# Web aplikacija za pregled i unošenje internih evidencijskih listi

**Dumančić, Ivan**

**VISOKO UČILIŠTE ALGEBRA**

ZAVRŠNI RAD

# Web aplikacija za pregled i unošenje internih evidencijskih listi

Ivan Dumančić

Zagreb, veljača 2018.

*„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor, te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada“.*

*U Zagrebu, 26.02.2018.*

# Predgovor

Želim se zahvaliti profesoru Bojanu Fulanoviću te ostalim profesorima na Visokom učilištu Algebra koji su mi svojim prenesenim znanjem i vještinama pomogli na akademskom putu.

Posebno želim zahvaliti svojoj obitelji, prijateljima i djevojci na neizmjernoj potpori tijekom obrazovanja. Njihova potpora mi je puno pomogla prilikom pisanja ovog rada.

**Prilikom uvezivanja rada, Umjesto ove stranice ne zaboravite umetnuti original potvrde o prihvaćanju teme završnog rada kojeg ste preuzeli u studentskoj referadi**

# Sažetak

Web aplikacija za unošenje i ažuriranje evidencijskih listi prikazuje alternativu za pisanje pisanih evidencijskih listi na papiru. Za razvoj aplikacije korišten je Visual Studio kao razvojno okruženje te programski okviri ASP.NET Web API 2, Unity, AngularJS i Entity Framework. Također, prilikom kodiranja su korišteni obrasci dizajna *Repository Pattern, Unit of Work, Response-Request* i *Factory*. Objavljivanje aplikacije je napravljeno pomoću Azure platforme. Arhitektura aplikacije je napravljena prema *Onion* arhitekturi. Aplikacija se sastoji od pet modula: Statistika, Evidencijske liste, Artikli, Zaposlenici i Ugostiteljski prostori. Potreba za web aplikacijama koje će olakšavati poslovne procese je uvijek prisutna. Svaki poslovni proces unutar bilo kojeg ugostiteljskog objekta se može poboljšati i automatizirati.

**Ključne riječi:** automatizacija, web aplikacija, evidencijska liste, poslovni procesi, arhitektura, programski okviri, obrasci dizajna

# Summary

A web application for entering and updating a list of records shows an alternative to writing written records on paper. Application development used Visual Studio as a development environment with program frameworks ASP.NET Web API 2, Unity, AngularJS and Entity Framework. Also, the design patterns Repository Pattern, Unit of Work, Response Request and Factory Patterns were used during coding. Publishing of application is made using the Azure platform. The architecture of the application is based on Onion architecture. The application consists of five modules: Statistics, Records, Items, Employees and Catering Spaces. The need for web applications that will facilitate business processes is always present. Every business process within any catering facility can be improved and automated.

**Keywords**: automation, web application, records, business processes, architecture, program frameworks, design patterns

# Sadržaj

# 1. Uvod

Ugostiteljski objekti svakodnevno se susreću sa pregledom stanja na kraju radnog vremena. Pregled stanja je proces pregleda ukupne količine robe na kraju radnog dana te unos tih podataka u internu evidencijsku listu. Interna evidencijska lista predstavlja više komada papira koji služe vlasniku ugostiteljskog objekta za osobni uvid u trenutno stanje robe. Zaposlenik na kraju svakog radnog dana, u prosjeku trideset minuta, pomoću fiskalne blagajne koja mu ispisuje trenutačno stanje robe, prepisuje to stanje na određeni evidencijski list za taj dan. Prilikom prepisivanja primljene robe i trenutnog stanja toga dana mora također uzeti u obzir i jučerašnje stanje te prema tome izračunati sveukupnu zaradu za taj dan.

Cilj ovog rada je utvrditi kako izrada aplikacije unapređuje vođenje ugostiteljskog objekta. Izradom web aplikacije za pregled i unos internih evidencijskih listi unaprijedilo bi se vođenje ugostiteljskog objekta na način da se unose i arhiviraju podaci koji predstavljaju brojčano stanje robe. Unosom tog stanja kroz aplikaciju, pretpostavila se smanjena potrošnja papira, smanjenje stalnog kopiranja i izrade listi. Namjeravale su se otkloniti greške prilikom ispunjavanja listi. Htjela se poboljšati preglednost stanja robe prema datumima, zaposleniku koji ju je upisao. Svi podaci i primjeri evidencijske liste korišteni u aplikaciji su preuzeti iz ugostiteljskog objekta Oldtimer koji se nalazi na adresi Ulica Ante Starčevića 60 u Sesvetama. Pristup podacima u svrhu izrade završnog rada omogućio je vlasnik.

Hipoteza ili pretpostavljeni odgovor na ciljeve istraživanja glasi: korištenje web aplikacije za pregled i unošenje internih evidencijskih listi ubrzat će unos stanja u evidencijske liste. Sljedeća hipoteza glasi: korištenje web aplikacije za pregled i unošenje internih evidencijskih listi smanjiti će potrošnju papira. Svrha rada je pragmatična odnosno doprinos unapređenju procesa unosa evidencijskih listi ugostiteljskog objekta.

Rad se sastoji od šest naslova. Uvod kao prvi naslov predstavlja objašnjenje osnovnih pojmova o kojima će se govoriti u radu, cilj i svrha rada te tema. Slijedeći naslov i njegovi podnaslovi opisuju tehnologije koje će se koristit za izradu aplikacije. Treći naslov označava strukturu baze podataka. Četvrti i peti naslov objašnjavaju dizajn i arhitekturu

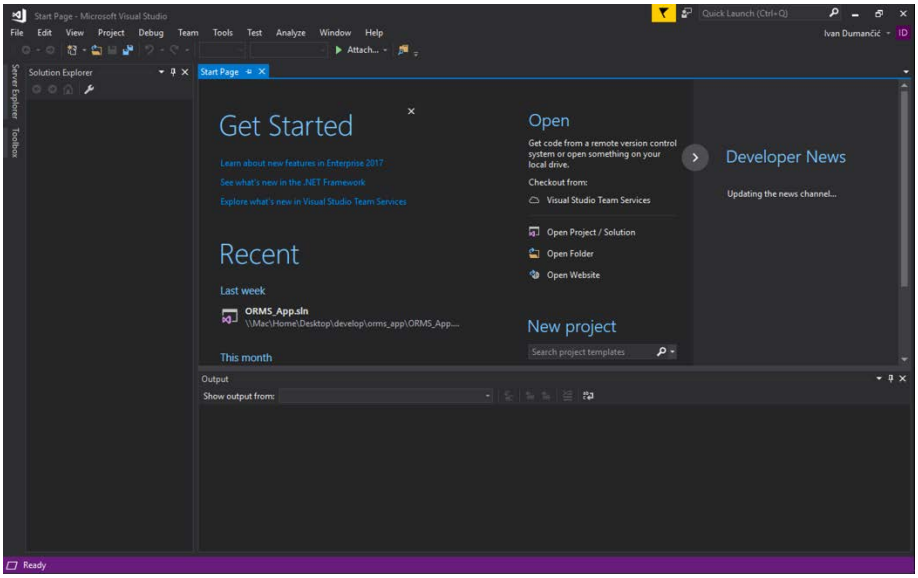
aplikacije. Šesti naslov objašnjava metode i rezultate testiranja. Zadnji naslov sadrži sveukupni zaključak završnog rada.

# 2. Tehnologije

Tehnologija je razvoj i primjena alata, strojeva, materijala i postupaka za izradbu nekog proizvoda ili obavljanje neke aktivnosti; također i znanost koja proučava primjenu znanja, vještine i organizacije u provedbi nekog procesa (Hrvatska enciklopedija, n.d.). Podnaslovi koji slijede ukratko objašnjavaju tehnologije i alate korištene za izradu web aplikacije koja je dio ovog rada.

## 2.1. Visual Studio 2017

Visual Studio (Slika 2.1) je interaktivno razvojno okruženje (engl. *Interactive Development Environment*, skraćeno IDE*)* koje služi za pregled, uređivanje programskog koda, otklanjanje neispravnosti (engl. *debug)* i izgradnju (engl. *build)*. Također, služi za objavljivanje (engl. *publish)* različitih vrsta aplikacija od Androida, iOS-a, Windowsa pa sve do weba i oblaka (engl. *cloud)* (Visual Studio IDE Overview, 2018). Prilikom izrade aplikacije, Visual Studio je bio glavno razvojno okruženje zbog njegove jednostavnosti i brzine izrade koda. S dolaskom Visual Studio 2017 verzije predstavljena je i puno bolja podrška za JavaScript[1], što je dodatno unaprijedilo razvoj web aplikacije.

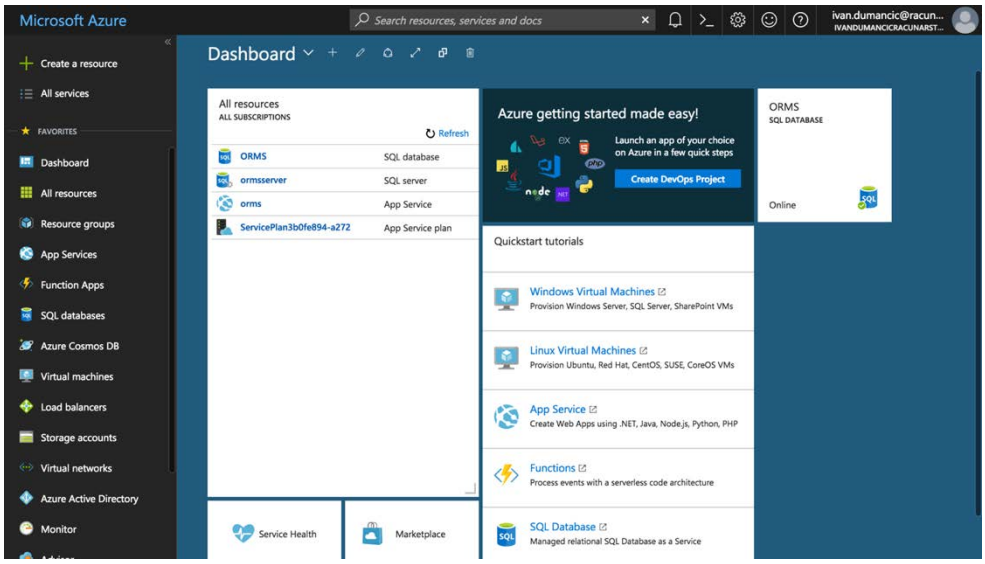



Slika 2.1 Interaktivno sučelje Visual Studio

[1] JavaScript je klijentski programski jezik za razvijanje web stranica.

## 2.2. Microsoft Azure

Microsoft Azure (Slika 2.2) je usluga računalnog oblaka (engl. *cloud computing service)* koja omogućava hostanje aplikacije svojim korisnicima, pojednostavljuje razvoj novih i postojećih aplikacija Azure, kao platforma za usluge, bazirana je u oblaku i nudi tri različite opcije usluga: infrastruktura kao usluga (engl. *Infrastructure as a Service*, skraćeno IaaS), platforma kao usluga (engl. *Platform as a Service*, skraćeno PaaS) i softver kao usluga (engl. *Software as a Service*, skraćeno SaaS). Najbrži način za objavljivanje aplikacija je putem njihovog App Service. Nadalje, postoji opcija Azure Virtual Machine, usluga u kojoj se kroz par koraka postavlja vlastita virtualna mašina, na kojoj će se izgraditi aplikacija (Azure Developer Guide, 2017). Ovaj rad će se koristiti infrastrukturom kao uslugom. Upotrijebiti će se usluga iznajmljivanja Azurove infrastrukture kako bi se hostala aplikacija, baza podataka i omogućila veća dostupnost aplikacije. Jedna od mnogih prednosti hostanja aplikacije na Azuru je zaštićenost aplikacije HTTPS[2] protokolom.



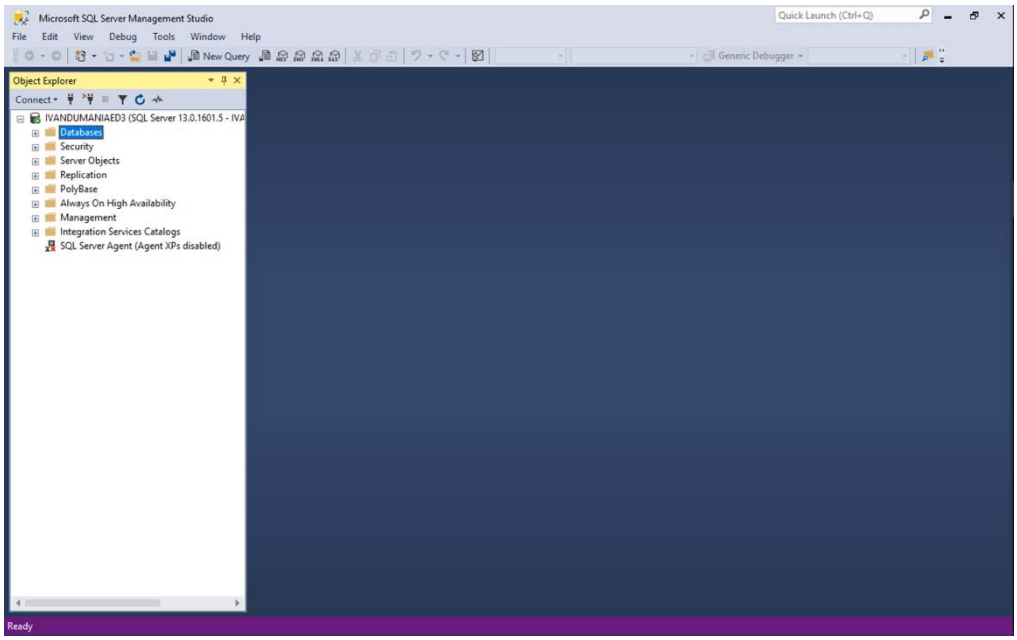

Slika 2.2 Kontrolna ploča (engl. *dashboard)* Microsoft Azura

## 2.3. Microsoft SQL Server

SQL[3] Server je centralni dio Microsoftove podatkovne platforme. SQL Server je industrijski predvodnik u operacijskim sustavima za upravljanje bazama podataka (engl. *Operational Database Management Systems*, skraćeno ODBMS). Microsoftov SQL Server

[2] HTTPS je osigurani protokol tj. skup pravila za prijenos informacija između poslužitelja i korisnika.
[3] SQL je programski jezik za pisanje strukturiranih upita prema bazi podataka.

sadrži dosta različitih tehnologija povezanih sa obradom podataka. Jedna od glavnih tehnologija je Database Engine. Glavna uloga Database Enginea je pohrana i obrada podataka (SQL Server Documentation, 2017). Prilikom razvoja aplikacije za lakši pristup bazi podataka i obradu podataka korišten je Microsoft SQL Server Management Studio (Slika 2.3). Sve potrebne skripte za kreiranje i popunjavanje tablica napisane su SQL Server Management Studiom. Detaljnije će se obrađivati u poglavlju Struktura baze podataka.




