APLIKACIJA ZA UPRAVLJANJE AUTOSERVISOM

Pupak, Luka

VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

APLIKACIJA ZA UPRAVLJANJE AUTOSERVISOM

Luka Pupak

Zagreb, listopad 2019.

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, 30. listopada 2019.

Luka Pupak

Predgovor

Zahvaljujem svome mentoru Bojanu Fulanoviću, dipl. ing., koji mi je pomogao svojim savjetima prilikom oblikovanja teme i izrade ovog završnog rada.

Također zahvaljujem i svim ostalim predavačima Visokog učilišta Algebra na svemu prenesenom znanju i iskustvu.

Posebnu zahvalnost iskazujem svojim roditeljima i sestri koji su uvijek bili uz mene i bili mi podrška tijekom cijelog perioda studiranja.

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

Sažetak

Tema ovog rada razvoj je softverskog rješenja koje bi služilo kao centralni informacijski sustav u radu autoservisa te time omogućilo njegovu veću produktivnost. Opisan je razvoj baze podataka, *web* aplikacijskog programskog sučelja (engl. *web application programming interface,* skraćeno *web* API) te *web* i mobilne aplikacije korištenjem modernih klijentskih i serverskih tehnologija kao što su ASP.NET Web API, Entity Framework, React i React Native.

Ključne riječi: autoservis, *web* aplikacija, mobilna aplikacija, ASP.NET, *web* API, Entity Framework, React, React Native

Abstract

This paper describes the development of a software solution that would serve as a central information system for an auto repair shop and therefore increase its productivity. Described is the development of a database, web application programming interface and also web and mobile application using modern client and server technologies such as ASP.NET Web API, Entity Framework, React and React Native.

Keywords: auto repair shop, web application, mobile application, ASP.NET, Web API, Entity Framework, React, React Native

Sadržaj

1. Uvod

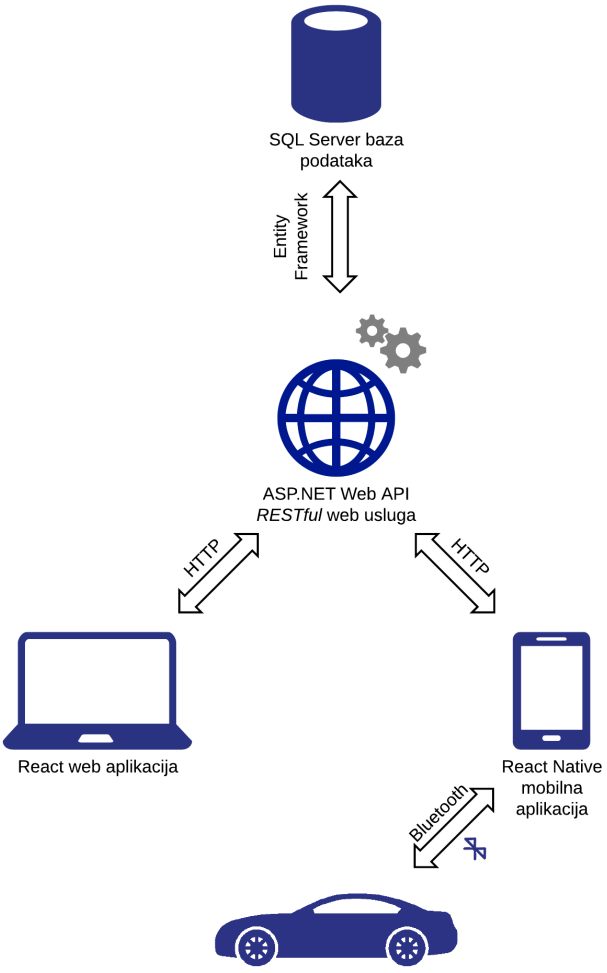
Informacijski sustavi važna su okosnica svakog poslovanja, pa tako i poslovanja autoservisa. Sustavi koji se trenutno koriste zastarjeli su te su neintuitivni za korištenje, stoga je cilj ovoga rada uz pomoć novih i modernih tehnologija i alata stvoriti kvalitetno softversko rješenje koje unaprjeđuje svakodnevno poslovanje autoservisa.

Neke od funkcionalnosti softverskog rješenja koje je razvijeno za potrebe ovog završnog rada su: a) praćenje podataka o klijentima i njihovim automobilima, b) praćenje podataka o servisnoj povijesti automobila, c) praćenje podataka o zaposlenicima, d) dijagnosticiranje kôdova kvara automobila, e) upravljanje podacima o dijelovima u skladištu, f) izrada ponuda, g) izrada radnih naloga te h) izrada izvještaja.

Rad je koncipiran na način da je prvo prikazana struktura praktičnog dijela završnog rada. Zatim su objašnjene korištene tehnologije i alati. Nakon toga opisana je struktura baze podataka razvijena za potrebe ovog završnog rada. Nadalje su opisane *web* i mobilna aplikacija. Naposlijetku se nalazi osvrt na testiranje kod potencijalnih korisnika te zaključak.

2. Struktura praktičnog dijela završnog rada

Praktični dio ovog završnog rada čini informacijski sustav koji se sastoji od četiri glavne komponente, a to su baza podataka, *web* aplikacijsko programsko sučelje (engl. *application programming interface*, skraćeno API) te *web* i mobilna aplikacija. Baza podataka je osnova informacijskog sustava koju *web* API koristi za skladištenje podataka kojima zatim poslužuje dva klijenta – *web* i mobilnu aplikaciju. Mobilna aplikacija bežičnim putem komunicira s automobilom tako što se spaja na adapter za dijagnostiku koji se nalazi u automobilu.



Slika 2.1 Dijagram odnosa komponenti sustava

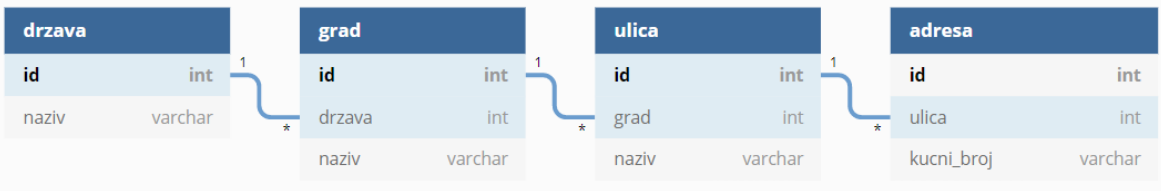
Na slici (Slika 2.1) je prikazan dijagram odnosa komponenti sustava. Tehnologija koja je korištena za razvoj ovog sustava objašnjena je u sljedećem poglavlju.

3. Korištene tehnologije

U ovom poglavlju objašnjene su tehnologije i alati korišteni prilikom izrade praktičnog dijela ovog završnog rada.

3.1. Baza podataka

Baza podataka je skup međusobno povezanih podataka te predstavlja osnovu informacijskog sustava. Postoji više modela baza podataka od kojih je najraširenij relacijski model u kojemu se baza podataka sastoji od skupa međusobno povezanih tablica, odnosno relacija. Relacije se mogu podijeliti na relacije jedan-prema-jedan, jedan-prema-više te više-prema-više. Relacija jedan-prema-jedan označava relaciju u kojoj je zapis u jednoj tablici povezan s jednim zapisom u drugoj tablici. Relacija jedan-prema-više je slična, s tim da nema ograničenja broja zapisa u drugoj tablici. Relacija više-prema-više povezuje više zapisa u jednoj tablici s više zapisa u drugoj tablici. Primjer relacije jedan-prema-više u bazi podataka prikazan je slikom (Slika 3.1).



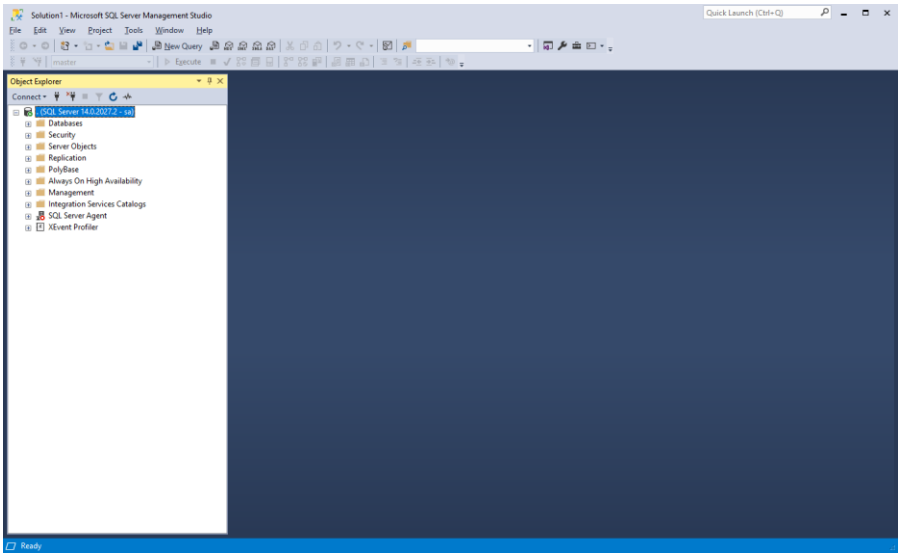
Slika 3.1 Primjer relacije jedan-prema-više u bazi podataka

“Zbog svoje jednostavnosti i prilagođenosti ljudskom shvaćanju podataka i odnosa među njima, relacijski model ima prednost pred ostalim modelima podataka“ (Kašmo, 2014), stoga je to ujedno i model koji se koristi u radu.

