Izrada desktop aplikacije u programskom okruženju Microsoft Visual Studio

Raguž, Luka

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Luka Raguž

Izrada desktop aplikacije u programskom okruženju Microsoft Visual Studio

DIPLOMSKI RAD

Varaždin, 2022.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Luka Raguž

Matični broj: 0016129635

Studij: *Informacijsko i programsko inženjerstvo*

Izrada desktop aplikacije u programskom okruženju Microsoft Visual Studio

DIPLOMSKI RAD

Mentor:

Prof. dr. sc. Neven Vrček

Varaždin, rujan 2022

Luka Raguž

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

„Microsoft Visual Studio“ je sveobuhvatan „IDE“ za „.NET“ i „C#“ programiranje te sadrži niz alata i značajki za podizanje i poboljšanje svake faze razvoja softvera. Cilj ovog diplomskog rada je izraditi „Desktop“ aplikaciju koristeći programsko okruženje „Microsoft Visual Studio“ gdje je u teorijskom dijelu rada potrebno opisati osnovne značajke i mogućnosti softverskog okvira te u praktičnom dijelu izraditi proizvoljnu aplikaciju koja sadrži niz funkcionalnosti koristeći isti okvir. Ovim radom bi se demonstrirao postupak rada sa „Microsoft Visual Studiom“ i njegovim mogućnostima kao što su jezik „C#“, „WPF“, „LINQ“, rad sa bazom podataka i drugo. Aplikacija treba koristiti bazu podataka, a samu aplikaciju je potrebno napraviti koristeći „.NET Framework“ što je izvorna implementacija „.NET-a“ koja podržava pokretanje „Desktop“ aplikacija uz korištenje objektno orijentiranog programskog jezika „C#“. Grafičko sučelje je potrebno izgraditi koristeći grafički podsustav otvorenog koda „WPF“ što je okvir za izradu „Desktop“ klijentskih aplikacija. Ovaj okvir koristi jezik oznaka „XAML“ koji služi pružanju deklarativnog modela za programiranje aplikacija. Uz izradu aplikacije podrazumijeva se i izrada same dokumentacije vezane uz aplikaciju.

Ključne riječi: Desktop aplikacije, .NET tehnologija, Visual Studio

Sadržaj

1. Uvod

U svijetu kakvog danas poznajemo sve se više i više susrećemo s tehnologijama koje nam život čine boljim tako da što god zamislimo ili nam predstavi problem, možemo olakšati korištenjem adekvatnog softvera ili alata. IT industrija od svojih začetaka bilježi kontinuirani rast i razvoj te je postala toliko napredna i uspješna da se danas aplikacije rade za stvari koje prije nismo mogli niti zamisliti. Kako je moguće razvijati aplikacije za različite platforme, za početak razvoja softvera potrebno je odgovoriti na pitanje što od alata i na koji način dobro iskoristiti? Kako u suvremeno doba skoro svaka obitelj posjeduje barem jedno računalo, odgovor na ovo pitanje rezultirao je odlukom o izradi računalne aplikacije (eng. „Desktop application") u programskom okviru „.NET Framework". Kako napreduje razvoj aplikacija u ovom području, normalno da je i konkurencija na tržištu velika stoga aplikacije s vremenom postaju sve bolje i iz njih se očekuje više, ali isto tako je i njihov razvoj sve više i više olakšan te prilagođen čovjeku što je postignuto kroz razne programske okvire.

Cilj ovog rada je napraviti potpun proizvod koji će na određeni način olakšati živote i poslove njezinih korisnika. Glavna inspiracija za ovaj rad je popularnost programskog okvira koja daje veliki izbor mogućnosti razvoja aplikacija te samo učenje i napredak karijere u polju razvoja softvera. Izrada aplikacije predstavlja praktični dio diplomskog rada, u kojemu je osim praktičnog, napisan i pismeni dio koji obuhvaća korištene metode i tehnike rada, njihova objašnjenja te dokumentaciju aplikacije. Osim „Microsoft Visual Studija" i korespondirajućih alata koji dolaze uz ovo razvojno okruženje, za funkcionalan rad ove aplikacije zaslužan je i „Firebase" na kojemu je smještena baza podatka, kao i „Azure storage" koji je poslužio za pohranu datoteka. Nešto više o metodama i alatima opisano je u sljedećem poglavlju rada.

Aplikacija napravljena u sklopu ovog diplomskog rada kao glavnu namjenu ima rezervaciju automobila po raznim auto kućama unutar Europe. Neke od glavnih funkcionalnosti aplikacije su da se korisnicima omogući kreiranje novih ureda na određenim lokacijama, zabilježavanje auta koje kuće posjeduju te rezervaciju istih od strane članova auto kuća. Aplikacija sadrži i druge osnovne funkcionalnosti koje su neophodne za rad današnjih aplikacija kao što su registracija i autentikacija, izmjena i pregled vlastitih informacija te drugih stvari vezanih uz tematiku što je korisnicima ograničeno s određenim ulogama. Uloge su smislene te predstavljaju hijerarhiju unutar same firme, a to su članovi auto kuća, njeni radnici te direktor auto kuće. Kako sam već spomenuo, kroz aplikaciju je moguće upravljati s više auto kuća koje su rasprostranjene po Europi čime se bavi najveća uloga, a to je administrator sustava. Administratoru je omogućeno kreiranje novih ureda te dodjeljivanje direktora samom uredu, kao i dodavanje radnika i članova. Cjelokupna dokumentacija navedena je u praktičnom dijelu rada.

2. Metode i tehnike rada

U ovom poglavlju opisane su metode, tehnike te programski alati korišteni pri razradi teme i izradi aplikacije. Prvotno, glavni alat korišten za izradu aplikacije je samo integrirano razvojno okruženje (eng. *Integrated development environment*, skraćeno IDE) „Microsoft Visual Studio“ u kojem je za izradu aplikacije korišten programski okvir „.NET Framework“. Kod korištenja ovog okvira, kao podloga za rad (eng. *Template*) odabrana je „WPF“ aplikacija u što se podrazumijeva korištenje programskog jezika „C#“ te aplikacijskoj jezika oznaka (eng. *Extensible Application Markup Language*, skraćeno XAML). „XAML“ je deklarativni jezik oznaka koji nam omogućuje izradu modernih korisničkih sučelja namijenjenog za sljedeće generacije, a prvenstveno služi za izradu sučelja „Windows“ i mobilnih aplikacija koje koriste „Windows Presentation Foundation“ (skraćeno WPF). Za izradu aplikacije korištene su razne mogućnosti koje nam ovaj okvir i jezici nude, kao što su i „NuGet“ paketi koji nam pružaju različite biblioteke koje možemo koristiti i na taj način si olakšati rad. Dosad Navedeni alati su korišteni u najvećoj mjeri te su opisani u sljedećem poglavlju, no za izradu praktičnog dijela potrebna je i baza podataka, za što se koristio „Firebase“.

Platforma „Firebase“, prvenstveno je korištena za omogućavanje registracije, autentikacije te kao baza podataka. Osim navedenog, platforma sadrži razne druge integrirane mogućnosti koje korisnicima mogu pomoći u razvoju i izradi aplikacija. Kod izrade autentikacije korištena je metoda elektroničke pošte i lozinke koja korisnicima omogućava registraciju sa svojom elektroničkom poštom te određenom lozinkom te nakon uspješne registracije korisnik treba verificirati svoju adresu pri čemu mu se omogućuje ulazak u aplikaciju. Za navedeno također je korišten „Firebase Auth REST API“ koji omogućuje postavljanje upita u pozadini. Prilikom kreiranja projekta unutar samog „Firebase-a“ administratoru se dodjeljuje ključ koji je potrebno postaviti unutar krajnje točke (eng. *endpoint*) gdje se koriste „GET“ i „POST“ metode za određene akcije. Svaki poziv ima određeni naziv koji je također potrebno staviti unutar krajnje točke te se prilikom poziva moraju slati parametri unutar tijela (eng. *Request body payload*). „Firebase Auth REST API“ sadrži mnoge krajnje točke dok su u aplikaciji korištene samo one koje su bile potrebne. Više o korištenju ove mogućnosti navedeno je u sljedećem poglavlju.

Zadnji, manje bitan alat, koji je također korišten pri izradi praktičnog dijela je „Inkscape“. Ovaj alat poslužio je za izradu logotipa aplikacije na način da je izrađeni logotip napravljen u vektorskoj grafici te se može prilagođavati tematici aplikacije.

Kao metode za učenje prije same izrade aplikacije korišteni su internet sadržaji i tečaji koji se vežu uz navedenu tematiku.

3. Opis korištenih alata

U prošlom poglavlju navedeni su svi alati korišteni za izradu praktičnog dijela. Ovo poglavlje teorijski opisuje svaki od alata, dok će praktični dio biti opisan u drugom dijelu rada. U najvećoj mjeri opisan je glavni alat, programsko okruženje „Microsoft Visual Studio“ te korespondirajući programski okvir i alati koji dolaze uz samo okruženje zatim platforma „Firebase“ te u manjoj mjeri sporedni alati kao što je „Inkscape“.

3.1. Programsko okruženje „Microsoft Visual Studio“

Kao što je već navedeno, „Microsoft Visual Studio“ je alat korišten u najvećoj mjeri za izradu praktičnog dijela diplomskog rada, a predstavlja opće poznato programsko okruženje koje se koristi za razvoj računalnih programa. Podržava različite programske jezike kao što su „C“, „C++“, „F#“ i „C#“ te se sastoji od niza alata i značajki za podizanje i poboljšanje svake faze razvoja softvera. Unutar ove obitelji alata spadaju i „Microsoft Studio for Mac“ te „Visual Studio Code“ koji nisu korišteni pri izradi praktičnog dijela rada. Osim navedenoga, prilikom preuzimanja ovog alata mogu se odabrati različite verzije i godine izdanja koji se razlikuju po nekim mogućnostima. Od mogućih izbora „Community“, „Professional“, „Enterprise“ korištena je besplatna verzija „Microsoft Visual Studija“, a to je „Visual Studio Community 2022“. Najnovija verzija nudi brže i pouzdanije 64-bitno integrirano razvojno okruženje te dolazi s verzijom „.NET-a 6“ uključujući i podršku za „MAUI“, „Blazor“ aplikacije te „Hot reload“, ali za izradu aplikacije nisu bili potrebni navedeni alati jer je odabir verzije programskog okvira bio „.NET Framework“ s verzijom 4.8. [1]

Kao osnova za početak rada odabrana je podloga za izradu „Windows Presentation Foundation“ (skraćeno WPF) aplikacija što je okvir za izradu korisničkog sučelja (eng. *UI Framework*) te je osnova za stvaranje klijentskih aplikacija za stolna računala (eng. *Desktop computers*). Koristeći ovaj alat omogućen je pristup širokom skupu značajki za razvoj aplikacija uključujući model aplikacije, kontrole, resurse, izgled, grafiku, uvezivanje podataka, dokumente i sigurnost. „WPF“ je dio navedenog „.NET“ okvira što se može usporediti sa „Windows Forms“ aplikacijama čije iskustvo pomaže u svladavanju ovog alata. Kao što je već spomenuto, „WPF“ koristi „XAML“ za pružanje deklarativnog modela za programiranje aplikacija te je isti jezik oznaka, zajedno sa „WPF-om“ detaljnije opisan u jednom od sljedećih poglavlja. Kao jedna od bitnih stvari također se treba spomenuti korištene „NuGet“ paketa što je mehanizam putem kojeg programeri mogu stvarati, dijeliti i koristiti koristan kod. Taj kod je povezan u pakete koji sadrže kompilirani kod te je napravljen kao „.dll“. Ovaj mehanizam definira kako se paketi za „.NET“ stvaraju i koriste te pruža alate za svaku od tih uloga. [2]

3.1.1. Programski okvir „.NET Framework“

Odabir ovog programskog okvira za izradu praktičnog dijela uslijedio je zbog posjedovanja ranijeg iskustva u istome što je bila dobra prilika za nadogradnju znanja u tom području. Tematika izrade aplikacije za stolno računalo u kombinaciji s ovim programskim okvirom činila se kao najbolja kombinacija jer je „.NET Framework“ tehnologija baš i osmišljena iz tog razloga na način da podržava izgradnju i pokretanje „Windows“ aplikacija i web usluga. Ova usluga nam pruža objektno orijentirano programsko okruženje bilo da se objektni kod pohranjuje i izvršava lokalno ili na webu. Odabirom ovog programskog okvira unaprijed osiguravamo sigurno izvršavanje koda, uključivalo to osobni napisani kod ili kod koji je izradila nepoznata ili polu-pouzdana treća strana. Osnova koja čini sam okvir je zajedničko jezično okruženje (eng. *Common Language Runtime*, skraćeno CLR) s bibliotekom klasa „.NET Framework-a“. [2]

Navedena osnova zapravo predstavlja dvije glavne komponente „.NET Frameworka“ među kojima je „CLR“ izvršni mehanizam za upravljanje pokrenutim aplikacijama. On nam pruža usluge poput upravljanja dretvama (eng. *Thread management*), skupljanja smeća (eng. *Garbage collection*), rukovanja iznimkama (eng. *Exception handling*) te upravljanja sigurnošću (eng. *Type-safety*). Druga navedena komponenta, odnosno biblioteka klasa „.NET Framework-a“ pruža skup sučelja za programiranje aplikacija (eng. *Application programming interface*, skraćeno API) i tipova za uobičajenu funkcionalnost. Ovo nam pruža poznate tipove podataka kao što su nizovi, datumi, brojevi i slično. Među ovu komponentu također spadaju i svima poznati „API-ji“ za čitanje i pisanje datoteka, povezivanje s bazama podataka i slično. Velika pogodnost koju nam ovaj programski okvir pruža je što neovisno o programskom jeziku u kojem je aplikacije pisana (C#, F#, Visual Basic), kod je kompajliran u zajednički srednji jezik (eng. *Common Intermediate Language*, skraćeno CIL) koji se prevodi te pohranjuje u sklopove (eng. *Assemblies*) koje dobivaju „.dll“ ili „.exe“ ekstenziju. Na ovaj način, kada se aplikacija pokrene, zajednički „runtime“ jezik (eng. *Common Language Runtime*, skraćeno CLR) preuzima sklop i koristi korespondirajući kompajler da ga pretvori u strojni kod koji se može izvršiti na specifičnoj arhitekturi računala na kojem se izvodi. [3]

