IZRADA WEB APLIKACIJE STUDIOLJEPOTE

Božić, Martina

Master's thesis / Specijalistički diplomski stručni

2021

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

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:228:060393

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-19**

Repository / Repozitorij:

Repository of University Department of Professional Studies

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Specijalistički diplomski stručni studij Informacijske tehnologije

Martina Božić

Z A V R Š N I R A D

Izrada web aplikacije StudioLjepote

Split, rujan 2021.

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Specijalistički diplomski stručni studij Informacijske tehnologije

Predmet: Napredno programiranje web aplikacija otvorenog kôda

Z A V R Š N I R A D

Kandidat: Martina Božić

Naslov rada: Izrada web aplikacije StudioLjepote

Mentor: Marina Rodić, predavač

Split, rujan 2021.

Sadržaj

Sažetak 1

Summary 1

1 Uvod 2

2 Tehnologije 3

3 Izvedba aplikacijske logike 5

3.1 Relacijska baza podataka 5

3.2 Korisničke uloge 8

3.2.1 Neregistrirani korisnik 8

3.2.2 Registrirani korisnik 13

3.2.3 Zaposlenik 28

3.2.4 Administrator 34

4 Konfiguracija 49

5 Zaključak 53

Literatura 54

Sažetak

U ovom završnom radu opisan je razvoj aplikacije StudioLjepote. Dokument počinje opisom korištene tehnologije, zatim opisuje implementaciju relacijske baze podataka i završava opisom funkcionalnosti. Aplikacija StudioLjepote ubrzava proces rezervacije termina u salonu te na jednostavan način omogućava otkazivanje termina. Zaposleniku salona je omogućen pregled rasporeda sa rezerviranim terminima i u slučaju potrebe može otkazati termin. Svaki vlasnik salona vodi brigu o zaradi salona, potražnjom usluga i radu zaposlenika. Prijavom u aplikaciju sve bitne informacije su mu dostupne i prikazane vlasniku. Za izradu web aplikacije studio ljepote korištene tehnologije su: .Net Core, Angular i SQL Express Server.

Ključne riječi: Angular, .Net Core, rezervacija termina, upravljanje salonom.

Summary

Development of web application StudioLjepote

This final paper describes the development of the application StudioLjepote. The beginning of the final paper describes used technologies, the implementation of the relational database and ends with a description of functionalities. The application StudioLjepote simplifies the process of reservation an appointment for the salon. The application also manages the cancelation of appointments. The employee of the salon can keep track of his schedule and has ability to cancel any pending appointment.

Salon owners keep track of the finances of the salon, service availability, and worker's performances. Following the login, the user is presented with all necessary information. Technologies used for the development of the application are .Net Core, Angular and SQL Express Server.

Keywords: Angular, appointment reservation, .Net Core, salon management.

1 Uvod

U današnje vrijeme većina ljudi služi se aplikacijama kako bi se olakšala svakodnevnica. Ljudi se služe aplikacijama za razne stvari poput naručivanja hrane, odjeće i obuće, preuzimanja dokumenata i slično. Za tu svrhu razvijena je aplikacija StudioLjepote za naručivanje u salon koji obavlja frizerske usluge, manikuru, pedikuru i druge.

Prilikom naručivanja u frizerski salon, osoba poziva telefonski broj salona i dogovara termin ovisno o tome kada je frizer(ka) slobodna. U slučaju da osoba mora odgoditi ili otkazati uslugu mora nazvati salon. Cilj aplikacije StudioLjepote je uštedjeti vrijeme frizeru i osobi koja se želi naručiti. Ušteda vremena je postignuta na način da osoba bira frizera/frizerku i lakše uspoređuje kada odabrani zaposlenik ima slobodan termin te kada se to uklapa u raspored osobe koja se želi naručiti. Osoba otkazuje termin jednostavnim klikom što je vidljivo zaposleniku salona. Prednost je što tada zaposlenik ima slobodan termin kojeg druga osoba može odabrati. S druge strane, zaposlenik također može otkazati termin i obavijestiti osobu elektroničkom porukom.

Aplikacija StudioLjepote olakšava posao i administratoru koji ima uvid u zaradu, najtraženije usluge koje se obavljaju i statistiku svakog zaposlenika. Za utvrđivanje efikasnosti radnika bilo je potrebno uvrstiti više kriterija. Uvidom u statistiku vlasniku se olakšava donošenje odluke za dodjelu bonusa i slično.

Nakon uvodnog poglavlja slijedi drugo u kojem su navedene korištene tehnologije i verzije istih za razvoj aplikacije. U trećem poglavlju je objašnjena relacijska baza podataka, korisnici sustava i koje su im opcije omogućene, zajedno s ispisima kôda za navedeno. Četvrto poglavlje objašnjava konfiguraciju aplikacije StudioLjepote, dok posljednje, peto poglavlje sadrži zaključak.

2 Tehnologije

Za razvoj web aplikacije StudioLjepote korišteni su alati: .Net Core verzija 3.1, Angular verzija 11.2 i SQL Express Server.

Platforma .Net Core je besplatna razvojna platforma otvorenog kôda za izradu aplikacija poput web aplikacija, mobilnih aplikacija, igara, strojno učenje i druge. Također .Net Core podržava aplikacija za mnoge operativne sustave. Platforma .Net Core podržava tri programska jezika, C#, F# i Visual Basic, za izradu završnog rada korišten je C# programski jezik.

Unutar .Net Corea omogućeno je spajanje na bazu podataka uz korištenje biblioteke Entity Framework (EF) Core [1]. EF Core omogućava rad s bazom podataka korištenjem deklarativnog kôda Language-integrated query (LINQ). Za razvoj web aplikacija .Net Core nudi već pripremljene predloške. Za izradu završnog rada odabran je predložak .Net Core i Angular. Zbog pregršt biblioteka i ugrađenih funkcionalnosti .Net Core dozvoljava brzu implementaciju poslovne logike. Bitni faktor u odabiru tehnologije je raširenost upotrebe. Raširenost upotrebe dozvoljava brzi pronalazak rješenja ukoliko postoje problemi prilikom razvoja. Kako je riječ o dosta velikom projektu od strane velike korporacije, sve izmjene u razvojnom okviru su dobro dokumentirane i dostupne putem dokumentacije. Dokumentacija također sadrži mnoge informacije vezane za organizaciju kôda, arhitektura i najbolju praksu.

Nužno je razlikovati AngularJs i Angular. AngularJs je prva verzija Angular i redovito se označava s Angular 1.x verzijom, dok je Angular druga inačica i sadrži sve ostale verzije nakon prve [2]. Kod pisanja Angular aplikacije zadana je jasna struktura organizacije kôda naziva Modul. Moduli su najveća organizacijska jedinica aplikacije. Moduli sadrže registraciju svih komponenti i drugih modula u upotrebi aplikacije. Komponente su dijelovi sučelja spojenih u jednu cjelinu i zajedničku funkcionalnost. Komponenta sadrži: klasu u TypeScriptu, HTML predložak i datoteku za kaskadne stilove (CSS). Osim komponenti, Angular sadrži direktive i pipeove kao alate za organizaciju kôda. Direktive su učestale funkcionalnosti koje se koriste tijekom pisanja aplikacije i najčešće se pišu u obliku atributa. Pipeovi su funkcionalnosti koje se upotrebljavaju prilikom interpolacije niza znakova (engl. *string*), funkcioniraju poput funkcija u programskim jezicima [3]. Jedna od prednosti Angulara je ta što je pogonjen predlošcima (engl. *template driven*). U fazi prevo-

đenja Angular aplikacije predlošci se pohranjuju u radnu memoriju što omogućuje brže inicijalno iscrtavanje na ekran. Razvojni okvir Angular funkcionira po principu jedne stranice (engl. *single page application*) po aplikaciji. Single page aplikacije funkcioniraju na način da iscrtavaju cijelu aplikaciju putem jedne datoteke vrste HTML [3]. Aplikacije razvijene na ovaj način nude puno bolje korisničko iskustvo. Jedna od glavnih funkcionalnosti koju svaka web aplikacija treba imati je rutiranje (engl. *routing).* Implementaciju rutiranja Angular podržava uz pomoć modula za rutiranje, što olakšava organizaciju kôda. Razvojni okvir Angular nudi opciju korištenja Angular Material, koja je UI (engl. *user interface*) biblioteka koja je implementacija Material Designa. Material Design je Googleov sustav dizajna kako bi se olakšala izrada aplikacija.

Baza podataka SQL server je Microsoftova implementacija relacijske baze podataka. Relacijske baze podataka su u upotrebi od 1970. zbog jednostavnosti organizacije podataka. Nedostatak relacijskih tipova baza je brzina dohvata podataka kada unos u tablice sadrži velik broj upisanih redaka. S obzirom na očekivanu količinu podataka koja će biti pohranjena tijekom korištenja aplikacije StudioLjepote, relacijska baza podataka je optimalna. Baza podataka SQL Server je odabrana zbog kompatibilnosti s razvojnim okvirom .Net Core i razvojnih alata. Razvojni alat koji omogućava lakšu interakciju s SQL Serverom je Management Studio. Management Studio omogućava laki pregled kreiranih baza podataka i tablica unutar njih. Također sadrži alate za pisanje i analizu upita, manipulaciju tablica, migraciju podataka i generiranje dijagrama baze podataka.