3.1.1. Microsoft SQL Server

Kako bismo u relacijsku bazu podataka mogli pohranjivati podatke, odnosno dohvaćati podatke iz nje, potreban nam je sustav za upravljanje relacijskom bazom podataka (engl. *relational database management system*, skraćeno RDBMS). RDBMS koji ćemo koristiti je Microsoftov SQL Server koji je jedan od najkorištenijih RDBMS-ova danas (2019 Database Trends, ožujak, 2019). Za komuniciranje s bazom podataka SQL Server koristi programski jezik visoke razine Transact-SQL (skraćeno T-SQL), koji je ekstenzija strukturnog upitnog jezika (engl. *Structured Query Language*, skraćeno SQL). SQL upiti mogu se podijeliti u pet

kategorija, a to su upiti za dohvaćanje podataka, upiti za umetanje, izmjenu te brisanje podataka, upiti za definiranje sheme baze podataka, upiti za upravljanje korisničkim pravima te upiti za upravljanje transakcijama (Types of SQL Commands, n.d.). Skup SQL upita naziva se SQL skripta. Iako je SQL skriptu moguće napisati u bilo kojem uređivaču teksta i zatim izvršiti koristeći *sqlcmd* alat koji se instalira uz SQL Server, efikasnije je koristiti zasebno integrirano okruženje za razvoj baze podataka kao što je SQL Server Management Studio (skraćeno SSMS), prikazano slikom (Slika 3.2). U SSMS-u objedinjeni su uređivač skripti i svi alati potrebni za pristup, konfiguraciju i upravljanje SQL Server bazom podatka.



Slika 3.2 Integrirano okruženje SQL Server Management Studio 18

3.2. *Web* API

Web API je aplikacijsko programsko sučelje s kojim klijenti komuniciraju preko mreže koristeći protokol HTTP.

3.2.1. HTTP

HTTP je protokol koji omogućava razmjenu podataka na mreži. Temeljen je na arhitekturi klijent – poslužitelj, u kojoj klijent (primjerice *web* preglednik) i poslužitelj (primjerice *web* API) međusobno razmjenjuju poruke. Takve se poruke u slučaju klijenta zovu zahtjevi (engl. *requests*), odnosno u slučaju poslužitelja odgovori (engl. *responses*). Struktura zahtjeva i odgovora je slična te se sastoji od početnog retka (engl. *start line*) u slučaju zahtjeva, odnosno statusnog retka (engl. *status line*) u slučaju odgovora te zaglavlja i tijela poruke (HTTP Overview, rujan, 2019). Početni redak zahtjeva sadrži HTTP metodu, jedinstveni

identifikator resursa (engl. *uniform resource identifier*, skraćeno URI) te verziju protokola. Neke od HTTP metoda i njihova namjena navedene su u tablici (Tablica 3.1).

Tablica 3.1 Neke od HTTP metoda i njihova namjena

GET	dohvaćanje resursa
POST	stvaranje novog resursa
PUT	ažuriranje / zamjena resursa
PATCH	djelomično ažuriranje / zamjena resursa
DELETE	brisanje resursa

Statusni redak odgovora sadrži statusni kôd koji upućuje na to je li zahtjev uspješno ispunjen. Neki od statusnih kôdova i njihovo značenje navedeni su u tablici (Tablica 3.2).

Tablica 3.2 Neki od HTTP statusnih kôdova i njihovo značenje

200 OK	zahtjev je uspješno ispunjen
404 Not Found	resurs nije pronađen
302 Found	resurs je pronađen, ali je preseljen na novu adresu

Zaglavlje sadrži dodatne komunikacijske parametre, veličinu i format tijela poruke i sl. Također, može sadržavati i dodatne informacije, kao što je primjerice autorizacijski token kojim se klijent predstavlja poslužitelju. Tijelo poruke čine podaci koje šaljemo poslužitelju, odnosno podaci koje nam poslužitelj vraća.

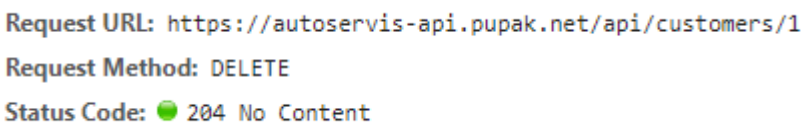
3.2.2. REST stil arhitekture

Representational State Transfer (skraćeno REST) stil je arhitekture *web* API-ja koji je izgrađen oko resursa. REST se u praksi najčešće koristi uz HTTP, iako se u načelu može koristiti i uz druge protokole. Razlog te popularnosti je što se radi o otvorenim standardima, što posljedično donosi veliku prednost, a to je da niti *web* API, niti klijentske aplikacije ne ovise o implementaciji REST-a. To znači da se *web* API baziran na REST-u, tj. *RESTful web* API može razviti koristeći, primjerice, ASP.NET Web API, a klijentske aplikacije mogu biti razvijene u nekom drugom jeziku, odnosno radnom okviru (engl. *framework*). Jedini je uvjet da moraju moći slati HTTP zahtjeve i obrađivati HTTP odgovore. U REST-u svaki resurs ima svoj jedinstveni identifikator, a klijenti putem API-ja izmjenjuju stanja resursa. Resursi mogu istovremeno biti prezentirani u različitim formatima, kao što su npr. XML i JSON.

Postoji nekoliko smjernica kojih bi se trebalo pridržavati prilikom dizajniranja *RESTful* API-ja, a to su:

- Jednoliko sučelje (engl. *uniform interface*) – svi zahtjevi koje klijent šalje API-ju moraju sadržavati sve informacije potrebne API-ju za uspješnu obradu zahtjeva i svaki odgovor koji API šalje klijentu mora sadržavati sve informacije potrebne klijentu za razumijevanje dobivenog odgovora.
- „Klijent – poslužitelj“ – klijentske aplikacije i poslužiteljska aplikacija moraju se moći razvijati neovisno jedna o drugoj. Sve što klijenti trebaju znati je URI resursa.
- Bez stanja (engl. *stateless*) – poslužiteljska aplikacija ne smije pamtiti ikakve podatke o klijentu, niti pratiti stanje prošlih zahtjeva.
- Međuspremanje podataka (engl. *caching*) – više klijenata može više puta tražiti isti resurs, stoga bi se takav resurs trebao spremiti u međuspremnik (engl. *cache*) poslužiteljske ili klijentske aplikacije kako bi se izbjegla nepotrebna obrada podataka i time poboljšale performanse.
- Slojevit sustav (engl. *layered system*) – između klijenta i poslužitelja mogu se nalaziti posrednici u vidu dodatnih slojeva sustava kao npr. autentifikacijski sloj, sloj za raspodjelu opterećenja (engl. *load-balancing layer*) i sl., te oni ne smiju utjecati na zahtjeve i odgovore između klijenata i API-ja (Representational State Transfer, 2000).

Na slici (Slika 3.3) je prikazan primjer HTTP zahtjeva za brisanjem resursa korištenjem HTTP metode DELETE te odgovor API-ja o uspješno obrađenom zahtjevu uz statusni kôd *204 No Content*.



Slika 3.3 Primjer HTTP zahtjeva i odgovora

Mnoštvo radnih okvira (engl. *frameworks*) omogućava razvoj *RESTful web* API-ja, a u ovom završnom radu korišten je ASP.NET Web API radni okvir.

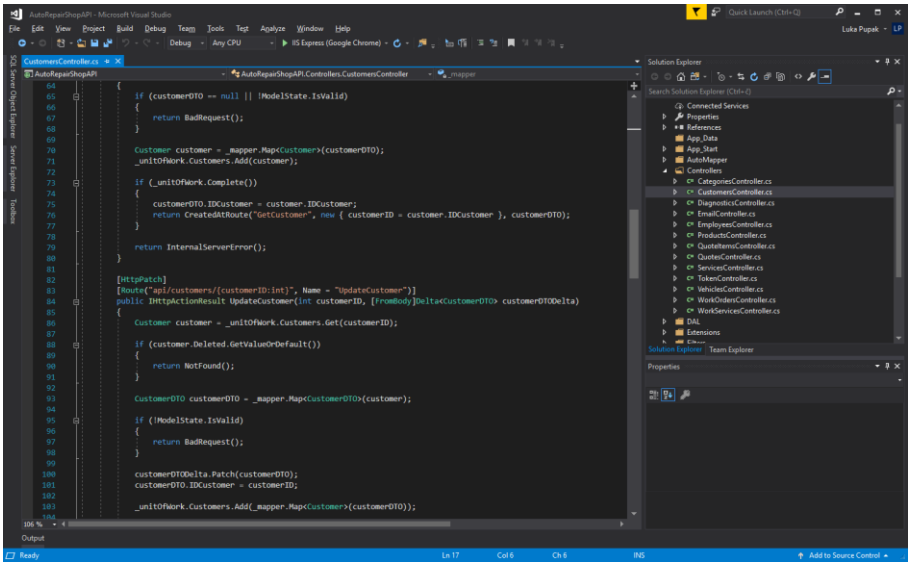
3.2.3. ASP.NET Web API

ASP.NET Web API dio je Microsoftovog ASP.NET radnog okvira (engl. *framework*) koji omogućuje izradu *RESTful* web API-ja (ASP.NET Overview, studeni, 2017).

Kada ASP.NET Web API primi HTTP zahtjev, provjerava URI unutar zahtjeva te se na osnovi tablice ruta zahtjev prosljeđuje na predviđeni *kontroler* te na točno predviđenu metodu, tj. *akciju* unutar kontrolera. Akcija obrađuje zahtjev te vraća odgovor klijentu u nekom od podržanih formata kao što su XML i JSON, ovisno o tome koji je format klijent zatražio putem *Accept* zaglavlja unutar HTTP zahtjeva. Odgovor može biti tekstualni, brojčani, binarni i dr. Bitna značajka (engl. *feature*) ASP.NET Web API-ja je *model binding*. *Model binding* omogućuje akciji da kao parametar primi model. Model je klasa koja predstavlja podatke i poslovnu logiku. Isto tako, akcija može kao odgovor vratiti model, a ASP.NET Web API će ga prije slanja serijalizirati u predviđeni format (Getting Started with ASP.NET Web API, siječanj, 2012).

Web API koji je razvijen za potrebe završnog rada, razvijen je korištenjem ovog radnog okvira u kombinaciji sa C# programskim jezikom[1].

Za uređivanje programskog kôda korišteno je Microsoft Visual Studio integrirano razvojno okruženje (engl. *integrated development environment*, skraćeno IDE), koje je prikazano na slici (Slika 3.4).



Slika 3.4 Microsoft Visual Studio 2017 IDE

[1] C# je Microsoftov objektno orijentirani programski jezik.