Na kraju ovog poglavlja, pošto smo svjesni da postoji „.NET“ i „.NET Framework“, navedene su sličnosti i razlike između istih okvira. Za početak se može reći da ova dva okvira dijele mnoge od istih komponente te se njihov kod dijeli. Neke od ključnih razlika su da je „.NET“ višeplatformski te radi na "Linuxu“, „MacOS-u“ i „Windowsu“, dok .NET Framework“ radi samo na „Windows-u“. „.NET“ je tip otvorenog koda (eng. *Opensource*) te se prihvaćaju doprinosi zajednice, odnosno drugih programera, dok je „.NET Framework“ kod dostupan, bez direktnog doprinosa zajednice te se ažurira na računalo putem „Windows-a“. [3]

3.1.2. Okvir za izradu korisničkog sučelja „WPF“

Odabir „WPF-a“ kao okvira za izradu korisničkog sučelja bila je laka odluka jer je to najnoviji „Microsoftov“ okvir za izradu grafičkog korisničkog sučelja (eng. *Graphical User Interface*, skraćeno GUI) te se koristi u kombinaciji s „.NET" okvirom. Kada spomenemo grafičko korisničko sučelje govorimo o okviru koji nam omogućuje da stvorimo aplikaciju sa širokim rasponom elemenata, kao što su oznake, tekstualni okviri te drugi dobro poznati elementi. Kada bi aplikaciju radili bez okvira za izradu grafičkog korisničkog sučelja, te elemente bi morali crtati ručno te rukovati svim scenarijima korisničke interakcije poput unosa teksta i klikova miša. Kada bi za svaku aplikaciju programer sam crtao elemente, to bi potrajalo jako dugo, stoga je umjesto toga većina programera navikla na korištenje izrađenih grafičkih korisničkih sučelja koji obavljaju osnovni posao umjesto njih. Postoji mnogo „GUI“ okvira, no za „.NET“ programere su najpoznatija dva, a to su „WinForms“ i „WPF“, od kojih je „WinForms“ stariji, ali se i dalje održava od strane Microsoft-a. [4]

Neke od glavnih razlika između „WinForms-a“ i „WPF-a“ su te što je „WPF“ noviji i samim time usklađeniji s trenutačnim standardima, sve se više i više koristi za izradu novih aplikacija te se više nadograđuje nego sam „WinForms“. „WPF“ koristi ranije naveden jezik oznaka „XAML“ koji olakšava stvaranje i uređivanje grafičkog korisničkog sučelja te omogućuje podjelu posla između dizajnera i programera. Isto tako, za ovo sučelje koristi se hardversko ubrzanje za crtanje „GUI-a“ te se ostvaraju bolje performanse. Jedna od prednosti koju „WinForms“ ima nad „WPF-om“ je ta što je stariji i samim time je više testiran i isproban te postoji puno veći broj kontrola treće strane koji se mogu kupiti ili preuzeti besplatno. [4]

U sljedećem primjeru naveden je klasičan primjer početka rada sa „WPF-om“ odnosno „XAML-om“. Ovaj kod predstavlja primarni prozor aplikacije odnosno onaj koji se prvi prikazuje prilikom pokretanja aplikacije. Ovo je prazan prozor unutar kojega, poznavajući „XAML“ možemo dodavati elemente kodirajući ili koristiti sučelje kako bi postavljali iste elemente na način povlačenja unutar prozora za dizajniranje. Praksa je pokazala da se puno više preporučuje poznavanje „XAML-a“ i ručno kodiranje nego povlačenje jer se na taj način može više postići. Nešto više o samom jeziku oznaka „XAML-u“ navedeno je u sljedećem poglavlju te su napravljeni osnovni demonstracijski primjeri. [4]

```
<Window x:Class="WpfApplication.MainWindow"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      Title="MainWindow" Height="350" Width="525">
      <Grid>

      </Grid>
</Window>
```

3.1.3. Jezik oznaka „XAML“

U ovom poglavlju detaljnije je opisan ranije spomenuti jezik oznaka „XAML“ što je zapravo kratica za „eXtensible Application Markup Language“. Ovaj jezik je „Microsoftova“ varijanta „XML-a“ koja služi za opisivanje grafičkog korisničkog sučelja. U prethodnim okvirima za grafičko korisničko sučelje, kao što je „WinForms“, elementi se stvaraju na istom jeziku koji se koristi i za interakciju s elementima, kao što je „C#“ ili „VB.NET“ kojeg održava dizajner. Sa „XAML-om“ je situacija drugačija te se ovaj jezik može usporediti sa „HTML-om“ gdje se može jednostavno pisati i na taj način dodavati i uređivati svoje grafički korisničko sučelje. Ovaj jezik predstavlja srž „WPF-a“ te se prilikom kreiranja stranice ili prozora kreira „XAML“ dokument te „CodeBehind“ dokument koji zajedno čine novonapravljeni prozor. „XAML“ datoteka opisuje sučelje sa svim njegovim elementima, dok „CodeBehind“ obrađuje sve događaje i ima pristup za upravljanje „XAML“ kontrolama. [4]

U sljedećem odjeljku objašnjeno je kako stvoriti kontrolu u „XAML-u“ kroz demonstracijski primjer. Stvaranje kontrole jednostavno je poput pisanja njezina naziva koji je okružen uglastim zagradama. „XAML“ oznake moraju se završavati, bilo pisanjem završne oznake ili stavljanjem kose crte na kraj početne oznake. Puno kontrola omogućuje stavljanje teksta između početne i završne oznake što onda predstavlja sadržaj oznake. Za razliku od „HTML-a“, „XAML“ razlikuje velika i mala slova jer naziv kontrole mora odgovarati korespondirajućem tipu unutar „.NET“ okvira, a isto vrijedi i za nazive atributa koji odgovaraju svojstvima kontrole. U sljedećem kratkom primjeru napravljen je gumb te su mu postavljenja svojstva „FontWeight“ i „Content“. Na ovaj način dobili smo gumb na kojemu piše „Gumb“ s podebljanim tekstom. Postavljanje kontenta se, kao što je već ranije navedeno, može obavljati i između početne i završne oznake, no ovo je kraći način. [4]

```
<Button FontWeight="Bold" Content="Gumb"/>
```

Većina modernih okvira za izradu grafičkog korisničkog sučelja je vođena događajima, pa je tako slučaj i sa „WPF-om“. Sve kontrole, uključujući i prozor koji nasljeđuje klasu kontrole, izlažu niz događaja na koju se ista kontrola može „pretplatiti“ (eng. *Subscribe*). Događaji se koriste da daju odgovore na interakciju korisnika s aplikacijom pomoću miša ili tipkovnice. Na većini kontrola se mogu pronaći neki osnovni događaji kao što su „KeyDown“, „KeyUp“, „MouseDown“, „MouseEnter“, „MouseLeave“, „MouseUp“ od kojih se svaki okida na određenu radnju koja je zapravo opisana unutar naziva događaja. Na sljedećem primjeru dan je gumb koji na sebi ima postavljen događaj koji će se okinuti ako kliknemo na gumb. Prilikom postavljanja događaja, automatski kreiramo metodu unutar „C#“ klase s istim imenom događaja unutar kojega postavljamo logiku. [4]

```
<Button FontWeight="Bold" Content="Gumb" Click="ButtonClick"/>
```

3.1.4. Programski jezik „C#“

Programski jezik korišten u izradi praktičnog dijela, koji ujedno čini i srž u radu sa „.NET“ programskim okvirom je već spomenuti „C#“ što je Microsoftov objektno orijentirani programski jezik koji se sastoji od kombinacije „C++-a“ i lakoće programiranja „Visual Basic-a“. Ovaj programski jezik sadrži mnoge značajke slične programskom jeziku „Java“ dok pojednostavljuje programiranje upotrebnom različitih jezika i protokola koji omogućuju pristup programskom objektu ili metodi bez potrebe da programer piše dodatni kod za svaki korak. Kako programeri mogu graditi postojeći kod očekuje se da će s vremenom „C#“ postati brži i jeftiniji za plasiranje novih proizvoda i usluga na tržište. [5]

Odabir ovog programskog jezika za izradu praktičnog dijela uslijedio je samim odabirom programskog okruženja i okvira te zbog ranijeg iskustva u radu s istim.

3.1.5. „NuGet“ paketi

Ovaj alat je danas neophodan za izradu aplikacija u „.NET“ programskom okviru. Predstavlja mehanizam putem kojeg programeri mogu stvarati i dijeliti koristan kod. Drugim riječima, cijeli paket je jedna „ZIP“ datoteka koja sadrži kompajlirani kod u obliku „.dll-a“ , druge datoteke povezane s tim kodom te opisni manifest koji uključuje informacije kao što je verzija paketa. Stvarajući ove pakete, njihovi tvorci ih mogu dijeliti i objavljivati na javnom mjestu gdje potrošači paketa dodaju iste u svoje projekte te povezuju funkcionalnost paketa u svom kodu projekta. Sam „NuGet“ obrađuje sve međupojedinosti i na taj način se tvorcima omogućava lagan i efikasan rad. [2]

U ovom odjeljku nabrojani su neki „NuGet“ paketi korišteni u praktičnom dijelu projekta, a ne spadaju u osnovnu skupinu paketa koji svaki projekt posjeduje. To su paketi koji omogućuju osnovni rad sa razmjenom slika na „Azure storage-u“, korištenje gotovih ikona, pomoć pri korištenju „JSON-a“ i slično.

Prvi od nabrojanih je „Azure storage Blob“ što je rješenje za pohranu podataka u oblaku (eng. *Cloud*). „Blob“ pohrana se u aplikaciji konkretno koristi za pohranu korisničkih slika te ovaj način je optimiziran za pohranjivanje velikih količina nestrukturiranih podataka, odnosno podataka koji se ne pridržavaju određenog podatkovnog modela ili definicije kao što su tekstualni ili binarni podaci. Sljedeći paket kojeg je bitno spomenuti je „MathApps Metro IconPack“ koji sadrži veliki broj ikonica za „WPF“ i „UWP“ aplikacije. Ikone iz ovog paketa korištene su za izradu navigacije te nekih gumbova. Također je korišten i „Newtonsoft Json“ paket koji pomaže pri serijalizaciji i deserijalizaciji “JSON“ objekata koji također može i koristiti za pretvorbu „LINQ-a“ u „JSON“. [6]

3.2. Firebase

U ovom poglavlju detaljnije je opisan ranije spomenuti „Firebase“ koji je u suštini iskorišten za autentikaciju i kao baza podataka aplikacije. „Firebase“ je „Google-ov“ proizvod koji programerima pomaže da jednostavno izgrade, upravljaju i razvijaju svoje aplikacije. Razvojnim programerima ubrzava proces izgradnje aplikacije uz garanciju sigurnosti te ne zahtijeva programiranje na strani „Firebase-a“ što olakšava učinkovitiju upotrebu njegovih značajki. Ova platforma pruža usluge za „Android“, „iOS“, „Web“ i „Unity“, omogućuje pohranu u oblaku te koristi „NoSQL“ za pohranu podataka unutar baze. Neke od najvećih usluga koje ova platforma nudi su baza podataka u stvarnom vremenu (eng. *RealTime Database*), pohrana podatka u oblaku (eng. *Cloud Firestore*), autentikacija, usluga daljinskog upravljanja, hosting i drugo. Od svega navedenoga, za izradu praktičnog dijela diplomskog rada korištena je baza podataka u stvarnom vremenu i autentikacija dok je za pohranu korisničkih fotografija korišten „Azure storage“. Spomenuta baza podataka u stvarnom vremenu je „NoSQL“ baza podataka temeljena na oblaku koja upravlja podacima jako velikom brzinom izraženom u milisekundama dok se promjene u bazi mogu odmah očitovati i na platformi i unutar aplikacije zbog čega i dolazi taj naziv. Najjednostavnije rečeno, ova baza podataka se može smatrati jednom velikom „JSON“ datotekom. Druga korištena stvar na ovoj platformi je autentikacija. Usluga „Firebase“ autentikacije pruža biblioteke korisničkog sučelja i „SDK-ove“ (eng. *Software Developer Kit*) jednostavne za autentifikaciju korisnika unutar aplikacije. Koristeći ovu uslugu mnogo se uštađuje na vremenu i smanjuje se radna snaga i napor koji je potreban za razvoj i održavanje usluge autentikacije korisnika. Ranije spomenuto daljinsko upravljanje se odnosni na daljinsko postavljanje konfiguracije što čini objavljivanje ažuriranja korisnicima lakšim. Promjene mogu varirati od promjene komponenti korisničkog sučelja do promjene ponašanja aplikacija te se ova stvar često koristi pri objavljivanju sezonskih ponuda i sadržaja u aplikaciji koja ima ograničen rok trajanja. U nastavku su navedene još neke značajke „Firebase-a“ koje nisu korištene u izradi praktičnog dijela rada, a uglavnom se odnose na poboljšanje kvalitete aplikacija, statistiku, dodatke i slično. Usluge koje se još nude su „Crashlytics“ što se koristi za dobivanje izvješća o padu u stvarnom vremenu što se kasnije može koristiti za poboljšanje kvalitete aplikacije. Sljedeća mogućnost je praćenje performansi (eng. *Performance monitoring*) što daje uvid u karakteristike performansi aplikacije. Također postoji usluga za testiranje aplikacija koja se naziva „Test Lab“ što je infrastruktura za testiranje aplikacija temeljena na oblaku koja podržava testiranje aplikacije na velikom broju uređaja i konfiguracija uređaja. Zadnja usluga koja je vrijedna spomena je usluga predviđanja (eng. *Predictions*) koja koristi strojno učenje za analitičke podatke aplikacije zatim stvara dinamičke korisničke segmente koji se temelje na ponašanju korisnika te su nakon toga dostupni za korištenje. [8]