Slika 2.3 Interaktivno sučelje Microsoft SQL Server Management Studio

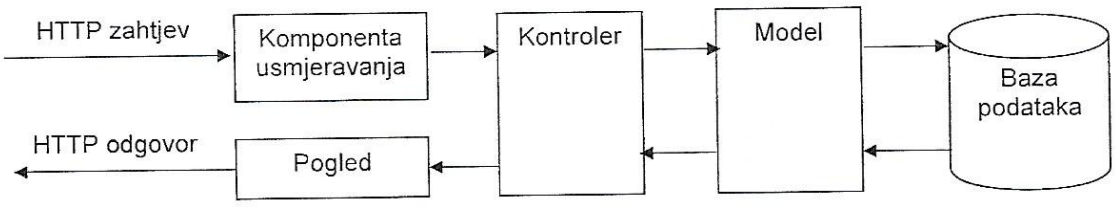
## 2.4. ASP.NET

ASP.NET je Microsoftov web okvir (engl. *framework*) za razvoj web aplikacija koji se koristi HTML[4], CSS[5] i JavaScript programskim jezicima. ASP.NET sadrži tri različita programska okvira za razvoj web aplikacija: Web Forms, ASP.NET Web Pages i ASP.NET MVC. Okvir koji također dolazi kao dio ASP.NET web okvira je Web API, koji nam služi za razvoj i izgradnju vlastitog API-ja[6] (ASP.NET overview, 2010). Za izradu ove aplikacije korištena su dva programska okvira: MVC i Web API 2. Svi navedeni okviri se mogu pronaći unutar Visual Studio IDE-a.

[4] HTML je označni programski jezik za konstrukciju web stranice.
[5] CSS je stilski programski jezik za definiranje ponašanja i izgleda web stranice.
[6] API je neka centralna točka koja nam služi pristup podacima sa bilo kojeg uređaja.

### 2.4.1. MVC

ASP.NET MVC uz Web Forms je najpoznatiji programski okvir za razvoj web aplikacija. Korištenjem *model-view-controller* obrasca za dizajniranje (engl. *design pattern*) omogućena je podjela odgovornosti aplikacije. Podjela odgovornosti predstavlja podjelu aplikacije na slojeve, što znači da svaki sloj aplikacije ima svoju odgovornost. Kada se napravi korisnički zahtjev u MVC aplikaciji prva komponenta na koju pristiže zahtjev je komponenta usmjeravanja (engl. *routing*) (Slika 2.4). Ona usmjerava korisnički zahtjev prema točno određenom kontroleru koji na sebi ima točno definiranu akciju na koju je zahtjev poslan. Slijedeća bitna komponenta je model. Model predstavlja poslovni sloj aplikacije, podatke s kojima će nadolazeći klijentski zahtjev raditi Komponenta pogled (engl. *view*) koristi se kao prezentacijski sloj, pretvara model u klijentski sadržaj koji će klijent nakon što poslužitelj obradi zahtjev primiti. Kontroler (engl. *controller*) je ulazna komponenta koja prima korisnički zahtjev, obrađuje podatke i priprema ih tako da odgovaraju modelu. Nadalje taj će se model predati pogledu za ispisivanje i prikazivanje klijentu (Đambić et al, 2014).




Slika 2.4 Tijek obrade MVC klijentskog zahtjeva (Đambić et al, 2014)

U razvoju aplikacije ASP.NET MVC je korišten za pokretanje izvornog pogleda koji će sadržavati AngularJS ng-view direktivu (engl. *directive*) koja služi za prikazivanje AngularJS parcijalnih (engl. *partial*) pogleda.

### 2.4.2. Web API 2

ASP.NET Web API 2 predstavlja programski okvir za izgradnju web API-ja na .NET web okviru. Olakšava izgradnju HTTP usluga koje mogu dosegnuti veliki broj klijenata uključujući web preglednike, mobitele i tradicionalne desktop aplikacije. Životni ciklus ASP.NET Web API korisničkog zahtjeva vrlo je sličan MVC-ovom. No, bitna razlika je u tome šta kontroler Web API-ja ne vraća MVC poglede nego vraća tekstualne, brojčane, binarne podatke u XML, HTML ili JSON formatu (Getting started with ASP.NET Web API 2, 2017). ASP.NET Web API programski okvir pripada velikom dijelu razvoja i

izgradnje aplikacije za interne evidencijske liste. Koristeći AngularJS programski okvir, pozivale su se HTTP metode GET i POST nad akcijama Web API kontrolera. Kontroler je nakon završenog zahtjeva vratio odgovarajući HTTP statusni kod i obrađene podatke u JSON formatu.

## 2.5. Entity Framework

Entity Framework (EF) je C#[7] bazirani ORM koji omogućava .NET programerima rad sa relacijskim podacima koristeći specifične domenske objekte Entity Framework jednako svakom programskom okviru ima svoje prednosti i mane. Jedna od bitnih prednosti je eliminiranje potrebe za pisanjem SQL koda kako bi se kreirala baza podataka i njoj pripadajuće tablice (Introduction to Entity Framework, 2016). Prema tome može se zaključiti da se ne mora ulaziti u SQL Server Management Studio ili neko drugo SQL razvojno okruženje kako bi se kreirala baza podataka. Kreiranje specifičnih domenskih modela (objekata) u C#-u i korištenje DbContext EF klase predstavlja EF Code First pristup kreiranju baze podataka. DbContext klasa predstavlja sesiju (engl. *session*) sa bazom podataka (Context Class in Entity Framework, n.d.). Također, postoji Database First i Model First pristup. Za izradu domenskih modela aplikacije za pregled i unošenje internih evidencijskih listi korišten je Code First pristup koji je baziran na postojećoj bazi podataka. U aplikaciji se koristi Entity Framework 6.

## 2.6. AngularJS

AngularJS je klijentski programski okvir koji se koristi za lakše, brže prikazivanje i obradu podataka na *frontend* strani web aplikacije. AngularJS je napisan u JavaScript programskom jeziku. Do danas postoji više verzija AngularJS programskog okvira. Za potrebe web aplikacije za pregled i unos internih evidencijskih listi korištena je 1.6.6 verzija AngularJS-a, jedna od zadnjih verzija koja je koristila JavaScript kao podlogu za razvoj. Novije verzije koriste TypeScript[8]. Jedna od dvije glavne prednosti AngularJS-a je dvosmjerno povezivanje podataka (engl. *two-way data binding)*. Dvosmjerno povezivanje podataka predstavlja automatsko sinkroniziranje između modela i komponenti pogleda

[7] C# je objektno orijentirani programski jezik.
[8] TypeScript je strogo tipizirani JavaScript koji više liči poznatijim objektno orijentiranim jezicima kao što su C# i Java.

Dvosmjerno povezivanje podataka omogućava da se promijene na klijentu prikazuju u stvarnom vremenu. Što znači da ne trebamo napraviti zahtjev (engl. *request*) prema poslužitelju i čekat njegov odgovor (engl. *response*) da bi vidjeli promjene. Trebamo poslati zahtjev ako želimo da se te promijene spreme. Druga glavna prednost je AngularJS usmjeravanje. Korištenjem AngularJS usmjeravanja u kombinaciji sa ng-view AngularJS direktivom omogućava se prikazivanje parcijalnih pogleda koji su napisani u čistom HTML-u (AngularJS Developer Guide, n.d.). Ključnu ulogu u usmjeravanju aplikacije za interne evidencijske liste ima znak # koji odvaja MVC usmjeravanje od AngularJS usmjeravanja.

### 2.6.1. Angular Chart i Chart.js

Angular Chart je omot (engl. *wrapper*) oko Chart.js (http://jtblin.github.io/angular-chart.js/). Chart.js je klijentski programski okvir napisan u JavaScript-u koji služi za statističko prikazivanje podataka pomoću stupčastih, linearnih, kružnih itd. grafova (http://www.chartjs.org/) Angular Chart olakšava korištenje Chart.js u kombinaciji sa AngularJS-om. Angular Chart u aplikaciji za uređivanje i unošenje evidencijskih listi služi za prikazivanje gore navedenih grafova kako bi se lakše prikazalo statističko stanje određenog skupa podataka.

### 2.6.2. Angular Paging

Angular Paging je AngularJS direktiva koja pomaže pri paginaciji velikog skupa podataka. Zahtjeva minimalne podatke o načinu prikazivanja stranice, kao što su broj stranice, količina podataka koja će se prikazivati i sveukupni zbroj podataka (https://github.com/brantwills/Angular-Paging). Angular Paging direktiva je bila od velike pomoći prilikom izrade aplikacije. Korištenjem ove direktive eliminiralo se vrijeme razvoja klijentske strane paginacije. Angular Paging direktiva za stiliziranje koristi Bootstrap 3. Kako bi se direktiva ispravno prikazivala u kombinaciji sa Bootstrap 4 programskim okvirom napravljene su neke izmjene u izvornom kodu koje nisu dio službene Angular Paging direktive.

## 2.7. Bootstrap 4

Bootstrap 4 je nasljednik Bootstrap 3 programskog okvira. Bootstrap je jedan od najpoznatijih programskih okvira za stiliziranje, responzivnost i mobilno prikazivanje web stranica. Baziran je na HTML-u, CSS-u i JavaScript-u. Bootstrap 4 donosi neke novitete u dizajniranju stranica, ali se ne razlikuje puno od prijašnjeg (https://getbootstrap.com/). Korištenjem Bootstrapa u aplikaciji za interne evidencijske liste osiguran je responzivan prikaz i na mobilnim uređajima.

## 2.8. SweetAlert 2

SweetAlert 2 je responzivna i prilagodljiva (engl. *customizable*) zamjena za JavaScript skočne (engl. *popup*) prozore (https://sweetalert2.github.io/). Aplikacija za interne evidencijske liste koristi SweetAlert 2 za prikazivanje uspješnosti dohvata podataka od poslužitelja i za prikazivanje skočnog prozora sa krugom učitavanja (engl. *loading circle*) prilikom dugog čekanja na odgovor poslužitelja.

## 2.9. Unity Container

Unity Container je lagani, proširivi programski spremnik za ubrizgavanje ovisnosti (engl. *dependency injection*). Omogućava razvoj i izgradnju slabo povezanih (engl. *loosely coupled*) aplikacija kroz slojeve apstrakcije (Unity Container, 2013). Upotrebom Unity Container u aplikaciji za interne evidencijske liste ne dolazi do poteškoća prilikom kreiranja objekata koji zahtijevaju ubrizgavanje ovisnosti. Unity Container ima mogućnost da to napravi, kada se definira koji objekt traži koju ovisnost.

# 3. Struktura baze podataka

Baza podataka je skup tablica koje sadrže neke podatke. Tablice sa podacima su pohranjene na vanjskoj memoriji, koji su istovremeno dostupne korisnicima i programima (Kaštelan, 2010). Poglavlje Struktura baze podataka prikazuje sve tablice baze podataka internih evidencijskih listi. Detaljno objašnjava veze između entiteta te njihove atribute.

## 3.1. Osnovni pojmovi

Osnovni pojmovi koji se koriste unutar baze podataka su (Kaštelan, 2010):

- **Entitet** – skup objekata, pojava ili događaja iz realnog svijeta o kojem želimo spremati podatke. Svaki entitet ima neka svoja svojstva koja ga opisuju. Ta svojstva se zovu atributi.
- **Atribut** – svojstva entiteta.
- **Veza** – je nešto što veže dva ili više entiteta.
- **Tablica** – skup zapisa u bazi podataka.
- **Primarni ključ** – atribut ili skup atributa koji jedinstveno identificiraju svaki element entiteta.
- **Strani ključ** – ograničenje koje označava relaciju između tablica.

Veza predstavlja bitan dio entiteta zato što ih povezuje u cjelinu koja ima smisao i koja čini bazu podataka. Razlikujemo tri vrste binarnih veza (Tablica 3.1) između entiteta (Kaštelan, 2010):

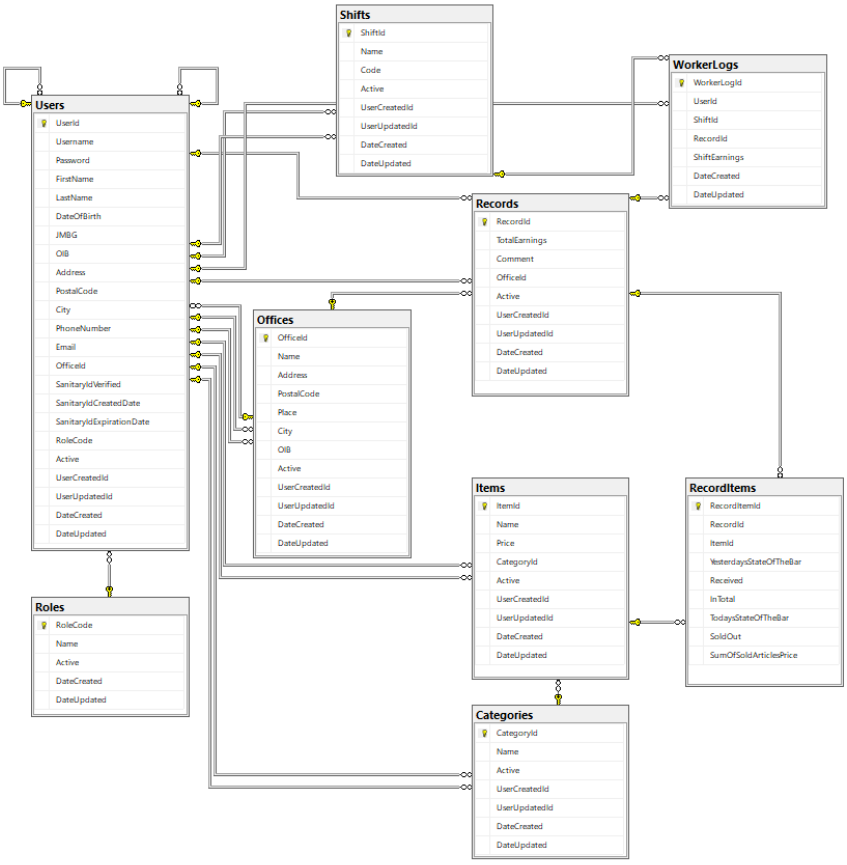
Tablica 3.1 Vrste veza između entiteta (Kaštelan, 2010)

| | |
|---|---|
| 1:1 veza | jednom zapisu iz jedne tablice odgovara jedan i samo jedan zapis iz druge tablice. |
| 1:N veza (ili 1: ∞) | jednom zapisu iz jedne tablice odgovara 0, 1 ili više zapisa iz druge. |
| M:N veza (ili ∞:∞) | jedan zapis prvog entiteta može biti u vezi s 0, 1 ili više primjeraka drugog entiteta, te također jedan zapis drugog entiteta može biti u vezi s 0, 1 ili više zapisa prvog entiteta. |

## 3.2. Tablice i entiteti

Baza podataka internih evidencijskih listi ukupno sadrži devet tablica. Dvije tablice predstavljaju vezne tablice između entiteta dok ostale tablice predstavljaju entitete koje možemo susresti u stvarnom svijetu. Korištenjem SQL skripte kreirana je baza podataka za uređivanje i unošenje evidencijskih listi.