3.2.4. SimplePatch

Kako omogućiti djelomično ažuriranje nekog entiteta jedno je od uobičajenih pitanja prilikom razvoja *RESTful web* usluge korištenjem ASP.NET Web API-ja. SimplePatch je biblioteka koja to na jednostavan način omogućava. Primjerice, ako imamo entitet `Korisnik` koji sadrži svojstva `Ime`, `Prezime`, `Email` i `Lozinka`, te želimo ažurirati vrijednosti svojstava `Email` i `Lozinka`, umjesto slanja zahtjeva *web* API-ju koji sadrži sve podatke, SimplePatch omogućava slanje samo onih podataka koje želimo ažurirati, a to su u ovom primjeru `Email` i `Lozinka`. Mogućnost takvog načina ažuriranja entiteta olakšava razvoj klijentskih aplikacija koje koriste API.

3.3. Entity Framework

Entity Framework je radni okvir za objektno-relacijsko preslikavanje (engl. *object-relational mapping*, skraćeno ORM) koji je razvio Microsoft. Služi za olakšavanje rada s bazom podataka iz programskog kôda, tj. za povezivanje aplikacije s bazom podataka. Postoje tri pristupa u korištenju Entity Frameworka, a to su *Code First*, *Model First* te *Database First* (Overview of Entity Framework, listopad, 2016). *Database First* pristup, koji je korišten u radu za povezivanje *web* API-ja s bazom podataka, preslikava strukturu već postojeće baze podataka u domenske modele, omogućava jednostavnu manipulaciju podacima korištenjem objekata unutar programskog kôda te eliminira potrebu za pisanjem SQL upita.

3.4. AutoMapper

AutoMapper je biblioteka (engl. *library*) koja kroz jednostavnu konfiguraciju, unutar programskog kôda, omogućava dvosmjerno preslikavanje jednog objekta na drugi. Time se eliminira ponavljanje kôda na više različitih mjesta i olakšava održavanje. U radu je AutoMapper korišten za preslikavanje objekata domenskih modela na objekte za prijenos podataka (engl. *data transfer object*, skraćeno DTO[2]).

[2] DTO je objekt koji služi za prijenos podataka između različitih komponenti sustava, primjerice za prijenos podataka između *web* API-ja i klijentskih aplikacija.

3.5. Simple Injector

Simple Injector je biblioteka, tj. *container* koji je u radu korišten za injektiranje ovisnosti (engl. *dependency injection*, skraćeno DI). DI je tehnika u programiranju koja kroz mapiranje sučelja (engl. *interface*) s konkretnim implementacijama sučelja omogućava veću fleksibilnost kôda. DI na temelju konfiguracije automatski pridružuje instance konkretne implementacije sučelja o kojima neka klasa ili metoda ovisi (Design Patterns Explained, n.d.). Time se omogućava razvoj slabo povezane (engl. *loosely-coupled*) aplikacije, što olakšava daljnje održavanje.

3.6. JSON Web Token

JSON Web Token (skraćeno JWT) je standard za kreiranje pouzdanih autentifikacijskih i autorizacijskih tokena u JSON formatu. Struktura tokena je kompaktna te se sastoji od tri dijela:

- zaglavlja – dio koji sadrži informacije o vrsti tokena te o algoritmu koji je korišten za stvaranje potpisa
- podataka – dio koji sadrži tvrdnje (engl. *claims*) o nekom entitetu, tipično o korisniku
- potpisa – potpis na temelju kojeg poslužitelj provjerava integritet podataka (Introduction to JSON Web Tokens, n.d.).

U radu je JWT korišten za autentifikaciju i autorizaciju korisnika *web* i mobilne aplikacije.

3.7. React

React je JavaScript[3] biblioteka koju je razvio Facebook, a namijenjena je za razvoj korisničkih sučelja. Kako se radi o biblioteci, a ne o radnom okviru (kao što je Angular s kojim se React često uspoređuje), React ne nameće odabir popratne tehnologije, već ga je moguće koristiti u kombinaciji s ostalim bibliotekama po želji. Tako je moguće odabrati biblioteku za HTTP komunikaciju s poslužiteljem, biblioteku za usmjeravanje (engl. *routing*) URL-ova unutar aplikacije i slično.

[3] JavaScript je skriptni programski jezik.

React se bazira na komponentama. Komponente su višestruko upotrebljivi dijelovi programskog kôda koji se slažu u veće cjeline, što znači da je React aplikacija jedna velika komponenta. Svaka komponenta unutar hijerarhije komponenti ima svoju logiku i *stanje*. Stanje komponente je stanje u kojem se ona nalazi u određenom trenutku, a sadrži podatke potrebne za njen prikaz (What is React?, 2019).

Primjerice, prilikom prvog učitavanja *web* aplikacije, aplikacija se nalazi u početnom stanju. Svaka akcija, poput otvaranja izbornika, pritiskanja dugmadi (engl. *buttons*) i sl., događaj je koji može promijeniti stanje aplikacije. Promjena podataka unutar stanja aplikacije rezultira osvježavanjem samo onih dijelova *web* aplikacije kojih se ta promjena tiče te je tako omogućen razvoj brzog i interaktivnog sučelja.

React je u radu korišten za razvoj klijentskog dijela *web* aplikacije.

3.7.1. React Router

React Router je biblioteka bazirana na Reactu koja omogućava jednostavno usmjeravanje URL-ova prema komponentama unutar aplikacije. Time je omogućeno stvaranje jednostranične aplikacije (engl. *single page application*, skraćeno SPA) čiji se sadržaj mijenja ovisno o trenutnom URL-u. Za razliku od višestranične aplikacije (engl. *multiple-page application*, skraćeno MPA), SPA se prilikom promjene URL-a ne učitava ponovno, već se dinamički mijenja dio sadržaja (React Router Quick Start, n.d.).

3.7.2. AG Grid React

AG Grid React je React komponenta koja omogućava stvaranje prilagodljivih preglednih tablica s mogućnostima kao što su filtriranje i prilagođavanje prikaza podataka po svakom stupcu, paginacija, parcijalno osvježavanja podataka i slično. U radu se koristi kao centralni dio većine stranica *web* aplikacije.

3.8. OBD-II

OBD-II je skraćenica za *On-Board Diagnostics II*, sustav za dijagnostiku automobila kojeg su svi automobili prodani unutar Europske unije zakonski dužni imati od 2001. g. u slučaju benzinskih automobila, odnosno od 2004. g. u slučaju dizelskih automobila. OBD-II sustav prati razne parametre ostalih sustava koji se nalaze u automobilu te u slučaju nepredviđenih vrijednosti pohranjuje podatak o grešci. Podatak se sastoji od dijagnostičkog kôda greške

(engl. *diagnostic trouble code*, skraćeno DTC). Pohranjeni DTC-ovi mogu se pročitati putem OBD-II čitača kako bi se ustanovio mogući kvar automobila. OBD-II čitač može biti zaseban uređaj koji izravno na ekranu prikazuje parametre, a može biti i adapter koji ih bežičnim putem šalje na drugi uređaj putem Bluetooth ili Wi-Fi tehnologije. Primjer OBD II Bluetooth adaptera vidljiv je na slici (Slika 3.5).

Slika 3.5 Primjer OBD II Bluetooth adaptera

Komunikacija s OBD-II sustavom je dvosmjerna. Kako bi OBD-II poslao parametre, čitač ih prvo mora zatražiti. Čitač može zatražiti parametar tako da OBD-II sustavu pošalje identifikator parametra (engl. *parameter ID*, skraćeno PID). Sustav zatim vraća vrijednost parametra ili podatak o neuspjelom čitanju, odnosno nepostojanju vrijednosti.

Postoje standardizirani PID-ovi i DTC-ovi čije je značenje poznato, no postoje i oni koje proizvođači automobila koriste za interne svrhe te nisu dostupni javnosti.

OBD II se u radu koristi za dijagnostiku kvarova u automobilu putem mobilne aplikacije.

3.9. React Native

React Native je JavaScript radni okvir baziran na React-u koji omogućuje razvoj višeplatformskih mobilnih aplikacija. Službene platforme koje su trenutno podržane su Android i iOS, no React Native zajednica radi i na podršci za platforme UWP, macOS i tvOS (Out of Tree Platforms, n.d.).

Tri su ključna dijela React Native aplikacije:

- Nativni kôd – kôd specifičan za platformu na kojoj se aplikacija izvodi. U slučaju Androida radi se o Java kôdu, a u slučaju iOS-a o Swiftu ili objektivnom C-u.
- JavaScript virtualna mašina (engl. *virtual machine*, skraćeno VM) – VM unutar koje se izvodi JavaScript kôd.
- React Native most (*React Native Bridge*) – most koji je zadužen za komunikaciju između nativnog kôda i JavaScript VM-a.

Prilikom pokretanja aplikacije pokreće se i JavaScript VM. VM ostvaruje vezu s nativnim kôdom te preslikava React Native komponente na njihove ekvivalentne nativne komponente. Stoga se aplikacije napisane uz pomoć React Nativea ponašaju i izgledaju kao da su napisane specifično za platformu na kojoj se izvode (React Native Internals, n.d.).

React Native u radu se koristi za razvoj mobilne aplikacije.

3.9.1. React Navigation

React Navigation je JavaScript biblioteka koja omogućava stvaranje navigacijske strukture mobilne aplikacije i navigaciju između različitih ekrana. Radi se o projektu otvorenog kôda (engl. *open-source*) koji je razvila React Native zajednica. Iznimno je korisna biblioteka jer pamti stanje aplikacije nakon promjene ekrana, podržava prosljeđivanje svojstava (engl. *properties*) između različitih ekrana i sl.

3.9.2. React Native OBD2

React Native OBD2 je React Native komponenta koja omogućuje povezivanje mobilne aplikacije s OBD-II čitačem putem Bluetooth tehnologije. Komponenta omogućava slanje proizvoljnih PID-ova i primanje odgovora, što je u radu iskorišteno za dijagnostiku kvarova u automobilu (objašnjeno u poglavlju 3.8.).

3.9.3. React Native Camera

React Native Camera je React Native komponenta koja omogućuje korištenje kamere mobilnog telefona unutar mobilne aplikacije u svrhu snimanja fotografija i videa, prepoznavanja lica, teksta i barkôdova. Lako se integrira unutar postojeće aplikacije te sadrži mnoštvo opcija kojima se ponašanje i izgled komponente može prilagoditi potrebama aplikacije. U radu je korištena za skeniranje barkôdova unutar mobilne aplikacije.