3 Izvedba aplikacijske logike

U ovom poglavlju pojašnjena je relacijska baza podataka, korisničke uloge aplikacije te funkcionalnosti koje svaki korisnik ima. Sučelja koja su prikazana korisnicima prilikom korištenja aplikacije prikazana su slikama zajedno s ispisima kôda.

3.1 Relacijska baza podataka

U nastavku poglavlja pojašnjeni su entiteti, atributi i veze između entiteta korištenih u aplikaciji StudioLjepote. Relacijski dijagram baze podataka prikazan je na Slika 1.

Slika 1 Relacijski dijagram

Razvoj aplikacije kreće od tablice Users, tablica Users služi za spremanje podataka vezanih za korisnika. Tablica korisnik sadrži sve nužne atribute o korisniku sustava, atributi su Email i Password. Za razlikovanje različitih korisničkih uloga u sustavu koristi se atribut Role. Atribut Role je i stupac diskriminator (engl. *discriminator column*) koji služi za mapiranje različitih vrsta objekata unutar poslužitelja. Stupac diskriminator za atribut Role prikazan je u Ispis 1. Apstraktnom klasom User, koja je prikazana u Ispis 2,

implementirana je tehnika tablica po hijerarhiji (engl. *table per hierarchy*), uz pomoć svojstva Role [4].

```
modelBuilder.Entity<User>()
    .HasDiscriminator(u => u.Role)
    .HasValue<Admin>(Enums.UserRole.Admin)
    .HasValue<Customer>(Enums.UserRole.Customer)
    .HasValue<Employee>(Enums.UserRole.Employee);
```

Ispis 1 Stupac diskriminator za atribut Role

U navedenom kôdu definirane su vrijednosti svojstva Role i odgovarajuće klase za mapiranje. Metodom HasDiscriminator definirano je svojstvo koje se promatra. Generičnom metodom HasValue<T> definirana je klasa koja će biti definirana ukoliko svojstvo odgovara vrijednosti navedenom u parametru, tako da za vrijednost Admin mapira se objekt tipa Admin. Ovakvim definiranjem mapiranja, dozvoljena je jednostavnija izvedba dijela sustava za prijavu. Također, omogućeno je spremanje dodatnih atributa i relacije za različite vrste objekata unutar jedne tablice, pod baznom klasom User. Stoga, klase Employee i Customer definirana kôdom mogu se razlikovati po svojstvima i biti spremljene u istu tablicu. Gdje će zaposlenik imati vrijednosti za OIB, DateOfBirth, PhoneNumber, StartOfEmployement i EndOfEmployement, dok će kupac imati vrijednosti za broj ostvarenih bodova. Oboje će imati vrijednosti definirane klasom User koje se koriste prilikom registracije i prijave u sustav.

```
public abstract class User
{
    public int Id { get; set; }
    public string Email { get; set; }
    public string Password { get; set; }
    public UserRole Role { get; set; }
}
public class Employee : User
{
    public string Oib { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public DateTime DateOfBirth { get; set; }
    public string PhoneNumber { get; set; }
    public DateTime StartOfEmployment { get; set; }
    public DateTime? EndOfEmployment { get; set; }
    public ICollection<EmployeeServiceCategory> EmployeeServiceCategories {
get; set; }
    public ICollection<Appointment> Appointments { get; set; }

      }

public class Customer : User
{
    public ICollection<Appointment> Appointments { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public int Points { get; set; }
}
```

Ispis 2 Apstraktna klasa User

Tablicom Service definirane se sve usluge koje su u ponudi unutar salona Studio-Ljepote. Usluge sadrže vrijednosti za naziv, cijenu i vrijeme trajanja. Za grupiranje usluga dodana je tablica ServiceCategories koja služi za stvaranje kategorija usluga. Vezom jedan na prema više između entiteta Service i ServiceCategories ostvaruje se grupiranje usluga po kategorijama. Entitet ServiceCategories sadrži više entiteta tipa Service. Atributom ServiceCategoryId ostvaren je strani ključ na tablicu ServiceCategories. Tablica EmployeeServiceCategories služi kao među tablica relacije Users s tablicom ServiceCategories. EmployeeServiceCategories relacija služi za relaciju više na više između kategorija usluga i zaposlenika. U svrhu spremanja rezervacija za obavljanje usluge korištena je tablica Appointements koja sadrži vrijednosti vezane za početak rezervacije, komentar, ocjenu, cijenu i broj bodova. Jednom rezervacijom u salonu ljepote može se obaviti više različitih usluga. Kako bi podatak o obavljenim uslugama bio spremljen, napravljena je među tablica AppointementsServices. U svrhu spremanja podataka o cijeni u trenutku rezervacije napravljen je atribut Price. Atribut Price je bitan zbog toga što se cijena usluga može mijenjati kroz vrijeme, dok jednom obavljena i plaćena rezervacija neće mijenjati cijenu. Kako bi bilo poznato koji zaposlenik je odgovoran za rezervaciju i koji je kupac rezervirao termin,

kreirane su dvije nove relacije, atribut CustomerId strani ključ kupca rezervacija, dok atribut EmployeeId predstavlja strani ključ zaposlenika koji će pružiti usluge.

3.2 Korisničke uloge

Tijekom razvoja aplikacija, potrebno je obratiti pažnju koja prava imaju određeni korisnici. Potrebno je podijeliti korisnike u određene skupine i dodijeliti im određena prava, unutar aplikacije StudioLjepote korisnici su podijeljeni u sljedeće skupine:

- Neregistrirani korisnik
- Registrirani korisnik
- Zaposlenik
- Administrator.

U sljedećim naslovima opisane su funkcionalnosti aplikacije i mogućnosti za pojedinog korisnika ovisno o ulozi.

3.2.1 Neregistrirani korisnik

Neregistrirani korisnik ima mogućnost pregledati usluge koje salon obavlja kao i njihovu cijenu. Također može vidjeti osnovne informacije o salonu, poput lokacije salona, kontakt informacija i ima mogućnost slanja upita salonu.

Postavljeni upit se šalje na elektroničku adresu salona. Pristup elektroničkoj adresi salona ima administrator koji odgovara na upite. Slika 2 prikazuje formu za unos upita od strane neregistriranog korisnika.

Slika 2 Forma za slanje upita

Sljedeći Ispis 3 kôda prikazuje kôd za slanje upita:

```
guestMessa-
ge(subject: string, email: string, content: string): Observable<any> {

      re-
turn this.httpClient.post<any>("api/Guest/GuestMessage", {subject, email, con
tent});
```

Ispis 3 Zahtjev za slanje upita

Preko aplikacije Angular, šalje se HTTP (engl. *Hypertext Transfer Protocol*) zahtjev koji poslužitelj obrađuje. Prosljeđuje se `subject`, `email` i `content`, odnosno naslov upita, elektronička adresa za odgovor na upit i sami upit, to jest poruka. Na poslužitelju je definirana `GuestMessage` metoda, koja za povratnu vrijednost ima tip `ActionResult`. U ASP.NET Coreu klasa ActionResult predstavlja status kôd protokola HTTP. Metoda `GuestMessage` deklarirana je s dva atributa `AllowAnonymous` i `HttpPost`, metoda je prikazana u Ispis 4. Atribut `AllowAnonymous` dozvoljava slanje upita na metodu bez tokena, dok atribut `HttpPost` predstavljava imenicu (engl. *verb*) protokola HTTP. Unutar `GuestMessage` se poziva metoda `SendMail` koja kao argument ima objekt `Message` sa vrijednostima svojstava (engl. *property*), konfiguracija za elektroničku poštu pojašnjena je u poglavlju 4. Ispis metode `SendMail` je prikazan u Ispis 5.

```
[AllowAnonymous]

[HttpPost(nameof(GuestMessage))]

public ActionResult GuestMessage(ContactFormModel model)

{

    _emaiSender.SendMail(new Message(new List<string> {
_emailConfiguration.From }, model.Subject, $"email: {model.Email}<br> content:
{model.Content}"));

    return Ok();

}
```

Ispis 5 Akcija za slanje upita

```
public void SendMail(Message message)

{

    var emailMessage = CreateEmailMessage(message);

    Send(emailMessage);

}
```

Ispis 4 Pozivi metoda za kreiranje i slanje elektroničke poruke

Metoda `SendMail` sadrži pozive na metodu `CreateEmailMessage`, koja kao argument ima `message` i metodu `Send` koja ima argument `emailMessage`:

Metoda `CreateEmailMessage` prilagođava objekt tipa `Message` u tip `MimeMessage` definiran od strane paketa MimeKit, pojašnjeno u poglavlju 4. Sljedeći Ispis 6 kôda prikazuje implementaciju metode `CreateEmailMessage`:

Metoda `CreateEmailMessage` vraća objekt tipa `MimeMessage`, koji se prosljeđuje kao argument metodi `Send`. Metoda `Send` se spaja na servis Gmail koristeći podatke za autentikaciju, nakon autentikacije metodom `Send` i objektom `mailMessage` šalje se elektronička

```
private MimeMessage CreateEmailMessage(Message message)

{

  var bodyBuilder = new BodyBuilder { HtmlBody = message.Content };

  var emailMessage = new MimeMessage();

  emailMessage.From.Add(new MailboxAddress(_emailConfiguration.From));

  emailMessage.To.AddRange(message.To);

  emailMessage.Subject = message.Subject;

  emailMessage.Body = bodyBuilder.ToMessageBody();

  return emailMessage;
```

Ispis 6 Prilagodba objekta Message u tip MimeMessage

pošta. Cijeli kôd obuhvaćen je try catch blokom u slučaju iznimke, metoda je prikazana u Ispis 7. Konačno, klijent se odspaja od servisa Gmail i oslobađa memoriju.

```
private void Send(MimeMessage mailMessage)

{

    using var client = new SmtpClient();

    try

    {

        client.Connect(_emailConfiguration.SmtpServer,
_emailConfiguration.Port, true);

        client.AuthenticationMechanisms.Remove("XOAUTH2");

        client.Authenticate(_emailConfiguration.Username,
_emailConfiguration.Password);

        client.Send(mailMessage);

    }

    catch

    {

        throw;

    }

    finally

    {

        client.Disconnect(true);

        client.Dispose();

    }

}
```

Ispis 7 Slanje elektroničke poruke

3.2.2 Registrirani korisnik

U svakom trenutku neregistrirani korisnik ima opciju registracije čime dobije veću razinu dozvola u sustavu. Registrirani korisnik ima sva prava kao neregistrirani korisnik, ali uz njih ostvaruje pravo:

- Rezervirati termin
- Otkazati termin
- Pregledati prethodne termine (povijest rezervacija)
- Uvid u stanje bodova
- Pregled nadolazećih termina
- Ocijeniti zaposlenika
- Ostaviti komentar na uslugu i/ili zaposlenika.

U nastavku poglavlja pojašnjena je svaka funkcionalnost koju registrirani korisnik ima, sučelje koje vidi te su prikazani ispisi kôda za svaku funkcionalnost.

Nakon što se korisnik registrira, korisnik vidi sučelje prikazano na Slika 3.

Slika 3 Sučelje novog registriranog korisnika

Klikom na *Rezerviraj termin* korisniku se prikazuje sučelje prikazano na Slika 4. Postupak rezervacije kreće odabirom željene usluge/a, čime se ukupni iznos za odabranu uslugu prikazuje na dnu forme. Nakon odabrane usluge, korisnik odabire željenog zaposlenika, lista zaposlenika prikazuje se prema odabranoj usluzi ili uslugama. Na primjer, korisnik odabere uslugu geliranje noktiju, ta usluga pripada kategoriji manikura i pedikura, lista zaposlenika prikazuje sve zaposlenike koji obavljaju usluge kategorije manikura i pediku-

ra. U slučaju da korisnik odabere više usluga iz različitih kategorija, tada lista zaposlenika sadrži sve zaposlenike koji obavljaju usluge tih kategorija. Izgled sučelja za primjer rezervacije usluga manikure i pedikure prikazano je na Slika 4.

Slika 4 Forma rezervacije termina

Kôd za dohvaćanje svih usluga u salonu je prikazan u Ispis 8.

```
public List<Service> GetAllServices()
{
    return _context
        .Services
        .Include(s => s.ServiceCategory)
        .ToList()
        .Select(s =>
        {
            s.ServiceCategory.Services = null;
            return s;
        })
        .ToList();
}
```

Ispis 8 Dohvaćanje usluga salona

Metoda `GetAllServices` iz baze podataka dohvaća sve usluge, zajedno s kategorijama kojima usluga pripada. Prilikom selektiranja usluga nad `ServiceCategory` navigacijsko svojstvo za usluge je `null`. Razlog tome je izbjegavanje cirkularnih referenci prilikom serijalizacije objekta u oblik pogodan za prijenos putem mreže, oblik pogodan za prijenos je json format.

Sljedeći ispisi kôda prikazuju metode za dohvaćanje zaposlenika za odabrane uslu-

```
getListOfEmployeesForService(services: number[]): Observable<Employee[]> {
  return this.httpClient.post<Employee[]>(
    "api/Employee/GetListOfEmployeesForService",
    services
  );
}
```

Ispis 9 Zahtjev za dohvaćanje zaposlenika

ge.

Aplikacija Angular šalje HTTP zahtjev koji je prikazan u Ispis 9, poslužitelju na daljnju obradu. Na poslužitelju je definira metoda `GetListOfEmployeesForService` s povratnim tipom `ActionResult` i deklariran je atributima `Authorize` i `HttpPost`, prikazanoj u Ispis 10. Atribut Authorize, za razliku od AllowAnonymous ne dozvoljava slanje upita na metodu bez tokena. Prilikom slanja upita na metodu vrši se provjera u kojoj se pomoću

```
[Authorize(Roles = "Customer")]
[HttpPost("GetListOfEmployeesForService")]
public ActionResult GetListOfEmployeesForService(List<int> services)
{
    var emoloyees =
_employeeRepository.GetEmployeeFromServiceCategoryId(services);
    return Ok(emoloyees);
}
```

Ispis 10 Akcija za dohvaćanje zaposlenika

tokena provjerava uloga korisnika, to jest upit poslan od strane registriranog korisnika.

Unutar metode `GetListOfEmployeesForService` poziva se metoda `GetEmployeeFromServiceCategoryId` koja iz baze podataka preko `servicesId` dohvaća

```
public List<Employee> GetEmployeeFromServiceCategoryId(List<int> servicesId)
{
    return _context.Employees
        .Where(e => e.EmployeeServiceCategories.Any(esc =>
esc.ServiceCategory.Services.Any(s => servicesId.Contains(s.Id))))
        .ToList();
}
```

Ispis 11 Dohvaćanje zaposlenika za odabranu kategoriju

zaposlenike koji odrađuju usluge, metoda je prikazana u Ispis 11.

Nakon odabira usluge i zaposlenika, korisnik odabire datum termina. Zatim korisnik treba odabrati vrijeme termina, odabir termina prikazan je na Slika 5.

Slika 5 Odabir datuma termina

Lista termina, odnosno vrijeme termina se formira na osnovu određenih kriterija. Kriteriji za listu termina su: odabrani datum termin, odabrani zaposlenik i vrijeme koliko odabrana usluga/e traje. Aplikacija Angular šalje zahtjev poslužitelju za listu slobodnih

```
getListOfFreeAppointments(
  dateOfAppointment: Date,
  employeeId: number,
  serviceIds: number[]
): Observable<AppointmentModel[]> {
  return this.httpClient.post<AppointmentModel[]>(
    "api/Appointment/GetFreeAppointments",
    { dateOfAppointment, employeeId, serviceIds }
  );
}
```

Ispis 12 Zahtjev za dohvaćanje slobodnih termina

termina, koji je prikazan u Ispis 12.

Na poslužitelju je definirana metoda `GetFreeAppointments` s povratnim tipom `ActionResult` i definiran je s atributima `Authorize` i `HttpPost`. Na Ispis 13 je prikazana metoda

```
[Authorize(Roles = "Customer")]
[HttpPost("GetFreeAppointments")]
public ActionResult GetFreeAppointments(FreeAppointments freeAppointments)
{
    var listOfAppointments =
_appointmentRepository.GetListOfFreeAppointments(freeAppointments.DateOfAppoin
tment, freeAppointments.EmployeeId, freeAppointments.ServiceIds);

    if(listOfAppointments == null)
    {
        return NotFound();
    }
    return Ok(listOfAppointments);
}
```

Ispis 13 Akcija za dohvaćanje slobodnih termina

`GetFreeAppointments`.