Nakon pokretanja skripte za kreiranje baze podataka dobit će se baza podataka spremna za korištenje. U skripti za kreiranje baze podataka postavljeni su klasterirani (engl. *clustered*) indeksi na primarne ključeve zbog boljih performansi baze podataka. Indeks je struktura na disku povezana sa tablicom, ubrzava dohvaćanje redaka iz tablice. U tablici može postojati samo jedan klasterirani indeks zato što se redovi u tablici mogu sortirati samo u jednom smjeru (Clustered and Nonclustered Indexes Described, 2017). Ograničenja (engl. *constraint*) za strane ključeve na tablicama nisu napravljena direktno prilikom kreiranja tablice, nego su kasnije posebno napisana tako da redoslijed kreiranja tablica nije bitan, ali je bitan redoslijed postavljanja ograničenja. Slika 3.1 grafički prikazuje relacijski model koji predstavlja kreiranu bazu podataka.




Slika 3.1 Grafički prikaz relacijskog modela

Nadalje, kako bi poslovni proces web aplikacije za uređivanje i unošenje evidencijskih listi pravilno funkcionirao, unijeti će se potrebne uloge u tablicu uloga (Kôd 3.1):

```
INSERT INTO [Roles] ([RoleCode], [Name]) VALUES
('ORMS_Admin', 'Administrator')

INSERT INTO [Roles] ([RoleCode], [Name]) VALUES
('ORMS_Owner', 'Owner')

INSERT INTO [Roles] ([RoleCode], [Name]) VALUES
('ORMS_Manager', 'Manager')

INSERT INTO [Roles] ([RoleCode], [Name]) VALUES
('ORMS_Worker', 'Worker')
```

Kôd 3.1 Popunjavanje tablice uloga sa podacima

Nakon izvršavanja Kôd 3.1 potrebno je dodati i nove korisnike koji će imati pristup kreiranju novih zapisa zaposlenika u web aplikaciji. Prvi korisnici u ovom slučaju će biti vlasnik i administrator (Kôd 3.2):

```
INSERT INTO [Users] ([Username], [Password], [FirstName],
[LastName], [RoleCode], [UserCreatedId], [UserUpdatedId])
VALUES ('admin', 'admin123', 'Admin', 'Admin', 'ORMS_Admin',
1, 1)

INSERT INTO [Users] ([Username], [Password], [FirstName],
[LastName], [RoleCode], [UserCreatedId], [UserUpdatedId])
VALUES ('vlasnik', 'vlasnik123', 'Vlasnik', 'Vlasnik',
'ORMS_Owner', 1, 1)
```

Kôd 3.2 Unošenje inicijalnih korisnika u tablicu korisnika

U nastavku rada tablice će biti kategorizirane prema veznim tablicama i tablicama koje predstavljaju entitete, kako bi se što bolje objasnila svrha tablica.

### 3.2.1. Entiteti kao tablice

Entitet kao skup nekih atributa možemo prikazati pomoću tablice. Pretvorba entiteta u tablicu se odvija pomoću sljedećeg načina (Kaštelan, 2010):

- svaki atribut ima svoj stupac u tablici
- u retke upisujemo zapise entiteta.
- svaki redak u tablici odgovara jednom elementu entiteta

- svaki stupac je odgovarajući atribut
- ime tablice jednako je nazivu entiteta

Koristeći navedeni način pretvorbe entiteta u tablice, kreirane su odgovarajuće tablice sa primarnim i stranim ključevima u bazi podataka za interne evidencijske liste:

- **Korisnik (engl. *user*)**

Tablica korisnika ima značajnu ulogu u ovoj bazi podataka. Sadrži sve bitne informacije o zaposleniku, kao što su: opći podaci (ime, prezime, datum rođenja, OIB, JMBG), podaci potrebni za prijavu u sustav (korisničko ime i lozinka), podaci o stanovanju (adresa, poštanski broj, grad), kontakt podaci (broj telefona i email), poslovni podaci (ugostiteljski prostor u kojem radi i ulogu koju ima u tom ugostiteljskom prostoru) i podaci od sanitarnoj iskaznici (ima li je, datum isticanja, datuma izrade). U korisničkoj tablici postoji i atribut (stupac) `Active` koji služi kao indikator aktivnog zapisa. Pomoću njega omogućila se mogućnost pregleda informacija obrisanog zapisa (engl. *records*) (npr. korisnik koji je izbrisao zapis) na način gdje nula (0) predstavlja obrisani, a jedinica (1) predstavlja aktivni zapis. Također, sadrži podatke o korisniku koji je kreirao (`UserCreatedId`) ili ažurirao korisnika (`UserUpdatedId`) te podatke o datumu kreiranja i ažuriranja. Ovi podaci predstavljaju povijesni prikaz mijenjanja podataka u tablici. Ovakav način obilježavanja povijesti zapisivanja podataka u tablici, provlači se kroz većinu tablica. Atributi `OfficeId` i `RoleId` predstavljaju strane ključeve koji pokazuju na zapise u tablicama ugostiteljskih prostora i korisničkih uloga, dok atribut `UserId` predstavlja jedinstveni primarni ključ (identifikator) korisnika. Atributi `UserCreatedId` i `UserUpdatedId` su strani ključevi koji se u ovom slučaju nalaze u istoj tablici gdje se nalazi primarni ključ te pokazuju na njega. Ovo je jedini takav slučaj, u ostalim tablicama ovi strani ključevi ne pokazuju na primarni ključ tablice u kojoj se nalaze nego na primarni ključ tablice korisnika.

- **Uloga (engl. *role*)**

Tablica uloga je atributno najmanja tablica u ovoj bazi podataka, ali je jako bitna za radni proces aplikacije. Baza podataka pohranjuje četiri različite uloge: administrator, vlasnik, manager i zaposlenik. Također, sadrži povijesne informacije o kreiranju i ažuriranju uloga te indikator aktivnog zapisa. Atribut `RoleId` predstavlja jedinstveni primarni ključ uloga.

- **Ugostiteljski prostor ili ured (engl. *office*)**

Tablica ugostiteljskih prostora sadrži informacije o imenu, adresi, poštanskom broju, mjestu, gradu, OIB-u ugostiteljskog prostora. Također, sadrži povijesne informacije o

kreiranju i ažuriranju prostora (ureda) te indikator aktivnog zapisa. Atribut `OfficeId` predstavlja jedinstveni primarni ključ ugostiteljskih prostora.

- **Smjena (engl. *shift*)**

Tablica smjena sadrži informacije o vrstama smjena. Vrste smjena koje su zapisane u bazi podataka su: prva smjena i druga smjena. Kreiranjem ove tablice ostavili smo mogućnost za buduće dodavanje potrebnih smjena s obzirom na korisničke zahtjeve. Također, sadrži povijesne informacije o kreiranju i ažuriranju smjena te indikator aktivnog zapisa. Atribut `ShiftId` predstavlja jedinstveni primarni ključ smjena.

- **Evidencija (engl. *record*)**

Tablica evidencija je glavna tablica i srž aplikacije za pregled i unošenje evidencijskih listi. Ova tablica sadrži evidencijske liste i informacije kao što su sveukupna zarada ugostiteljskog prostora za određeni dan, ugostiteljski prostor ili ured za koji je unesena lista te komentare. Sama po sebi tablica nije atributno velika, ali je centralna točka za dohvaćanje bitnih informacija o evidencijskoj listi pomoću primarnog ključa. Također, sadrži povijesne informacije o kreiranju i ažuriranju evidencijskih listi te indikator aktivnog zapisa. Atribut `OfficeId` je strani ključ koji pokazuje na zapis ugostiteljskog prostora za koji je unesena lista. Atribut `RecordId` predstavlja jedinstveni primarni ključ evidencijskih listi.

- **Artikl (engl. *item*)**

Tablica artikala je šifrarnik svih dostupnih artikala s kojima ugostiteljski prostori mogu raditi. Sadrži informacije o imenu artikla, cijeni i kategoriji pod koju spada. Također, sadrži povijesne informacije o kreiranju i ažuriranju artikala te indikator aktivnog zapisa. Atribut `CategoryId` je strani ključ koji pokazuje na zapis u tablici kategorija. Atribut `ItemId` predstavlja jedinstveni primarni ključ artikala.

- **Kategorija (engl. *category)***

Tablica kategorija je druga atributno najmanja tablica u bazi podataka. Sadrži informacije o nazivima kategorija artikala. Također, sadrži povijesne informacije o kreiranju i ažuriranju kategorija te indikator aktivnog zapisa. Atribut `CategoryId` predstavlja jedinstveni primarni ključ kategorija.

### 3.2.2. Vezne tablice

Vezne tablice povezuju tablice. To su tablice koje ne predstavljaju entitete, nego predstavljaju povezivanje podataka dvaju ili više različitih entiteta:

- **Radni dnevnik (engl. *worker log)***

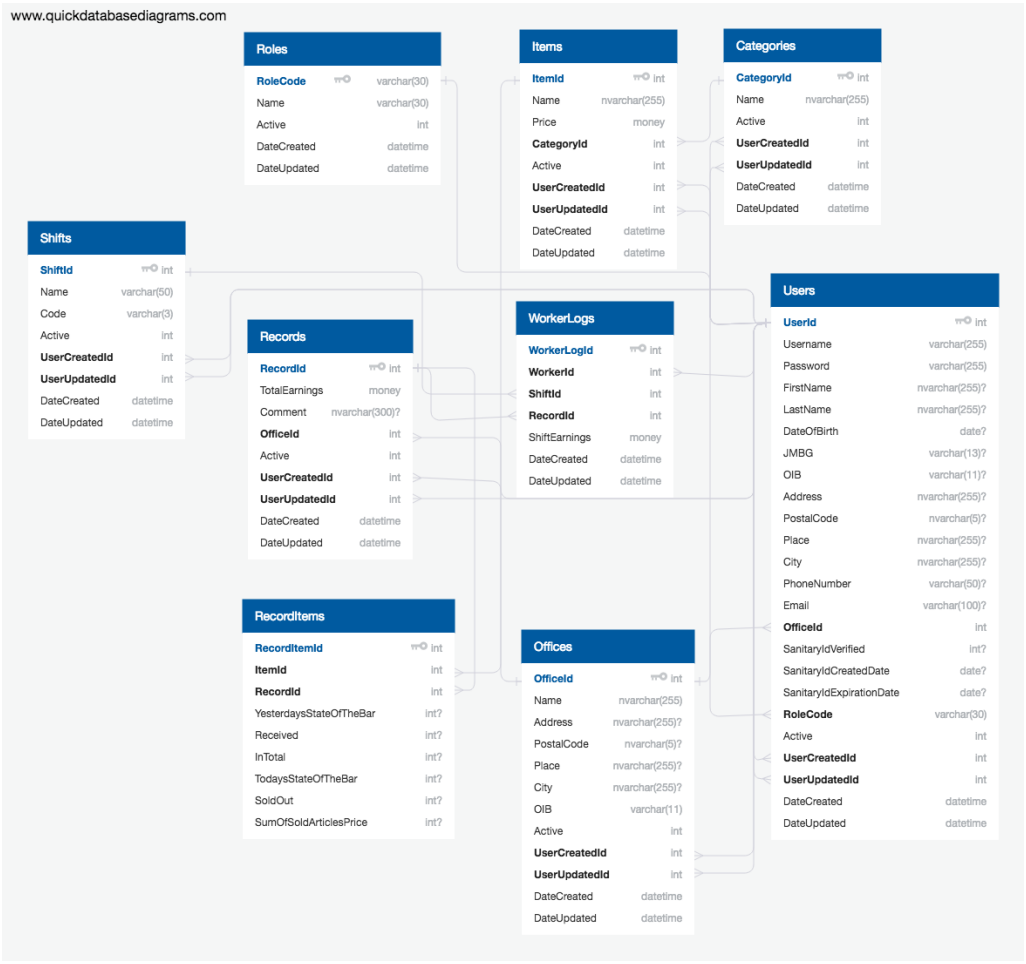
Tablica radni dnevnik sadrži informacije o vrstama smjena koje su se odradile za određeni dan (`ShiftId`), zaposleniku (`UserId`) koji je radio određenu smjenu, sveukupnu zaradu određene smjene i kojoj listi (`RecordId`) zapis u dnevniku pripada. Također, sadrži povijesne informacije o datumu kreiranja i ažuriranja određenog zapisa. Atributi `ShiftId`, `UserId` i `RecordId` su strani ključevi. Atribut `WorkerLogId` je jedinstveni primarni ključ koji označava jedinstveni zapis u radnom dnevniku.