3.9.4. NativeBase

NativeBase je kolekcija React Native komponenti za izgradnju korisničkog sučelja. Omogućuje stvaranje boljeg korisničkog iskustva kroz stvaranje konzistentnog izgleda aplikacije na različitim platformama. NativeBase bazira se na nativnim komponentama te je u odnosu na njih promijenjen samo izgled, dok je ponašanje komponenti ostalo isto.

4. Baza podataka

U ovom poglavlju objašnjena je struktura baze podataka aplikacije za upravljanje autoservisom.

Struktura baze podataka razvija se kroz prepoznavanje entiteta, njihovih atributa te veza između entiteta. Entitet predstavlja skup objekata ili događaja iz stvarnog svijeta koji ima neka svojstva, a ta svojstva se unutar terminologije baza podataka zovu atributi (Carić et al., 2015). U relacijskom modelu baza podataka entiteti se modeliraju kao tablice, a atributi kao stupci tih tablica.

4.1. Tablice

Baza podataka aplikacije za upravljanje autoservisom sastoji se od jedanaest tablica:

- **Zaposlenik (engl. *Employee*)**
  Sadrži sve potrebne podatke o zaposlenicima autoservisa kao što su ime, prezime, *e-mail* adresa, lozinka, telefonski broj, adresa stanovanja i razina ovlasti unutar aplikacije.

- **Klijent (engl. *Customer*)**
  Sadrži sve potrebne podatke o klijentima autoservisa kao što su ime, prezime, *e-mail* adresa, telefonski broj i adresa stanovanja.

- **Vozilo (engl. *Vehicle*)**
  Sadrži sve potrebne podatke o vozilima kao što su jedinstveni identifikator vlasnika vozila, proizvođač, model, broj šasije, registracijska oznaka, boja, broj prijeđenih kilometara i godina proizvodnje.

- **Kategorija (engl. *Category*)**
  Sadrži sve potrebne podatke o kategorijama u kojima se mogu nalaziti proizvodi i usluge rada. Ti podaci obuhvaćaju jedinstveni identifikator kategorije, naziv kategorije te jedinstveni identifikator roditeljske kategorije (ukoliko postoji). Takva struktura omogućava neograničeno grananje kategorija u smislu da svaka kategorija može biti unutar neke druge kategorije.

- **Proizvod (engl. *Product*)**

  Sadrži sve potrebne podatke o proizvodima u skladištu kao što su naziv proizvoda, jedinstveni identifikator kategorije proizvoda, cijena, opis, kataloški broj i količina u skladištu.

- **UslugaRada (engl. *WorkService*)**

  Sadrži sve potrebne podatke o uslugama rada autoservisa kao što su naziv usluge rada, jedinstveni identifikator kategorije usluge rada, cijena i opis.

- **Ponuda (engl. *Quote*)**

  Sadrži sve potrebne podatke o ponudama koje su izdane klijentima. Ti podaci obuhvaćaju broj ponude koji se generira sekvencijski unutar baze podataka, jedinstveni identifikator klijenta, jedinstveni identifikator vozila na koje se ponuda odnosi, jedinstveni identifikator zaposlenika koji je stvorio ponudu, datum stvaranja ponude, postotak popusta na ponudu i ukupan iznos ponude prije popusta.

- **StavkaPonude (engl. *QuoteItem*)**

  Sadrži sve potrebne podatke o stavkama koje se nalaze na nekoj ponudi. Stavka može biti proizvod ili usluga rada. Podaci koji se nalaze u ovoj tablici su jedinstveni identifikator proizvoda ili usluge rada i količina.

- **RadniNalog (engl. *WorkOrder*)**

  Sadrži sve potrebne podatke o radnim nalozima. Radni nalog predstavlja prihvaćenu ponudu koja je dodijeljena nekom zaposleniku te je vezana uz određeni termin. Podaci koji se nalaze u ovoj tablici su broj radnog naloga koji se generira sekvencijski unutar baze podataka, jedinstveni identifikator zaposlenika koji je stvorio radni nalog, jedinstveni identifikator zaposlenika kojem je radni nalog dodijeljen, datum i vrijeme stvaranja, opis i podatak je li radni nalog obavljen.

- **Servis (engl. *Service*)**

  Sadrži sve potrebne podatke o obavljenom servisu automobila kao što su stanje kilometara u trenutku servisa, opis servisa te jedinstveni identifikator radnog naloga na osnovu kojeg je servis obavljen.

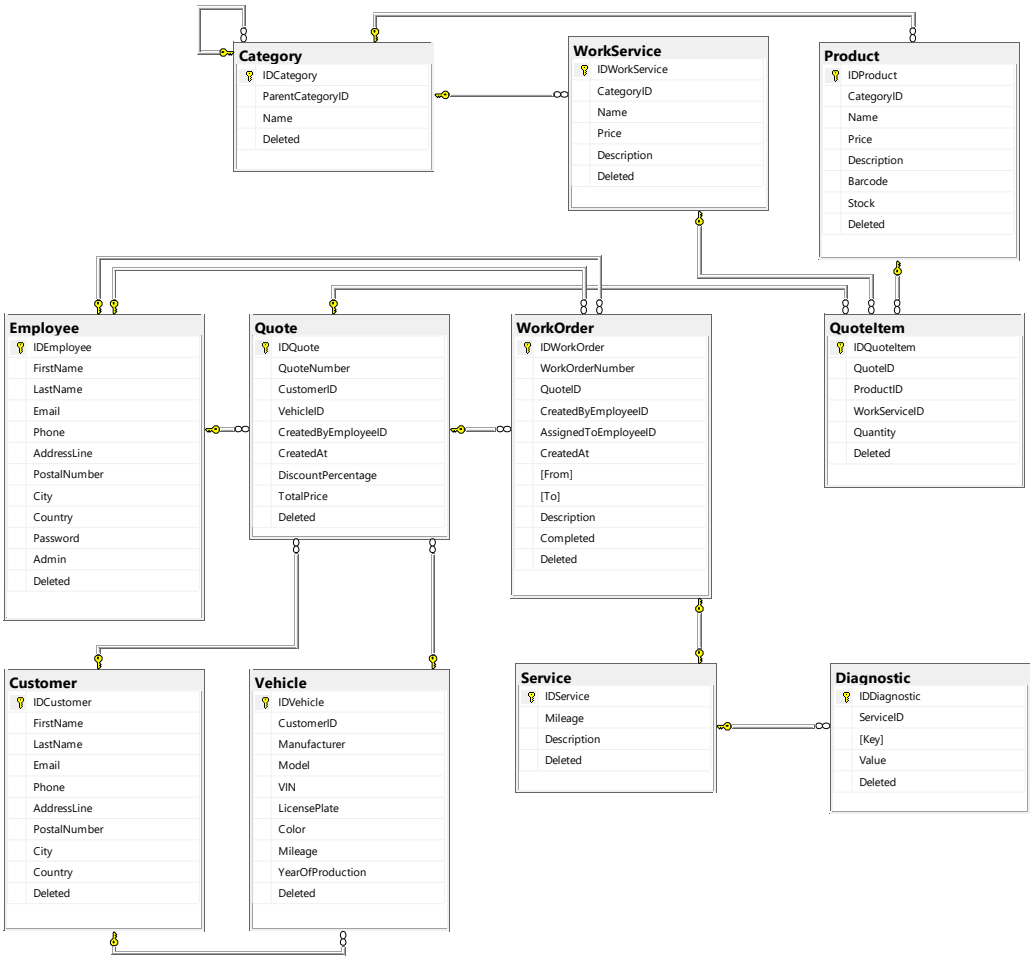
- **Dijagnostika (engl. *Diagnostic*)**

  Sadrži sve potrebne podatke o dijagnostici automobila obavljenoj prilikom određenog servisa. Ti podaci obuhvaćaju naziv dijagnostičkog parametra, njegovu vrijednost te jedinstveni identifikator servisa tijekom kojeg je obavljena dijagnostika.

Sve tablice unutar baze podataka sadrže atribut `Obrisano` (engl. `Deleted`) čija vrijednost predstavlja stanje zapisa. Ako je vrijednost atributa „1“, smatra se da je zapis obrisan. Takva metoda označavanja zapisa unutar baze podataka zove se *soft deletion*, a omogućava pregled i oporavak povijesnih podataka.

4.2. ER dijagram

Relacije između entiteta u bazi podataka mogu se prikazati ER dijagramom (engl. *entity-relationship diagram*). ER dijagram baze podataka prikazan je na slici (Slika 4.1).



Slika 4.1 ER dijagram baze podataka

5. *Web* API

U ovom poglavlju objašnjena je arhitektura te funkcionalnosti *web* API-ja koji predstavlja centralnu točku praktičnog dijela ovog završnog rada.

Web API služi kao poveznica između baze podataka i klijentskih aplikacija. S bazom podataka povezan je uz pomoć Entity Frameworka, a s klijentima preko krajnjih točaka (engl. *endpoints*) koje nudi. Klijenti prema krajnjim točkama šalju HTTP zahtjeve, na koje API zatim odgovara.

Web API je podijeljen na dva sloja – podatkovni i aplikacijski. Kako se radi o relativno jednostavnim operacijama s podacima u repozitoriju, zaseban sloj koji bi sadržavao poslovnu logiku izostavljen je radi jednostavnosti.

5.1. Podatkovni sloj

Podatkovni sloj je sloj za pristup podacima. Implementiran je pomoću obrasca repozitorija (*engl. repository pattern*) i obrasca jedinice rada (engl. *unit of work pattern*, skraćeno UoW), koji su neki od oblikovnih obrazaca (engl. *design patterns*)[4] dizajna programskih rješenja. Repozitoriji predstavljaju sučelja za pristup podacima, dok UoW predstavlja kontekst jedne transakcije unutar koje je moguće koristiti više repozitorija. Svaki entitet autoservisa ima svoj repozitorij koji je implementacija specifičnog sučelja za taj repozitorij, a sva specifična sučelja implementiraju jedno generičko sučelje te je tako stvoreno unificirano sučelje za pristup podacima.

Kôd 5.1 prikazuje generičko sučelje repozitorija koje sadrži metode za dohvaćanje entiteta na osnovu jedinstvenog identifikatora, dohvaćanje svih entiteta, dohvaćanje entiteta na osnovu nekog uvjeta, dodavanje entiteta, dodavanje više entiteta odjednom, brisanje entiteta te brisanje više entiteta odjednom.