3.2.1. Firebase Auth REST API

Kako za autentikaciju, tako i za neke druge osnovne stvari vezane oko korisničkih računa, koristi se „Firebase Auth REST API“. Na ovaj „REST API“ se mogu postavljati upiti ako se posjeduje „API_KEY“ što je „Web API“ ključ koji se može dobiti na stranici postavki projekta unutar administratorske konzole na „Firebase-u“. Za korištenje ove usluge unutar svakog zahtjeva se treba poslati tijelo zahtjeva (eng. *Request body payload*) u kojemu se nalaze određeni parametri. Prilikom implementacije ove značajke unutar aplikacije jako je korisno koristiti dokumentaciju kako bi se znalo na koju krajnju točku poslati zahtjev te koje parametre postaviti prije slanja kako bi se mogao dobiti uspješan odgovor. U ovom odjeljku nabrojane su samo korištene značajke ovog „API-ja“ uz kratak opis svake. Prva i najbitnija korištena značajka je registracija korisnika s adresom elektroničke pošte i lozinkom te omogućavanje logiranja korisnika nakon registracije. Nakon logiranja koriste se krajnja točka koja omogućava razmjenu tokena osvježavanja (eng. *Refresh token*) kako bi se resetirao korisnikov „ID token“ koji je potreban za dobivanje odgovora s drugih krajnjih točaka „Rest API-ja“. Nadalje, korištena je značajka koja omogućuje slanje zahtjeva za resetiranje lozinke te verifikacija potvrde lozinke koja korisniku omogućuje da promjeni lozinku koja će istome biti poslana na adresu elektroničke pošte. Nakon ulogiranja unutar aplikacije, korisniku je s ovom značajkom omogućena i promjena adrese elektroničke pošte, promjena lozinke te ažuriranje profila. Prilikom prve registracije korisnik je obavezan verificirati svoju adresu elektroničke pošte kako bi se potvrdilo da je adresa validna i njegova. Za svaku krajnju točku ukoliko se pošalju validni parametri unutar tijela zahtjeva dobiva se odgovor. Unutar odgovora se može dobiti tekst pogreške ukoliko nešto nije dobro napravljeno što programeru jasno daje do znanja u čemu je problem. Unutar aplikacije ova značajka je iskorištena kako bi se korisniku ispisivale jasne statusne poruke u slučaju greške. [7]

3.3. Inkscape

Zadnji alat koji je korišten u manjoj mjeri je alat „Inkscape“. Ovaj alat je besplatni uređivač vektorske grafike otvorenog koda za „Windows-e“. Nudi skup značajki i naširoko se koristi za umjetničke i tehničke ilustracije kao što su crteži, logotipi, tipografija i slično. U ovoj aplikaciji alat je poslužio za izradu logotipa. S ovim alatom omogućeno je da logotip bude napravljen koristeći vektorsku grafiku te da se s bojama može prilagoditi tematici aplikacije. Koristeći vektorsku grafiku omogućuje se oštar ispis i prikaz u neograničenoj rezoluciji te nije vezan na fiksni broj piksela kao rasterska grafika. Napravljen je logotip u dvije boje koji se koristi unutar same aplikacije. [9]

4. Specifikacija zahtjeva

U narednom poglavlju navedena je specifikacija softverskih zahtjeva stavljajući fokus na opis aplikacije koji obuhvaća svrhu, funkcionalnosti, opis sučelja, korisnika i karakteristika korisnika. Nakon opisa aplikacije navedeni su specifični i nefunkcionalni zahtjevi gdje u specifične zahtjeve spada podjela na funkcionalne zahtjeve po ulogama korisnika. Sukladno aplikaciji postoje funkcionalni zahtjevi za modul korisnika (eng. *Member*), funkcionalni zahtjevi za modul radnika (eng. *Worker*), funkcionalni zahtjevi za modul direktora (eng. *Director*) te finalno funkcionalni zahtjevi za modul administratora (eng. *Administrator*). Navedene uloge predstavljaju člana auto kluba, običnog radnika, direktora auto kluba te administratora koji je ujedno i vlasnik aplikacije. Finalno u nefunkcionalnim zahtjevima obuhvaćeni su zahtjevi koji određuju atribute kvalitete sustava.

4.1. Svrha specifikacije zahtjeva

Danas je u razvoju softvera veoma bitno imati specifikaciju zahtjeva jer se tako prikupljajući informacije dobiva šira slika o aplikaciji te opis kako bi se sustav trebao ponašati. O bilo kojoj vrsti sustava ili aplikaciji da je riječ, uvijek je dobro napisati specifikaciju zahtjeva prije faze implementacije, a pogotovo ukoliko su korisnik i programer dvije različite strane. U ovoj situaciji to nije slučaj, no u narednom poglavlju svakako je navedena specifikacija zahtjeva u potpunosti kako bi faza implementacije bila lakša. Kada bi u projektu bile uključene obje strane na ovaj način se pruža zajednička slika o tome kako bi sustav trebao izgledati te je to odličan period u kojemu se može dogovoriti za određene pojedinosti o aplikaciji. Štoviše, specifikacija zahtjeva najviše pomaže programerima kad dođe do faze implementacije, a čak je korisna i u fazi testiranja kada se funkcionalnosti uspoređuju s unaprijed definiranim zahtjevima. Tokom cijelog razvoja bitno je ažurirati dokument specifikacije zahtjeva kako bi sve bilo u skladu, a pogotovo ako s druge strane imamo klijenta koji može postaviti nove funkcionalne zahtjeve te se na taj način može izbjeći bilo kakvo kašnjenje ili problemi. Kada govorimo o vremenu koje je potrebno posvetiti na specifikaciju zahtjeva, ne možemo ga lako predvidjeti. Svaki projekt je jedinstven i drugačijeg obuhvata što otežava vremensko predviđanje, no ova stavka se može izvući iz statističkih podataka prošlih projekata. U nekima od statističkih podataka „NASA-e“ pokazana je jasna veza između ukupnog troška projekta i udjela resursa uloženih u početne faze razvoja softvera (što uključuje i specifikaciju zahtjeva). Ovi rezultati su pokazali ukoliko se manje od 5% proračuna dodijeli fazi planiranja, krajnji troškovi znaju varirati od 40% do 170%, dok u situacijama uloga od 10% do 20%, krajnji troškovi budu ispod 30%. [10]

4.2. Opis aplikacije

U ovom poglavlju površno je opisana sama aplikacija gdje se uz opis navodi i njena svrha te opseg. Tema za aplikaciju proizašla je iz toga što je danas sve više i više turista u različitim gradovima diljem svijeta, a putovanje s lokacije na lokaciju bez privatnog automobila postalo je nemoguće. Kao rješenje sve se više i više razvijaju firme koje nude usluge prijevoza, a isto tako i usluge iznajmljivanja automobila i različitih voznih sredstava. Kao što se već moglo zaključiti, tematika aplikacije ide u tom smjeru, a njen cilj je olakšati korisnicima samo iznajmljivanje automobila u auto kućama koje su rasprostranjeni po većim gradovima u Europi. U nastavku je detaljnije navedena svrha aplikacije, odnosno što aplikacija u pravilu omogućava te njen opseg, odnosno za koga je sve aplikacija namijenjena.

4.2.1. Svrha aplikacije

Što se tiče svrhe aplikacije, njen cilj je olakšati život njenim korisnicima, a isto tako i radnicima auto kuća koji su na kraju krajeva isto tako korisnici aplikacije. Ovisno o ulozi koju korisnik posjeduje, aplikacija sadrži različite mogućnosti. Iz aspekta korisnika člana auto kuće, funkcionalnost je jednostavna za korištenje koja korisniku omogućuje registraciju nakon čega je moguće koristiti aplikaciju. Na samom ulazu u aplikaciju članu je ponuđen kratak statistički osvrt na poslovanje firme koja iznajmljuje aute te mu se ovisno o auto kući u koju je dodan nudi ponuda svih vozila namijenjenih za iznajmljivanje. Ukoliko korisniku paše vozilo, a isto tako i njegova cijena, može ga rezervirati na određeni broj dana i preuzeti u istoj poslovnici. Aspekt djelatnika je drugačiji jer se sastoji od dvije uloge, a to je radnik i direktor. Kao direktor i radnik funkcionalnosti aplikacije su slične, no direktor ima malo veće ovlasti od radnika što je opisano u daljnjim funkcionalnim zahtjevima korisnika. U ovom odjeljku opisuje je svrha iz pogleda radnika i direktora, a to je da im je za poslovnicu u kojoj rade, omogućena administracija vozila te prezentacija istih svojim korisnicima. Administrator sustava je osoba koja se brine da sve radi uredno te ima posebne mogućnosti kojima ne mogu pristupiti ostale uloge koje korisnici posjeduju. Detaljnije o ulogama se može pročitati u funkcionalnim specifikacijama koje su navedene u jednom od sljedećih poglavlja.

4.2.2. Opseg aplikacije

Pod pojmom opseg aplikacije se podrazumijeva za koga je aplikacija namijenjena, a to može biti bilo koji servis za iznajmljivanje auta koji posluje na više poslovnica. Ova aplikacija kompatibilna je bilo kojoj „auto kući“ jer se rezervacija/iznajmljivanje automobila (eng. *Rent a car*) danas više manje provodi svagdje na isti način. Što se tiče korisnika aplikacije, dijele se na uloge što omogućava korist aplikacije svima,.od njenog vlasnika sve do krajnjeg korisnika.

4.3. Funkcionalni i nefunkcionalni zahtjevi

U narednom poglavlju detaljno su obrađene sve funkcionalnosti aplikacije uz što se ubrajaju i karakteristike korisnika što podrazumijeva funkcionalne zahtjeve za svaku od ranije navedenih uloga, a to su funkcionalni zahtjevi za modul člana, radnika, direktora i administratora. Na kraju su navedeni nefunkcionalni zahtjevi u što spadaju zahtjevi koji određuju atribute kvalitete sustava.

4.3.1. Funkcionalnosti aplikacije

Ovo poglavlje obuhvaća sve funkcionalnosti koje aplikacija nudi raspoređene po većim cjelinama. U veće cjeline mogu se rasporediti registracija i autentikacija korisnika, kratki statistički osvrt na urede i korisnike, rezervacija automobila, manipulacija nad posjedovnim vozilima, administracija korisnika, ažuriranje i pregled vlastitog profila i administracija ureda.

4.3.1.1. Registracija korisnika

Funkcionalnost registracije je namijenjena za svakog novog korisnika aplikacije, bio član, radnik ili direktor nakon čega je administrator zadužan za dodjelu ovlasti i ureda ovisno o korisnikovoj poziciji. Korisniku se prikazuje sučelje unutar kojega upisuje svoju adresu elektroničke pošte, ime i prezime te lozinku uz potvrdu lozinke. Korisnik prilikom registracije mora unijeti validnu adresu elektroničke pošte, u suprotnom neće moći verificirati svoj račun te se neće moći ulogirati u aplikaciju. Osim same verifikacije, adresa mora pratiti standard za pisanje adresa elektroničke pošte te ukoliko je adresa već zauzeta, korisnik ne može napraviti svoj račun. Kada korisnik unese sve navedeno za kraj mora potvrditi svoju lozinku te mu se otključava gumb za registraciju. U slučaju bilo kakve greške korisniku se ispisuje statusna poruka s jasnim značenjem.

4.3.1.2. Autentikacija korisnika

Funkcionalnost autentikacije je omogućena svakom korisniku te bez nje ne može pristupiti ostalim funkcionalnostima aplikacije. Na prvom prikazanom zaslonu prilikom svakog pokretanja aplikacije korisnik mora unijeti svoje kredencije kako bi se ulogirao u aplikaciju. Korisniku je omogućena jednostavna kretnja između registracije i prijave unutar početnog zaslona. Unutar ove funkcionalnosti spada i funkcionalnost resetiranja lozinke ukoliko ju je korisnik zaboravio. Klikom na poveznicu ispod forme za registraciju korisniku se otvara nova forma u kojoj može unijeti svoju adresu elektroničke pošte te na taj način dobiti zahtjev za reset lozinke. Nakon što korisnik unese potrebne kredencije može ući u aplikaciju te koristiti ostale funkcionalnosti. Kao i kod funkcionalnosti registracije, u slučaju bilo kakve greške korisniku se ispisuje statusna poruka s jasnim značenjem.

4.3.1.3. Pregled statistike (naslovna stranica) i navigacija

Nakon autentifikacije korisniku se otvara naslovna stranica aplikacije koja nudi funkcionalnost kratkog osvrta na statistiku poslovanja auto kuća napravljena u stilu modernijih aplikacija. Uz naslovnu stranicu korisniku se nudi funkcionalnosti navigacije po većim cjelinama navedenima u uvodu ovog poglavlja. Funkcionalnost navigacije korisniku omogućava kretanje po raznim drugih funkcionalnostima koje su mu dostupne ovisno o dodijeljenoj ulozi. Globalne mogućnosti neovisno o ulozi nude kretanje po cjelinama za rezervacije automobila, manipulaciju na posjedovnim vozilima, administraciju korisnika, ažuriranje i pregled vlastitog profila te administraciju ureda. Na samom kraju navigacije korisnik ima mogućnost odjavljivanja iz aplikacije čime završava njegova sesija.