Unutar metode `GetFreeAppointments` poziva se metoda `GetListOfFreeAppointments`, koja za argument ima objekt `FreeAppointments`. Metoda `GetListOfFreeAppointments` prikazana je na Ispis 14.

```
public List<AppointmentModel> GetListOfFreeAppointments(DateTime dateOfAppoin-
tment, int employeeId, ICollection<int> serviceIds)

{

    var services = _context.Services.Where(s => serviceI-
ds.Contains(s.Id)).ToList();

    var appointmentsForDay = GetCalendarForEmployee(employeeId, dateOfAppoin-
tment, dateOfAppointment);

    var totalDuration = services.Sum(s => s.Duration.Ticks);

    var startOfShift = new DateTime(dateOfAppointment.Year, dateOfAppoin-
tment.Month, dateOfAppointment.Day, 8, 0, 0);

    var endOfShift = startOfShift.AddHours(8);

    var timeSpanOfAllServices = new TimeSpan(totalDuration);

    var appointments = new List<AppointmentModel>();

    var hoursInRange = endOfShift.Hour - startOfShift.Hour;

    for (var hourItterator = 0; hourItterator < hoursInRange; hourItterator++)

    {

        var startOfProposedHour = startOfShift.AddHours(hourItterator);

        var endOfProposedHour = startOfProposedHo-
ur.Add(timeSpanOfAllServices);

        if (!appointmentsForDay.Any(afd => afd.StartOfAppointment < endOfPro-
posedHour && startOfProposedHour < afd.EndOfAppointment) && startOfProposedHo-
ur >= startOfShift && endOfProposedHour <= endOfShift)

        {

            appointments.Add(new AppointmentModel

            {

                StartOfAppointment = startOfProposedHour,

                EndOfAppointment = endOfProposedHour

            });

        }

    }

    return appointments;

}
```

Ispis 14 Dohvaćanje slobodnih termina

Unutar metode `GetListOfFreeAppointments` prvo se dohvate usluge koje je korisnik odabrao kako bi se izračunalo trajanje odabrane usluge ili više usluga. Vremenski raspon u kojem je formira lista slobodnih termina određeno je pomoću varijabli `startOfShift` i `endOfShift`. Varijabla `appointmentsForDay` sadrži listu termina kada je zaposlenik slobodan te se vrši provjera da li trajanje odabrane usluge/a je dulje od radnog vremena zaposlenika, u slučaju da nije, termin, odnosno vrijeme termina se sprema u listu preko koje korisnik odabire vrijeme termina. Lista termina prikazana je na Slika 6.

Slika 6 Odabir vremena termina

Također tijekom rezervacije korisniku je omogućeno da ostavi napomenu za zaposlenika, ali nije obavezno, što je prikazano na Slika 4.

Za redovite klijente salona, omogućeno je sakupljanje bodova čime korisnici mogu ostvariti popust. Minimalni popust je 10%, a maksimalni 50% te je popust moguće iskoristiti na bilo koju uslugu. Skupljanje bodova zamišljeno je na način da 10 HRK je jednako 1

bod, na primjer, ukoliko korisnik odabere uslugu *Šišanje kratke kose,* cijena te usluge je *70 HRK,* što znači da će korisnik ostvariti 7 bodova. Kako bi korisnik ostvario minimalni popust od 10% to bi značilo da mora ostvariti minimalno 100 bodova (100 bodova = 10%), maksimalan popust od 50% jednak je 500 bodova. Bodovi se dodjeljuju nakon što zaposlenik označi da je odradio traženu uslugu, odnosno termin.

Ukoliko korisnik želi iskoristiti popust, Angular aplikacija šalje zahtjev poslužitelju, zahtjev je prikazan na Ispis 15.

```
getPriceForService(
  serviceIds: number[],
  isDiscount: boolean
): Observable<AppointmentPriceModel> {
  re-
turn this.httpClient.post<AppointmentPriceModel>("api/Appointment/GetPriceForS
ervice", {
    serviceIds,
    isDiscount,
  });
}
```

Ispis 15 Zahtjev za dohvaćanje cijene

Na poslužitelju je definirana metoda `GetPriceForService` s povratinom tipom `ActionResult` te je definirana s atributima `Authorize` i `HttpPost`. Metoda `GetListOfFreeAppointments` prikazana je na Ispis 16.

```
[Authorize(Roles = "Customer")]
[HttpPost("GetPriceForService")]
public ActionResult GetPriceForService(GetPriceModel model)
{
    var priceForService =
_appointmentRepository.GetCostForAppointment(model.ServiceIds, mo-
del.IsDiscount);

    return Ok(priceForService);
}
```

Ispis 16 Akcija za dohvaćanje cijene

Metoda `GetPriceForService` koja je prikazana u Ispis 17, poziva metodu `GetCostForAppointment` koja vrši izračun cijene usluge s popustom te vraća izračun.

```
public AppointmentPriceModel GetCostForAppointment(List<int> serviceIds, bool
isDiscount)
{
    var services = _context.Services.Where(s => serviceIds.Contains(s.Id));
    var totalCost = Convert.ToInt32(services.Sum(c => c.Cost));

    if(isDiscount)
    {
        var user = _context.Customers.Find(_claimProvider.GetUserId());
        if(user.Points >= 500)
        {
            return new AppointmentPriceModel
            {
                Cost = totalCost,
                PointsUsed = 500,
                CostAfterPoints = totalCost - Convert.ToInt32(totalCost * 0.5)
            };
        }
        if(user.Points < 100)
        {
            return new AppointmentPriceModel
            {
                Cost = totalCost
            };
        }
        var pointsRounded = Convert.ToInt32(Math.Floor((decimal)user.Points /
100) * 100);
        var discount = totalCost * ((decimal)pointsRounded / 1000);
        return new AppointmentPriceModel
        {
            Cost = totalCost,
            PointsUsed = pointsRounded,
            CostAfterPoints = totalCost - Convert.ToInt32(discount)
        };
    }
    return new AppointmentPriceModel
    {
        Cost = totalCost
    };
}
```

Ispis 17 Dohvaćanje cijene

Kao što je već navedeno, prilikom rezervacije korisnik može iskoristiti popust i

umanjiti ukupnu cijenu, ukoliko broj bodova nije dovoljan korisniku će biti prikazana odgovarajuća poruka koja je prikazana na Slika 7, u slučaju da korisnik ima dovoljan broj bodova cijena će biti umanjena što je prikazano na Slika 8.

Slika 7 Ne dovoljan broj bodova

Slika 8 Primijenjeni popust na cijenu

Klikom na *Potvrdi* se rezervira termin i isti termin je vidljiv odabranom zaposleniku u kalendaru, o tome je moguće pročitati više u poglavlju 3.2.3.

Korisnik preko sučelja može pratiti nadolazeće termine te detalje istih, prethodne termine, stanje prikupljenih bodova i može otkazati termin. Spomenute akcije i sučelje koje korisnik vidi prikazani su na Slika 9.

Slika 9 Sučelje registriranog korisnika

Otkazivanje termina se provodi klikom na ikonu izbriši *(*engl. *delete)*, ispis kôda za otkazivanje termina:

Aplikacija Angular šalje zahtjev poslužitelju, koji je prikazan u Ispis 18.

```
deleteAppointment(id: number): Observable<any> {
  return this.httpClient.delete<any>(`api/Appointment/${id}`);
}
```

Ispis 18 Zahtjev za otkazivanje termina

Na poslužitelju je definirana metoda `DeleteAppointment` s povratnim tipom `Action-Result` te je definirana s atributima `Authorize` i `HttpDelete`, pomoću atributa `FromRoute` dohvaća se jedinstveni identifikator iz rute zahtjeva. Metoda je prikazana Ispis 19.

```
[Authorize(Roles = "Employee,Customer")]
[HttpDelete("{id}")]
public ActionResult DeleteAppointment([FromRoute] int id)
{
    var appointmentToDelete = _appointmentRepository.DeleteAppointment(id);

    if(!appointmentToDelete)
    {
        return BadRequest();
    }
    return Ok();
}
```

Ispis 19 Akcija za otkazivanje termina

Unutar metode `DeleteAppointment` poziva se metoda `DeleteAppointment` s argumentom `idOfTheAppointmentToDelete`. Unutar metode pomoću `idOfTheAppointmentToDelete` u bazi podataka se pronađe termin te se on izbriše, u slučaju da je termin otkazan od strane zaposlenika odgovarajuća elektronička poruka će biti poslana, ali o tome više u poglavlju 3.2.3, u Ispis 20 kôda prikazano je brisanje termina:

```
public bool DeleteAppointment(int idOfTheAppointmentToDelete)
{
    var appointmentToDelete = _context.Appointments.Include(a =>
a.Employee).Include(a => a.Customer).FirstOrDefault(a => a.Id == idOfTheAppo-
intmentToDelete);

    if (appointmentToDelete == null)
    {
        return false;
    }

    var customer = appointmentToDelete.Customer;
    if (_claimProvider.GetUserId() != appointmentToDelete.CustomerId)
    {
        _emailSender.SendMail(
        new Message(new List<string> { customer.Email },
        "Otkazani termin",
        $"Poštovana/i {appointmentToDelete.Customer.FirstName}  {appointmen-
tToDelete.Customer.LastName}, " +
        $"<br><br>" +
        $"Ovim putem Vas obavještavamo da je Vaš termin u " +
        $"{appointmentToDelete.StartOfService.ToString("dd-MM-yyyy")} " +
        $"kod zaposlenika {appointmentToDelete.Employee.FirstName} otkazan" +
        $"<br><br>" +
        $"Lijep pozdrav," +
        $"<br>StudioLjepote")
        );
    }
    _context.Appointments.Remove(appointmentToDelete);
    _context.SaveChanges();
    return true;
}
```

Ispis 20 Otkazivanje termina

Nakon što zaposlenik označi da je odradio uslugu nakon rezerviranog termina, kori-

snik ima mogućnost ocijeniti zaposlenika i ostaviti komentar. Ocjena i komentari za pojedinog zaposlenika su vidljivi administratoru, o tome više u poglavlju 3.2.4.