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

[4] Oblikovni obrasci su obrasci dizajna softverskog rješenja koji su općeprihvaćeni te se često koriste kako bi riješili neke uobičajene probleme.

```
    IEnumerable<TEntity> Find(Expression<Func<TEntity, bool>> expr);
    void Add(TEntity entity);
    void AddRange(IEnumerable<TEntity> entities);

    void Remove(TEntity entity);
    void RemoveRange(IEnumerable<TEntity> entities);
}
```

Kôd 5.1 Generičko sučelje repozitorija

Entiteti su predstavljeni domenskim modelima koje je Entity Framework stvorio na osnovu tablica koje se nalaze u bazi podataka. Na temelju njih stvorena su specifična sučelja repozitorija za pojedine entitete. Primjerice, za entitet Klijent (engl. *Customer*) stvoreno je specifično sučelje `ICustomerRepository` koje je prikazano kôdom (Kôd 5.2.).

```
public interface ICustomerRepository : IRepository<Customer>
{
    Customer GetWithVehicles(int id);
    Customer GetWithQuotes(int id);

    IEnumerable<Customer> GetAllWithVehicles();
    IEnumerable<Customer> GetAllWithQuotes();

    IEnumerable<Customer> GetAllFilteredByName(string name);
}
```

Kôd 5.2 Primjer specifičnog sučelja repozitorija `ICustomerRepository`

Prethodno navedeni primjer specifičnog sučelja sadrži dodatne metode koje olakšavaju rad s repozitorijem klijenata kao što su dohvaćanje klijenta na osnovu jedinstvenog identifikatora zajedno s njegovim vozilima, dohvaćanje klijenta na osnovu jedinstvenog identifikatora zajedno s ponudama koje su mu izdane, dohvaćanje svih klijenata i njihovih vozila, dohvaćanje svih klijenata i ponuda koje su im izdane te dohvaćanje kolekcije klijenata filtriranih po imenu.

Prilikom transakcija koje uključuju podatke koji se nalaze u različitim repozitorijima želimo da se sve akcije uspješno izvrše ili da se u slučaju greške niti jedna ne izvrši. Primjer takve transakcije u kontekstu autoservisa je zatvaranje radnog naloga. Prilikom zatvaranja radnog naloga potrebno je korigirati količinu proizvoda u skladištu koji su upotrijebljeni/potrošeni prilikom njegovog obavljanja. U slučaju da se iz nekog razloga količina određenog

proizvoda u skladištu ne može ažurirati, ne želimo da se ažurira količina ni jednog proizvoda, niti da se radni nalog označi kao zatvoren. Kako bi to bilo moguće, potrebno je na neki način povezati repozitorije. Jedan od načina je korištenje *Unit of Work* obrasca, koji je prethodno spomenut.

Unit of Work obrazac sastoji se od klase koja objedinjuje instance svih repozitorija i konteksta baze podataka[5] koje navedeni repozitoriji koriste za komuniciranje s bazom. Dio sučelja koje takva klasa implementira, u kontekstu *web* API-ja izrađenog u sklopu ovog završnog rada, prikazan je kôdom (Kôd 5.3).

```
public interface IUnitOfWork : IDisposable
{
      ICustomerRepository Customers { get; }
      IEmployeeRepository Employees { get; }
      ...

      bool Complete();
}
```

Kôd 5.3 Dio `IUnitOfWork` sučelja

Metoda `Complete()`, prikazana u prethodnom kôdu, izvršava transakciju te vraća rezultat tipa `bool` koji je pozitivan (`true`) u slučaju uspješnog izvršenja transakcije, odnosno negativan (`false`) u slučaju neuspješnog izvršenja transakcije.

5.2. Injektiranje ovisnosti

Pridruživanje instanci repozitorija i konteksta baze podataka unutar implementacije `IUnitOfWork` sučelja postignuto je korištenjem metode injektiranja ovisnosti (engl. *dependency injection*) koju koristi Simple Injector biblioteka, opisane u odlomku 3.5.

Kôdom (Kôd 5.4) prikazan je dio `UnitOfWork` klase, u kojem je vidljivo kako se nigdje eksplicitno ne kreiraju objekti, već se injektiraju pomoću parametara konstruktora[6]. Prilikom

[5] Kontekst baze podataka je klasa unutar Entity Frameworka zadužena za preslikavanje baze podataka na domenske modele i obrnuto.

[6] Konstruktor je posebna vrsta metode koja se zove isto kao i klasa u kojoj se nalazi, a izvršava se automatski prilikom kreiranja instance te iste klase (Constructors, n.d.).

kreiranja instance `UnitOfWork` klase, DI *container* (SimpleInjector) pridružit će odgovarajuće objekte parametrima.

```
public class UnitOfWork : IUnitOfWork
{
      private readonly DbContext _context;

      public ICustomerRepository Customers { get; private set; }
      public IEmployeeRepository Employees { get; private set; }
      ...

      public UnitOfWork(DbContext context,
          ICustomerRepository customersRepository,
          IEmployeeRepository employeesRepository,
          ...
          {
              _context = context;

              Customers = customersRepository;
              Employees = employeesRepository;
              ...
```

Kôd 5.4 Dio `UnitOfWork` klase

Isto tako, pomoću metode injektiranja ovisnosti osigurano je stvaranje jednog `UnitOfWork` objekta po jednom HTTP zahtjevu.

Dio konfiguracije DI *containera* u kojem je vidljiv način mapiranja sučelja s konkretnom implementacijom prikazan je kôdom (Kôd 5.5).

```
// Unit of Work
_container.Register<IUnitOfWork, UnitOfWork>(Lifestyle.Scoped);
```

Kôd 5.5 Dio konfiguracije DI *containera*

5.3. Aplikacijski sloj

Aplikacijski sloj predstavljaju kontroleri koji obrađuju HTTP zahtjeve. Za svaki entitet autoservisa postoji zaseban kontroler koji sadrži akcije za stvaranje, dohvaćanje, ažuriranje i brisanje entiteta.

Svaka akcija povezana je s određenim URI-jem te predstavlja krajnju točku (engl. *endpoint*) API-ja. Uloga akcije je da obradi zahtjev i vrati odgovor s odgovarajućim statusnim kôdom.

Pri komunikaciji između *web* API-ja i klijenata korišteni su prilagođeni modeli podataka, tj. objekti za prijenos podataka (engl. *data transfer object*, skraćeno DTO). Takvi modeli bazirani su na domenskim modelima, no uključuju samo one podatke koji su potrebni klijentima. Primjer prilagođenog modela podataka baziranog na domenskom modelu Servis (engl. *Service*) prikazan je kôdom (Kôd 5.6).

```
public class ServiceDTO
{
	public WorkOrderDTO WorkOrder { get; set; }
	public int? IDService { get; set; }
	public int? Mileage { get; set; }
	public IList<DiagnosticDTO> Diagnostics { get; set; }
	public string Description { get; set; }
}
```

Kôd 5.6 Primjer DTO modela

Preslikavanje domenskih modela na DTO modele i obrnuto postignuto je korištenjem biblioteke AutoMapper koja je objašnjena u odlomku 3.4.

Primjer akcije za dohvat podataka o klijentu na osnovu jedinstvenog identifikatora prikazan je kôdom (Kôd 5.7).

```
[HttpGet]
[Route("api/customers/{customerID:int}", Name = "GetCustomer")]
public IHttpActionResult GetCustomer(int customerID)
{
	Customer customer = _unitOfWork.Customers.Get(customerID);

	if (customer == null)
	{
		return NotFound();
	}

	CustomerDTO customerDTO = _mapper.Map<CustomerDTO>(customer);
	return Ok(customerDTO);
}
```

Kôd 5.7 Primjer akcije za dohvat podataka o klijentu

U gore prikazanom primjeru vidljivo je kako se krajnja točka API-ja za dohvat podataka o klijentu na osnovu jedinstvenog identifikatora nalazi na adresi `/api/customers/id`. Akcija pokušava dohvatiti podatke o klijentu iz predviđenog repozitorija te ukoliko ih ne

može naći, vraća rezultat metode `NotFound`, koja kreira HTTP odgovor sa statusnim kôdom `404 Not Found`. Ukoliko repozitorij vrati tražene podatke, oni se preslikavaju u DTO te se pozivanjem metode `Ok` stvara HTTP odgovor koji unutar tijela sadrži DTO, a statusni kôd mu je `200 OK`.

Akcije za djelomično ažuriranje entiteta ne zahtijevaju slanje cjelovitog modela, već je dovoljno poslati samo ona svojstva koja se žele mijenjati. Ta mogućnost postignuta je korištenjem biblioteke SimplePatch koja je objašnjena u odlomku 3.2.4.

5.4. Autentifikacija i autorizacija

Kako bi *web* API uspješno odgovorio na HTTP zahtjev, on mora sadržavati zaglavlje `Authorization` čija se vrijednost sastoji od riječi `Bearer` i JWT tokena (objašnjeno u odlomku 3.6). Kako bi dobio token, klijent ga prvo mora zatražiti slanjem `POST` zahtjeva na krajnju točku API-ja čija je ruta `/api/token`. Krajnja točka očekuje podatke u sljedećem obliku:

```
Email: <e-mail adresa zaposlenika>
Password: <lozinka zaposlenika>
```

Ukoliko su primljeni podaci valjani, API odgovara statusnim kôdom `200 OK`, a u tijelo odgovora postavlja JWT token unutar kojeg se nalazi i razina ovlasti klijenta. Ukoliko podaci nisu valjani, API odgovara statusnim kôdom `401 Unathorized`.

Nakon dobivanja tokena, klijent treba postaviti autorizacijsko zaglavlje prilikom svakog sljedećeg zahtjeva API-ju.

Ukoliko zahtjev ne sadrži autorizacijsko zaglavlje, a klijent pokušava pristupiti zaštićenom resursu, API odgovara statusnim kôdom `401 Unathorized`.