- **Artikl evidencije (engl. *record item*)**

Tablica artikala za evidencijske liste predstavlja jedan zapis u evidencijskoj listi. Sadrži informacije o evidencijskoj listi kojoj određeni artikl pripada te informacije o stanju artikla za određenu listu. Informacije o stanju artikla su slijedeće: jučerašnje količinsko stanje artikla, koliko se količinski artikla primilo danas, sveukupni količinski zbroj stanja artikla, današnje količinsko stanje artikla, koliko se količinski artikla prodalo danas i sveukupni zbroj količine i cijene artikla za određeni dan. Tablica ne sadrži nikakve povijesne informacije zato što je usko povezana sa tablicom evidencijskih listi, što znači ako se dogodi promjena na evidencijskoj listi kao npr. lista se obriše eliminirat će se pristup zapisima u ovoj veznoj tablici. Atributi `ItemId` i `RecordId` su strani ključevi koji gledaju na jedinstvene zapise u tablicama za artikle i evidencijske liste. Atribut `RecordItemId` je jedinstveni primarni ključ artikla evidentiranog za određenu evidenciju.

## 3.3. ER Dijagram

ER dijagram (engl. *Entity Relationship diagram*) prikazuje relacije između entiteta u sustavu. Relacije među entitetima mogu biti unarne, binarne ili n-arne. Unarna relacija predstavlja jedan na prema jedan relaciju između entiteta. Relacija je binarna ako sudjeluju dva entiteta, a n-arna ako ih ima više od dva (Kaštelan, 2010). U nastavku je prikazan ER dijagram sa vezama između entiteta (Slika 3.2):




Slika 3.2 ER dijagram baze podataka za evidencijske liste

# 4. Obrasci dizajna web aplikacije

Obrasci dizajna (engl. *design patterns*) su predlošci sa visokom razinom apstraktnih rješenja. Obrasci dizajna nisu primjenjivi samo u razvoju softvera; mogu se primijeniti i u drugim dijelovima života od arhitekture do inženjerstva. Podjelu obrazaca za dizajn po kategorijama napravila je skupina iskusnih programera koji su poznatiji pod imenom *Gang of Four*. Sakupili su dvadeset i tri obrasca i kategorizirali ih u tri grupe (Millet, 2010):

- **Obrasci kreiranja (engl. *creational patterns*)** – bave se konstrukcijom objekata i referenciranjem.
- **Obrasci ponašanja (engl. *behavioral patterns*)** – bave se komunikacijom između objekata.
- **Strukturni obrasci (engl. *structural patterns*)** - bave se odnosima između objekata te kako oni međusobno djeluju.

Obrasci dizajna se mogu primijeniti na bilo koji objektno orijentirani jezik. Što znači da nisu usko povezani sa problemom nastalim u radu sa jednim objektno orijentiranim jezikom. Cilj obrazaca je višekratna upotreba gotovih rješenja na već postojeće probleme (Millet, 2010).

U razvoju web aplikacije za unošenje i ažuriranje evidencijskih listi korišteni su određeni obrasci dizajna kao što su: *Repository Pattern, Unit of Work, Request-Response, Factory.* Svaki od ovih obrazaca ima neku određenu ulogu u razvoju aplikacije koja će biti optimizirana i slabo spojena (engl. *loosely coupled*). Slabo spojena aplikacija predstavlja aplikaciju koja sadrži slojeve koji nisu usko povezani sa ostalima aplikativnim slojevima i ne ovise o promjenama koji se događaju u drugim slojevima (Karatoprak, 2012). Nadalje, kako bi se što bolje shvatio dizajn aplikacije, objašnjeni su korišteni obrasci dizajna.

## 4.1. The Repository Pattern

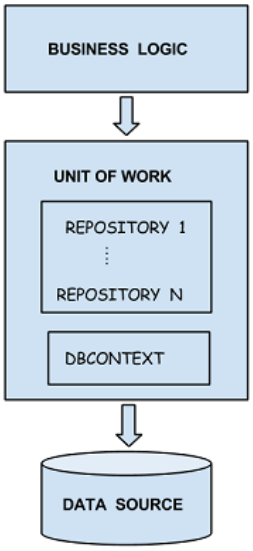
*Repository Pattern* predstavlja jedan od poznatijih strukturnih obrazaca. Koristi se u aplikativnom sloju za pristup podacima (engl. *Data Access Layer*, skraćeno DAL). Repozitorij, kao glavni dio ovog obrasca, ponaša se kao kolekcija u memoriji koja u potpunosti izolira poslovne entitete od podatkovne infrastrukture (Millet, 2010). Korištenje ovog obrasca ima mnoge prednosti kao što su (Kumar, 2016):

- poslovna logika može biti testirana bez logike za pristupanje podacima

- kod za pristup bazi podataka je višekratno upotrebljiv
- kod za pristup bazi podataka je centralno upravljan
- lako je za implementirat domensku logiku.

## 4.2. Unit of Work

*Unit of Work,* kao obrazac dizajna, dizajniran je da sadrži listu poslovnih objekata koji su bili promijenjeni nekom poslovnom transakcijom, bilo to dodavanje, ažuriranje ili brisanje. Jedna od velikih prednosti korištenja ovog obrasca je što sve liste poslovnih objekata gledaju na isti kontekst (engl. *context*) baze podataka[9] (Millet, 2010). Što dovodi do slijedećeg zaključka. Ako prilikom neke poslovne transakcije više lista poslovnih objekata u isto vrijeme napravi promijene na kontekstu baze podataka i na samo jednoj od njih dođe do greške, dogodit će se vraćanje (engl. *rollback*) podataka svih listi na validno početno stanje (Millet, 2010).




Slika 4.1 *Unit of Work*[10]

## 4.3. The Request-Response Pattern

*Request-Response* obrazac dizajna nastao je korištenjem *Document Message* obrasca. *The Document Message* obrazac omogućava generalizirani pristup komunikacije sa metodama

[9] Kontekst baze podataka je objekt koji sadrži sve tablice koje su dio baze podataka. Pomoću njega možemo vršiti CRUD operacije nad tablicama.

[10] Izvor: https://www.codeproject.com/Articles/825646/Generic-Repository-and-UnitofWork-patterns-in-MVC

servisa (engl. *service*). Korištenjem ovog obrasca izbjegavamo RPC način pozivanja metoda (Kôd 4.1) (Millet, 2010).

```
List<User> GetUsersBy(string firstName);
List<User> GetUsersBy(string firstName, string lastName);
```

Kôd 4.1 RPC način pozivanja metoda

Ovakav način pozivanja metoda gledajući u budućnost razvoja aplikacije ne bi bio održiv. Razlog tome je taj, da ako dođe do potrebe da se npr. napravi još jedan potpis metode koji sadrži još jedan dodatan parametar po kojem bi dohvaćali korisnike i tako n puta, imali bi nepreglednu i neodrživu metodu. Korištenjem *Document Message* obrasca ćemo enkapsulirat sve navedene parametre u jedan objekt (koji predstavlja zahtjev korisnika) (Kôd 4.2) što će eliminirat potrebu za stalnim mijenjanjem potpisa metode te će nastati čisti i programski održiv potpis metode (Millet, 2010).

```
List<User> GetUsersBy(UserSearchRequest request);
```

Kôd 4.2 Potpis metode korištenjem *Document Message* obrasca

*Request-Response* obrazac primjenjuje sva ista pravila, samo što još osigurava da i objekt kojeg vraćamo (koji predstavlja odgovor poslužitelja) primjenjuje ista pravila *Document Message* obrasca, kao i ulazni objekt (zahtjev) metode što možemo vidjeti iz Kôd 4.3.

```
UserSearchResponse GetUserBy(UserSearchRequest request);
```

Kôd 4.3 Potpis metode korištenjem *Request-Response* obrasca

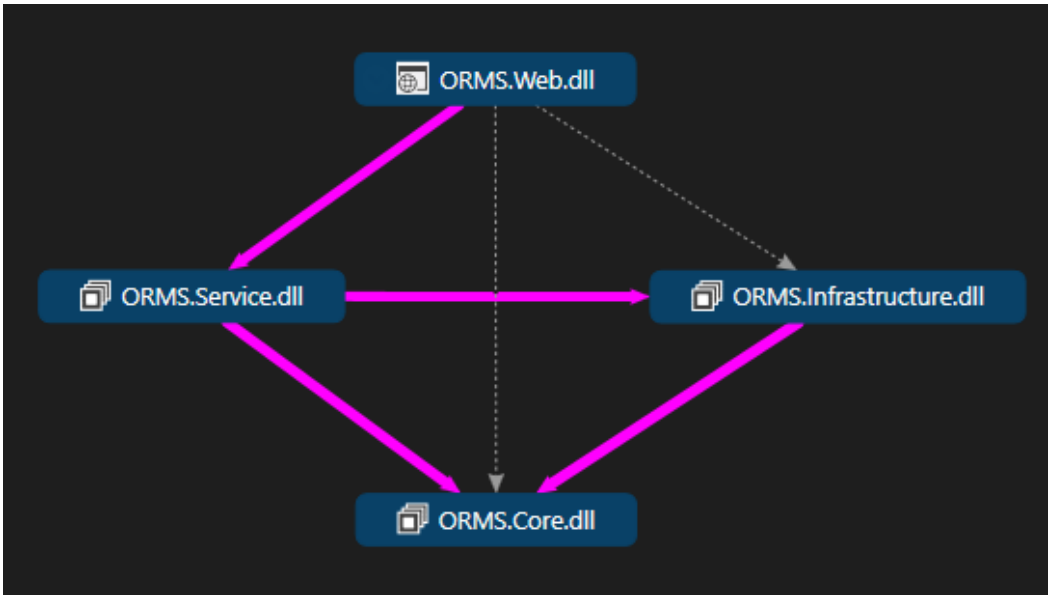
## 4.4. Factory

*Factory*, kao obrazac dizajna, pripada obrascima kreiranja. Svrha ovog obrasca je skrivanje kompleksnosti kreiranja nekog objekta. Klasa koja implementira ovaj obrazac tipično ima neku statičnu metodu koja će se pozivat i vraćat apstraktnu klasu ili sučelje (engl. *interface*) (Millet, 2010). Ovaj obrazac web aplikacija za unošenje i ažuriranje evidencijskih listi koristi za kreiranje usluga (engl. *service*) koje će web API pozivat kada klijent napravi zahtjev.

# 5. Razvoj web aplikacije

Web aplikacija (engl. *application*) je aplikacija kojoj korisnici mogu pristupiti putem web preglednika (engl. *browser*). Preglednik komunicira sa web poslužiteljem koristeći HTTP zahtjeve. Poslužitelj obrađuje zahtjev te vraća korisniku podatke ili HTML stranice koje preglednik može prikazati (Designing Web Applications, n.d.). Poglavlje Razvoj web aplikacije obrađuje glavne probleme u razvoju aplikacije te njihova odgovarajuća rješenja u kôdu. Uključuje i slike aplikacija za unošenje i ažuriranje evidencijskih listi u mobilnom izdanju te izdanju za računala.

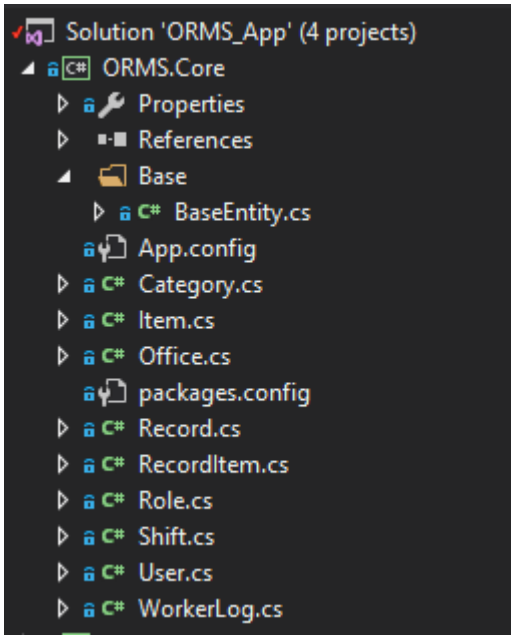
Za arhitekturu aplikacije odabrana je *Onion* arhitektura. *Onion* arhitektura sadrži slojeve podijeljene od *Core* do *Service* sloja. Koristeći ovu arhitekturu dobila se slabo povezana aplikacija gdje se komunikacija između slojeva nikad ne odvija pomoću konkretnih implementacija klasa nego preko sučelja. Također, riješio se problem podijele odgovornosti (engl. *separation of concerns*). Centralni dio ove arhitekture je *Core* aplikativni sloj koji sadrži sve domenske modele. Sloj *Infrastructure* sadrži sve operacije koje se odvijaju prema bazi podataka. *Service* sloj dohvaća podatke iz repozitorija koje *Infrastructure* sloj sadrži i obrađuje zahtjeve koji dolaze od web aplikacije. Glavno pravilo ove arhitekture je da vanjski slojevi smiju ovisiti prema više centralnim slojevima, ali ne smiju ovisiti prema vanjskim slojevima kao što i Slika 5.1 prikazuje (Vihite, 2015).




Slika 5.1 Prikaz ovisnosti između slojeva aplikacije

## 5.1. Core

*Core* sloj web aplikacije za unošenje i ažuriranje evidencijskih listi sadrži sve potrebne domenske modele koji će se koristiti u poslovnim transakcijama prema bazi podataka. Domenski modeli su kreirani koristeći Entity Framework *Code First* pristup. Struktura *Core* sloja je vidljiva iz Slika 5.2.




