postigla što veća objektivnost. Nakon provedenog testiranja, ljudi su pokazali zadovoljstvo zbog jednostavnog dizajna i poruka koje su im pomogle da dobiju dojam kada je nešto prošlo ispravno ili gdje je nastupio problem. Razgovor s njima je potvrdio da je podnošenje i pregled zahtjeva pristupačan za korištenje i time je aplikacija uspješno prošla test korisničkog zadovoljstva.

Posljednji test koji se provodio iz ovog djela je bio test odobravanja zahtjeva. Testirala se situacija kada administrator odbije zahtjev za rezervaciju i hoće li netko drugi moći rezervirati to radno mjesto. Test je pokazao da se u trenutku odbijanja rezervacije za jedno radno mjesto, to isto radno mjesto se automatski oslobađa i drugi korisnici ga mogu rezervirati. Testiran je slučaj kada postoji jedno radno mjesto i korisnik koji ga je uspio rezervirati i administrator mu je odobrio rezervaciju, ali onda je korisnik podnio zahtjev za otkazivanje. Kada korisnik podnese zahtjev za otkazivanje, radno mjesto se ne oslobađa automatski skroz dok administrator ne odobri zahtjev za otkazivanje što je i potvrđeno kroz ovaj test.

5.3. Rad s podatcima

Administrator ima mogućnost kreiranja, uređivanja i brisanja podataka. Kako bi se provjerila uspješnost rada s podatcima, dva administratora su istovremeno radili s podatcima kako bi se testiralo koliko brzo su podatci vidljivi drugoj osobi. Kako je aplikacija rađena u sinkronom načinu rada, tako je to došlo do izražaja u slučajevima kada bi oba administratora pokušala istovremeno obrisati istog djelatnika, brži administrator bi obrisao djelatnika, dok bi se drugom pojavila poruka o neuspješnom brisanju i djelatnik bi nestao. Osim slučaja kada se pokušava istovremeno nešto uređivati, aplikacija nije pokazala nikakve probleme. Test je pokazao kako bi se u budućnosti aplikacija mogla prebaciti u asinkroni način rada kako bi se automatski vidjele promjene bez potrebe za osvježavanjem stranice.

6. ZAKLJUČAK

Nakon završetka COVID-19 pandemije mnoge firme će prijeći na hibridni način rada koji će biti reguliran pomoću raznih aplikacija za organizaciju radnog prostora. Cilj ovog rada je bio napraviti aplikaciju koja se temelji na maksimalnoj iskorištenosti radnog prostora kako bi se ostvarile uštede na najmu prostora i ostvarila ravnomjerna iskorištenost radnih mjesta i opreme. Za razliku od ostalih aplikacija na tržištu ne sadrži mogućnost izbora radnog mjesta, nego se automatski dodjeljuje najmanje korišteno radno mjesto. U budućnosti je cilj napraviti aplikaciju koja će napraviti kompromis između interesa tvrtke i zadovoljstva zaposlenika kroz omogućavanje zaposlenicima odabir i dodjeljivanje željenih radnih mjesta uz uvjet da se nalaze u grupi najmanje korištenih radnih mjesta.

Zaposlenici imaju mogućnost podnošenja zahtjeva za rezervaciju radnog mjesta, otkazivanje rezervacije i pregled zahtjeva, dok administratori imaju i dodatne funkcionalnosti kao što je odobravanje zahtjeva i mogućnost uređivanja i brisanja svih podataka vezanih za zaposlenike i tvrtku kako bi aplikacija omogućavala i pregled osnovnih informacija o samom poduzeću. Aplikacija radi na principu podnošenja i odobravanja zahtjeva te su te radnje popraćene e-mailovima kako bi djelatnici dobili povratne informacije o statusu njihovog zahtjeva i van same aplikacije.

Aplikacija je namijenjena za rad u višekorisničkom okruženju i kao takva je testirana na Internetu. Testiranje je pokazalo da aplikacija zadovoljava postavljene ciljeve i jednostavnost korištenja, ali i male nedostatke sinkronog rada koji ne utječu izravno na samu funkcionalnost aplikacije. Između ostalih mogućnosti nadogradnje aplikacije, ona se planira prebaciti u asinkroni način rada.

LITERATURA

[1] „Istraživanje: Svaki 5. zaposlenik koji radi od doma jedva čeka povratak u ured, glavni problem - nedostatak samodiscipline“, *Dnevnik.hr*. https://dnevnik.hr/vijesti/koronavirus/istrazivanje-o-radu-od-kuce---602627.html (pristupljeno lip. 22, 2021).