4.3.1.4. Pregled i ažuriranje korisničkog profila

Funkcionalnost koja je dostupna svakom korisniku neovisno o njegovoj ulozi je pregled vlastitog profila. Ova funkcionalnost korisniku nudi razne druge mogućnosti koje se mogu navesti kao manje funkcionalnosti ove cjeline. Prilikom ulaska unutar cjeline za pregled vlastitog računa/profila (unutar aplikacije, eng. *Account*) korisnik može vidjeti svoje podatke kao što su njegov vlastiti identifikator, ime i prezime, adresu elektroničke pošte, vrijeme registracije u sustav te mu je vidljiva kratka forma koja služi za promjenu lozinke (što se razlikuje od reset-a lozinke što je funkcionalnost koja je omogućena korisnicima prije autentikacije). Osim pregleda, korisnik može promijeniti vlastito ime i prezime (što je omogućeno jer može postojati slučaj da ga je prvotno krivo naveo) te svoju adresu elektroničke pošte. Funkcionalnost promjene vlastite fotografije profila svrstava se u ovu cjelinu jer je iste tematike, a promjena fotografije odvija se na dijelu u kojem se nalazi i navigacija. Ulaskom u cjelinu vlastitog korisničkog profila uređivanje podataka nije jedina funkcionalnost koja se nudi. Korisnik je u mogućnosti kretati se između sporednih cjelina među kojima su profil i privilegije (unutar aplikacije, eng. *Profile* i *Privileges*). Na prozoru za privilegije korisnik, neovisno o ulozi, može vidjeti sve mogućnosti aplikacije među kojima su kvačicom označene njegove ovlasti. Isto tako korisnik vidi ime svoje uloge unutar aplikacije. Korisnicima je onemogućeno mijenjane svojih ovlasti te im ova funkcionalnost omogućuje samo pregled i služi u informativne svrhe. Ovlasti su raspoređene po svakoj pojedinoj cjelini te se ispred svake ovlasti može vidjeti i cjelina kojoj pripada te da li je ta cjelina uopće vidljiva trenutno logiranom korisniku. Kao primjer može se navesti cjelina uredi što predstavlja auto kuće/poslovnice (unutar aplikacije, eng. *Offices*) gdje korisnik može vidjeti ima li pristup određenoj cjelini te ako ima, kojim točno sporednim funkcionalnostima ima pristup i njima može upravljati. Točne ovlasti vezane uz pojedinu ulogu navedene su u funkcionalnim zahtjevima vezanog uz svaki pojedini modul korisnika gdje najviše ovlasti ima korisnik s ulogom administratora, zatim direktor, radnik te na kraju sam član auto kuće.

4.3.1.5. Pregled i administracija poslovnica

Navedena funkcionalnost je dostupna administratoru sustava i djelatnicima te omogućuje pregled svih dosadašnjih poslovnica, administraciju nad istima te kreiranje novih poslovnica uz dodjeljivanje direktora pojedinoj poslovnici. Prilikom ulaska u cjelinu poslovnica (unutar aplikacije, eng *Offices*) korisnik vidi pregled poslovnica unutar tablice (eng. *Datagrid*) zajedno sa brojem ukupnih zapisa koja mu osim samog pregleda svih postojećih poslovnica omogućuje i detalj pregleda za pojedini zapis te različite operacije nad odabranim zapisom. Korisnik unutar tablice može vidjeti ime poslovnice, državu u kojoj poslovnica posluje, grad, adresu, status rada te operacije nad odabranom poslovnicom koje su reprezentirane gumbovima s relevantnom oznakom koja predstavlja akciju. Operacije koje su mu omogućene su zatvaranje poslovnice, otvaranje poslovnice ukoliko ju je prvotno zatvorio (Po početnim vrijednostima poslovnica nakon kreiranja dobiva vrijednost statusa rada otvoreno) te brisanje odabrane poslovnice. Vrijednosti u tablici korisnik može filtrirati po ključnim riječima vrijednosti koje predstavljaju sadržaj tablice. Prilikom odabira poslovnice pritiskom na jednu unutar tablice, korisnik dobiva detalje poslovnice u kojoj osim navedenih podataka unutar tablice može vidjeti još neke podatke kao što su identifikatori poslovnica i direktora te ime i prezime direktora poslovnice. Neki od podataka unutar detalja omogućeni su administratoru na izmjenu kao što je samo ime poslovnice, država poslovanja, grad i adresa dok su mu za identifikatore određenih polja omogućeni gumbovi za kopiranje vrijednosti u međuspremnik ukoliko su mu dalje potrebni. Nakon izmjene željenih podataka, administrator može ažurirati poslovnicu s novim vrijednostima. Ukoliko administrator pritisne gumb za kreiranje nove poslovnice otvara mu se novi zaslon unutar istog prozora koji mu nudi novu tablicu u kojoj se nalaze svi korisnici sa ulogom direktor. Ovi zapisi su namijenjeni za pregled i odabir te na ovom zaslonu nije moguće njihovo uređivanje. Moguće je filtriranje po ključnim riječima vrijednosti unutar tablice te se nakon odabira novog direktora ispod tablice automatski popunjavaju vrijednosti koje su vezane uz direktora. Nakon odabira, administratoru preostaje unijeti ime poslovnice, državu poslovanja, grad i adresu nakon čega može kreirati novi ured. Nakon kreiranja ureda ili ako neki od podataka nisu popunjeni, administratoru se ispisuje jasna statusna poruka na vrhu ekrana s određenim uputama. Administratoru je također i na ovom zaslonu omogućeno kopiranje identifikatora direktora u međuspremnik što može rezultirati bržem pronalasku direktora unutar druge cjeline, ako je to potrebno. Nakon kreiranja nove poslovnice automatski se ažurira tablica na pregledu poslovnica te su moguće druge operacije s novom poslovnicom. Sve dosad navedene stvari ove cjeline dostupne su samo administratoru sustava, dok je direktoru i radnicima omogućen pregled poslovnica i svih radnika njihove poslovnice, bez mogućnosti kreiranja novih poslovnica, uređivanja ili provedbe određenih operacija. Ovih ulogama također je omogućeno filtriranje poslovnica i zaposlenika po ključnim riječima vrijednosti iz tablica.

4.3.1.6. Pregled i administracija svih korisnika

Pregled i ažuriranje svih korisnika je jedina funkcionalnost koja je dostupna samo administratoru sustava. Ova cjelina se sastoji od više zaslona koja unutar sebe sadrži manju navigaciju gdje se administrator može kretati između manjih cjelina. Manje cjeline koje su dostupne unutar cjeline korisnika (unutar aplikacije, eng. *Users*) su korisnici, radnici i članovi (unutar aplikacije, eng. *Users*, *Workers* i *Members*). Prvi zaslon omogućuje pregled svih korisnika aplikacije neovisno o njihovim ulogama te su korisnici prikazani unutar tablice. Podaci koje administrator može vidjeti unutar tablice su ime i prezime korisnika, uloga, adresa elektroničke pošte, identifikator korisnika, status o mogućnosti ulaska unutar aplikacije te operacije nad korisnicima. Operacije koje su dostupne administratoru je da blokira korisnika te ga odblokira ukoliko ga je prvotno blokirao (prilikom registracije svaki korisnik ima mogućnost ulaska u aplikaciju). Osim operacije za blokiranje, administrator je u mogućnosti dodjeljivati uloge korisnicima. Odabirom pojedinog korisnika popunjavaju se detalji o tom korisniku ispod tablice. Svaki korisnik može ažurirati svoje podatke te podaci o pojedinom korisniku nisu mogući za ažuriranje, čak ni administratoru sustava. Podaci koje administrator može vidjeti su identifikatori korisnika, adresa elektroničke pošte, ime i prezime, uloga, vrijeme registracije te vrijeme zadnjeg korištenja aplikacije. Osim pregleda moguće je i filtriranje po ključnim riječima vrijednosti unutar tablice. Kada se novi korisnici registriraju unutar aplikacije zaključane su im sve mogućnosti dok im administrator ne dodijeli prava. Nakon dodjele uloge slijedi pridruživanje radnika ili članova određenoj auto kući. Drugi zaslon administratoru nudi pregled tablice sa svim zaposlenicima i njihovim detaljima te pregled tablice s poslovnicama i njihovim detaljima. Prilikom odabira radnika i poslovnice automatski se popunjava forma ispod tablica s imenom radnika te imenom direktora odabrane poslovnice kojemu se radnik dodjeljuje. Nakon odabira obje stavke, administrator može pokrenuti akciju dodjeljivanja radnika određenom uredu, a ukoliko je jedna od stavki neodabrana, administrator dobiva jasnu statusnu poruku gdje se navodi da je potrebno odabrati sve vrijednosti. Ovaj zaslon također nudi filtriranje radnika i poslovnica po ključnim riječima vrijednosti unutar korespondirajućih tablica. Zadnji zaslon ove cjeline je zaslon s članovima auto kuća, a ima istu ulogu kao i prethodni zaslon, odnosno dodjeljivanje članova auto kući kojoj pripadaju. Administrator ovdje vidi tablicu sa svim članovima i svim poslovnicama. Kao i u prethodnom zaslonu, s odabirom određenog člana i odabirom auto kuće omogućuje se dodjela, u suprotnom korisnik dobiva statusnu poruku kako određene stavke nisu odabrane. Nakon što administrator sustava odradi svoj dio posla, radnicima i članovima se otključavaju mogućnosti aplikacije ovisno o ovlastima koje im je dodijelio. Radnicima se otključavaju ovlasti za manipulaciju nad posjedovnim vozilima poslovnice kojoj su dodijeljeni, a članovima se otključavaju ovlasti za rezervaciju vozila auto kuće kojoj pripadaju. Na ovaj način ograničavaju se nedozvoljene radnje te valjan i kontroliran rad sustava.

4.3.1.7. Pregled i manipulacija nad posjedovnim vozilima

Sljedeća cjelina razlikuje se od ostalih iz razloga što ima različite poglede iz različitih uloga. Ovu cjelinu mogu vidjeti djelatnici poslovnica te administrator. Radnik ili direktor poslovnice mogu vidjeti vozila koja njihova poslovnica posjeduje te izvršavati operacije nad istima. Mogu vidjeti tablicu s podacima i vozilima što uključuje brend automobila, model, godinu proizvodnje, dnevnu cijenu najma te dostupnost. Operacije koje se nude nad automobilima su označavanje dostupnosti automobila, označenje nedostupnosti automobila te njegovo brisanje. Prilikom odabira jednog od automobila djelatnik ispod tablice može vidjeti njegove podatke te ažurirati isti automobil što se odnosi na njegov brend, model, godinu proizvodnje te dnevnu cijenu najma. Kao i na svim drugim cjelinama, djelatnicima su omogućene filtracije nad tablicom po ključnim riječima vrijednosti koje se u tablici nalaze. Djelatnicima je omogućeno dodavanje novih automobila u sustav nakon čega automobil automatski postaje dostupan članovima za iznajmljivanje. Razlika kod uloge administratora je što administrator ima pravo dodavanja automobila u bilo koju poslovnicu što mu je omogućeno na način da uz tablicu automobila ima i pregled poslovnica te automobile može filtrirani po svakoj poslovnici. Prvotni pogled koji administrator ima su sve poslovnice i svi automobili gdje se nakon odabira poslovnice količina automobila sužava.

4.3.1.8. Pregled vlastitih rezervacija te rezervacija novog automobila

Sljedeća cjelina je ona cjelina koja je ključna za članove gdje se omogućava rezervacija automobila unutar auto kuće kojoj član pripada. Ova cjelina je omogućena svakoj ulozi te se u svakoj ulozi i razlikuje. Kao administrator sustava vidljiv je pregled svih poslovnica s korespondirajućim rezervacijama za svaku poslovnicu. Administrator može filtrirati podatke ovisno o poslovnici te ažurirati iste, a također mu je i omogućena filtracija po ključnim riječima vrijednosti iz tablica. Sljedeći pogledi su pogledi od strane radnika i direktora koji mogu vidjeti automobile poslovnice u kojoj rade te isto tako ažurirati i filtrirati iste. Najključnija uloga u ovoj cjelini je uloga člana kojemu su vidljive sve vlastite rezervacije i kreiranje novih rezervacija. Član može ažurirati i filtrirati vlastite registracije te kod kreiranja odabrati raspon dana unutar kalendara u kojemu će automobil biti iznajmljen. Kako bi se izbjegle nedozvoljene radnje kod odabira raspona postavljena su ograničenja koja članu blokiraju odabir datuma koji su prošli te odabir početka najma koji je veći od odabira završetka najma. Prilikom rezervacije članu se automatski računa cijena ukupnog najma koja se dobije s izračunom dana u kojima je automobil u njegovom posjedu sa cijenom najma po danu. Nakon rezervacije i zabilježbe u sustavu, član može preuzeti automobil u poslovnici te podmiriti navedeni dug preko uplate ili prilikom preuzimanja automobila. Prilikom rezervacije automobila, ostalim korisnicima automobil nije više dostupan za iznajmljivanje te se prilikom vraćanja automobila član zabilježava te se dostupnost resursa postavlja na inicijalno stanje.

4.3.2. Karakteristike korisnika

U ovom poglavlju objašnjene su karakteristike aplikacije iz pogleda svakog korisnika te predstavljaju funkcionalne zahtjeve za svaki od modula korisnika. Unutar poglavlja koje jasno opisuje funkcionalnosti aplikacije navede su već neke stvari, no ovaj dio prikazuje tablični prikaze za svaku pojedinu ulogu i njezinim mogućnostima.

4.3.2.1. Funkcionalni zahtjevi za modul „administrator“

Administrator je korisnik i vlasnik aplikacije koji ima najveće ovlasti od svih uloga. Osim svega navedenoga unutar tablice, administrator također može pristupiti bazi podataka te sudjelovati u održavanju sustava.