Slika 5.2 Struktura *Core* sloja

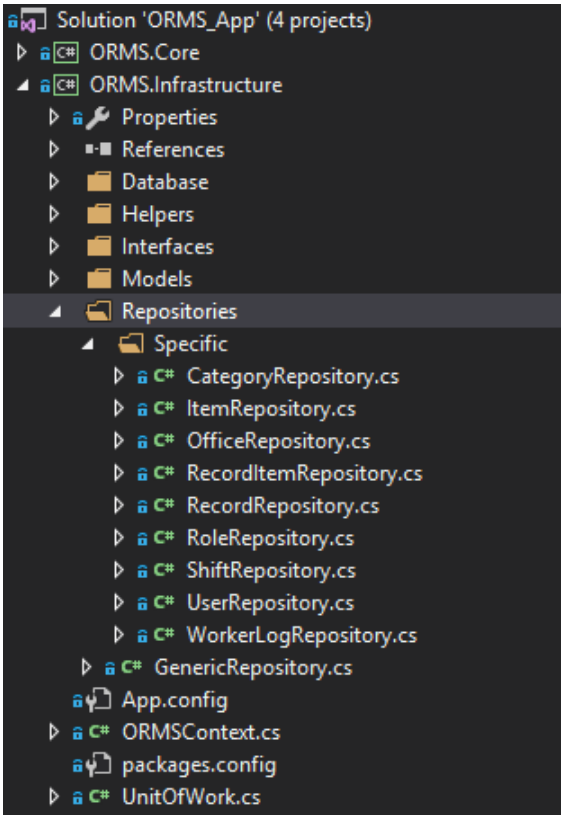
`BaseEntity` (Kôd 5.1) je apstraktna klasa koja sadrži svojstva (engl. *properties*) koja dijele svi domenski modeli. Razlog implementacije ove apstraktne klase je DRY princip dizajna (engl. *design principle*). Cilj DRY principa je izbjegavanje ponavljanja bilo kojeg dijela sustava izvlačenjem svojstava koja su zajednička i postavljanje tih svojstava na jedno mjesto (Millet, 2010).

```
public abstract class BaseEntity
{
	public int Active {get; set;}
	public int UserCreatedId {get; set;}
	public int UserUpdatedId {get; set;}
	public DateTime DateCreated {get; set;}
	public DateTime DateUpdated {get; set;}
}
```

Kôd 5.1 Apstraktna klasa `BaseEntity`

## 5.2. Infrastructure

*Infrastructure* sloj je najudaljeniji sloj *Onion* arhitekture koji se bavi sa potrebama infrastrukture i pruža implementacije repozitorijskih sučelja. Samo ovaj sloj zna koji se programski okviri za dohvaćanje podataka koriste dok drugi slojevi ne znaju ništa o tome (Vihite, 2015). Struktura *Infrastructure* sloja je vidljiva iz Slika 5.3.




Slika 5.3 Struktura *Infrastructure* sloja

Repozitoriji su implementirani koristeći *Repository Pattern.* Podijeljeni su na specifične i generičke. Specifične implementacije repozitorija su implementacije striktno vezane uz domenske modele koje predstavljaju. Također, sadrže metode koje nisu implementirane nasljeđivanjem generičkog repozitorija nego implementacijom domenskog sučelja repozitorija (Kôd 5.2).

```
public class UserRepository : GenericRepository<User>,
IUserRepository
{
      public ORMSContext Context {get {return _context as
      ORMSContext;}}
```

```
        public UserRepository(ORMSContext context) :
        base(context){}
        ...
}
```

Kôd 5.2 Primjer implementacije specifičnog repozitorija

Koristeći princip dizajna ubrizgavanje ovisnosti (engl. *dependency injection*) ubrizgana je implementacija `ORMSContext` Entity Framework kontekstne klase koja u pozadini nasljeđuje `DbContext` klasu. Ubrizgavanje ovisnosti je princip koji predstavlja dostavljanje zavisne klase kroz svojstvo, konstruktor ili metodu (Millet, 2010). Iz priloženog koda se također vidi da konstruktor prosljeđuje implementaciju ubrizgane kontekstne klase baznom konstruktoru klase koju nasljeđuje. Klasa koju nasljeđuje je u ovom slučaju generička implementacija domenskog repozitorija koja sadrži sve potrebne CRUD metode.

```
public class GenericRepository<TEntity> :
IRepository<TEntity> where TEntity : class
{
        protected readonly DbContext _context;
        protected readonly DbSet<TEntity> _dbSet;

        public GenericRepository(DbContext context)
        {
            this._context = context;
            this._dbSet = _context.Set<TEntity>();
        }
        ...
}
```

Kôd 5.3 Primjer implementacije generičkog repozitorija

Korištenjem `DbContext` klase u Kôd 5.3 umjesto direktne implementacije `ORMSContext` klase nije došlo do uskog povezivanja između generičkog repozitorija i programskog okvira Entity Framework. Generički repozitorij implementira generičko sučelje `IRepository` koje sadrži potpise metoda koje generički repozitorij implementira. Implementacije specifičnih sučelja repozitorija također implementiraju `IRepository` (Kôd 5.4).

```
public interface IRepository<TEntity> where TEntity : class
{
     IEnumerable<TEntity> GetAll();
     TEntity Get(object id);
```

```
        void Add(TEntity entity);
        void AddRange(IEnumerable<TEntity> entities);
        void Update(TEntity entity);
        void Remove(TEntity entity);
        void Remove(object id);
        void RemoveRange(IEnumerable<TEntity> entities);
        int Count();
    }
```

Kôd 5.4 Generičko sučelje `IRepository`

Nadalje, *Infrastructure* sloj također koristi *Unit of Work* obrazac dizajna. Klasa `UnitOfWork` implementira `IUnitOfWork` sučelje (Kôd 5.5) koje sadrži svojstva repozitorijskih sučelja domenskih modela i jednu metodu .

```
public interface IUnitOfWork : IDisposable
{
      ICategoryRepository Categories { get; }
      IItemRepository Items { get; }
      IOfficeRepository Offices { get; }
      IRecordItemRepository RecordItems { get; }
      IRecordRepository Records { get; }
      IRoleRepository Roles { get; }
      IShiftRepository Shifts { get; }
      IUserRepository Users { get; }
      IWorkerLogRepository WorkerLogs { get; }
      int Commit();
}
```

Kôd 5.5 `IUnitOfWork` sučelje

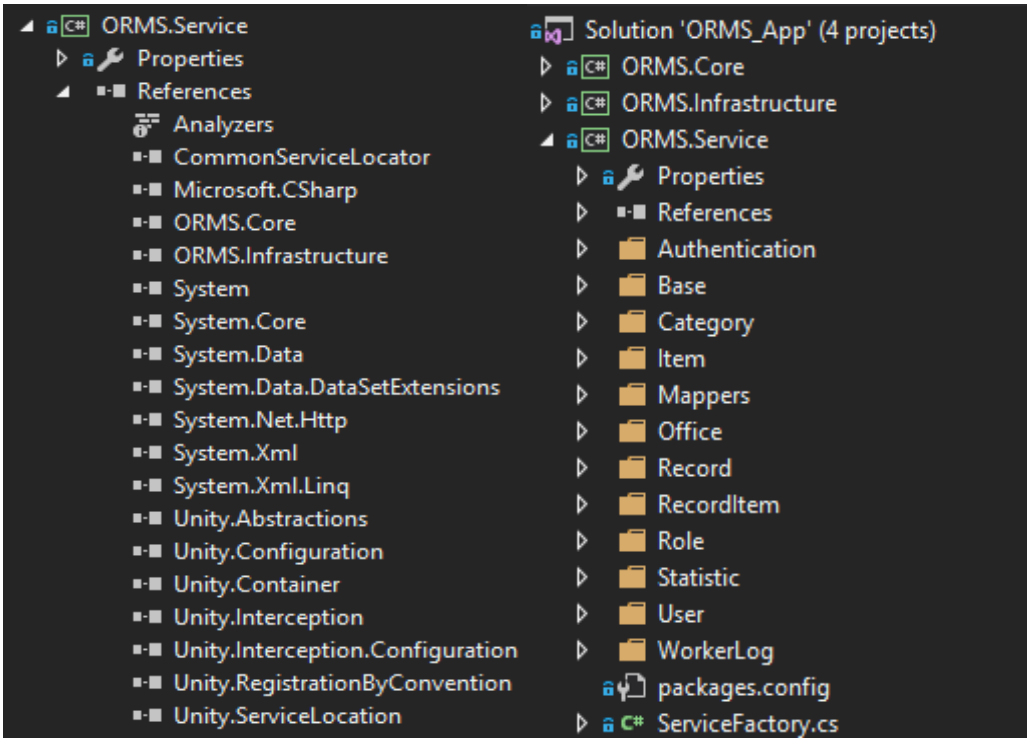
Metoda `Commit()` ima veliku ulogu u ovoj klasi zato što poziva završavanje transakcije i spremanje promjena na svim instancama repozitorija koje koriste isti bazni kontekst. Klasa `UnitOfWork` pomoću ubrizgavanja ovisnosti prima u konstruktor `ORMSContext` instancu klase koju dalje prosljeđuje svakoj novo kreiranoj instanci domenskog repozitorija kojeg je naslijedila putem sučelja.

Obrasci *Repository Pattern* i *Unit of Work* su već implementirani unutar Entity Framework (EF) programskog okvira. Razlog korištenja još jednog sloja apstrakcije nad apstrakcijom koju već programski okvir EF sadrži je jednostavan, htjela se izbjeći uska povezanost sloja za pristup podacima sa programskim okvirom kojeg koristi. Ovom implementacijom

omogućena je promjena programskog okvira ili načina dohvaćanja podataka iz baze podataka bez utjecaja na ostale slojeve aplikacije.

## 5.3. Service

*Service* sloj se ponaša kao posrednik između *Infrastructure* i *Web* sloja. Sadrži implementacije usluga koje primaju klijentske zahtjeve od *Web* sloja i dobivene podatke iz baze podataka od *Infrastructure* sloja te ih obrađuju (Vihite, 2015). Ovaj sloj nema nikakvog doticaja sa programskim okvirom Entity Framework, što znači da ne zna od kud dolaze podaci i kako dolaze. To je vidljivo iz strukture (Slika 5.4) *Service* sloja.




Slika 5.4 Struktura *Service* sloja

*Service* sloj sadrži implementacije usluga koje implementiraju njima odgovarajuća sučelja. Implementacije sučelja, kao i u *Infrastructure* sloju, predstavljaju DIP princip dizajna koji se odlično slaže sa ubrizgavanjem ovisnosti. Cilj DIP principa je izoliranje klasa od njihovih konkretnih implementacija korištenjem apstraktnih klasa ili sučelja, što dovodi do fleksibilnosti u slojevima gdje aplikacija nije nužno vezana uz jednu implementaciju (Millet, 2010). Implementacije usluga se također koriste *Request-Response* obrascem dizajna koji je detaljno objašnjen u poglavlju 4.3. Primjer implementacije *Request-*

*Response* obrasca i DIP principa u `AuthenticationService` klasi i `IAuthenticationService` sučelju prikazan je u Kôd 5.6 i Kôd 5.7

```
public interface IAuthenticationService : IBaseService
{
      AuthenticationResponse
      Authenticate(AuthenticationRequest authRequest);
}
```

Kôd 5.6 Primjer implementacije `IAuthenticationService` sučelja

Kôd 5.6 koji prikazuje `IAuthenticationService` sučelje, prikazuje i implementiranje `IBaseService` generičkog sučelja koje sadrži potpis `IUnitOfWork` svojstva kojeg klasa `AuthenticationService` implementira.

```
public class AuthenticationService : IAuthenticationService
{
      private IUnitOfWork unitOfWork;

      public IUnitOfWork UnitOfWork { get { return
      unitOfWork; } }

      public AuthenticationService(IUnitOfWork unitOfWork)
      {
            this.unitOfWork = unitOfWork;
      }

      public AuthenticationResponse
      Authenticate(AuthenticationRequest authRequest)
      {
            var response = new AuthenticationResponse();

            try
            {
                  var user =
                  unitOfWork.Users.Get(authRequest.Username,
                  authRequest.Password);

                if (user != null)
                {
                    authenticatedUser = user;
                    response.Success = true;
                    response.User = user.ToUserViewModel();
```

```
                }

            }
            catch (Exception ex)
            {
                Debug.Print($"AUTHORIZATION FAILED:
                ${ex.ToString()}");
                response.Success = false;
            }

            return response;
        }
    ...
}
```

Kôd 5.7 Primjer implementacije `AuthenticationService` klase

Iz priloženog Kôd 5.7 prikazuje se i ovisnost konstruktora `AuthenticationService` klase o `IUnitOfWork` sučelju. Kako se web aplikacija ne bi trebala brinuti o tome na koji način se kreira određena usluga, na snagu dolazi Unity programski okvir. Svrha Unity programskog okvira je detaljnije objašnjena u poglavlju 2.9 korištenih tehnologija. Njegova svrha u *Service* sloju je ubrizgavanje ovisnosti u konstruktor usluga. Konkretna implementacija i kreiranje `UnityContainer` klase Unity programskog okvira sa registracijom tipova koje treba ubrizgati u konstruktor usluge odvija se u `ServiceFactory` klasi (Kôd 5.8). Klasa `ServiceFactory`, pomoću *Factory* obrasca dizajna, (4.4) implementira statične metode koje kreiraju i vraćaju apstraktnu implementaciju (sučelje) usluga.

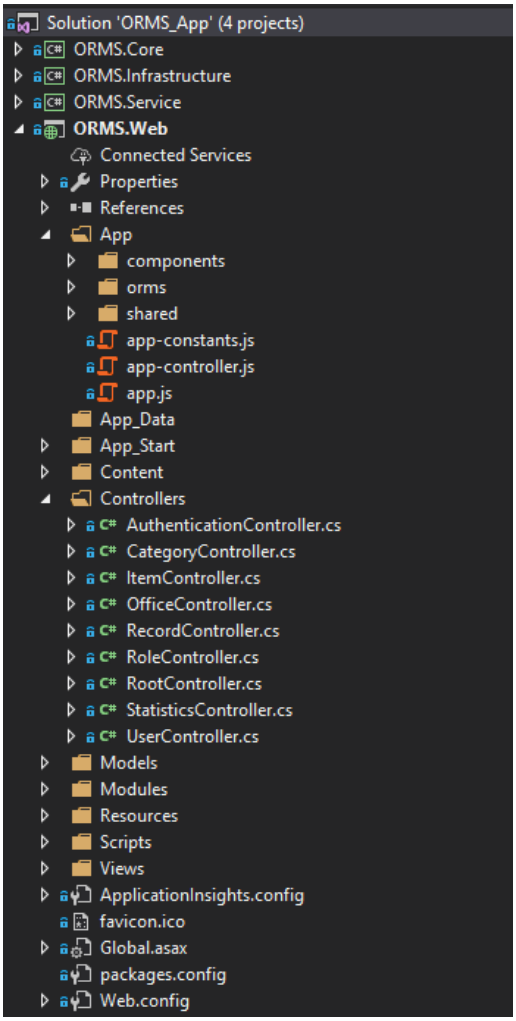
```
public static class ServiceFactory
{
    private static T Initialize<T>()
    {
        var container = new UnityContainer();
        container.RegisterType<IUnitOfWork,
        UnitOfWork>();
        container.RegisterType<ORMSContext>();
        return container.Resolve<T>();
    }

    public static IAuthenticationService
    CreateAuthenticationService()
```

```
        {
            return Initialize<AuthenticationService>();
        }
        ...
}
```