Ocjenjivanje zaposlenika se vrši preko tablice *Povijest termina* klikom na ikonu u stupcu *Opcija*. Korisniku se otvara forma u koju unosi ocjenu i komentar, koja je prikazana na Slika 10.

Slika 10 Forma za ocjenu zaposlenika

Preko aplikacije Angular šalje se HTTP zahtjev, kojeg poslužitelj obrađuje. Uz pomoć atributa `FromRoute` dohvaća se jedinstveni identifikator iz rute zahtjeva, dok se iz

```
[Authorize(Roles = "Customer")]
[HttpPost("Rate/{id}")]
public ActionResult Rate([FromRoute] int id, RateModel model)
{
    var isSuccessful = _appointmentRepository.Rate(id, model);

    if (!isSuccessful)
    {
        return NotFound();
    }
    return Ok();
}
```

Ispis 21 Akcija za ocjenjivanje usluge i/ili zaposlenika

tijela zahtjeva dohvaća model koji sadrži ocjenu i komentar. Ispis 21 prikazuje metodu Rate.

Izvučeni jedinstveni identifikator i model prosljeđuje se metodi `Rate` na repozitoriju Appointment. Metoda `Rate` prvo dohvaća odgovarajući Appointment te ukoliko pronađe odgovarajući termin, vrši izmjene na entitetu Appointment i dodaje vrijednosti `Custommer-`

```
public bool Rate(int id, RateModel model)
{
    var appointment = _context.Appointments.Find(id);

    if (appointment == null)
    {
        return false;
    }
    appointment.CustomerComment = model.CustomerComment;
    appointment.CustomerRating = model.CustomerRating;

    _context.SaveChanges();
    return true;
}
```

Ispis 22 Ocjenjivanje usluge i/ili zaposlenika

`Comment` i `CustomerRating` za termin. Metoda `Rate` prikazana je na Ispis 22.

3.2.3 Zaposlenik

Korisnik uloge zaposlenik je osoba zaposlena u salonu StudioLjepote. Zaposlenik ima mogućnost:

- Pregledati vlastiti raspored
- Otkazati nadolazeći termin
- Označiti termin da je gotov

Prilikom zaposlenja administrator ispunjava podatke o zaposleniku, forma koju administrator ispunjava prikazana je na Slika 11.

Slika 11 Forma za dodavanje novog zaposlenika

Administrator ispunjava *Email,* to jest elektroničku adresu na koju će se poslati poruka zaposleniku za postavljanje lozinke te kojom će se zaposlenik služiti kako bi se prijavio u aplikaciju. O dodavanju novog zaposlenika, moguće je pročitati u poglavlju 3.2.4.

Nakon što administrator doda novog zaposlenika, na elektroničku adresu se pošalje poveznica kako bi zaposlenik postavio lozinku za svoj korisnički račun, primjer poruke prikazan je na Slika 12.

Slika 12 Elektronički poziv novom zaposleniku

Klikom na poveznicu zaposleniku se otvara forma koja je prikazana na Slika 13 gdje unosi svoju lozinku. Nakon postavljene lozinke i klikom na *Postavi lozinku,* korisnika se preusmjerava na prijavu. Nakon same prijave, korisniku vidi sučelje koje je prikazano na Slika 14.

Slika 13 Postavljanje lozinke

Slika 14 Sučelje novog zaposlenika

Zaposlenik preko sučelja ima pregled rasporeda i termina. Na Ispis 23 kôda prika-

```
getEmployeeCalendar(
       startDate: Date,
       endDate: Date
): Observable<AppointmentModel[]> {
       return this.httpClient.post<AppointmentModel[]>(
          "api/Employee/GetEmployeeCalendar",
          { startDate, endDate }
       );
```

Ispis 23 Zahtjev za dohvaćanje rasporeda

zano je dohvaćanje rasporeda zaposlenika:

Angular aplikacija šalje zahtjev poslužitelju, a unutar tijela zahtjeva prosljeđuju su `startDate` i `endDate`, koje sadrže vrijednost početnog i krajnjeg datuma za kojeg se zahtjeva pregled termina unutar kalendara.

Na poslužitelju je definirana metoda `GetEmployeeCalendar`, prikazana je na Ispis 24 s povratnim tipom `ActionResult` te je definirana s atributima `Authorize` i `HttpDelete`. Metoda `GetEmployeeCalendar` dohvaća trenutnog korisnika, odnosno zaposlenika koji je prijavljen te poziva metodu `GetCalendarForEmployee` koja vraća listu termina. Ispis 25 prikazuje metodu `GetCalendarForEmployee`.

```
public ICollection<AppointmentModel> GetCalendarForEmployee(int employeeId,
DateTime startDate, DateTime endDate)
{
    var appointments = _context
        .Appointments
        .Include(a => a.Customer)
        .Include(a => a.AppointmentServices)
        .ThenInclude(x => x.Service)
        .Where(ae => ae.EmployeeId == employeeId && ae.StartOfService.Date >=
startDate.Date && ae.StartOfService.Date <= endDate.Date)
        .ToList();
    var currentEmployeeId = _claimProvider.GetUserId();
    return appointments.Select(a => new AppointmentModel
    {
        Id = a.Id,
        StartOfAppointment = a.StartOfService,
        EndOfAppointment =
a.StartOfService.AddTicks(a.AppointmentServices.Sum(ap =>
ap.Service.Duration.Ticks)),
        Comment = a.Comment,
        CustomerEmail = a.Customer.Email,
        NameOfServices = a.AppointmentServices.Select(sa =>
sa.Service.Name).ToList(),
        IsDone = a.IsDone
```

Ispis 24 Akcija za dohvaćanje rasporeda zaposlenika

```
[Authorize(Roles = "Employee")]
[HttpPost("GetEmployeeCalendar")]
public ActionResult GetEmployeeCalendar(StartEndDateModel model)
{
    var appointments =
_appointmentRepository.GetCalendarForEmployee(currentEmployeeId, mo-
del.StartDate, model.EndDate);
    if(appointments == null)
    {
        return BadRequest();
    }
    return Ok(appointments);
```

Ispis 25 Dohvaćanje rasporeda zaposlenika

Zaposlenik ima mogućnost otkazivanja termina što je prikazano Ispis 26 kôda:

```
public bool (int idOfTheAppointmentToDelete)
{
    var appointmentToDelete = _context.Appointments.Include(a =>
a.Employee).Include(a => a.Customer).FirstOrDefault(a => a.Id == idOfTheAppo-
intmentToDelete);

    if (appointmentToDelete == null)
    {
        return false;
    }

    var customer = appointmentToDelete.Customer;
    if (_claimProvider.GetUserId() != appointmentToDelete.CustomerId)
    {
        _emailSender.SendMail(
        new Message(new List<string> { customer.Email },
        "Otkazani termin",
        $"Poštovana/i {appointmentToDelete.Customer.FirstName}  {appointmen-
tToDelete.Customer.LastName}, " +
        $"<br><br>" +
        $"Ovim putem Vas obavještavamo da je Vaš termin u " +
        $"{appointmentToDelete.StartOfService.ToString("dd-MM-yyyy")} " +
        $"kod zaposlenika {appointmentToDelete.Employee.FirstName} otkazan" +
        $"<br><br>" +
        $"Lijep pozdrav," +
        $"<br>StudioLjepote")
        );
    }
    _context.Appointments.Remove(appointmentToDelete);
    _context.SaveChanges();
    return true;
}
```

Ispis 26 Elektronička poruka u slučaju da zaposlenik otkazuje termin

Unutar metode `DeleteAppointment` vrši se provjera korisnika te se šalje elektronič-

ka poruka korisniku da je termin otkazan, primjer poruke prikazan je na Slika 15.

Slika 15 Elektronička poruka za otkazivanje termina

Nakon što je odradio uslugu, zaposlenik termin označi kao gotov. Bodovi koje je korisnik skupio dodaju se njegovom ukupnom broju bodova. Osim toga, korisniku se nudi mogućnost ocjenjivanja zaposlenika i/ili usluge.

```
[Authorize(Roles = "Employee")]
[HttpPatch("{id}/SetDone")]
public ActionResult SetDone([FromRoute] int id)
{
    var isSuccessful = _appointmentRepository.ConfirmDone(id);

    if(!isSuccessful)
    {
        return BadRequest();
    }
    return Ok();
}
```

Ispis 27 Akcija za označavanje završetka termina

Na poslužitelju je definirana metoda `SetDone` s povratnim tipom `ActionResult` definirana je s atributima `Authorize` i `HttpDelete`, pomoću atributa `FromRoute` dohvaća se jedinstveni identifikator iz rute zahtjeva. Ispis 27 prikazuje metodu `SetDone` . Unutar metode `SetDone`, poziva se metoda `ConfirmDone` s argumentom `id`. Unutar `ConfirmDone` vrši se računanje bodova i termin se označava kao gotov, što je prikazano Ispis 28 kôda:

```
public bool ConfirmDone(int id)
{
    var appointment = _context.Appointments
        .Include(a => a.AppointmentServices)
        .ThenInclude(sa => sa.Service)
        .Include(a => a.Customer)
        .FirstOrDefault(a => a.Id == id && !a.IsDone);

    if (appointment == null)
    {
        return false;
    }

    var totalPointsGained = appointment.AppointmentServices.Sum(sa => Con-
vert.ToInt32(sa.Service.Cost / 10));
    appointment.Customer.Points += totalPointsGained;

    appointment.IsDone = true;
    _context.SaveChanges();
    return true;
}
```

Ispis 28 Označavanje završetka termina

3.2.4 Administrator

Administrator upravlja aplikacijom StudioLjepote, a to podrazumijeva:

- Upravljanje kategorijama
- Upravljanje uslugama
- Upravljanje zaposlenicima
- Uvid u zaradu
- Uvid u najtraženije usluge
- Pregled statistike za svakog zaposlenika
- Pregled kalendara svih zaposlenika

Nakon što se administrator prijavi u aplikaciju, sučelje koje vidi prikazano je na Slika 16.