[2] „Workplace app | smart data-driven room booking app | Spacewell“, *Spacewell | A Nemetschek Company*. https://spacewell.com/solutions/workplace-solutions/smart-workplace-app/personal-assistant/ (pristupljeno svi. 17, 2021).

[3] „Hot Desk Booking Software - Envoy Desks“, *Envoy*. https://envoy.com/products/hot-desk-booking-software/ (pristupljeno svi. 17, 2021).

[4] „Desk Booking System“, *Condeco Software*. https://www.condecosoftware.com/products/desk-booking/ (pristupljeno svi. 17, 2021).

[5] Skedda, „Desk Booking System - Workplace Software | Skedda“. https://www.skedda.com/home/desk-booking-system (pristupljeno svi. 17, 2021).

[6] „The CXApp, An Inpixon Company - Desk Booking App“, *The CXApp, An Inpixon Company*. https://thecxapp.com/desk-benefits (pristupljeno svi. 17, 2021).

[7] tdykstra, .„NET introduction and overview“. https://docs.microsoft.com/en-us/dotnet/core/introduction (pristupljeno lip. 26, 2021).

[8] „Five Things You Should Know About .NET 5“, *Auth0 - Blog*. https://auth0.com/blog/dotnet-5-whats-new/ (pristupljeno lip. 26, 2021).

[9] A. FREEMAN, *Pro ASP.NET Core MVC*. Apress, 2016.

[10] „Introduction to ASP.NET Core MVC“, *YogiHosting*, pros. 17, 2018. https://www.yogihosting.com/aspnet-core-introduction/ (pristupljeno lip. 26, 2021).

[11] freeCodeCamp.org, *ASP.NET Core MVC Course (.NET 5)*. Pristupljeno: lip. 26, 2021. [Na internetu Video]. Dostupno na: https://www.youtube.com/watch?v=Pi46L7UYP8I

[12] W. Ofner, „View Components in ASP.NET Core MVC“, *Programming With Wolfgang*, svi. 09, 2019. https://www.programmingwithwolfgang.com/view-components-in-asp-net-core-mvc/ (pristupljeno srp. 09, 2021).

[13] J. Ciliberti, *ASP.NET Core Recipes: A Problem-Solution Approach*, 2nd ed. 2017. Berkeley, CA: Apress : Imprint: Apress, 2017. doi: 10.1007/978-1-4842-0427-6.

[14] „Understanding ASP.NET Core Tag Helpers“. https://www.dotnettricks.com/learn/aspnetcore/aspnet-core-tag-helpers (pristupljeno srp. 07, 2021).

[15] „Tag Helpers in ASP.NET Core MVC“, *Dot Net Tutorials*. https://dotnettutorials.net/lesson/tag-helpers-in-asp-net-core-mvc/ (pristupljeno srp. 07, 2021).

[16] „A Developer Guide to ASP.NET Core Tag Helpers“, *EzzyLearning.net*, lis. 03, 2020. https://www.ezzylearning.net/tutorial/a-developer-guide-to-asp-net-core-tag-helpers (pristupljeno srp. 07, 2021).

[17] „Routing in ASP.NET Core MVC“, *Dot Net Tutorials*. https://dotnettutorials.net/lesson/routing-asp-net-core-mvc/ (pristupljeno srp. 07, 2021).

[18] „What is Bootstrap? An Awesome Beginner’s Guide“. https://careerfoundry.com/en/blog/web-development/what-is-bootstrap-a-beginners-guide/ (pristupljeno lip. 27, 2021).

[19] „Who Uses SQL Server Management Studio and Why“. https://www.mssqltips.com/sqlservertutorial/9275/who-uses-sql-server-management-studio-and-why/ (pristupljeno srp. 08, 2021).

[20] „What is MVC? - The ASP.NET Core MVC Tutorial“. https://asp.mvc-tutorial.com/introduction/what-is-mvc/ (pristupljeno lip. 26, 2021).

[21] „Model and ViewModel in ASP.NET Core MVC Pattern“, *TekTutorialsHub*, sij. 04, 2018. https://www.tektutorialshub.com/asp-net-core/asp-net-core-model-and-viewmodel/ (pristupljeno srp. 08, 2021).

[22] „Razor View Engine & Razor Syntax in ASP.NET Core“, *TekTutorialsHub*, stu. 28, 2017. https://www.tektutorialshub.com/asp-net-core/asp-net-core-razor-view-engine/ (pristupljeno srp. 09, 2021).

[23] „ASP.NET Core default project structure explained (part 1)“. https://korzh.com/blog/asp-net-core-project-structure-explained-part1 (pristupljeno srp. 10, 2021).