Tablica 1: Funkcionalni zahtjevi za modul „administrator“

Cjelina	Akcije
Stranica prijave	Registracija Autentifikacija Reset lozinke
Početna stranica	Pregled naslovne stranice Pregled statistike
Rezervacije	Filtriranje po ključnim riječima i poslovnicama Pregled rezervacija svih poslovnica Ažuriranje rezervacija svih poslovnica Brisanje rezervacija svih poslovnica Potvrđivanje uplate vezane uz rezervaciju Ukidanje potvrde uplate vezane uz rezervaciju
Automobili	Filtriranje po ključnim riječima i poslovnicama Kreiranje automobila u bilo kojoj poslovnici Pregled automobila svih poslovnica Ažuriranje automobila svih poslovnica Brisanje automobila svih poslovnica Označavanje automobila dostupnim Označavanje automobila nedostupnim
Korisnici	Filtriranje po ključnim riječima Dodjeljivanje uloga korisnicima Blokiranje korisnika Odblokiranje korisnika Pregled korisnika svih poslovnica Pregled radnika svih poslovnica Dodjeljivanje radnika poslovnicama Pregled članova svih poslovnica Dodjeljivanje članova poslovnicama
Poslovnice	Filtriranje po ključnim riječima Kreiranje novih poslovnica Pregled svih poslovnica Ažuriranje svih poslovnica Brisanje poslovnica Dodjeljivanje direktora određenoj poslovnici Otvaranje poslovnice Zatvaranje poslovnice
Račun	Pregled vlastitih podataka Ažuriranje profila Promjena adrese elektroničke pošte Promjena lozinke Pregled privilegija vezanih uz njegovu ulogu

4.3.2.2. Funkcionalni zahtjevi za modul „direktor“

Direktor je korisnik koje je vlasnik poslovnice te mu je uloga dodijeljena od strane administratora. Direktor kao nadređeni u svojoj poslovnici ima nešto veća prava od radnika poslovnice. Direktor poslovnice u dogovoru s administratorom može svoju ulogu dodijeliti nekoj drugoj osobi te isto tako dati otkaz (izbrisati iz sustava) određenog radnika koji je zaposlen u njegovoj poslovnici.

Tablica 2: Funkcionalni zahtjevi za modul „direktor“

Cjelina	Akcije
Stranica prijave	Registracija Autentifikacija Reset lozinke
Početna stranica	Pregled naslovne stranice Pregled statistike
Rezervacije	Filtriranje po ključnim riječima Pregled rezervacija svoje poslovnice Ažuriranje rezervacija svoje poslovnice Brisanje rezervacija svoje poslovnice Potvrđivanje uplate vezane uz rezervaciju Ukidanje potvrde uplate vezane uz rezervaciju
Automobili	Filtriranje po ključnim riječima Kreiranje automobila u svojoj poslovnici Pregled automobila svoje poslovnice Ažuriranje automobila svoje poslovnice Brisanje automobila svoje poslovnice Označavanje automobila dostupnim Označavanje automobila nedostupnim
Poslovnice	Filtriranje po ključnim riječima Pregled svih poslovnica Pregled radnika vlastite poslovnice Otvaranje vlastite poslovnice Zatvaranje vlastite poslovnice Brisanje vlastite poslovnice
Račun	Pregled vlastitih podataka Ažuriranje profila Promjena adrese elektroničke pošte Promjena lozinke Pregled privilegija vezanih uz njegovu ulogu

4.3.2.3. Funkcionalni zahtjevi za modul „radnik“

Radnik je korisnik koji je djelatnik poslovnice te mu je uloga dodijeljena od strane administratora. Radnik ima sva potrebne ovlasti za rad unutar svoje poslovnice kako bi sustav mogao uspješno funkcionirati. Radnik ima slična prava kao i direktor poslovnice jer su na kraju krajeva obje uloge djelatnici iste poslovnice, ali su mu neke mogućnosti koje zahtijevaju odluku nadređenog ipak onemogućene. Radnici imaju uvid u cijene svih rezervacija vezanih uz poslovnicu u kojoj su zaposleni te naplaćuju iste prilikom preuzimanja automobila unutar poslovnice.

Tablica 3: Funkcionalni zahtjevi za modul „radnik“

Cjelina	Akcije
Stranica prijave	Registracija Autentifikacija Reset lozinke
Početna stranica	Pregled naslovne stranice Pregled statistike
Rezervacije	Filtriranje po ključnim riječima Pregled rezervacija svoje poslovnice Ažuriranje rezervacija svoje poslovnice Brisanje rezervacija svoje poslovnice Potvrđivanje uplate vezane uz rezervaciju Ukidanje potvrde uplate vezane uz rezervaciju
Automobili	Filtriranje po ključnim riječima Kreiranje automobila u svojoj poslovnici Pregled automobila svoje poslovnice Ažuriranje automobila svoje poslovnice Brisanje automobila svoje poslovnice Označavanje automobila dostupnim Označavanje automobila nedostupnim
Poslovnice	Filtriranje po ključnim riječima Pregled svih poslovnica Pregled suradnika poslovnice
Račun	Pregled vlastitih podataka Ažuriranje profila Promjena adrese elektroničke pošte Promjena lozinke Pregled privilegija vezanih uz njegovu ulogu

4.3.2.4. Funkcionalni zahtjevi za modul „član“

Član je korisnik aplikacije koji je ujedno i član auto kuće koja iznajmljuje aute. Član ima najmanje ovlasti, ali dovoljne za ispunjenje cilja aplikacije, a to je rezervacija vozila.

Tablica 4: Funkcionalni zahtjevi za modul „član“

Cjelina	Akcije
Stranica prijave	Registracija Autentifikacija Reset lozinke
Početna stranica	Pregled naslovne stranice Pregled statistike
Rezervacije	Filtriranje po ključnim riječima Kreiranje novih rezervacija Pregled dostupnih automobila poslovnice Pregled vlastitih rezervacija Ažuriranje vlastitih rezervacije Brisanje vlastitih rezervacija Slanje potvrde o uplati Ukidanje potvrde o uplati
Račun	Pregled vlastitih podataka Ažuriranje profila Promjena adrese elektroničke pošte Promjena lozinke Pregled privilegija vezanih uz njegovu ulogu

4.3.3. Nefunkcionalni zahtjevi

U sljedećem poglavlju govori se o važnosti nefunkcionalnih zahtjeva uz njihovo objašnjenje te su navedeni nefunkcionalni zahtjevi aplikacije u što se podrazumijevaju zahtjevi koji određuju atribute kvalitete sustava.

Nefunkcionalni zahtjevi su skup specifikacija koje opisuju operativne mogućnosti i ograničenja sustava te na taj način pokušavaju poboljšati njegovu funkcionalnost. Kada govorimo o nefunkcionalnim zahtjevima govorimo o zahtjevima koji ocrtavaju koliko će dobro funkcionirati sustav uzimajući u obzir stvari poput brzine, sigurnosti, pouzdanosti, integriteta podataka i slično. Kako nam i funkcionalni i nefunkcionalni zahtjevi nude opis specifičnih karakteristika koje proizvod mora imati, iz naziva se može zaključiti da je fokus usmjeren na različite stvari. Dok funkcionalni zahtjevi definiraju što softverski proizvod mora raditi, njegove značajke i funkcije, nefunkcionalni zahtjevi određuju atribute kvalitete sustava kao što su performanse, skalabilnost, prenosivost, kompatibilnost, pouzdanost, mogućnost održavanja, dostupnost, sigurnost, lokalizaciju, upotrebljivost i drugo. [11]

4.3.3.1. Performanse i skalabilnost

Nefunkcionalni zahtjevi performansi i skalabilnosti odgovaraju na pitanje koliko brzo sustav vraća rezultate te koliko će se izvedba odgovaranja promijeniti s većim radnim opterećenjima. Performanse definiraju koliko brzo softverski sustav ili njegov određeni dio reagira na radnje određenih korisnika pod određenim radnim opterećenjem, drugim riječima to znači da ova metrika objašnjava koliko dugo korisnik mora čekati prije nego se dogodi ciljna operacija. Što se tiče skalabilnosti, ono procjenjuje najveća radna opterećenja pod kojima će sustav i dalje ispunjavati zahtjeve performansi. Ovaj nefunkcionalni zahtjev je veoma bitan za uzeti u obzir jer se broj sesija u aplikaciji može uvelike povećati nakon provođena marketinške kampanje te sustav na to mora biti spreman. [11]

4.3.3.2. Prenosivost i kompatibilnost

Prenosivost i kompatibilnost su sljedeća dva ključna pojma u svijetu nefunkcionalnih zahtjeva. Kada čujemo atribut nefunkcionalnih zahtjeva prenosivost misli se na određivanje kako se sustav ili njegov element mogu pokrenuti unutar jednog ili drugog okruženja. Ovdje spadaju specifikacije hardvera, softvera ili drugih korisničkih platformi. Drugim riječima, utvrđuje se koliko se radnje izvedene putem jedne platforme izvode na drugoj platformi. Drugi atribut, odnosno kompatibilnost definira kako se sustav može nositi s drugim sustavom u istom okruženju. Pod ovim nefunkcionalnim zahtjevom se podrazumijeva da softver instaliran na određenom operacijskom sustavu mora moći koegzistirati i biti kompatibilan s drugim programima kao što su na primjer antivirusni sustavi ili vatrozid. [11]

4.3.3.3. Pouzdanost, mogućnost održavanja i dostupnost

Ova tri atributa stavljena su pod isto poglavlje budući da pristupaju istom problemu, ali iz različitih pogleda. Navedeni zahtjevi su jako teški za izraziti u smislu proračuna, stoga ih mnogi pružatelji sustava rijetko kad i dokumentiraju. Atribut pouzdanosti određuje kolika je vjerojatnost da će sustav ili njegov element raditi bez kvara u određenom vremenskom razdoblju pod unaprijed definiranim uvjetima. Ovaj nefunkcionalni zahtjev se u većini slučajeva izražava u postotcima na određeno vrijeme, kao na primjer da sustav mora raditi u 90% slučajeva u razdoblju od jednog mjeseca. Mogućnost održavanja je sljedeći atribut te je veoma bitan i povezan s prijašnjim atributom. Ovaj nefunkcionalni zahtjev definira vrijeme potrebno da se rješenje ili njegova komponenta popravi od strane programera u svrhu povećanja izvedbe sustava. Kao i pouzdanost, ovaj se atribut izražava u postotcima što znači da se može navesti kao postotak koji predstavlja šansu da se kvar otkloni u određenom razdoblju, na primjer u jednome danu. Zadnji atribut je dostupnost sustava koji opisuje koliko je vjerojatno da će sustav biti dostupan korisniku u određenom trenutku. Ovaj nefunkcionalni zahtjev se može odrediti kao očekivani postotak uspješnih zahtjeva, a isto tako kao i postotak u prethodnim zahtjevima, a to je postotak vremena u kojem je sustav dostupan za rad tijekom nekog vremenskog razdoblja. Kako bi se ovaj nefunkcionalni zahtjev definirao, potrebno je imati procjene za pouzdanost i mogućnost održavanja. [11]

4.3.3.4. Sigurnost

Sigurnost je jedan od najbitnijih nefunkcionalnih zahtjeva koji osigurava da će svi podaci unutar sustava biti zaštićeni od zlonamjernih napada ili neovlaštenog pristupa. Kako bi se unutar aplikacije zaštitilo od neovlaštenog pristupa, ova stvar se rješava uvođenjem tijeka prijave i različitih korisničkih uloga koji definiraju ovlasti i radnje korisnika. [11]

4.3.3.5. Lokalizacija

Atribut lokalizacije podrazumijeva koliko je sustav u skladu s kontekstom budućeg lokalnog tržišta što uključuje lokalne jezike, kulturu, pravopis, valute i druge aspekte. Bez ovog zahtjeva sustav ima manju šansu za uspjeh kod određene ciljane publike. [11]

4.3.3.6. Upotrebljivost

Zadnji nefunkcionalni zahtjev obrađen unutar ovog rada je upotrebljivost. Ovaj atribut ukazuje na to koliko je teško koristiti se sustavom. Postoje mnoge vrste kriterija upotrebljivosti, a jedna vrsta je ta koja pokriva pet dimenzija: mogućnost učenja, učinkovitost, memorabilnost, greške i zadovoljstvo. Ove dimenzije odgovaraju na pitanja koliko brzo korisnik može postići svoj cilj, koliko često korisnici griješe, je li dizajn ugodan za korištenje i slično. Navedeni zahtjev se teško može odrediti unaprijed te zahtjeva korisničko testiranje sustava. [11]

4.3.3.7. Nefunkcionalni zahtjevi aplikacije

Praktični dio , odnosno aplikacija, je napravljena u svrhu diplomskog rada te samim time ne zahtijeva da se navode nefunkcionalni zahtjevi, ali kada bi aplikacija išla u produkciju nefunkcionalni zahtjevi su obavezna komponenta dokumentacije te će se svakako navesti u ovom poglavlju. Nefunkcionalni zahtjevi su u sljedećoj tablici podijeljeni na smislenim cjelinama kako je to navedeno i objašnjeno u prijašnjim poglavljima.

Tablica 5: Nefunkcionalni zahtjevi

Atributi	Nefunkcionalni zahtjev
Performanse i skalabilnost	Aplikacija ako podržava 1000 korisnika po satu mora osigurati vrijeme odaziva od 5 sekundi ili manje ukoliko se koristi valjanja internetska konekcija.
Prenosivost i kompatibilnost	Aplikacija mora raditi na sustavu Windows obavezno uključujući Windows 10 i Windows 11 bez značajnih promjena u performansama.
Pouzdanost, mogućnost održavanja i dostupnost	Aplikacija, odnosno sustav, mora raditi bez greške u 90% slučajeva u razdoblju od jednog mjeseca. Održavatelji sustava moraju imati 50% i veću šansu prilikom održavanja tijekom jednoga dana. Nakon kritičnog kvara srednje vrijeme vraćanja sustava ne smije biti veće od jednoga dana. Mogućnost rezervacije automobila mora biti dostupna korisnicima u 99% vremena svakog mjeseca te radnicima i direktoru u 99% slučajeva tijekom radnog vremena.
Sigurnost	Sustav nudi garanciju da će korisnici biti zaštićeni od zlonamjernih napada na način da baza podataka nije dostupna vanjskom izvoru. Sustav nudi garanciju da drugi korisnici neće biti u mogućnosti izvršavati neovlaštene radnje tako što postoje ograničenja ovlasti po ulogama korisnika.
Lokalizacija	Aplikacija mora biti na engleskom jeziku. Format datuma mora biti po europskim standardima. Korištena valuta mora biti po europskim standardima.
Upotrebljivost	Aplikacija mora biti napravljena tako da član može jednostavno, brzo, lako i efikasno odabrati što je tražio kako bi se povećalo zadovoljstvo korisnika. Aplikacija treba jasno ukazivati na greške koje rade korisnici te ako ih ima jasno ih obavijestiti u čemu griješe. Mogućnost učenja unutar aplikacije treba biti omogućena za radnike na način da se stvari koje se izvršavaju repetitivno izvršavaju bez grešaka te dodatnih uloga rada i napora radnika. Stopa pogrešaka u aplikaciji ne smije premašivati 10%.