Kôd 5.8 Primjer implementacije `ServiceFactory` klase

## 5.4. Web

Web sloj predstavlja konačan rezultat sveukupnog razvoja web aplikacije za unošenje i ažuriranje evidencijskih listi. Sadrži AngularJS aplikaciju i Web API kontrolere s kojima će AngularJS aplikacija komunicirati putem HTTP poziva. Konačna struktura web aplikacije izgleda ovako (Slika 5.5):

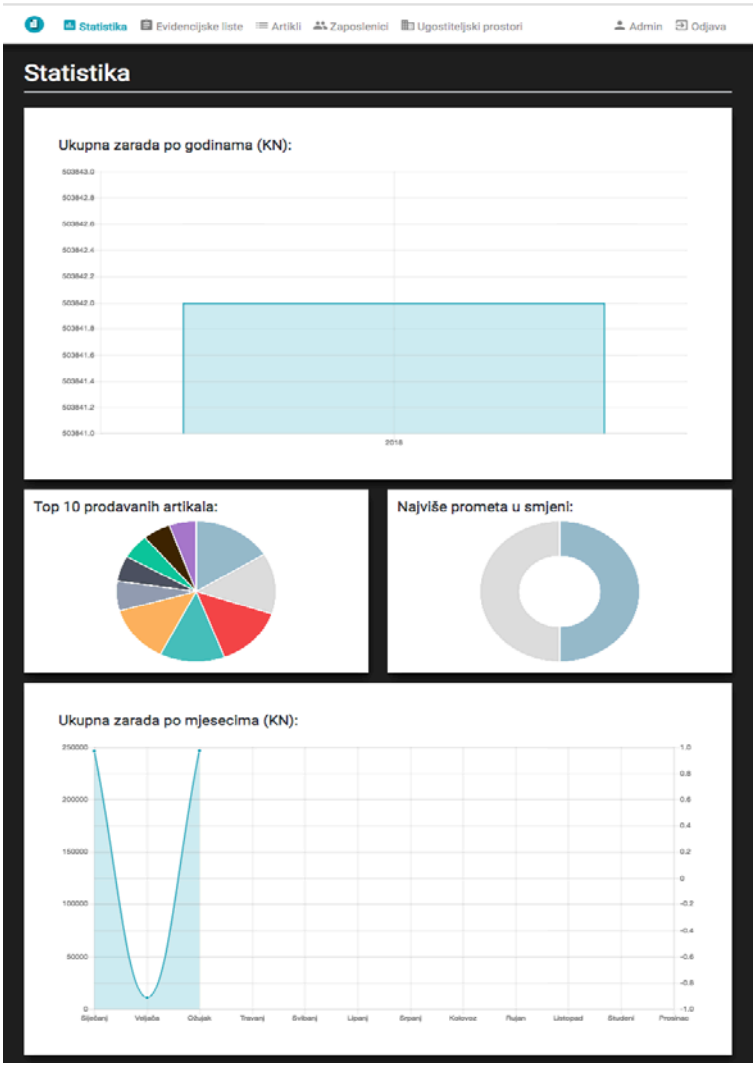



Slika 5.5 Struktura *Web* sloja

Web aplikacija za unošenje i ažuriranje evidencijskih listi sadrži pet modula. Svaki od tih modula ima određenu svrhu u radnom procesu unošenja evidencijskih listi ugostiteljskog prostora. Nadalje, kako bi se dobio bolji dojam aplikacije, svaki modul će se obrađivati posebno i uz njega će biti priložena odgovarajuća slika koja prikazuje njegov izgled u aplikaciji. Moduli aplikacije su:

- **Statistika**

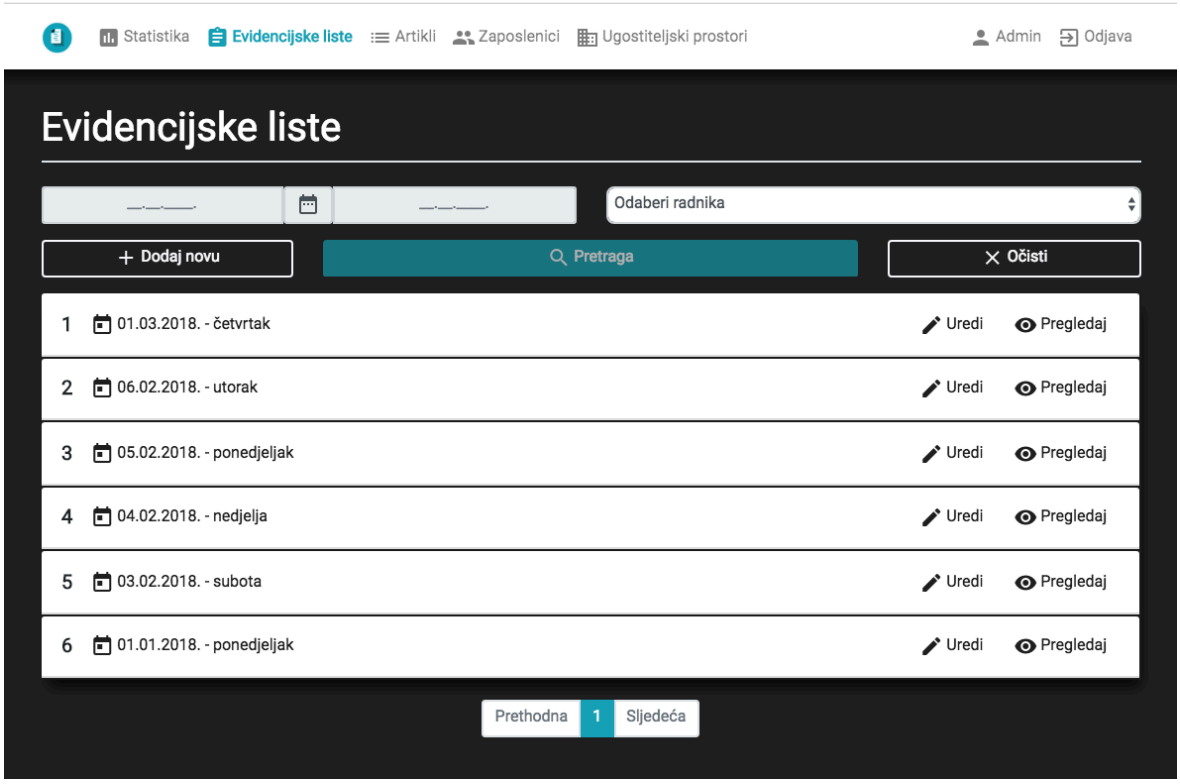
Statistika (Slika 5.6) kao modul obrađuje podatke unesene putem evidencijskih listi i prikazuje ih u obliku stupčastih, linearnih i kružnih grafova. Za prikaz grafova korišten je Angular Chart koji je detaljnije objašnjen u poglavlju 2.6.1. Pomoću ovih grafova korisnik u ulozi vlasnika ili administratora ima uvid u statističke podatke (ukupna zarada po godinama, top deset prodavanih artikala, najviše prometa u smjeni, ukupna zarada po mjesecima) aplikacije.



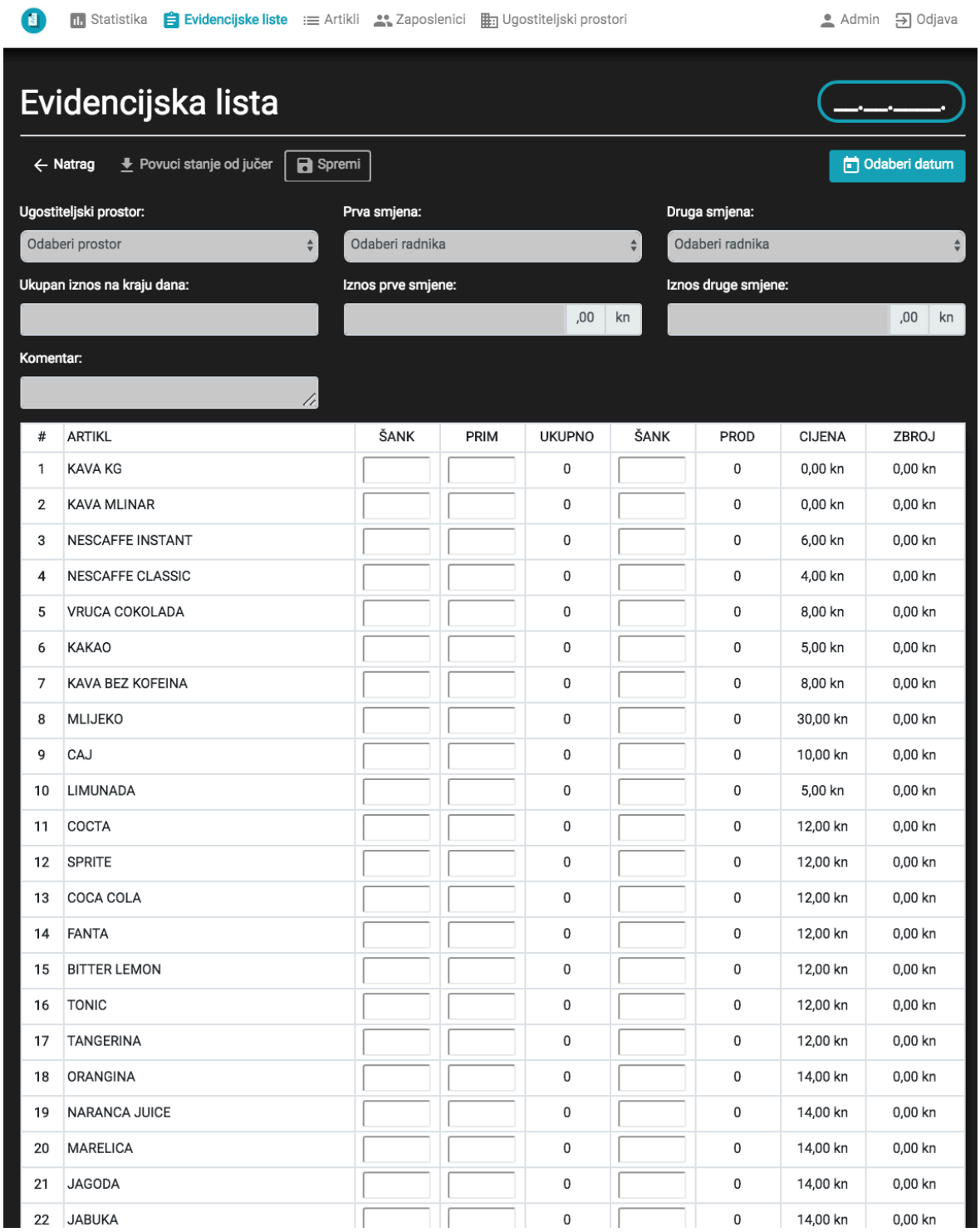

Slika 5.6 Modul – Statistika

- **Evidencijske liste**

Modul Evidencijske liste (Slika 5.7) je srž aplikacije. U ovom modulu korisnik u ulozi administratora, vlasnika, managera ili zaposlenika može unositi i ažurirati evidencijske liste za određeni dan (Slika 5.8). Korisnici u ulozi administratora, vlasnika ili managera mogu pretraživati liste po određenom vremenskom razdoblju te po zaposleniku, dok korisnici u ulozi zaposlenika imaju opciju pretrage samo po vremenskom razdoblju. Evidencijske liste su sortirane od novijih datuma prema starijim datumima. Vlasnik ili administrator ima i opciju izvoza podataka evidencijskih listi ispisom ili skidanjem .csv formata. Za paginaciju evidencijskih listi i artikala korišten je Angular Paging koji je detaljnije opisan u poglavlju 2.6.2.