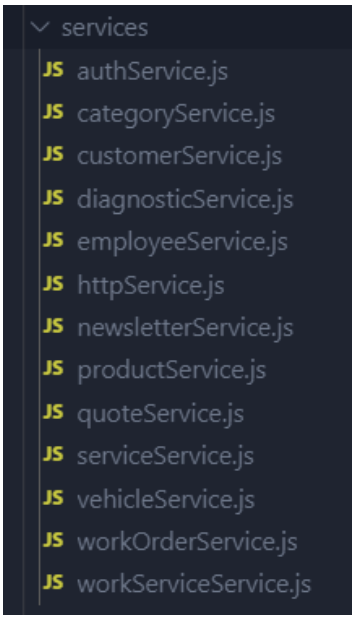
Kada primi zahtjev koji unutar zaglavlja sadrži JWT token, API provjerava valjanost tokena, te ukoliko je token valjan prosljeđuje zahtjev na predviđenu akciju. Ukoliko je token valjan, no klijent nije autoriziran, tj. razina ovlasti klijenta nije dovoljna za ostvarenje zahtjeva, API odgovara statusnim kôdom `403 Forbidden`.

6. *Web* aplikacija

U ovom poglavlju predstavljena je *web* aplikacija koja je izrađena kao dio ovog završnog rada. *Web* aplikacija koristi React biblioteku (odlomak 3.7) za izgradnju korisničkog sučelja te predstavlja prezentacijski sloj cjelokupnog sustava. Služi za pregled, unos i izmjenu podataka, a s *web* API-jem povezana je preko krajnjih točaka (engl. *endpoints*) API-ja.

6.1. Struktura aplikacije

Aplikacija se sastoji od mnoštva manjih komponenata, od kojih svaka ima svoju programsku logiku, a za rad s podacima koristi servise. Servisi su manji JavaScript moduli[7] koji sadrže funkcije za stvaranje, dohvaćanje, ažuriranje i brisanje entiteta. Svaki entitet unutar konteksta autoservisa ima svoj vlastiti servis, a svi koriste zajednički HTTP servis za komunikaciju s API-jem. Postojeći servisi aplikacije prikazani su slikom (Slika 6.1).



Slika 6.1 Postojeći servisi *web* aplikacije

Spajanjem komponenata stvorene su veće cjeline, tj. moduli, a neke od njihovih funkcionalnosti opisane su u sljedećim odlomcima.

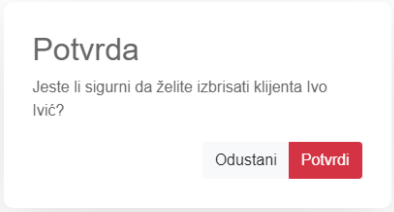
[7] JavaScript moduli su izdvojeni dijelovi JavaScript kôda.

6.2. Modul Klijenti

Modul Klijenti putem pregledne tablice prikazuje osnovne podatke o klijentima kao što su ime, prezime, *e-mail* adresa, broj telefona i adresa stanovanja. Tablica se može sortirati uzlaznim ili silaznim redoslijedom po svim stupcima. Alatna traka koja se nalazi iznad tablice nudi opcije dodavanja novog klijenta, slanja *newslettera* postojećim klijentima te sadrži polje za pretraživanje klijenata koje omogućuje filtriranje tablice po svim stupcima. Odabirom klijenta omogućuju se dodatne opcije unutar alatne trake kao što su pregled detaljnih podataka o klijentu, uređivanje podataka o klijentu, brisanje klijenta te se s desne strane pojavljuje nova komponenta koja prikazuje preglednu tablicu vozila čiji je klijent vlasnik. Alatna traka koja se nalazi ispod te komponente nudi opciju dodavanja novog vozila. Odabirom postojećeg vozila omogućuju se dodatne opcije, a to su pregled detaljnih podataka o vozilu te brisanje vozila.

Odabir opcije dodavanja novog klijenta ili uređivanja podataka o klijentu preusmjerava aplikaciju na novu komponentu koja sadrži formu za dodavanje klijenta, odnosno za uređivanje podataka o klijentu. U formu se unose svi potrebni podaci, a unos je olakšan na način da se podaci o adresi dohvaćaju putem Google Maps API-ja[8] što omogućuje djelomičan unos adrese nakon kojeg se ostala polja poput poštanskog broja, grada i države automatski popunjuju.

Odabir opcije brisanja klijenta, vozila ili bilo kojeg drugog entiteta otvara potvrdni dijaloški okvir koji osigurava korisnika aplikacije od nehotičnog brisanja zapisa. Primjer takvog dijaloškog okvira prikazan je slikom (Slika 6.2).

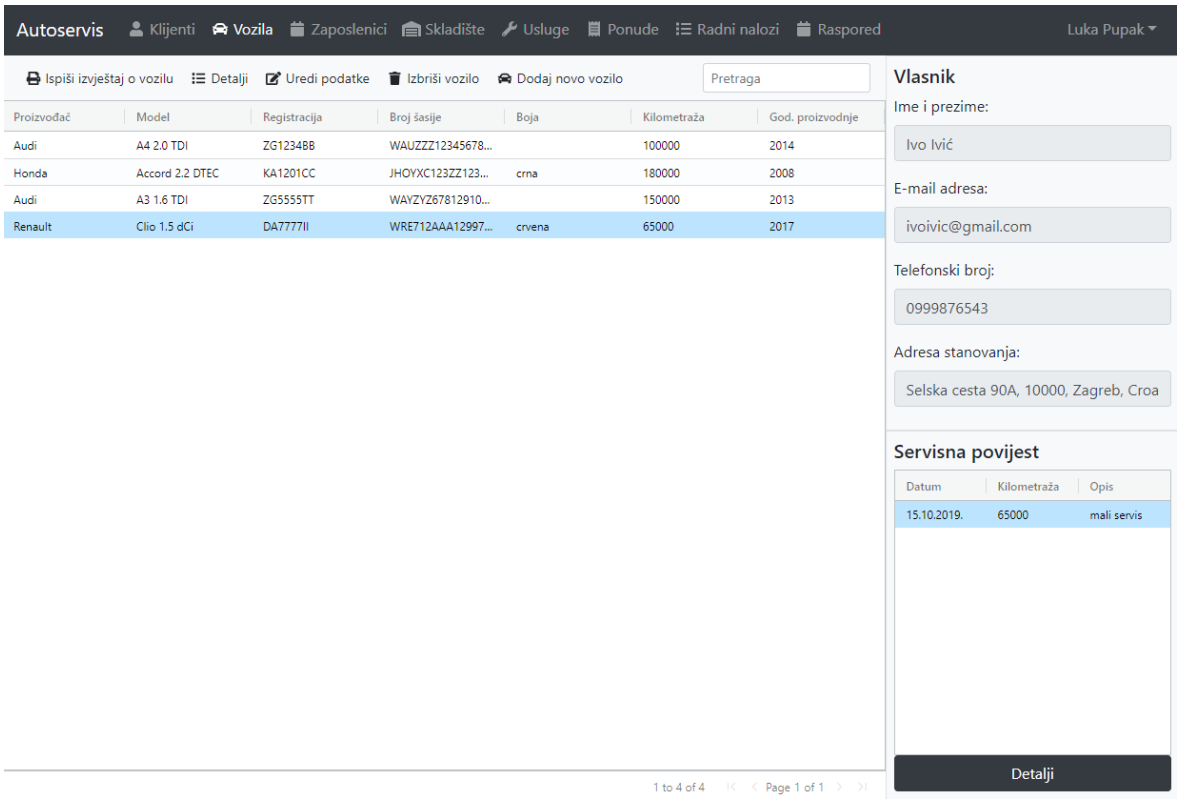


Slika 6.2 Potvrdni dijaloški okvir

[8] Google Maps API je API koji pruža korisne podatke o zemljopisnim lokacijama.

6.3. Modul Vozila

Slično kao i modul Klijenti, modul Vozila koristi tablicu za prikaz podataka o vozilima kao što su proizvođač, model, registracija, broj šasije, boja, broj prijeđenih kilometara te godina proizvodnje. Alatna traka koja se nalazi iznad tablice nudi opcije dodavanja novog vozila te sadrži polje za pretraživanje vozila. Odabirom vozila omogućuju se dodatne opcije unutar alatne trake kao što su ispis izvještaja o vozilu, pregled detaljnih podataka o vozilu, uređivanje podataka o vozilu, brisanje vozila te se s desne strane pojavljuju dvije nove komponente, jedna ispod druge, koje prikazuju podatke o vlasniku te o servisnoj povijesti vozila. Podaci koji su prikazani su ime i prezime vlasnika, *e-mail* adresa, telefonski broj te adresa stanovanja. Servisna povijest prikazana je u obliku tablice koja sadrži datum, stanje kilometara u trenutku servisa te opis servisa. Odabirom pojedinog servisa na alatnoj traci ispod komponente omogućuje se opcija detaljnog pregleda podataka o odabranom servisu. Izgled modula Vozila prikazan je slikom (Slika 6.3).



Slika 6.3 Izgled modula Vozila

6.4. Modul Zaposlenici

Modul Zaposlenici baziran je na modulu Klijenti, a umjesto upravljanja podacima o klijentima pruža mogućnost upravljanja podacima o zaposlenicima. Opcije koje nudi su dodavanje novih zaposlenika, pregled i uređivanje podataka o postojećim zaposlenicima te brisanje zaposlenika.

6.5. Modul Skladište

Modul Skladište putem pregledne tablice prikazuje podatke o proizvodima u skladištu kao što su naziv, kategorija, opis, cijena, kataloški broj te količina proizvoda.

Pruža slične mogućnosti kao i prethodno navedeni moduli, no sadrži i dodatne opcije kao što je upravljanje kategorijama. Uz standardno dodavanje kategorija, omogućeno je i grananje kategorija u smislu da svaka kategorija može biti unutar neke druge kategorije te je na taj način moguće stvoriti hijerarhiju u koju se mogu smještati proizvodi i usluge.

6.6. Modul Usluge

Modul Usluge baziran je na modulu Skladište, s razlikom da umjesto pregleda i upravljanja proizvodima nudi pregled i upravljanje uslugama rada koje autoservis nudi.

6.7. Modul Ponude

Modul Ponude omogućuje izradu servisnih ponuda za klijente. Prilikom izrade ponude korisnik aplikacije odabire klijenta, vozilo te dodaje stavke ponude koje mogu biti proizvodi i usluge rada koje autoservis nudi. Da bi se proizvodi mogli odabrati, mora postojati bar jedan proizvod u skladištu. Za svaku stavku može se odabrati količina te se na ukupan iznos ponude može primijeniti popust koji je izražen kroz postotak. Nakon izrade ponude, ona se uz ostale opcije može i ispisati.