5. „MVVM“ arhitektura sustava

U narednom poglavlju detaljno je opisana korištena arhitektura za izradu aplikacije. Naime, kao što sam naslov poglavlja kaže, radi se o „MVVM“ arhitekturi (eng. *Model-View-ViewModel*) čija skraćenica dolazi od početnih slova punog imena gdje slovo M predstavlja „Model“, V predstavlja „View“ (eng. View), a VM predstavlja „ViewModel“. U nastavku je za početak u potpunosti objašnjena arhitektura dok su u poglavlju nakon objašnjenje i svake od njenih komponenata.

5.1. Uvod i objašnjenje „MVVM“ arhitektura

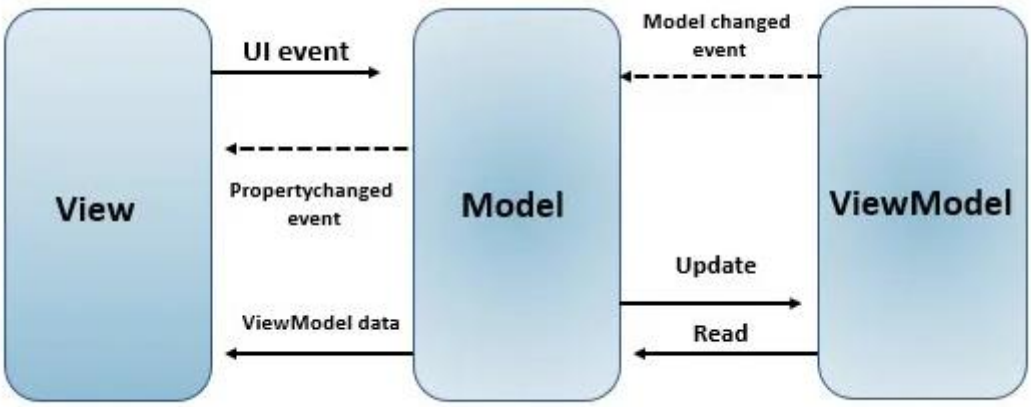
Kroz ovu arhitekturu gradi se uređena struktura sustava koja koristi tehniku recikliranja za organiziranje novoizgrađenog koda. Koristeći ovu arhitekturu stvara se organiziran kod i kompaktan kod koji se može održavati i testirati u svim proširivim aplikacijama, što je slučaj i s aplikacijom izrađenom u praktičnom dijelu rada. „Model“ arhitekture koristi se za čuvanje informacija koje se sastoje od poslovne taktike i logike, a formatirane podatke čuva „View“ te ih prezentira korisniku. „ViewModel“ je poveznica između prijašnjih komponenti te karakterizira uspostavljenu vezu između „Model-a“ i „View-a“ kako bi kod bio što bolji i kompaktniji. „MVVM“ je već utvrđeni arhitekturalni dizajn koji služi za razvoj softverskih aplikacija i uređaja te ja jako popularan u razvoju aplikacija unutar „Microsoft Visual Studija“. Najveći cilj ove arhitekture je odvojiti kod koji se koristi za implementaciju grafičkog korisničkog sučelja od poslovne logike. Kako se iz ranijih poglavlja već moglo naslutiti prilikom spominjanja jezika oznaka, izrada korisničkog sučelja unutar ove arhitekture radi se označnim jezikom ili „GUI“ kodom. Po mom mišljenju, najznačajniju ulogu ima „ViewModel“ komponenta koja se koristi za pretvaranje vrijednosti koje daju utjecajno značenje za istraživanje podatkovnih objekata iz modela jednostavnim kontroliranjem i predstavljanjem ulaznih objekata. Ova komponenta upravlja najvećim dijelom logike te je najnaprednija od svih komponenti. [12]

5.2. Prednosti korištenja „MVVM“ arhitekture

Korištenje ove arhitekture programerima omogućuje razne prednosti. Najbitnija prednost je segregacija između pogleda i modela te fleksibilnost i prilagodljivost promjenama bez potrebe za pregledom. Sve komponente prisutne unutar „MVVM“ arhitekture mogu funkcionirati neovisno jedne o drugima što uvelike omogućava mogućnost testiranja. Još jedna od prednosti je da održivost „MVVM-a“ pomaže programerima držati se agilnog razvoja te nastaviti se kretati brzim tempom, odnosno brzim izdavanjem isporuka. [12]

5.3. Komponente „MVVM“ arhitekture

U ovom poglavlju dan je slikovni prikaz arhitekture u kojoj su detaljnije objašnjenja svaka od njezinih komponenti te njihova međusobna povezanost. Sljedeća slika prikazuje sve tri komponente i strelice koje ukazuju na komunikaciju između komponenti. Prva komponenta koja predstavlja „Model“ naziva se domenom ili objektno orijentiranom tehnikom koja se koristi za označavanje sadržaja u stvarnom vremenu ili za sloj pristupa podacima. Druga komponenta predstavlja pogled (eng. *View*) te predstavlja sadržaj koji krajnji korisnik gleda na svom ekranu. Proširuje prikaz modela koji prima interakciju korisnika, a događaji se prosljeđuju na model putem atributa vezanja podataka kao što su povratni pozivi događaja (eng. *Event callbacks*). Treća komponenta je „ViewModel“ komponenta koja pretvara apstrakciju u prikaz. Opcije kontrolera „MVC“ uzorka i opcija prezentera „MVP“ uzorka zamijenjena je u „MVVM“ s opcijom vezanja koja olakšava interakciju između ograničenih atributa „ViewModel-a“ i „View-a“. [12]



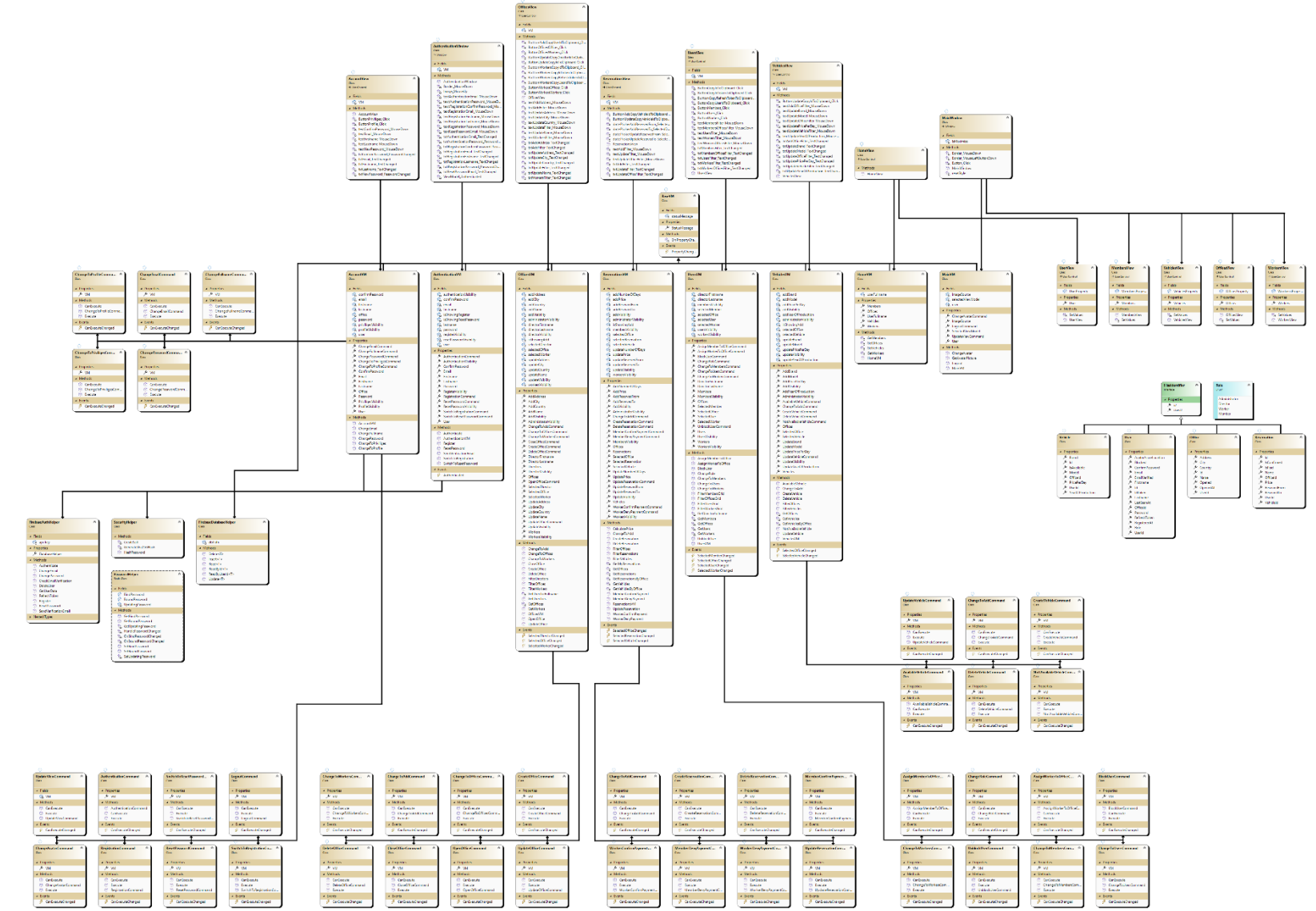
Slika 1 Komponente "MVVM" arhitekture sustava

Unutar aplikacije Model predstavlja klase čiji atributi odgovaraju onima unutar baze podataka. Svaka od cjelina navedenih unutar funkcionalnosti aplikacija ima zaseban pogled unutar kojeg se elementi prikazuju koristeći jezik oznaka „XAML“ te svaki pogled ima korespondirajući „ViewModel“ koji se veže uz njega te se za svaki element veže (eng. *Binding*) određeni atribut iz „ViewModel-a“. Prilikom promjene modela „ViewModel“ obavještava „Model“ te se podaci mijenjaju, a prilikom promjene atributa objekta iz modela, pomoću „PropertyChangedEvent-a“ se može okinuti neka radnja koja će utjecati na pogled. S događajima unutar korisničkog sučelja možemo pokrenuti određene akcije i tako utjecati na „Model“, a za izvršavanje akcija preko „ViewModel-a“ koriste se klase koje implementiraju „ICommand“ sučelje te okidaju određene radnje iz „ViewModel-a“ na ciljanje korisnikove akcije.

6. Dijagram klasa

U sljedećem poglavlju opisan je i prikazan dijagram klasa što u programskom inženjerstvu u univerzalnom jeziku modeliranja (eng. *Unified Modeling Language*, skraćeno UML) predstavlja dijagram statičke strukture koji opisuje strukturu sustava prikazujući klase njihove atribute, metode i međusobne odnose. [13]

Na idućoj slici možemo vidjeti dijagram klasa koji odgovara klasama aplikacije koja je odrađena u praktičnom dijelu rada. U gornjem dijelu se nalaze klase „AccountView“, „AuthenticationWindow“, „OfficeView“, „ReservationView“, „UsersView“ , „VehiclesView“, „HomeView“ te „MainWindow“ koji su zaslužni za prezentaciju podataka i unutar aplikacije predstavljaju sve moguće poglede. Svaki od navedenih pogleda ima odgovarajući „ViewModel“ s istim imenom „prefix-a“ za poglede. Kao što na dijagramu možemo vidjeti svaki pogled ima vezu asocijacije na korespondirajući „ViewModel“. „ViewModel-i“ su jedne od glavnih klasa aplikacije te predstavljaju jezgru, a to su „AccountVM“, „AuthenticationVM“, OfficesVM“, „ReservationVM“, „UsersVM“, „VehiclesVM“, „HomeVM“ te „MainVM“. Svaki od „ViewModel-a“ nasljeđuje baznu klasu koja se naziva „BaseVM“. S desne strane dijagrama možemo vidjeti pet klasa koje predstavljaju korisničke kontrole, a služe nam za prezentaciju grupa podataka unutar aplikacije kao što su korisnički podaci unutar „MainWindow-a“ te statistički podaci na „HomeView“. S lijeve strane dijagrama nalazi se pomoćne klase koje uglavnom služe za rad s bazom podataka. Klasa „FirebaseAuthHelper“ sadrži sve metode koje su potrebne za autentikaciju te izmjenu korisničkih podataka dok „FirebaseDatabaseHelper“ uglavnom pomaže pri unosu, čitanju, ažuriranju i brisanju podataka (eng. *Create, Read, Update, Delete*, skraćeno CRUD). Ostale dvije klase „SecurityHelper“ i „PasswordHelper“ sadrže metode za heširanje lozinke te pomagalo prilikom povezivanja lozinke s korisnikom. Na dnu dijagrama nalaze se „Command“ klase koje predstavljaju naredbu koja će se izvršiti prilikom neke korisnikove akcije. Svaki naziv klase opisuje korisnikovu akciju, a svaka klasa implementira sučelje „ICommand“ te sadrži metode koje govore kad se akcija može izvršiti te što će se izvršiti prilikom ostvarivanja iste. Na desnoj strani dijagrama vidimo model klase, a to su „User“ koja predstavlja korisnika, „Office“ koja predstavlja poslovnicu, „Vehicle“ koja predstavlja automobil te „Reservation“ koja predstavlja korisničke rezervacije. Svaka od ovih klasa implementira sučelje „IHasIdentifier“ te na taj način osigurava da klasa mora imati svoj identifikator i korisnički identifikator što se unutar „Firebase-a“ razlikuje u tome što svaki korisnik ima alfanumerički identifikator s kojim je zabilježen unutar baze podataka te drugi identifikator „UserId“ koji korisnik dobije prilikom registracije u sustav. „UserId“ u klasama predstavlja koja je osoba odgovorna za određenu radnju, npr. koji korisnik je kreirao automobil, koji korisnik je direktor poslovnice te kojem korisniku pripada rezervacija.