Slika 5.7 Modul - Evidencijske liste



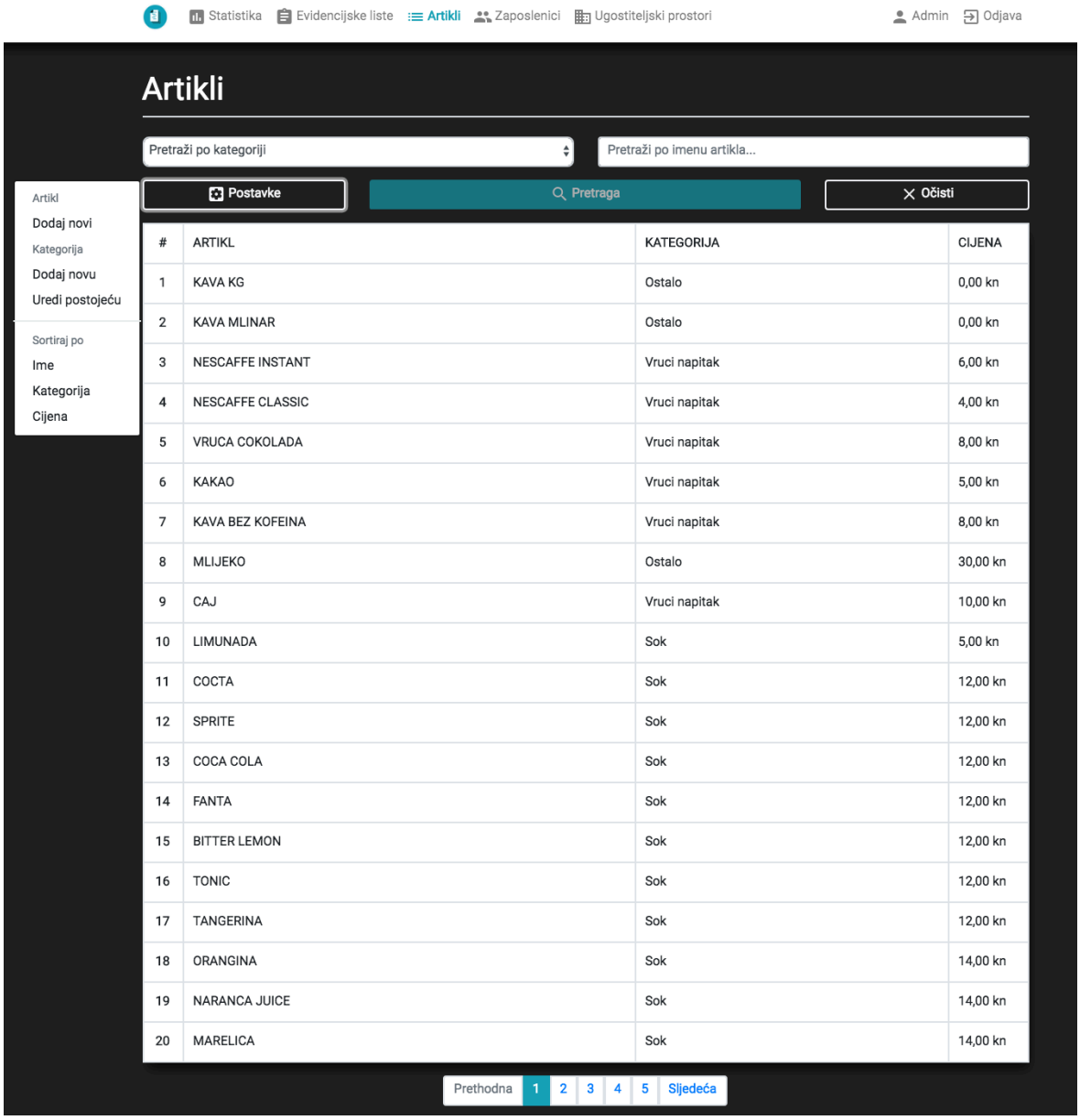

| # | ARTIKL | ŠANK | PRIM | UKUPNO | ŠANK | PROD | CIJENA | ZBROJ |
|---|---|---|---|---|---|---|---|---|
| 1 | KAVA KG | | | 0 | | 0 | 0,00 kn | 0,00 kn |
| 2 | KAVA MLINAR | | | 0 | | 0 | 0,00 kn | 0,00 kn |
| 3 | NESCAFFE INSTANT | | | 0 | | 0 | 6,00 kn | 0,00 kn |
| 4 | NESCAFFE CLASSIC | | | 0 | | 0 | 4,00 kn | 0,00 kn |
| 5 | VRUCA COKOLADA | | | 0 | | 0 | 8,00 kn | 0,00 kn |
| 6 | KAKAO | | | 0 | | 0 | 5,00 kn | 0,00 kn |
| 7 | KAVA BEZ KOFEINA | | | 0 | | 0 | 8,00 kn | 0,00 kn |
| 8 | MLIJEKO | | | 0 | | 0 | 30,00 kn | 0,00 kn |
| 9 | CAJ | | | 0 | | 0 | 10,00 kn | 0,00 kn |
| 10 | LIMUNADA | | | 0 | | 0 | 5,00 kn | 0,00 kn |
| 11 | COCTA | | | 0 | | 0 | 12,00 kn | 0,00 kn |
| 12 | SPRITE | | | 0 | | 0 | 12,00 kn | 0,00 kn |
| 13 | COCA COLA | | | 0 | | 0 | 12,00 kn | 0,00 kn |
| 14 | FANTA | | | 0 | | 0 | 12,00 kn | 0,00 kn |
| 15 | BITTER LEMON | | | 0 | | 0 | 12,00 kn | 0,00 kn |
| 16 | TONIC | | | 0 | | 0 | 12,00 kn | 0,00 kn |
| 17 | TANGERINA | | | 0 | | 0 | 12,00 kn | 0,00 kn |
| 18 | ORANGINA | | | 0 | | 0 | 14,00 kn | 0,00 kn |
| 19 | NARANCA JUICE | | | 0 | | 0 | 14,00 kn | 0,00 kn |
| 20 | MARELICA | | | 0 | | 0 | 14,00 kn | 0,00 kn |
| 21 | JAGODA | | | 0 | | 0 | 14,00 kn | 0,00 kn |
| 22 | JABUKA | | | 0 | | 0 | 14,00 kn | 0,00 kn |

Slika 5.8 Izgled evidencijske liste u aplikaciji

- **Artikli**

Artikli kao modul predstavljaju šifrarnik artikala za koje se unosi stanje u evidencijske liste (Slika 5.9). Pomoću ovog modula mogu se kreirati novi artikli i kategorije artikala te ažurirati postojeći. Također, omogućuje sortiranje artikala po imenu, cijeni i kategoriji kao

i njihovu pretragu po kategoriji i imenu. Ovom modulu mogu pristupiti samo korisnici u ulozi administratora ili vlasnik.

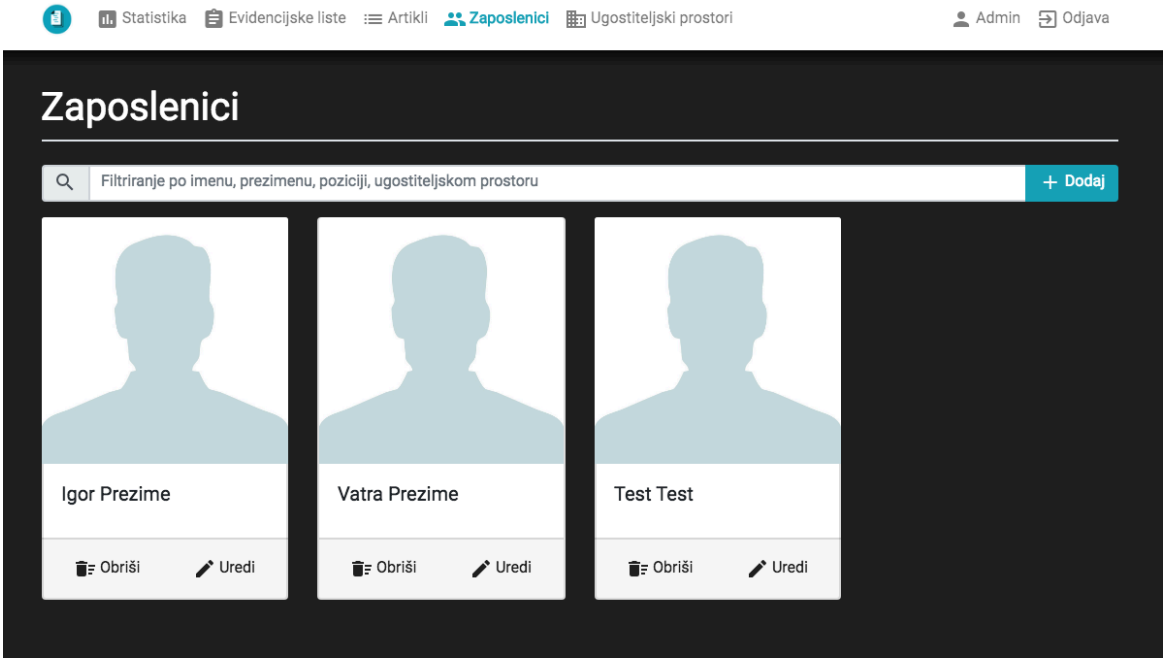



Slika 5.9 Modul – Artikli

- **Zaposlenici**

Modul Zaposlenici (Slika 5.10) bavi se menadžmentom osobnih informacija (ime, prezime, datum rođenja, OIB itd.) zaposlenika koji rade u određenom ugostiteljskom prostoru. Kreiranjem ovog modula omogućili smo vlasniku obrta da ima uvid u bitne informacije svog zaposlenika (npr. datum isticanja sanitarne iskaznice) te da može dodavati nove zaposlenike i ažurirati postojeće. Modul ima opciju filtriranja zaposlenika po imenu,

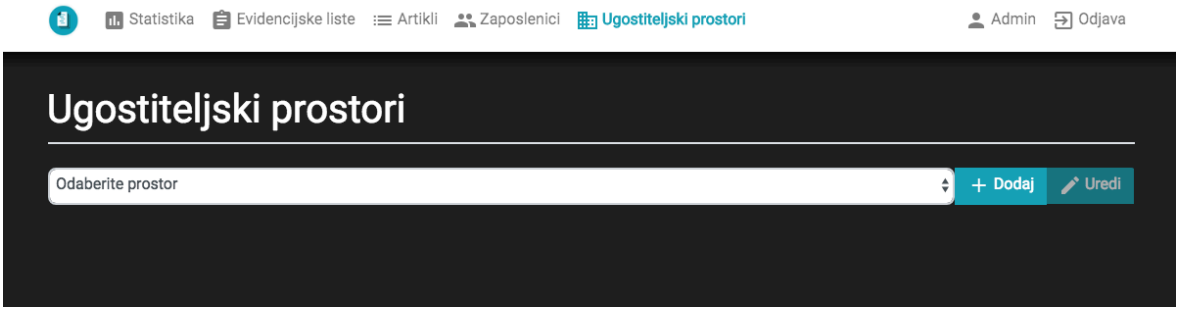
prezimenu, ugostiteljskom prostoru te prema njegovoj ulozi u ugostiteljskom prostoru. Pristup ovom modulu imaju korisnici u ulozi administratora, vlasnika ili managera.




Slika 5.10 Modul – Zaposlenici

- **Ugostiteljski prostori**

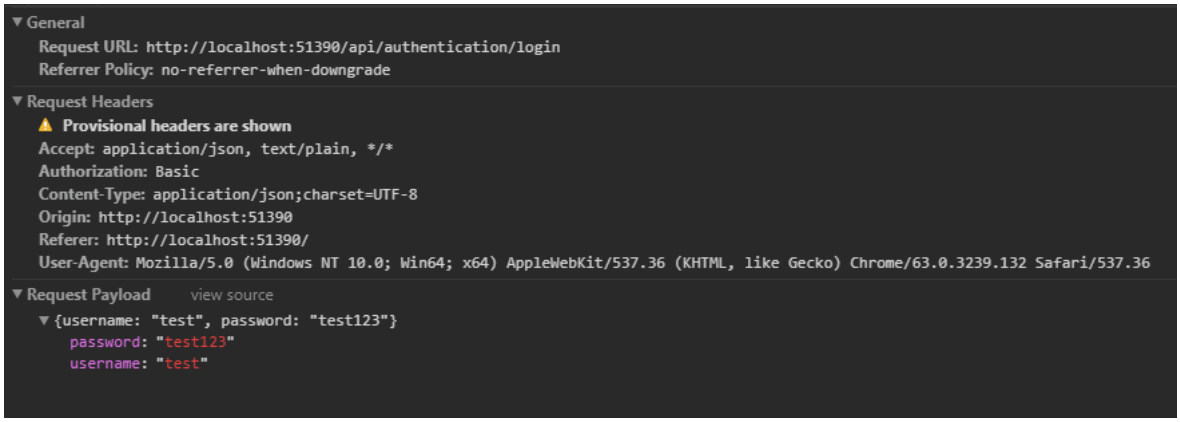
Ugostiteljski prostori (Slika 5.11), kao zadnji modul u aplikaciji, predstavlja menadžment ugostiteljskih prostora nekog obrta. Podatkovno je najmanji modul. Sadrži osnovne informacije o prostoru i služi za dodavanje novih prostora te ažuriranje postojećih. Pristup ovom modulu imaju samo korisnici u ulozi administratora ili vlasnika.




Slika 5.11 Modul - Ugostiteljski prostori

Nadalje, uzevši u obzir količinu koda i kompleksnost razvoja ovog sloja, prikazat će se samo jedan primjer komunikacije između AngularJS-a i Web API. Za primjer će se uzeti prijava korisnika u sustav. Prijava korisnika započinje od stranice za prijavu i završava pokazivanjem skočnog prozora za uspješnu prijavu (skočni prozor je prikazan pomoću SweetAlert 2 programskog okvira) te prebacivanjem korisnika na početnu stranicu

aplikacije. Kada korisnik unese podatke potrebne za prijavu u aplikaciju klikom na gumb "Prijava" , pomoću AngularJS usluge, podaci se putem HTTP POST metode šalju na Web API (Slika 5.12).




Slika 5.12 Login - zahtjev

Kada zahtjev dođe na Web API prvo se gleda koja je URL ruta pozvana i prema toj ruti usmjeritelj poziva akcijsku metodu određenog kontrolera. U ovom slučaju poziva se akcijska metoda `Login` na `AuthenticationController`-u (Kôd 5.9). Akcijska metoda `Login` prima korisničko ime i lozinku kao parametre. Unutar akcijske metode poziva se autorizacijska metoda novonastalog `IAuthenticationService` sučelja, koja u *Service* sloju obrađuje zahtjevom predane podatke i uspoređuje ih sa podacima iz baze podataka.

```
private IAuthenticationService authService =
ServiceFactory.CreateAuthenticationService();

[HttpPost]
public IHttpActionResult
Login([FromBody]AuthenticationRequest authRequest)
{
      var authResponse =
      authService.Authenticate(authRequest);

      if (authResponse.Success)
      {
            authResponse.Message = ORMSMessage.LOGIN_SUCCESS;
            var response =
            Request.CreateResponse(HttpStatusCode.OK,
            authResponse);
            return ResponseMessage(response);
      }
```

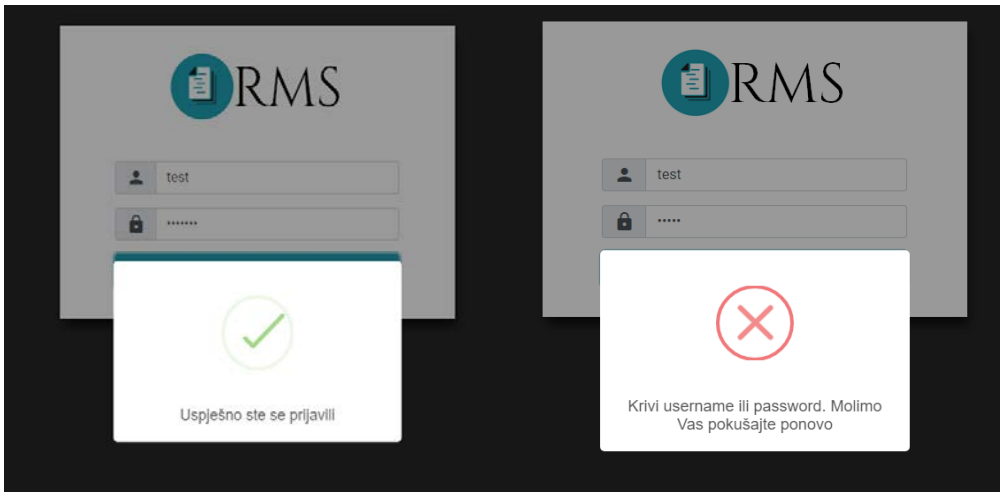
```
            authResponse.Message = ORMSMessage.LOGIN_FAILED;
            var error = Request.CreateResponse(HttpStatusCode.OK,
            authResponse);
            return ResponseMessage(error);
        }
```

Kôd 5.9 Primjer Login akcijske metode

Kada podaci dođu u *Service* sloj, pozivom implementirane metode `unitOfWork.Users.Get(authRequest.Username, authRequest.Password)` (Kôd 5.7) zalazimo u *Infrastructure* sloj koji jedini komunicira sa bazom podataka. Implementirana metoda provjerava postoji li zapis korisnika u bazi podataka te vraća `null` vrijednost ako korisnik ne postoji ili podatke korisnika ako postoji. Na osnovi vrijednosti koju metoda `unitOfWork.Users.Get` vrati, *Service* sloj vraća objekt u obliku odgovora koji sadrži podatke korisnika te dva svojstva koja govore o uspješnosti obrade zahtjeva. Kada vraćeni objekt *Service* sloja dođe ponovo u akcijsku metodu `Login`, ona prema dobivenim informacijama u objektu vraća odgovarajuću poruku i HTTP status klijentu (Slika 5.13).