Modul putem pregledne tablice prikazuje podatke o ponudama kao što su broj ponude, ime klijenta, vozilo, ime zaposlenika koji je stvorio ponudu, datum stvaranja ponude, iznos prije popusta, postotak popusta te ukupan iznos ponude s popustom.

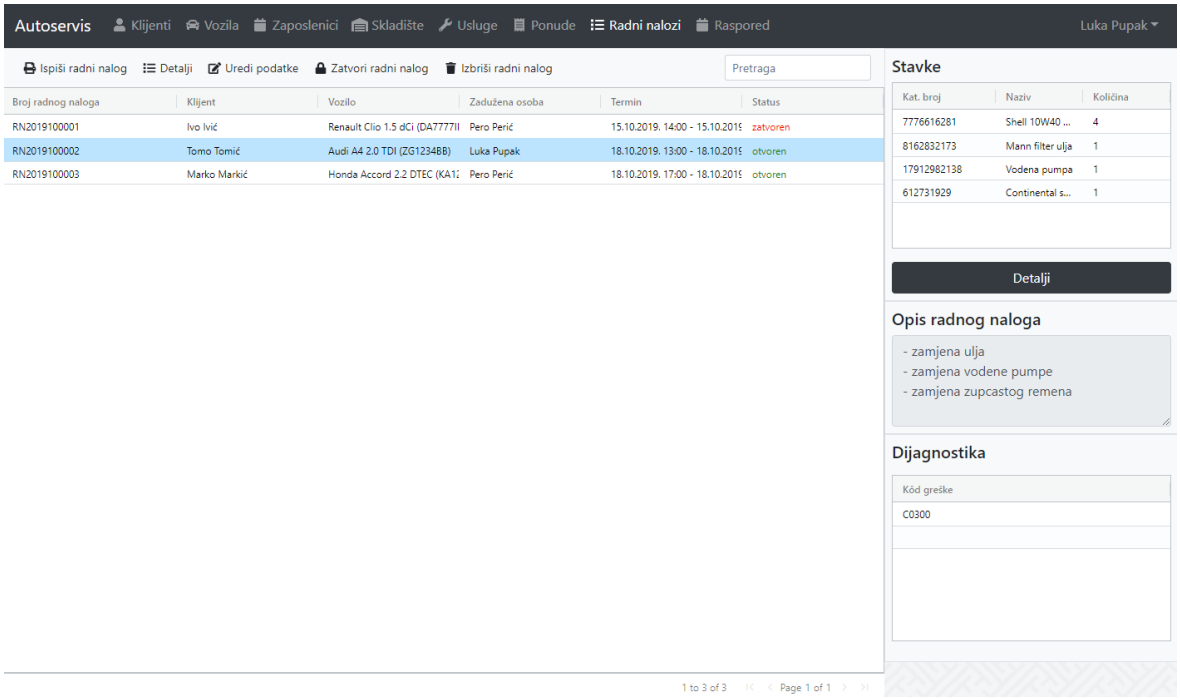
Korisničko sučelje modula Ponude izvedeno je na isti način kao i korisnička sučelja prethodno opisanih modula. Odabirom postojeće ponude s desne se strane prikazuje nova

komponenta koja putem pregledne tablice prikazuje podatke o radnim nalozima koji su povezani s ponudom. Podaci koji su prikazani su termin radnog naloga te zadužena osoba. Alatna traka koja se nalazi ispod te komponente nudi opciju stvaranja novog radnog naloga. Odabirom postojećeg radnog naloga omogućuju se dodatne opcije, a to su pregled i brisanje radnog naloga.

Odabir opcije stvaranja novog radnog naloga preusmjerava aplikaciju u modul Radni nalozi.

6.8. Modul Radni nalozi

Modul Radni nalozi objedinjuje sve prethodno opisane module te omogućuje izradu radnih naloga na temelju izdanih ponuda. Prilikom stvaranja radnog naloga korisnik aplikacije odabire zaduženu osobu, termin te unosi opis radnog naloga. Ukoliko ponuda na temelju koje se radni nalog kreira sadrži usluge rada, opis će se automatski popuniti na temelju tih usluga te tako korisniku aplikacije olakšati unos podataka. Izgled modula Radni nalozi prikazan je slikom (Slika 6.4).



Slika 6.4 Izgled modula Radni nalozi

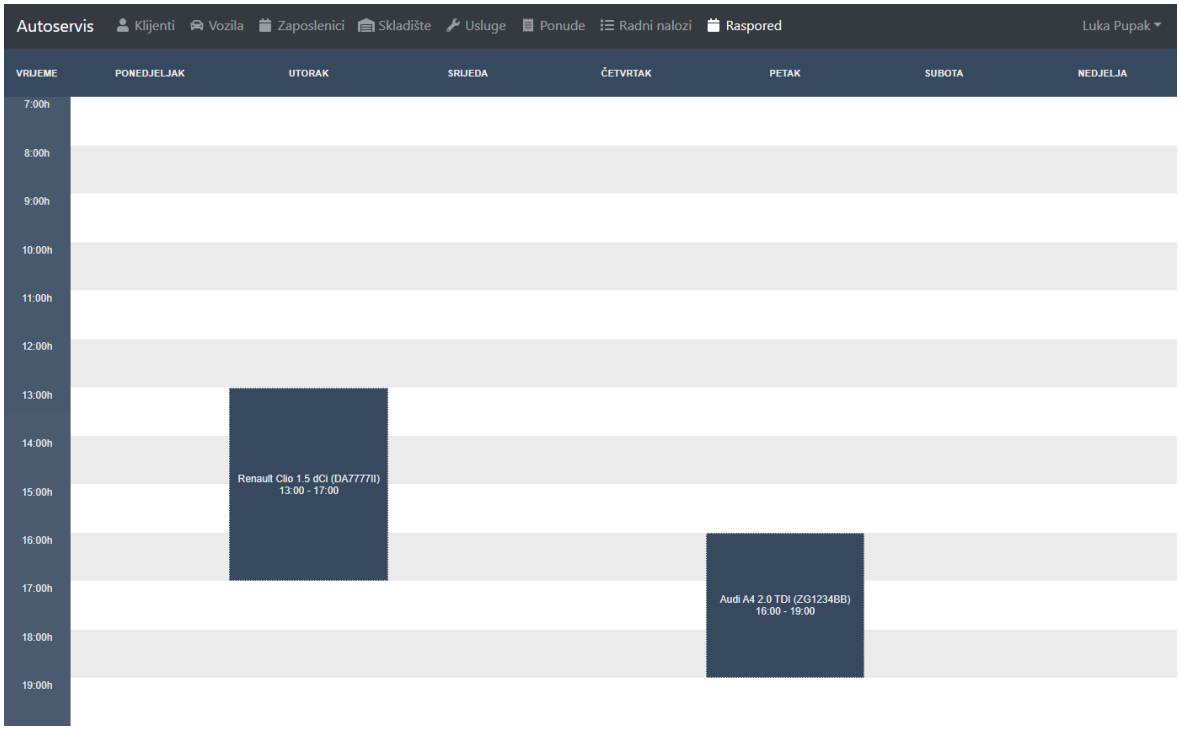
Modul putem pregledne tablice prikazuje podatke o radnim nalozima kao što su broj radnog naloga, ime klijenta, vozilo, ime zadužene osobe, termin te status radnog naloga koji može biti otvoren ili zatvoren.

Odabirom postojećeg radnog naloga omogućuju se dodatne opcije alatne trake kao što su ispis i zatvaranje radnog naloga. Također, s desne se strane pojavljuju tri nove komponente, jedna ispod druge, koje prikazuju podatke kao što su stavke ponude uz koju je radni nalog vezan, opis radnog naloga te dijagnostički podaci prikupljeni tijekom servisa vozila koji se obavlja na temelju radnog naloga.

Odabirom opcije zatvaranja radnog naloga otvara se dijaloški okvir koji nudi unos opisa završenog posla te unos trenutnog stanja kilometara vozila. To su podaci na temelju kojih se kreira servisna povijest vozila. Nakon potvrde zatvaranja radnog naloga, on se više ne može uređivati te se korigira količina proizvoda u skladištu koji su iskorišteni tijekom servisa.

6.9. Modul Raspored

Modul Raspored prikazuje radne naloge unutar tekućeg tjedna. Odabirom radnog naloga na rasporedu aplikacija se preusmjerava na modul Radni nalozi gdje se prikazuju detalji odabranog radnog naloga. Izgled modula Raspored prikazan je slikom (Slika 6.5).



Slika 6.5 Izgled modula Raspored

7. Mobilna aplikacija

U ovom poglavlju predstavljena je mobilna aplikacija koja je osmišljena kao produžetak informacijskog sustava autoservisa. Razvijena je korištenjem React Native radnog okvira (odlomak 3.9), a za izgradnju korisničkog sučelja iskorištena je NativeBase biblioteka (odlomak 3.9.4). Za komunikaciju s *web* API-jem iskorišteni su servisni moduli kreirani prilikom razvoja *web* aplikacije.

Aplikacija je testirana na Android platformi, no radi odabrane tehnologije lako se može prilagoditi za iOS platformu ako se to u budućnosti pokaže potrebnim.

Dvije su glavne mogućnosti koje ova aplikacija nastoji pružiti, a to su čitanje dijagnostičkih kôdova grešaka (DTC) automobila bežičnim putem te registracija proizvoda u skladištu korištenjem barkôd čitača. U sljedećim odlomcima opisane su mogućnosti glavnih modula aplikacije – modula Dijagnostika te modula Skladište.