Slika 16 Administratorsko sučelje

Prijavom u sustav administrator ima uvid u statistiku salona StudioLjepote. Prvi dijagram je statistika zarade salona, omogućen je pregled za dnevnu, mjesečnu, kvartalnu i godišnju zaradu. Osim dijagrama za zaradu, administrator ima uvid u dijagram za top tri najtraženije usluge dnevno, kvartalno, mjesečno i godišnje. Također, pomoću dijagrama je prikazana statistika za svakog zaposlenika StudioLjepote.

Klikom na izbornik s lijeve strane, administrator može upravljati kategorijama, uslugama, zaposlenicima ili pregledati raspored svih zaposlenika. Upravljanje kategorijama odnosi se na dodavanje novih kategorija, brisanja, uređivanje i pregled. Klikom na *Kategorija,* otvara se sučelje koje je prikazano Slika 17.

Slika 17 Administratorsko sučelje za upravljanje kategorijama

Tablica dohvaća sve kategorije unutar baze podataka, u Ispis 29 kôda prikazano je dohvaćanje kategorija:

```
[Authorize(Roles = "Admin")]
[HttpGet(nameof(Categories))]
public ActionResult<List<ServiceCategory>> Categories()
{
    var categoryList = _serviceCategoryRepository.GetAllCategories();

    return Ok(categoryList);
}

       Metoda ListOfServices tipa ActionResult poziva metodu GetAllCategories

public List<ServiceCategory> GetAllCategories()
{
    return _context.ServiceCategories.ToList();
}
```

Ispis 29 Akcija za dohvaćanje kategorija

```
deleteCategory(serviceCategoryId: number) : Observable<any>{

          re-
turn this.httpClient.delete<any>(`api/ServiceCategory/${serviceCategoryId}`);

}
```

Ispis 30 Zahtjev za brisanje kategorije

Unutar tablice pod stupcem Opcije administrator može izmijeniti kategoriju ili obrisati. Klikom na ikonu *delete* kategorija se briše, što je prikazano Ispis 30 kôda:

Aplikacija Angular šalje `id` kategorije na poslužitelj, metoda `Delete` tipa `Action-Result` iz rute dohvaća jedinstveni identifikator kategorije i poziva metodu `DeleteServiceCategory`. Ispis 31 prikazuje metodu `Delete`.

```
[Authorize(Roles = "Admin")]
[HttpDelete("{id}")]
public ActionResult Delete([FromRoute] int id)
{
    var isServiceCategoryDeleted =
_serviceCategoryRepository.DeleteServiceCategory(id);

    if (isServiceCategoryDeleted)
    {
        return Ok();
    }
    return BadRequest("Neuspješno brisanje");
}
```

Ispis 31 Akcija za brisanje kategorije

```
public bool DeleteServiceCategory(int idOfServiceCategoryToDelete)
{
    var serviceCategoryToDelete =
_context.ServiceCategories.Find(idOfServiceCategoryToDelete);

    if (serviceCategoryToDelete == null) return false;

    _context.Remove(serviceCategoryToDelete);
    _context.SaveChanges();

    return true;
}
```

Ispis 32 Brisanje kategorije

Metoda `DeleteServiceCategory` pretražuje bazu podataka i traži kategoriju s jedinstvenim identifikatorom koji je odgovara identifikatoru iz argumenta. Ako ne postoji kategorija u bazi podataka s identifikatorom metoda vraća *false*, inače se kategorija briše iz baze podataka i vraća *true* i kategorija se briše iz baze podataka, opisano je prikazano sljedećim Ispis 32.

Izmjena kategorije se odvija na sličan način, uz jedinstveni identifikator prosljeđuje se i objekt tipa `ServiceCategory`, što je prikazano u Ispis 33 kôda:

```
edit(serviceCategoryId: number, serviceCategory: ServiceCategory): Observable<
any>{

    re-
turn this.httpClient.put<any>(`api/ServiceCategory/${serviceCategoryId}`, serv
iceCategory);

}
```

Ispis 33 Akcija za uređivanje kategorije

Poslužitelj iz rute dohvaća jedinstveni identifikator koji se prosljeđuje metodi `EditServiceCategory` zajedno s objektom `serviceCategory`. Metoda `Edit` prikazana je na Ispis 34.

```
[Authorize(Roles = "Admin")]
[HttpPut("{id}")]
public ActionResult Edit([FromRoute] int id, ServiceCategory serviceCategory)
{
    var isEditSuccessful = _serviceCategoryRepository.EditServiceCategory(id,
serviceCategory);

    if(isEditSuccessful)
    {
        return Ok();
    }

    return BadRequest("Kategorija već postoji");

}
```

Ispis 34 Akcija za uređivanje kategorije

```
public bool EditServiceCategory(int id, ServiceCategory editedServiceCategory)
{
    var doesServiceCategoryExist =
_context.ServiceCategories.Any(serviceCategory =>
        serviceCategory.Name.ToLower() ==
editedServiceCategory.Name.ToLower());

    if (doesServiceCategoryExist) return false;

    var serviceCategoryToEdit = _context.ServiceCategories.Find(id);
    if (serviceCategoryToEdit == null) return false;

    serviceCategoryToEdit.Name = editedServiceCategory.Name;
    serviceCategoryToEdit.About = editedServiceCategory.About;
    _context.SaveChanges();

    return true;
}
```

Ispis 35 Uređivanje kategorije

```
addCategory(serviceCategory: ServiceCategory) : Observable<any>{

    return this.httpClient.post<any>('api/ServiceCategory', serviceCategory);

}
```

Ispis 36 Zahtjev za dodavanje nove kategorije

Vrši se provjera da li u bazi podataka postoji kategorija s jedinstvenim identifikatorom koja odgovara identifikatoru iz argumenta, ako ne postoji metoda vraća *false*, u slučaju da pronađe podudaranje vrši se izmjena nad kategorijom, opisano je prikazano Ispis 35.

Klikom na *Dodaj* administrator može unijeti novu kategoriju u bazu podataka, aplikacija Angular šalje poslužitelju objekt tipa `ServiceCategory`, što je prikazano u Ispis 36.

Na poslužitelju je definirana metoda `AddCategories` koja je prikazana na Ispis 37.

```
[Authorize(Roles = "Admin")]
[HttpPost]
public ActionResult AddCategories(ServiceCategory serviceCategory)
{
    var isServiceCategoryAdded =
_serviceCategoryRepository.AddServiceCategory(serviceCategory);

    if (isServiceCategoryAdded)
    {
        return Ok();
    }
    return BadRequest("Neuspješni unos");
}
```

Ispis 37 Akcija za dodavanje nove kategorije

```
public bool AddServiceCategory(ServiceCategory serviceCategoryToAdd)
{
    var doesServiceCategoryExist =
_context.ServiceCategories.Any(serviceCategory =>
        serviceCategory.Name.ToLower() ==
serviceCategoryToAdd.Name.ToLower());

    if (doesServiceCategoryExist) return false;
    _context.ServiceCategories.Add(serviceCategoryToAdd);
    _context.SaveChanges();

    return true;
}
```

Ispis 38 Dodavanje nove kategorije

Unutar metode `AddCategories` poziva se metoda `AddServiceCategory`

Radi se provjera postoji li kategorija s istim nazivom kategorije koja je proslijeđena kao argument metode `AddServiceCategory`, u slučaju da ne postoji, kategorija s nazivom i opisom se sprema u bazu podataka te metoda vraća *true*, u suprotnom *false*. Opisano je prikazano Ispis 38.

Administrator također može upravljati uslugama koje StudioLjepote odrađuje, klikom na Usluge iz izbornika, administrator vidi tablicu prikazanu na Slika 18.