Slika 2 Dijagram klasa

7. ERA dijagram

Sljedeće poglavlje opisuje dijagram odnosa entiteta (eng. *Entity relation diagram*, skraćeno ER Diagram) što je vrsta dijagrama toka koji ilustrira kako se entiteti kao što su korisnici, vozila, poslovnice i rezervacije međusobno povezuju unutar sustava. Ova vrsta dijagrama se najčešće koristi za dizajn ili otklanjanje pogrešaka u relacijskim bazama podataka u područjima softverskog inženjerstva, poslovnih informacijskih sustava, istraživanja i slično. Baza podataka je izgrađena unutar platforme „Firebase“ te je „NoSQL“ baza podataka, no i dalje se može napraviti dijagram odnosa entiteta koji prikazuje njihovu povezanost. [14]

Na sljedećim tablicama nalaze se atributi entiteta koji su također prikazani unutar ERA dijagrama. Svaki atribut unutar tablice sadrži kolonu koja ukazuje na tip podatka i kolonu koja opisuje atribut.

Tablica 6: Entitet korisnik (eng. *User*)

Atribut	Tip podatka	Opis
Id	string	Vrijednost koja označava alfanumerički identifikator korisnika
Email	string	Vrijednost koja označava adresu elektroničke pošte korisnika
EmailVerified	bool	Vrijednost da/ne koja označava verifikaciju adrese korisničke pošte
Firstname	string	Vrijednost koja označava ime korisnika
Lastname	string	Vrijednost koja označava prezime korisnika
Password	string	Vrijednost koja označava heširanu lozinku korisnika
AzureStorageLocation	string	Vrijednost koja označava putanju do korisničke slike koja se čuva na „Azure storage-u“
Blocked	bool	Vrijednost da/ne koja označava blokiranost korisnika
IdToken	string	Vrijednost koja označava identifikacijski token korisnika koji je potreban za dohvaćanje podataka sa „Firebase Auth REST API-ja“
LastSeenAt	datetime	Vrijednost koja označava vrijeme zadnjeg korištenja aplikacije za određenog korisnika
RefreshToken	string	Vrijednost koja označava token dodijeljen korisniku prilikom registracije koji služi za osvježavanje identifikacijskog tokena
RegisteredAt	datetime	Vrijednost koja označava vrijeme registracije korisnika
Role	string	Vrijednost koja označava ulogu korisnika

Tablica 7: Entitet poslovnica (eng. *Office*)

Atribut	Tip podatka	Opis
Id	string	Vrijednost koja označava alfanumerički identifikator poslovnice
UserId	string	Vrijednost koja označava vanjski ključ na tablicu korisnika
Country	string	Vrijednost koja označava državu u kojoj poslovnica posluje
City	string	Vrijednost koja označava grad u kojemu poslovnica posluje
Name	string	Vrijednost koja označava ima poslovnice
Address	string	Vrijednost koja označava adresu u kojoj poslovnica posluje
Opened	bool	Vrijednost da/ne koja označava otvorenost poslovnice
OpenedAt	datetime	Vrijednost koja označava vrijeme otvaranja poslovnice

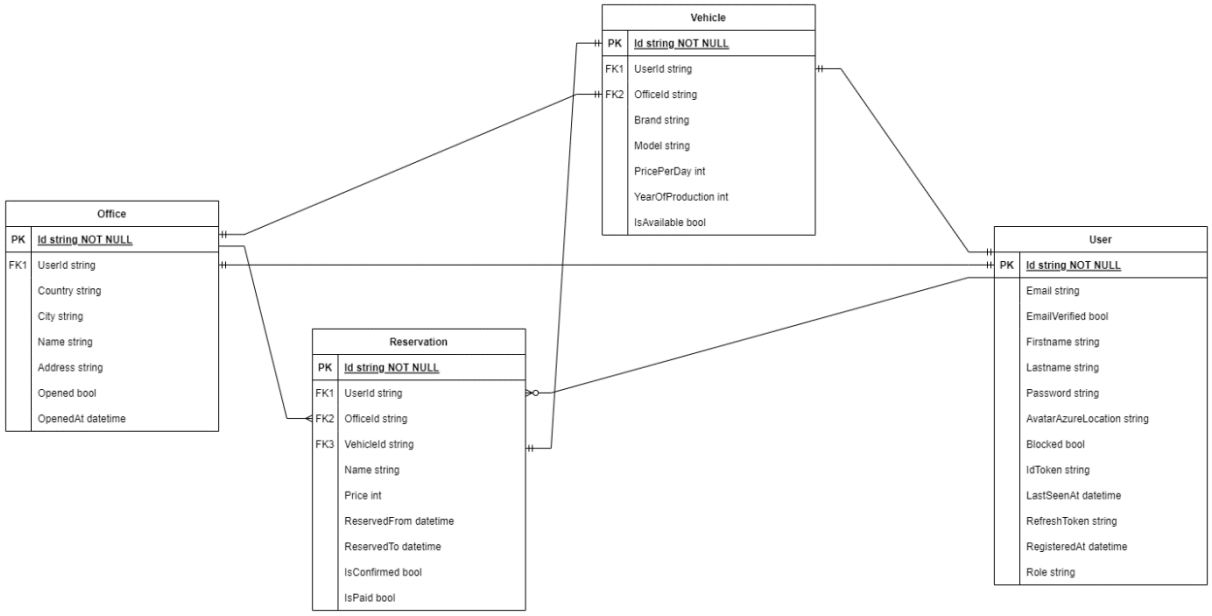
Tablica 8: Entitet rezervacije (eng. *Reservation*)

Atribut	Tip podatka	Opis
Id	string	Vrijednost koja označava alfanumerički identifikator rezervacije
UserId	string	Vrijednost koja označava vanjski ključ na tablicu korisnika
OfficeId	string	Vrijednost koja označava vanjski ključ na tablicu poslovnica
VehicleId	string	Vrijednost koja označava vanjski ključ na tablicu vozila
Name	string	Vrijednost koja označava ima rezervacije (u pravilu se ime kreira kombinacijom brenda i modela vozila)
Price	int	Vrijednost koja označava iznos rezervacije koji je potrebno podmiriti. Jedinica mjerenja za ovaj atribut su euri (€)
ReservedFrom	datetime	Vrijednost koja označava vrijeme od kada je korisnik rezervirao automobil
ReservedTo	datetime	Vrijednost koja označava vrijeme do kada je korisnik rezervirao automobil
IsConfirmed	bool	Vrijednost da/ne koja označava potvrdu plaćanja od strane djelatnika poslovnice. Vrijednost potvrđuje djelatnik nakon provjere
IsPaid	bool	Vrijednost da/ne koja označava potvrdu plaćanja od strane člana poslovnice. Vrijednost potvrđuje član nakon plaćanja

Tablica 9: Entitet vozila (eng. *Vehicle*)

Atribut	Tip podatka	Opis
Id	string	Vrijednost koja označava alfanumerički identifikator vozila
UserId	string	Vrijednost koja označava vanjski ključ na tablicu korisnika
OfficeId	string	Vrijednost koja označava vanjski ključ na tablicu poslovnica
Brand	string	Vrijednost koja označava brend automobila
Model	string	Vrijednost koja označava model automobila
PricePerDay	int	Vrijednost koja označava cijenu najma automobila po danu. Jedinica mjerenja za ovaj atribut su euri (€)
YearOfProduction	int	Vrijednost koja označava godinu proizvodnje automobila
IsAvailable	bool	Vrijednost koja označava dostupnost automobila

Na sljedećoj slici nalazi se ERA dijagram koji odgovara aplikaciji napravljenoj unutar praktičnog dijela rada. Na dijagramu možemo vidjeti četiri tablice koje odgovaraju vozilima, poslovnicama, rezervacijama i korisnicima. Također se mogu vidjeti međusobne veze između tablica te tipovi podataka i atributi koji su objašnjeni u prijašnjim tablicama.



Slika 3 ERA Dijagram

8. Opis izrade aplikacije

Sljedeće poglavlje opisuje praktični dio rada, a to je izrada programskog rješenja. Programsko rješenja predstavlja aplikaciju izrađenu u tehnologijama navedenima na početku rada naziva „Rent a Car“. U nastavku će kroz poglavlja biti prikazano rješenje uz sliku i kod te opis kako je ta cjelina nastala. Pošto aplikacija sadrži velik broj klasa i koda kroz cjeline su opisani samo ključni dijelovi koji su bitni za rad aplikacije, dok su sporedni samo spomenuti. Prvo je opisana model komponenta zajedno sa pomoćnim klasama jer su zapravo dosta povezane, a zatim se po cjelinama opisuju različiti pogledi uz svoje „ViewModel-e“.

8.1. Opis komponente „Model“

U sljedećem dijelu opisana je komponenta modela uz priložen kod za svaku od model klasa. Idući dijelovi koda prikazuju sučelje koje svaka od model klasa implementira, a to sučelje određuje da svaka klasa koja ga implementira mora imati identifikator i korisnički identifikator.

```
public interface IhasIdentifier
{
    string Id { get; set; }
    string UserId { get; set; }
}
```

Model klase koje implementiraju ovo sučelje su redom korisnik (unutar aplikacije, eng. *User*), poslovnica (unutar aplikacije, eng. *Office*), rezervacija (unutar aplikacije, eng. *Reservation*) te vozilo (unutar aplikacije, eng. *Vehicle*). Osim model klasa postoji i jedna Enum klasa koja sadrži korisničke uloge. U sljedećem dijelu prikazan je kod za pojedinu od navedenih klasa. Za početak je navedena model klasa za objekt korisnika.

```
public class User : IHasIdentifier
{
    public string Id { get; set; }
    public string UserId { get; set; }
    public string IdToken { get; set; }
    public string RefreshToken { get; set; }
    public string OfficeId { get; set; }
    public string Firstname { get; set; }
    public string Lastname { get; set; }
    public string Email { get; set; }
    public bool EmailVerified { get; set; }
    public bool Blocked { get; set; }
    public string Password { get; set; }
    public string ConfirmPassword { get; set; }
    public string Role { get; set; }
    public string AvatarAzureLocation { get; set; }
    public DateTime RegisteredAt { get; set; }
    public DateTime LastSeenAt { get; set; }
}
```

U sljedećem dijelu koda prikazan je dio koji se odnosi na model klasu za objekt poslovnice.

```
public class Office : IHasIdentifier
{
    public string Id { get; set; }
    public string Name { get; set; }
    public string UserId { get; set; }
    public string Country { get; set; }
    public string City { get; set; }
    public string Address { get; set; }
    public DateTime OpenedAt { get; set; }
    public bool Opened { get; set; }
}
```

U sljedećem dijelu koda prikazan je dio koji se odnosi na model klasu za objekt rezervacije.

```
public class Reservation : IHasIdentifier
{
    public string Id { get; set; }
    public string UserId { get; set; }
    public string VehicleId { get; set; }
    public string OfficeId { get; set; }
    public string Name { get; set; }
    public DateTime ReservedFrom { get; set; }
    public DateTime ReservedTo { get; set; }
    public int Price { get; set; }
    public bool IsPaid { get; set; }
    public bool IsConfirmed { get; set; }
}
```

U sljedećem dijelu koda prikazan je dio koji se odnosi na model klasu za objekt automobila.

```
public class Vehicle : IHasIdentifier
{
    public string Id { get; set; }
    public string UserId { get; set; }
    public string OfficeId { get; set; }
    public string Brand { get; set; }
    public string Model { get; set; }
    public string YearOfProduction { get; set; }
    public int PricePerDay { get; set; }
    public bool IsAvailable { get; set; }
}
```

U sljedećem dijelu koda prikazan je dio koji se odnosi na enum klasu koja sadrži sve uloge unutar aplikacije.

```
public enum Role
{
    Administrator,
    Director,
    Worker,
    Member
}
```

8.2. Pomoćne klase

Sljedeće klase opisuju pomoćne klase koje su bitna komponenta aplikacije. Među pomoćnim klasama nalaze se klase koje služe za komunikaciju s bazom podataka, odnosno za kreiranje, čitanje, ažuriranje i brisanje zapisa te klasa koja šalje zahtjeve „Firebase Auth REST API-ju“. Nešto manje bitne pomoćne klase služe za heširanje lozinke te za pomoć prilikom povezivanja (eng. *binding*) lozinke sa svojstvom (eng. *propertie*). Sljedeći kod prikazuje „FirebaseDatabaseHelper“ klasu koja unutar aplikacije služi za povezivanje s bazom. Kako metode za „CRUD“ operacije primaju parametar generičkog tipa ove metode se mogu koristiti za bilo koji tip podataka. Osim navedenih metoda unutar klase postoje i metode za dohvat po identifikatoru.

```
public class FirebaseDatabaseHelper
{
    private static string dbPath = "https://thesisapp-740ad-default-
rtdb.europe-west1.firebasedatabase.app/";

    public static async Task<bool> Insert<T>(T item)
    {
        var jsonBody = JsonConvert.SerializeObject(item);
        var content = new StringContent(jsonBody, Encoding.UTF8,
"application/json");

        using (var client = new HttpClient())
        {
            var result = await
client.PostAsync($"{dbPath}{item.GetType().Name.ToLower()}.json", content);

            if (result.IsSuccessStatusCode)
            {
                return true;
            }
            else
            {
                return false;
            }
        }
    }

    public static async Task<List<T>> Read<T>() where T : IHasIdentifier
    {
        using (var client = new HttpClient())
        {
            var result = await
client.GetAsync($"{dbPath}{typeof(T).Name.ToLower()}.json");
            var jsonResult = await result.Content.ReadAsStringAsync();

            if (result.IsSuccessStatusCode)
            {
                var objects =
JsonConvert.DeserializeObject<Dictionary<string, T>>(jsonResult);

                List<T> list = new List<T>();
                if (objects != null)
```

```
                {
                    foreach (var o in objects)
                    {
                        o.Value.Id = o.Key;
                        list.Add(o.Value);
                    }
                }

                return list;
            }
            else
            {
                return null;
            }
        }
    }

    public static async Task<bool> Update<T>(T item) where T :
IHasIdentifier
    {
        var jsonBody = JsonConvert.SerializeObject(item);
        var content = new StringContent(jsonBody, Encoding.UTF8,
"application/json");

        using (var client = new HttpClient())
        {
            var result = await
client.PutAsync($"{dbPath}{item.GetType().Name.ToLower()}/{item.Id}.json",
content);

            if (result.IsSuccessStatusCode)
            {
                return true;
            }
            else
            {
                return false;
            }
        }
    }

    public static async Task<bool> Delete<T>(T item) where T :
IHasIdentifier
    {
        using (var client = new HttpClient())
        {
            var result = await
client.DeleteAsync($"{dbPath}{item.GetType().Name.ToLower()}/{item.Id}.json
");

            if (result.IsSuccessStatusCode)
            {
                return true;
            }
            else
            {
                return false;
            }
        }
    }
}
```