[24] S. Kılıçarslan, „ASP.NET Core MVC Web Application (Project Structure)“, *Medium*, stu. 21, 2020. https://medium.com/net-core/asp-net-core-mvc-web-application-project-structure-3ccaa244fa66 (pristupljeno srp. 10, 2021).

[25] „IEnumerable Vs IQueryable In LINQ“. https://www.c-sharpcorner.com/uploadfile/a20beb/ienumerable-vs-iqueryable-in-linq/ (pristupljeno srp. 09, 2021).

[26] WADE, „Using MailKit To Send And Receive Email In ASP.NET Core“, *.NET Core Tutorials*, stu. 02, 2017. https://dotnetcoretutorials.com/2017/11/02/using-mailkit-send-receive-email-asp-net-core/ (pristupljeno kol. 16, 2021).

[27] „Sessions - The ASP.NET Core MVC Tutorial“. https://asp.mvc-tutorial.com/hu/480/httpcontext/sessions/ (pristupljeno kol. 16, 2021).

[28] V. P. | U. D. Sep 10 i 2020 | 9, „HTTP - Overview of the Basic Concepts“, *Code Maze*, lip. 19, 2017. https://code-maze.com/http-series-part-1/ (pristupljeno kol. 20, 2021).

SAŽETAK

Godina 2020/2021. je obilježena COVID-19 pandemijom koja je utjecala na život ljudi i dovela do velikih promjena u poslovnom svijetu. Veliki dio tvrtki, koje su mogle prilagoditi svoju vrstu poslovanja, se odlučuju na rad od kuće kako bi sačuvale zdravlje svojih zaposlenika. Takav način rada je pozitivno prihvaćen što je dovelo do razmatranja ideje o uvođenju hibridnog modela rada po završetku pandemije, tj. kombinacija rada od kuće i u tvrtki.

Na tržištu postoje razne aplikacije koje nude uslugu organizacije radnog prostora s mogućnošću izbora radnog mjesta, dok je u ovom radu cilj bio napraviti aplikaciju za postizanje maksimalne iskorištenost radnih mjesta uz sustav odobravanja zahtjeva zaposlenika. Djelatnik ima mogućnost podnošenje zahtjeva za rezervaciju i otkazivanje te pregled svojih zahtjeva, dok je administrator zadužen za održavanje podataka i odobravanja zahtjeva.

Aplikacija je postavljena na Internet kako bi se testirala u višekorisničkom okruženju. Pokazala je zadovoljavajuće rezultate i ispunila postavljene ciljeve.

Ključne riječi: ASP.NET Core, C#, dijeljenja radna mjesta, MVC

ABSTRACT

Title: Web Application for Shared Work Desks Reservation

Year 2020/2021. was marked by the COVID-19 pandemic which has affected people's lives and led to major changes in the business world. Many companies, which have been able to adapt their type of business, choose to work from home to preserve the health of their employees. This way of working was positively accepted, which led to the consideration of the idea of introducing a hybrid model of work after the end of the pandemic, i.e. a combination of work from home and in the company.

There are various applications on the market that offer a workplace organization service with the possibility of choosing a work desk, while in this paper the goal was to create an application to achieve maximum utilization of work space and to make a system of approving employee requests. The employee can submit reservation and cancellation requests and review their requests while the administrator is in charge of maintaining the data and approving requests.

The application is set up on the Internet so it can be tested in a multi-user environment. It showed satisfactory results and fulfilled the set goals.

Key words: ASP.NET Core, C#, MVC, shared work desks

ŽIVOTOPIS

Valentina Gotal rođena je 26. travnja 1997. godine u Osijeku, Republika Hrvatska. Prvih šest razreda osnovne škole pohađa u osnovnoj školi Mladost u Osijeku, nakon čega se seli u Bilje i tamo završava svoje osnovnoškolsko obrazovanje u OŠ Bilje. Nakon osnovne škole upisuje III. Gimnaziju, Osijek. 2016. godine dobiva izravan upis na preddiplomski sveučilišni studij elektrotehnike na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija te ga upisuje te iste godine. Na 2. godini preddiplomskog sveučilišnog studija opredjeljuje se za izborni blok Komunikacije i informatika. Preddiplomski sveučilišni studij završava 2019. godine i upisuje diplomski sveučilišni studij Automobilsko računarstvo i komunikacije te potpisuje ugovor o stipendiranju s tvrtkom PBZ. Tijekom studiranja dobiva Priznanje za postignuti uspjeh u studiranju na 2. godini preddiplomskog i diplomskog studija te nagradu Kompot za realizaciju Pametne komode.

PRILOZI

Programski kod je priložen na CD-u uz ovaj rad.