Slika 18 Administratorsko sučelje za upravljanje uslugama

Tablica sadrži sve usluge koje nalaze u bazi podataka, dohvaćanje svih usluga odvija se na način:

```
[AllowAnonymous]
[HttpGet(nameof(Services))]
public ActionResult<List<Service>> Services()
{
    var servicesList = _serviceRepository.GetAllServices();

    return Ok(servicesList);
}
```

Ispis 39 Akcija za dohvaćanje usluga

metoda `Services` tipa `ActionResult`, koja je prikazana u Ispis 39, poziva metodu `GetAllServices` koja je prikazana u Ispis 40.

```
public List<Service> GetAllServices()
{
     return _context
         .Services
         .Include(s => s.ServiceCategory)
        .ToList()
        .Select(s =>
        {
            s.ServiceCategory.Services = null;
            return s;
        })
     .ToList();
}
```

Ispis 40 Dohvaćanje usluga

```
[Authorize(Roles = "Admin")]
[HttpDelete("{id}")]
public ActionResult Delete([FromRoute] int id)
{
    var isServiceDeleted = _serviceRepository.DeleteService(id);

    if (isServiceDeleted)
    {
        return Ok();
    }
    return BadRequest("Neuspješno brisanje");
}
```

Ispis 41 Akcija za brisanje usluge

Pojašnjeno u poglavlju 3.2.2. Kao i kategorije, preko tablice administrator može izbrisati uslugu ili promijeniti. Kao i brisanje kategorije, poslužitelj iz rute dohvaća jedinstveni identifikator, traži uslugu u bazi podataka s istim jedinstvenim identifikatorom u slučaju da ne postoji vraća *false*, u suprotnom briše uslugu i vraća *true*. Ispisi kôda za opisano su Ispis 41 i Ispis 42:

```
public bool DeleteService(int idOfTheServiceToDelete)
{
    var serviceToDelete = _context.Services.Find(idOfTheServiceToDelete);

    if (serviceToDelete == null) return false;

    _context.Remove(serviceToDelete);
    _context.SaveChanges();

    return true;
}
```

Ispis 42 Brisanje usluge

Izmjena usluge se odvija kao i izmjena kategorije, ispisi kôda za izmjenu su Ispis

```
[Authorize(Roles = "Admin")]
[HttpPut("{id}")]
public ActionResult Edit([FromRoute] int id, Service service)
{
    var isEditSuccessful = _serviceRepository.EditedService(id, service);

    if (isEditSuccessful)
    {
        return Ok();
    }

    return BadRequest("Usluga već postoji");

}
```

Ispis 43 Akcija za uređivanje usluge

43 i Ispis 44:

```
public bool EditedService(int id, Service editedService)
{
    var doesServiceExist = _context.Services.Any(service =>
        service.Name.ToLower() == editedService.Name.ToLower() && service.Id
!= id);

    if (doesServiceExist) return false;

    var serviceToEdit = _context.Services.Find(id);
    if (serviceToEdit == null) return false;

    serviceToEdit.Name = editedService.Name;
    serviceToEdit.Cost = editedService.Cost;
    serviceToEdit.Duration = editedService.Duration;
    serviceToEdit.ServiceCategoryId = editedService.ServiceCategoryId;

    _context.SaveChanges();

    return true;
}
```

Ispis 44 Uređivanje usluge

Klikom na *Dodaj* administrator dodaje novu uslugu, sučelje je prikazano na Slika 19.

Slika 19 Forma za unos nove usluge

Administrator unosi naziv usluge, cijenu, trajanje usluge i kojoj kategoriji pripada, lista kategorija se dohvaća iz baze podataka. Ispis 45 kôda prikazano je dodavanje nove usluge:

```
addService(service: Service): Observable<any> {
    return this.httpClient.post<any>("api/Service", service);
}
```

Ispis 45 Zahtjev za dodavanje usluge

Na poslužitelju je definirana metoda `AddServices` tipa `ActionResult`, koja je prikazana Ispis 46.

```
[Authorize(Roles = "Admin")]
[HttpPost]
public ActionResult AddServices(Service service)
{
    var isServiceAdded = _serviceRepository.AddService(service);

    if (isServiceAdded)
    {
        return Ok();
    }
    return BadRequest("Neuspješni unos");
}
```

Ispis 46 Akcija za dodavanje usluge

Koja poziva metodu `AddService`, prikazanoj u Ispis 47.

```
public bool AddService(Service serviceToAdd)
{
    var doesServiceExist = _context.Services.Any(service =>
        service.Name.ToLower() == serviceToAdd.Name.ToLower());

    if (doesServiceExist) return false;

    _context.Services.Add(serviceToAdd);
    _context.SaveChanges();

    return true;
}
```

Ispis 47 Dodavanje usluge

Unutar metode `AddService` je li u bazi podataka već postoji usluga s imenom koju administrator želi unijeti. U slučaju da postoji metoda `AddService` vraća *false*, u suprotnom se usluga sprema u bazu podataka.

Kao što je već navedeno administrator ima mogućnost upravljanja zaposlenicima. Iz baze podataka se dohvaća lista zaposlenika i prikazuje administratoru u obliku tablice, prikazanoj na Slika 20.

Slika 20 Administratorsko sučelje za upravljanje zaposlenicima

Kao što upravlja kategorijama i uslugama, administrator preko tablice može izbrisati ili izmijeniti podatke o zaposleniku. Sučelje koje administrator vidi prilikom izmjene zaposlenika prikazano je na Slika 21.

Slika 21 Forma za uređivanje informacija o zaposleniku

Unosom datuma u polje *Kraj radnog odnosa* zaposlenik ostaje u bazi, ali se ne prikazuje korisniku kao mogućnost odabira prilikom rezervacije termina za uslugu. Dodavanje novog zaposlenika izvršava se na isti način kao i dodavanje nove kategorije i usluge prethodno objašnjene, primjer sučelja je prikazan na Slika 11.

Administrator ima uvid u sve termine svih zaposlenika, ali nema prava otkazivanja. Primjer pregleda prikazan je na Slika 22.

Slika 22 Pregled svih termina

Klikom na određeni termin, administrator ima uvid u detalje zakazanog termina, primjer detalja termina je prikazan na Slika 23.

Slika 23 Detalji termina

Ispis 48, Ispis 49 i Ispis 50 u nastavku, su ispisi kôda za dohvaćanje svih termina.

```
getCalendarForEmployees(
        startDate: Date,
        endDate: Date
      ): Observable<CustomerAppointmentModel[]> {
        return this.httpClient.post<CustomerAppointmentModel[]>(
          "api/Employee/GetAllEmployeeCalendar",
          { startDate, endDate }
        );
}
```

Ispis 48 Zahtjev za dohvaćanje rasporeda svih zaposlenika

```
[Authorize(Roles = "Admin")]
[HttpPost("GetAllEmployeeCalendar")]
public ActionResult GetAllEmployeeCalendar(StartEndDateModel model)
{
    var appointments =
_appointmentRepository.GetCalendarForEmployees(model.StartDate, mo-
del.EndDate);

    if (appointments == null)
    {
        return BadRequest();
    }
    return Ok(appointments);

}
```

Ispis 49 Akcija za dohvaćanje rasporeda svih zaposlenika

```
public ICollection<CustomerAppointmentModel> GetCalendarForEmployees(DateTime
startDate, DateTime endDate)
{
    var appointments = _context
        .Appointments
        .Include(a => a.Customer)
        .Include(a => a.AppointmentServices)
        .ThenInclude(x => x.Service)
        .Include(a => a.Employee)
        .Where(ae => ae.StartOfService.Date >= startDate.Date &&
ae.StartOfService.Date <= endDate.Date)
        .ToList();

    return appointments.Select(a => new CustomerAppointmentModel
    {
        Id = a.Id,
        StartOfAppointment = a.StartOfService,
        EndOfAppointment =
a.StartOfService.AddTicks(a.AppointmentServices.Sum(ap =>
ap.Service.Duration.Ticks)),
        Comment = a.Comment,
        CustomerEmail = a.Customer.Email,
        NameOfServices = a.AppointmentServices.Select(sa =>
sa.Service.Name).ToList(),
        IsDone = a.IsDone,
        CustomerComment = a.CustomerComment,
        CustomerRating = a.CustomerRating,
        EmployeeFullName = $"{a.Employee.FirstName} {a.Employee.LastName}",
        Price = a.Price,
        PointsUsed = a.PointUsed
    }).ToList();
}
```

Ispis 50 Dohvaćanje rasporeda svih zaposlenika

4 Konfiguracija

U svrhu lakšeg konfiguriranja aplikacije razvojni okvir .Net Core omogućava dinamično postavljanje varijabli van razvojnog okruženja. Kako bi vrijednosti koje će se koristiti prilikom izvršavanja bile dostupne korištena je datoteka appsettings.json. Unutar datoteke dostupan je jedan objekt zapisan u JSON formatu. Dodavanje vrijednosti u konfiguraciju izvršava se uz pomoć postavljanja vrijednosti svojstava objekta. Sva namještana svojstva su dostupna prilikom pokretanja projekta u objektu IConfiguration. Zadane vrijednosti unutar konfiguracije dohvaćaju se uz pomoć metoda dostupnih u objektu IConfiguration. Koristeći metodu `GetSection` moguće je dohvatiti vrijednost iz konfiguracije. Nakon dohvaćanja vrijednosti moguće je registrirati u .Net Core kontajner za inverziju kontrole uz pomoć metode Configure<T>. Gdje je T tip objekta koji se registrira, a vrijednost argumenta je vrijednost namještene konfiguracije. Nakon registracije vrijednosti u kontajner, ona postaje dostupna za korištenje unutar konstruktora diljem aplikacije. Prilikom korištenja vrijednosti upotrebljava se IOptions<T> interfejs koji sadrži vrijednost konfiguracije. Odabrane sekcije za konfiguraciju su: ConnectionStrings, JwtConfiguration i Email. Vrijednosti spremljene u appsettings.json su skrivene u slučaju dekompajliranja izvornog kôda poslužitelja.