Slika 5.13 Login – uspješna prijava - odgovor

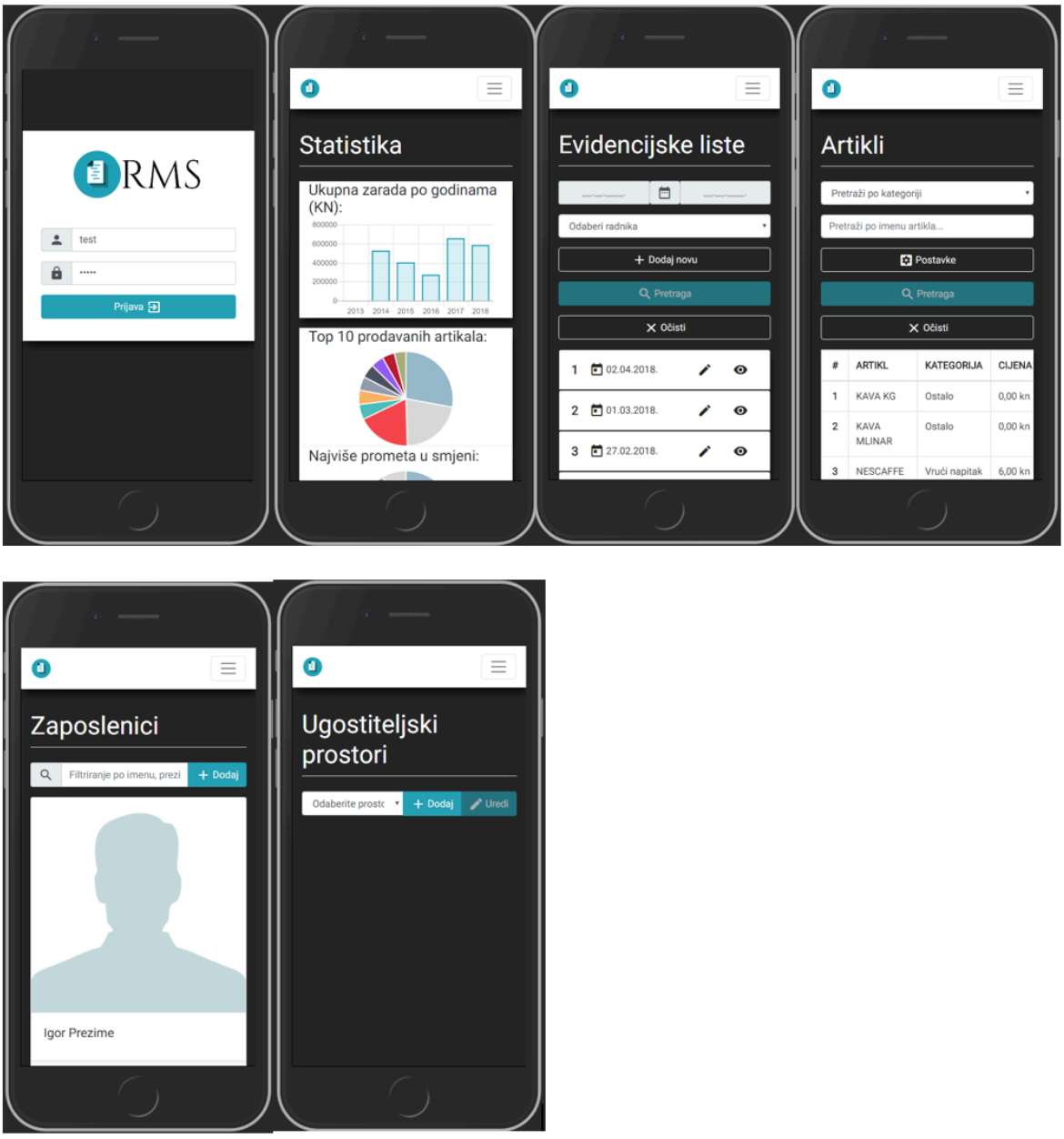
Kada klijent dobije odgovor od Web API-a u obliku JSON objekta, prema dobivenim podacima korisniku prikazuje skočni prozor sa odgovarajućom porukom (Slika 5.14). Ako je došlo do uspješne prijave korisnik je prebačen na početnu stranicu aplikacije.




Slika 5.14 Skočni prozor uspješne i neuspješne prijave korisnika

## 5.5. Mobilna verzija

Koristeći Bootstrap klijentski programski okvir omogućena je responzivna i mobilna verzija aplikacije koja je prikazana pomoću Google Chrome Developer Tools-a (Slika 5.15).




Slika 5.15 Mobilni izgled aplikacije

## 5.6. Logo

Logo (Slika 5.16) web aplikacije za unošenje i ažuriranje evidencijskih listi je napravljen pomoću https://www.freelogodesign.org/index.html web aplikacije.

Slika 5.16 Logo aplikacije

# 6. Testiranje

Testiranje aplikacije za unos internih evidencijskih listi u samom ugostiteljskom objektu nije bilo moguće iz razloga  što je aplikacija objavljena na Azure platformi koja se, prema besplatnoj studentskoj licenci, nalazi u Americi. Iz tog razloga, koji je otežavao normalan rad aplikacije, nije bilo mogućnosti testiranja aplikacije u ugostiteljskom objektu. Slaba internet konekcija ugostiteljskog objekta nije omogućavala ispravan rad aplikacije.

Zbog navedenih razloga nemogućnosti testiranja aplikacije unutar ugostiteljskog objekta, aplikacija je bila testirana od strane vlasnika aplikacije i vlasnika ugostiteljskog objekta. Testiranje se odvijalo unutar stambenog objekta sa dobrom internet konekcijom. U razdoblju od pet radnih dana, aplikacija je bila testirana tako da je vlasnik aplikacije unosio podatke koji su odražavali trenutačno i stvarno stanje robe unutar ugostiteljskog objekta. Uvid u te podatke, nakon svakog radnog dana, vlasniku aplikacije uručio je vlasnik ugostiteljskog objekta. Vlasnik ugostiteljskog objekta je potom pristao da upisuje podatke sa računa fiskalne blagajne na papir evidencijske liste. U isto vrijeme, vlasnik aplikacije je počeo unositi podatke sa istog ispisa fiskalne blagajne u aplikaciju. Tijekom pet radnih dana, štopericom su bilježeni podaci upisivanja evidencijskih listi u aplikaciju i upisivanja evidencijskih listi na papire. Došlo se do zaključka, da u prosjeku, upisivanje podataka sa ispisa fiskalne blagajne u aplikaciju zahtjeva 10 do 15 minuta. Vrijeme prijepisa sa ispisa fiskalne blagajne na evidencijski listu (papir), mjereno je štopericom te se došlo do zaključka da u prosjeku, za jedan upis, potom preračunavanje i provjeru treba 20 do 25 minuta. Takvi rezultati mogu se objasniti činjenicom da je u aplikaciji omogućeno automatsko zbrajanje stanja robe kako bi se došlo do informacije o ukupnoj zaradi za određeni artikl, pa određenom danu. Prema dobivenim rezultatima testiranja hipoteze postavljene u uvodu rada su prihvaćene.

# Zaključak

Korištenjem modernih tehnologija i alata kao što su Visual Studio, Azure, AngularJS i Entity Framework te korištenjem dobrih praksi kodiranja kao što su principi dizajna i obrasci dizajna moguće je napraviti web aplikaciju koja će poboljšati vođenje ugostiteljskog objekta.

Uvođenjem web aplikacija koje koriste moderne tehnologije koje su u skladu s vremenom u poslovne procese ugostiteljskih objekata pokazalo se kako se i mali poslovni procesi unutar ugostiteljskih objekata mogu automatizirati. Prilikom korištenja i testiranja aplikacije od strane vlasnika ugostiteljskog objekta za kojeg je i prvobitno napravljena aplikacija došlo se do slijedećeg zaključka. Smanjile su se moguće greške napravljene od strane ljudskog faktora, ubrzao se radni proces i došlo je do smanjenja potrošnje papira. Također, olakšalo se vođenje ugostiteljskog objekta i arhiviranje internih evidencijskih listi o trenutačnom stanju robe.

Gledajući na budućnost ove aplikacije, aplikacija bi mogla biti implementirana i kod drugih ugostiteljskih objekata koji imaju slični princip vođenja evidencijskih listi. Implementacijom ove aplikacije drugim bi se vlasnicima koji imaju više ugostiteljskih objekata olakšao rad, vođenje objekta, pojednostavio bi se i ubrzao proces unošenja trenutnog stanja robe na kraju radnog dana.

Aplikacija je dostupna na https://orms.azurewebsites.com/.

# Popis kratica

| | | |
|---|---|---|
| IDE | *Interactive Development Environment* | interaktivno razvojno okruženje |
| IaaS | *Infrastructure as a Service* | infrastruktura kao usluga |
| SaaS | *Software as a Service* | softver kao usluga |
| PaaS | *Platform as a Service* | platforma kao usluga |
| HTTP | Hypertext Transfer Protocol | protokol prijenosa informacija |
| HTTPS | Hypertext Transfer Protocol Secure | osigurani protokol prijenosa informacija |
| SQL | *Structured Query Language* | jezik za strukturirani upit |
| HTML | *Hypertext Markup Language* | tekstualni označni jezik |
| CSS | *Cascading Style Sheets* | kaskadni stilski listovi |
| API | *Application Programming Interface* | programsko sučelje aplikacije |
| XML | *eXtensible Markup Language* | jezik za označavanje podataka |
| JSON | *JavaScript Object Notation* | JavaScript označavanje objekata |
| ORM | *Object Relational Mapper* | objektni relacijski maper |
| CRUD | *Create Retrieve Update Delete* | kreiraj dohvati ažuriraj obriši |
| RPC | *Remote Procedure Call* | udaljeni poziv procedure |
| UML | *Unified Modeling Language* | jedinstveni jezik modeliranja |
| DRY | *Don't Repeat Yourself* | nemoj se ponavljati |
| DIP | *Dependency Inversion Principle* | princip inverzije ovisnosti |
| URL | *Uniform Resource Locator* | jedinstveni lokator resursa |

# Popis slika

# Popis tablica

# Popis kodova

# Literatura


[1] Hrvatska enciklopedija, Tehnologija, http://www.enciklopedija.hr/Natuknica.aspx?ID=60658, veljača. 2018.

[2] Microsoft, Visual Studio IDE Overview, 2018., https://docs.microsoft.com/en-us/visualstudio/ide/visual-studio-ide, veljača. 2018.

[3] Microsoft, Azure Developer Guide, 2017., https://docs.microsoft.com/en-us/azure/guides/developer/azure-developer-guide, veljača. 2018.

[4] Microsoft, SQL Server Documentation, 2017., https://docs.microsoft.com/en-us/sql/sql-server/sql-server-technical-documentation, veljača. 2018.

[5] Microsoft, ASP.NET Overview, 2010., https://docs.microsoft.com/hr-hr/aspnet/overview, veljača. 2018.

[6] Đambić, G., Zdešić, V. *Razvoj web aplikacija: priručnik*. Zagreb: Algebra, 2014.

[7] Microsoft, Getting started with ASP.NET Web API 2, 2017., https://docs.microsoft.com/hr-hr/aspnet/web-api/overview/getting-started-with-aspnet-web-api/tutorial-your-first-web-api, veljača. 2018.

[8] Microsoft, Introduction to Entity Framework, 2016., https://msdn.microsoft.com/en-us/library/aa937723(v=vs.113).aspx, veljača. 2018.

[9] Entity Framework Tutorial, Context Class in Entity Framework, http://www.entityframeworktutorial.net/basics/context-class-in-entity-framework.aspx, veljača. 2018.

[10] AngularJS, AngularJS Developer Guide, https://code.angularjs.org/snapshot-stable/docs/guide, veljača. 2018.

[11] https://github.com/brantwills/Angular-Paging, veljača. 2018.

[12] http://jtblin.github.io/angular-chart.js/, veljača. 2018.

[13] http://www.chartjs.org/, veljača. 2018.

[14] https://getbootstrap.com/, veljača. 2018.

[15] https://sweetalert2.github.io/, veljača. 2018.

[16] Microsoft, Unity Container, 2013., https://msdn.microsoft.com/en-us/library/ff647202.aspx, veljača. 2018.

[17] Kaštelan, T. *Uvod u baze podataka: priručnik.* Zagreb: Algebra, 2010.

[18] Microsoft, Clustered and Nonclustered Indexes Described, 2017., https://docs.microsoft.com/en-us/sql/relational-databases/indexes/clustered-and-nonclustered-indexes-described, veljača. 2018.

[19] Millet, S. *Professional ASP.NET Design Patterns.* Indianapolis: Wiley Publishing, Inc., 2010.

[20] Microsoft, Designing Web Applications, https://msdn.microsoft.com/en-us/library/ee658099.aspx, veljača. 2018.

[21] Karatoprak, Y. Difference Between Loose Coupling and Tight Coupling, 2012., http://www.c-sharpcorner.com/UploadFile/yusufkaratoprak/difference-between-loose-coupling-and-tight-coupling/, veljača. 2018.

[22] Kumar, D. How to Implement the Repository Pattern in ASP.NET MVC Application, 2016., https://www.infragistics.com/community/blogs/b/dhananjay_kumar/posts/how-to-implement-the-repository-pattern-in-asp-net-mvc-application, veljača. 2018.

[23] Vihite, C. Understanding Onion Architecture, 2015., http://blog.thedigitalgroup.com/chetanv/2015/07/06/understanding-onion-architecture/, veljača. 2018.