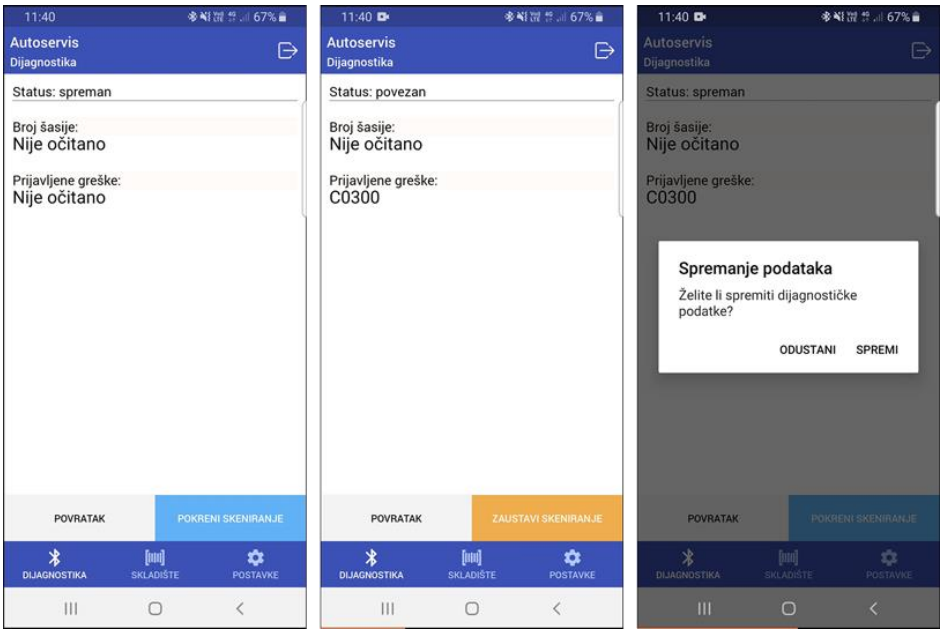
7.1. Modul Dijagnostika

Modul Dijagnostika omogućava čitanje dijagnostičkih kôdova grešaka putem Bluetooth tehnologije tako što se aplikacija spaja s OBD-II Bluetooth čitačem koji se nalazi u automobilu. Za tu svrhu iskorištena je React Native OBD2 biblioteka (odlomak 3.9.2).

Proces dijagnostike počinje odabirom vozila s popisa vozila za koje trenutno postoje otvoreni radni nalozi. Odabirom vozila pojavljuje se izbornik u kojem su prikazani radni nalozi vezani uz odabrano vozilo. Nakon odabira radnog naloga nudi se opcija skeniranja dijagnostičkih kôdova grešaka. Nakon što je skeniranje obavljeno, aplikacija nudi mogućnost spremanja podataka u informacijski sustav. U slučaju da OBD-II adapter uspije pročitati[9] i broj šasije automobila, on će se također pohraniti u informacijski sustav ukoliko već nije upisan.

Izgled modula Dijagnostika prikazan je slikom (Slika 7.1).

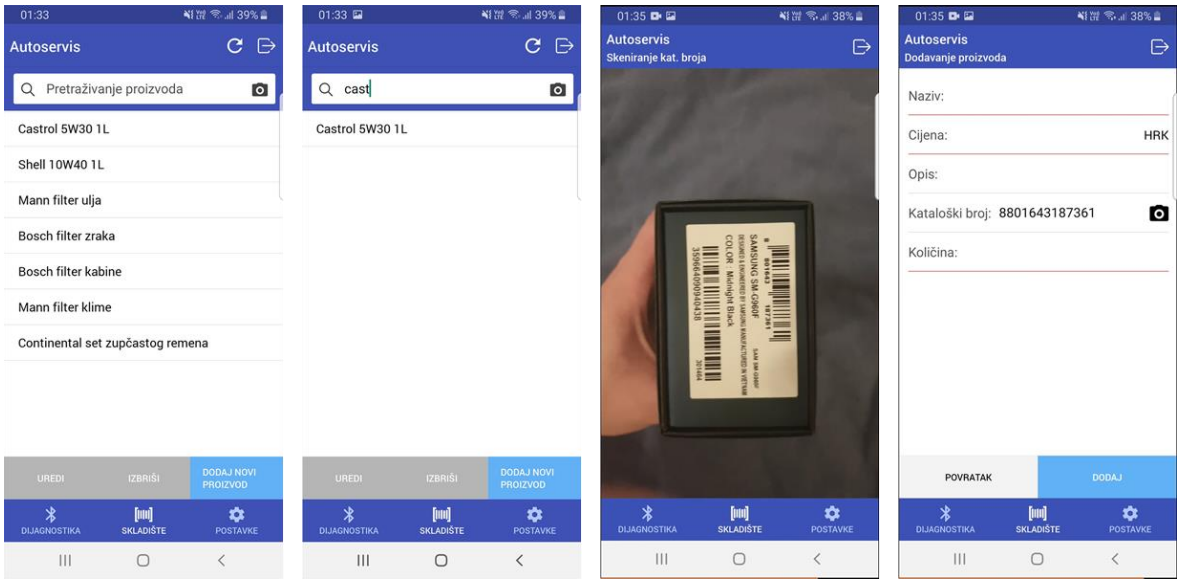
[9] Uspjeh čitanja broja šasije ovisi o OBD-II sustavu kojim automobil raspolaže.



Slika 7.1 Izgled modula Dijagnostika

7.2. Modul Skladište

Drugi modul mobilne aplikacije je modul Skladište koji omogućava pregled proizvoda u skladištu, pretraživanje proizvoda po imenu ili barkôdu, dodavanje novih proizvoda uz mogućnost registracije kataloškog broja korištenjem barkôd čitača, uređivanje podataka o postojećim proizvodima te brisanje proizvoda. Izgled modula Skladište prikazan je slikom (Slika 7.2).



Slika 7.2 Izgled modula Skladište

8. Testiranje kod potencijalnih korisnika

Informacijski sustav izrađen u sklopu ovog završnog rada ponuđen je na testiranje nekolicini automehaničarskih obrta. U trenutku izrade pismenog dijela završnog rada jedan obrt je prihvatio testirati sustav, dok se od ostalih čeka povratna informacija.

Obrt koji je pristao testirati sustav je Automehaničarski obrt KOS, Poljana.

Ukoliko se prilikom suradnje s ovim i potencijalno drugim obrtima ustanovi potreba za izmjenama sustava, one će biti implementirane kroz naredne verzije te objavljene na *web* stranici https://autoservis.pupak.net.

Zaključak

Svrha ovog završnog rada bila je stvoriti moderan informacijski sustav koji bi unaprijedio svakodnevno poslovanje autoservisa.

Kroz rad je opisan informacijski sustav razvijen korištenjem modernih tehnologija i alata te su prikazane neke od mogućnosti korištene tehnologije poput ASP.NET Web API-ja, Entity Framework-a, React-a i React Native-a.

Redom su opisane sve komponente sustava kojeg čine baza podataka, *web* aplikacijsko programsko sučelje te *web* i mobilna aplikacija, te je napravljen osvrt na testiranje kod potencijalnih korisnika.

Smatra se da je rad ispunio svoju svrhu jer je razvijen učinkovit informacijski sustav koji može unaprijediti svakodnevno poslovanje autoservisa.

Popis kratica

API	Application Programming Interface	aplikacijsko programsko sučelje
DI	Dependency Injection	injektiranje ovisnosti
DTC	Diagnostic Trouble Code	dijagnostički kôd greške
DTO	Data Transfer Object	objekt za prijenos podataka
IDE	Integrated Development Environment	integrirano razvojno okruženje
JWT	JSON Web Tokens	JSON mrežni tokeni
MPA	Multiple-Page Application	višestranična aplikacija
ORM	Object-Relational Mapping	objektno-relacijsko preslikavanje
PID	Parameter Identifier	identifikator parametra
RDBMS	Relational Database Management System	sustav za upravljanje relacijskom bazom podataka
REST	Representational State Transfer	prijenos reprezentativnog stanja
SPA	Single-Page Application	jednostranična aplikacija
SQL	Structured Query Language	strukturni upitni jezik
URI	Uniform Resource Identifier	jedinstveni identifikator resursa
URL	Uniform Resourse Locator	jedinstveni lokator resursa
VM	Virtual Machine	virtualna mašina

Popis slika

Popis tablica

Popis kôdova

Literatura

[1] CARIĆ, T.;BUNTIĆ, M., Uvod u relacijske baze podataka, Zagreb, 2015.

[2] GEEKS FOR GEEKS, C# | Constructors, https://www.geeksforgeeks.org/c-sharp-constructors/, listopad, 2019.

[3] JWT.IO, Introduction to JSON Web Tokens, https://jwt.io/introduction/, listopad, 2019.

[4] KAŠMO, D., Normizacija terminologije u geoinformatici, Diplomski rad, Sveučilište u Zagrebu, 2014.

[5] MICROSOFT, Getting Started with ASP.NET Web API, https://docs.microsoft.com/en-us/aspnet/web-api/overview/getting-started-with-aspnet-web-api/, listopad, 2019.

[6] MICROSOFT, ASP.NET Overview, https://docs.microsoft.com/en-us/aspnet/web-api/overview/, listopad, 2019.

[7] MICROSOFT, Overview of Entity Framework, https://docs.microsoft.com/en-us/ef/ef6/, listopad, 2019.

[8] MINIGRANTH, Types Of SQL Commands, https://www.minigranth.com/sql-tutorial/types-of-sql-commands/, listopad, 2019.

[9] MOZILLA, HTTP Overview, https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview, listopad, 2019.

[10] REACT, What is React?, https://reactjs.org/tutorial/tutorial.html#what-is-react, listopad, 2019.

[11] REACT NATIVE GUIDE, React Native Internals, https://www.reactnative.guide/3-react-native-internals/3.1-react-native-internals.html, listopad, 2019.

[12] REACT ROUTER, React Router Quick Start, https://reacttraining.com/react-router/web/guides/quick-start, listopad, 2019.

[13] REACT NATIVE, Out of Tree Platforms, https://facebook.github.io/react-native/docs/out-of-tree-platforms, listopad, 2019.

[14] SCALEGRID, 2019 Database Trends, https://scalegrid.io/blog/2019-database-trends-sql-vs-nosql-top-databases-single-vs-multiple-database-use/, listopad, 2019.

[15] STACKIFY, Design Patterns Explained, https://stackify.com/dependency-injection/, listopad, 2019.

ALGEBRA
VISOKO UČILIŠTE

APLIKACIJA ZA UPRAVLJANJE AUTOSERVISOM

Pristupnik: Luka Pupak, 0321006665

Mentor: Bojan Fulanović, dipl. ing.