Sekcija ConnectionStrings je predefinirana vrijednost za organizaciju stringova za spajanje na različite baze podataka. Usluga koja se konfigurira u ovom slučaju je baza podataka SQL Server. Konfigurira se na način da se koriste parovi ključ vrijednost (engl. *key value pair*). Za pristup vrijednosti unutar ove sekcije koristi se metoda `GetConnectionString` nad objektom tipa `IConfiguration`. Korištenje za registraciju spoja na bazu za SQL Server unutar projekta što je prikazano Ispis 51:

```
services.AddDbContext<BeautyDbContext>(options => op-
tions.UseSqlServer(Configuration.GetConnectionString("BeautyDbContext")));
```

Ispis 51 Konfiguracija DBConxtext

Sekcija JWTConfiguration koja je prikazana u Ispis 52, koristi se prilikom namještanja enkripcijskih parametara tokena JWT. Vrijednost `Issuer` govori koja je aplikacija odgovorna za kreaciju tokena. Vrijednost `AudienceId` je jedinstveni identifikator aplikacije za koju je izdan token. Vrijednost `AudienceSecret` je tajni ključ koji se koristi za enkripciju potpisa (engl. *signature*) JWT tokena. Jako je bitno da `AuidenceSecret` nije javno dostu-

pna informacija, razlog tome je što se uz pomoć njega validira token. Vrijednost

```
"JwtConfiguration": {
    "Issuer": "beauty",
    "AudienceId": "beauty-audience",
    "AudienceSecret":
"735035B417D1BFE4F4B426A590CD31EF8966A9E714B5BB1B683D939B53A66C77",
    "ExpiryMinutes": 9999
}
```

Ispis 52 JWT parametri za konfiguraciju

`ExpireyMinutes` predstavlja vrijeme u minutama o duljini valjanosti tokena.

Dohvaćene vrijednosti se dodjeljuju ugrađenoj funkcionalnosti za autorizaciju na

```
services
    .AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
    {
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidIssuer = jwtConfiguration.Issuer,
            ValidateAudience = true,
            ValidAudience = jwtConfiguration.AudienceId,
            ValidateIssuerSigningKey = true,
            IssuerSigningKey = new SymmetricSecuri-
tyKey(jwtConfiguration.GetAudienceSecretBytes())
        };
});
```

Ispis 53 Konfiguracija ugrađene autorizacije u .Net Core okruženju

poslužitelju koja je prikazana u Ispis 53.

Za kreaciju tokena napravljen je odvojen servis `JwtService`, koji je prikazan u Ispis 54. Uz pomoć biblioteke Jose.JWT koji sadrži implementaciju najkorištenijih enkripcijskih

```
private static DateTime GetUtcBaseDateTime => new DateTime(1970, 1, 1, 0, 0,
0, DateTimeKind.Utc);
private static double GetCurrentSeconds => (DateTime.UtcNow -
GetUtcBaseDateTime).TotalSeconds;

public const JwsAlgorithm Algorithm = JwsAlgorithm.HS256;

private readonly JwtConfiguration _jwtConfiguration;
private readonly IUserRepository _userRepository;

public JwtService(IOptions<JwtConfiguration> jwtConfigurationOptions,
                  IUserRepository userRepository)
{
       _jwtConfiguration = jwtConfigurationOptions.Value;
       _userRepository = userRepository;
}
public string GetJwtTokenForUser(User user)
{
       var expiresAt = GetCurrentSeconds + _jwtConfiguration.ExpiryMinutes *
60;
       var payload = new Dictionary<string, string>
       {
           {"iss", _jwtConfiguration.Issuer},
           {"aud", _jwtConfiguration.AudienceId},
           {"exp", expiresAt.ToString(CultureInfo.InvariantCulture)},
           {RoleClaim.UserId, user.Id.ToString()},
           {RoleClaim.Role, user.Role.ToString()}
       };
       return JWT.Encode(payload, _jwtConfiguration.GetAudienceSecretBytes(),
Algorithm);
}
```

Ispis 54 Metoda za kreaciju JWT tokena

algoritama izrađuje se token.

Metoda `GetJwtTokenForUser` je odgovorna za stvaranje tokena koristeći vrijednosti na objektu tipa `User`. Metoda `Encode` iz biblioteke Jose.Jwt prima vrijednosti za enkripciju u obliku rječnika (engl. *dictionary*). Vrijednosti `iss`, `aud` i `exp` su definirane standardom RFC7519 [5] od strane istraživačkog tijela za internet inženjerstvo. Vrijednosti `UserId` i `Role` su dodatno dodane u svrhu implementacije poslovne logike poslužitelja. Drugi parametar metode `Encode` je tajni ključ u obliku bajtova. Treći parametar je tip algoritma kojeg želimo odabrati za enkripciju tokena, odabran je `HS-256`. HS-256 odabran je zbog toga što koristi jednu vrijednost za enkripciju i dekripciju tokena uz pomoć jedne vrijednost. Kako je generiranje i validacija tokena dužnost poslužitelja algoritam odgovara aplikaciji.

U svrhu slanja elektroničke pošte korisnicima odabran je paket NETCore.MailKit, koji sadrži implementaciju za slanje poruka putem protokola SMTP. Odabrani servis za

```
"Email": {
    "From": "ljepotastudiousluge@gmail.com",
    "SmtpServer": "smtp.gmail.com",
    "Port": 465,
    "Username": "ljepotastudiousluge@gmail.com",
```

Ispis 55 Email konfiguracija

slanje poruka je Gmail, zbog pristupačnosti u procesu stvaranja računa i pregleda elektroničke pošte. Za spajanje na elektroničku poštu korištena je konfiguracija Email, koja je prikazana u Ispis 55.

Prilikom korištenja servisa EmailSender pojašnjenog u poglavlju 3.2.1 upotrebljavaju se navedene vrijednosti unutar metode `Send`.

5 Zaključak

Prilikom izrade završnog rada odabrane tehnologije omogućile su jednostavan razvoj aplikacije. Za razvoj klijentske strane razvojni okvir Angular je pružio potporu pri organizaciji kôda i izvedbi većine aplikacijske logike. Korištenjem paketa Angular Material koji je omogućen od strane Angulara korisnička sučelja je bilo lako organizirati i sve potrebne informacije prikazati krajnjem korisniku. Prilikom unosa informacija u formu, Angular ima ugrađenu funkcionalnost za validaciju forme što je bilo korisno. Korištenjem ugrađene validacije smanjena je potreba za dodatnim kôdom prilikom validacije svojstava forme. Na poslužitelju je korišten EntityFramework i LINQ pomoću kojih su se na jednostavan način izvršavali SQL upiti na bazu podataka i promjene nad podacima. EntityFramework i LINQ su paketi unutar .NET Corea koji su preuzeti preko upravitelja paketa Nuget. S obzirom da je korišten .NET Core koji je razvijen od Microsofta dokumentacija je opširna i detaljno objašnjena, što je je od velike pomoći prilikom razvoja aplikacije.

Prilikom izrade planiranih funkcionalnosti uočeni su nedostatci. Aplikacija bi se mogla proširiti na način da zaposlenik ima mogućnost zatražiti godišnji odmor koji bi administrator mogao prihvatiti ili odbiti. Također, aplikacija se može nadograditi da podržava upravljanje više različitih salona sa više administratora gdje bi svaki salon imao svog administratora. Na taj način bi se omogućila jedinstvena platforma za salone ljepote.

Literatura

[1] Microsoft documentation, „.Net Core“, https://docs.microsoft.com/en-us/dotnet/core/introduction (posjećeno 1.5.2021.).

[2] Angular blog, „Branding Guidelines for Angular and AngularJS“, http://blog.angularjs.org/2017/01/branding-guidelines-for-angular-and.html (posjećeno 30.7.2021.).

[3] Angular documentation, „Angular“, https://v11.angular.io/docs (posjećeno 5.5.2021.).

[4] Microsoft documentation, „Table-per-hierarchy and discriminator configuration“, https://docs.microsoft.com/en-us/ef/core/modeling/inheritance#table-per-hierarchy-and-discriminator-configuration (posjećeno 1.5.2021.).

[5] Internet Engineering Task Force (IETF), „JSON Web Token (JWT)“, https://datatracker.ietf.org/doc/html/rfc7519 (posjećeno 1.7.2021.).