Prethodni kod koristi gotove metode za komunikaciju sa „Firebase-om“ gdje je za svaku od operacija potrebna samo „JSON“ serijalizacija te slanje asinkronog zahtjeva na putanju baze podataka. U nastavku se nalazi kod klase „FirebaseAuthHelper“ koji služi za slanje zahtjeva na „Firebase Auth REST API“. Kako ova klasa sadrži velik broj metoda u radu su navedeni samo dijelovi, dok se sve metode mogu iščitati iz dijagrama klasa. Kao dvije najbitnije metode iz ove klase odabrane su metode za registraciju i autentikaciju. Na kraju koda nalazi se klasa „FirebaseResult“ koja je potrebna kako bi se primio odgovor na poslani zahtjev. Ova klasa sadrži atribute koji odgovaraju onima vraćenim od strane „Firebase Auth REST API-ja“.

```
public class FirebaseAuthHelper
{
    private static string api_key =
"AIzaSyCUuTUZLSJsbXsO86uxUmTFh8YTvhXWMUc";

    public static object DatabaseHelper { get; private set; }

    public static async Task<string> Authenticate(User user)
    {
        using (HttpClient client = new HttpClient())
        {
            var body = new
            {
                email = user.Email,
                password = user.Password,
                returnSecureToken = true
            };

            string bodyJson = JsonConvert.SerializeObject(body);
            var data = new StringContent(bodyJson, Encoding.UTF8,
"application/json");

            var response = await
client.PostAsync($"https://identitytoolkit.googleapis.com/v1/accounts:signI
nWithPassword?key={api_key}", data);

            if (response.IsSuccessStatusCode)
            {
                string resultJson = await
response.Content.ReadAsStringAsync();
                var result =
JsonConvert.DeserializeObject<FirebaseResult>(resultJson);
                List<User> Users = await
FirebaseDatabaseHelper.ReadByUserId<User>(result.localId);
                App.User = Users.FirstOrDefault();
                App.User.RefreshToken = result.refreshToken;
                await RefreshToken(App.User);
                return await CheckEmailVerification();
                return "";
            }
            else
            {
                string errorJson = await
response.Content.ReadAsStringAsync();
                var error =
JsonConvert.DeserializeObject<Error>(errorJson);
```

```
                return FirebaseErrorMessage.GetByKey(error.error.message);
            }
        }
    }

    public static async Task<string> Register(User user)
    {
        using (HttpClient client = new HttpClient())
        {
            var body = new
            {
                email = user.Email,
                password = user.Password,
                returnSecureToken = true
            };

            string bodyJson = JsonConvert.SerializeObject(body);
            var data = new StringContent(bodyJson, Encoding.UTF8,
"application/json");

            var response = await
client.PostAsync($"https://identitytoolkit.googleapis.com/v1/accounts:signU
p?key={api_key}", data);

            if (response.IsSuccessStatusCode)
            {
                string resultJson = await
response.Content.ReadAsStringAsync();
                var result =
JsonConvert.DeserializeObject<FirebaseResult>(resultJson);
                App.User.UserId = result.localId;
                App.User.IdToken = result.idToken;
                App.User.RefreshToken = result.refreshToken;

                return "";
            }
            else
            {
                string errorJson = await
response.Content.ReadAsStringAsync();
                var error =
JsonConvert.DeserializeObject<Error>(errorJson);

                return FirebaseErrorMessage.GetByKey(error.error.message);
            }
        }
    }

    public class FirebaseResult
    {
        public string kind { get; set; }
        public string idToken { get; set; }
        public string email { get; set; }
        public string refreshToken { get; set; }
        public string expiresIn { get; set; }
        public string localId { get; set; }
        public bool registered { get; set; }
    }
}
```

Od pomoćnih klasa još je spomenuta „SecurityHelper“ klasa koja uglavnom služi za heširanje lozinke kako bi se ista mogla spremiti unutar baze podataka. Ova klasa sadrži tri metode od kojih jedna služi za kreiranje soli (eng. S*alt*) gdje se na postojeću lozinku dodaje odrađeni broj znakova te se novokreirana lozinka sa soli kriptira „Sha256“ algoritmom. Metoda koja povezuje sve ove stvari je metoda „HashPassword“ koja za primljenog korisnika s normalnom lozinkom, vraća objekt korisnika sa heširanom lozinkom.

```
public class SecurityHelper
{
    public static User HashPassword(User user)
    {
        string salt = CreateSalt(5);
        string hashedPassword = GenerateSha256Hash(user.Password, salt);
        hashedPassword = hashedPassword.Replace("-",
string.Empty).Substring(0, 16);
        user.Password = hashedPassword;
        user.ConfirmPassword = hashedPassword;
        return user;
    }

    private static String CreateSalt(int size)
    {
        var cryptoRNG = new RNGCryptoServiceProvider();
        var buffer = new byte[size];
        cryptoRNG.GetBytes(buffer);
        return Convert.ToBase64String(buffer);
    }

    private static String GenerateSha256Hash(string password, string salt)
    {
        byte[] bytes = Encoding.UTF8.GetBytes(password + salt);
        SHA256Managed shaString = new SHA256Managed();
        byte[] hash = shaString.ComputeHash(bytes);

        return BitConverter.ToString(hash);
    }
}
```

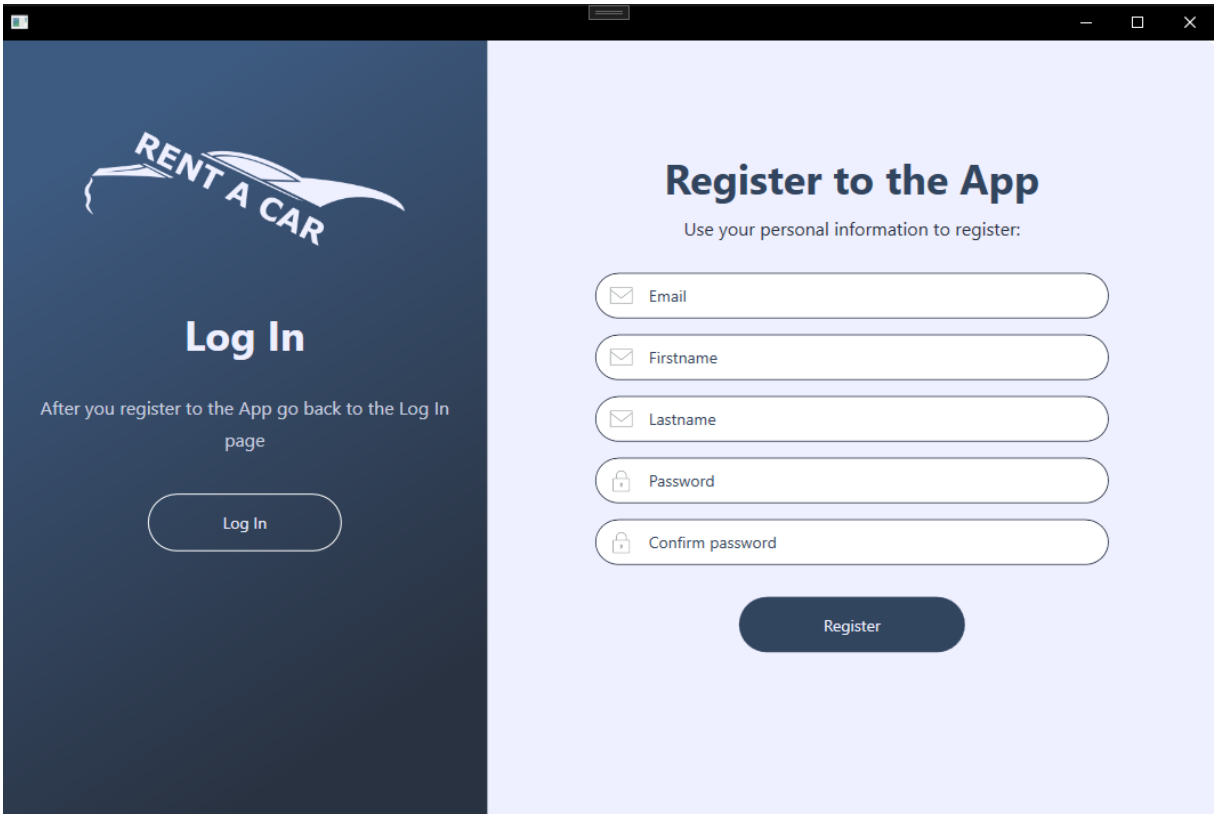
Od pomoćnih klasa još postoji „PasswordHelper“ klasa koja nije navedena u diplomskome radu, a napravljena je iz razloga što se tekst korišten za upis lozinke (zvjezdice) ne može povezati (eng. *binding*) sa svojstvom „ViewModel-a“ te se kao rješenje koristi gotova navedena klasa koja je dostupna na internetu. Pomoću ove klase omogućuje se rad s elementom „PasswordBox“ unutar „XAML-a“ na isti način kao da je to „TextBox“. Ova klasa koristi koncept prilaganja svojstava (eng. *Attach properties*) koji omogućuju vrlo jednostavno proširenje kontrola. Klasa „PasswordHelper“ proširuje „PasswordBox“ kontrolu s priloženim svojstvom koje omogućuje podatkovno vezanje lozinke. Kod korišten za ovaj dio može se naći unutar literature priložene uz ovaj odlomak. Na ovaj način ipak je omogućeno ne odstupanje od „MVVM“ koncepta te se uz navedenu klasu držati „data bindinga“. Iako ovaj način radi, nije se pokazao kao najsigurnija opcija stoga se u kombinaciji s ovim koristi i „SecurityHelper“ klasa. [15]

8.3. Opis komponenti „View“ i „ViewModel“

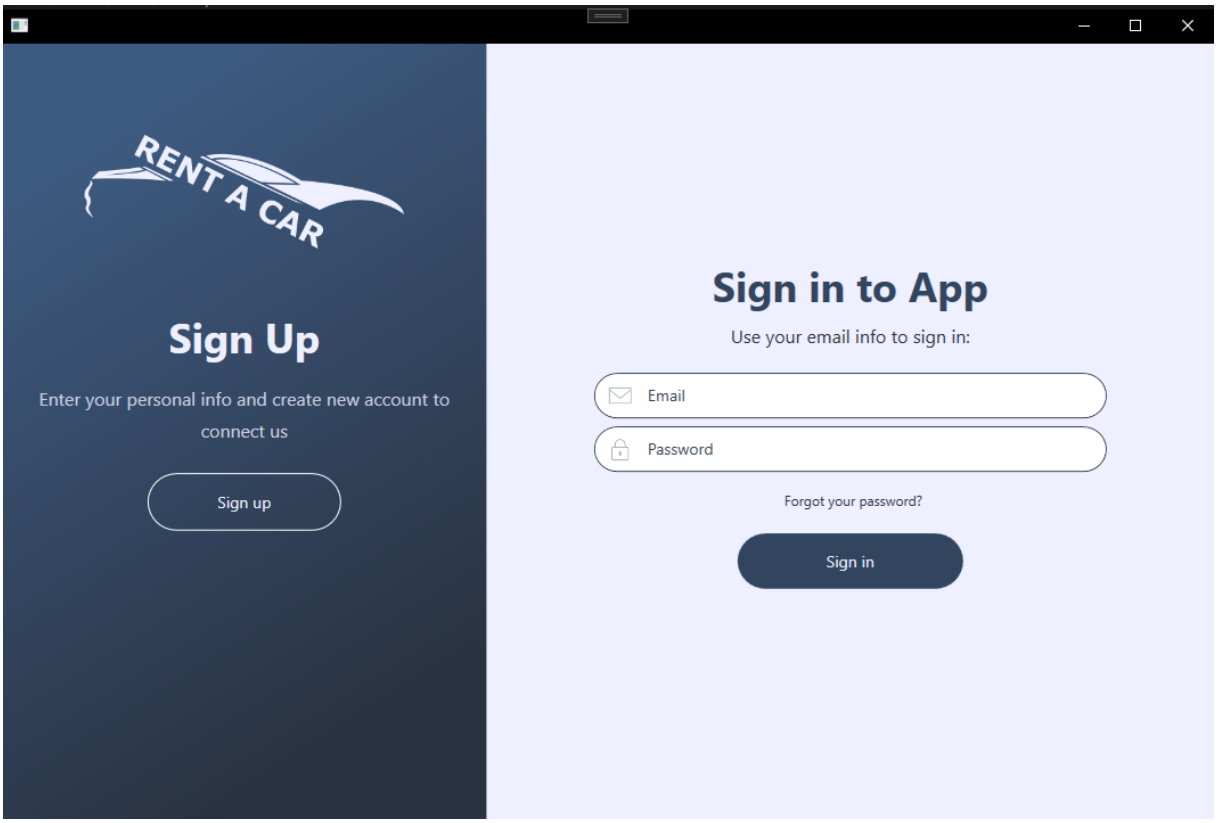
U sljedećem poglavlju su kroz cjeline opisani pojedini dijelovi aplikacije. Svaki dio sadrži opis, sliku i najključniji kod bitan za realizaciju pogleda. Kod svake cjeline bila je potrebna izrada korisničkog sučelja kroz jezik oznaka „XAML“ te svaki od pogleda sadrži korespondirajući „ViewModel“ za poslovnu logiku. Pogledi su prikazivani redoslijedom kako se u aplikaciji i koriste.

8.3.1. Pogled registracije i autentikacije

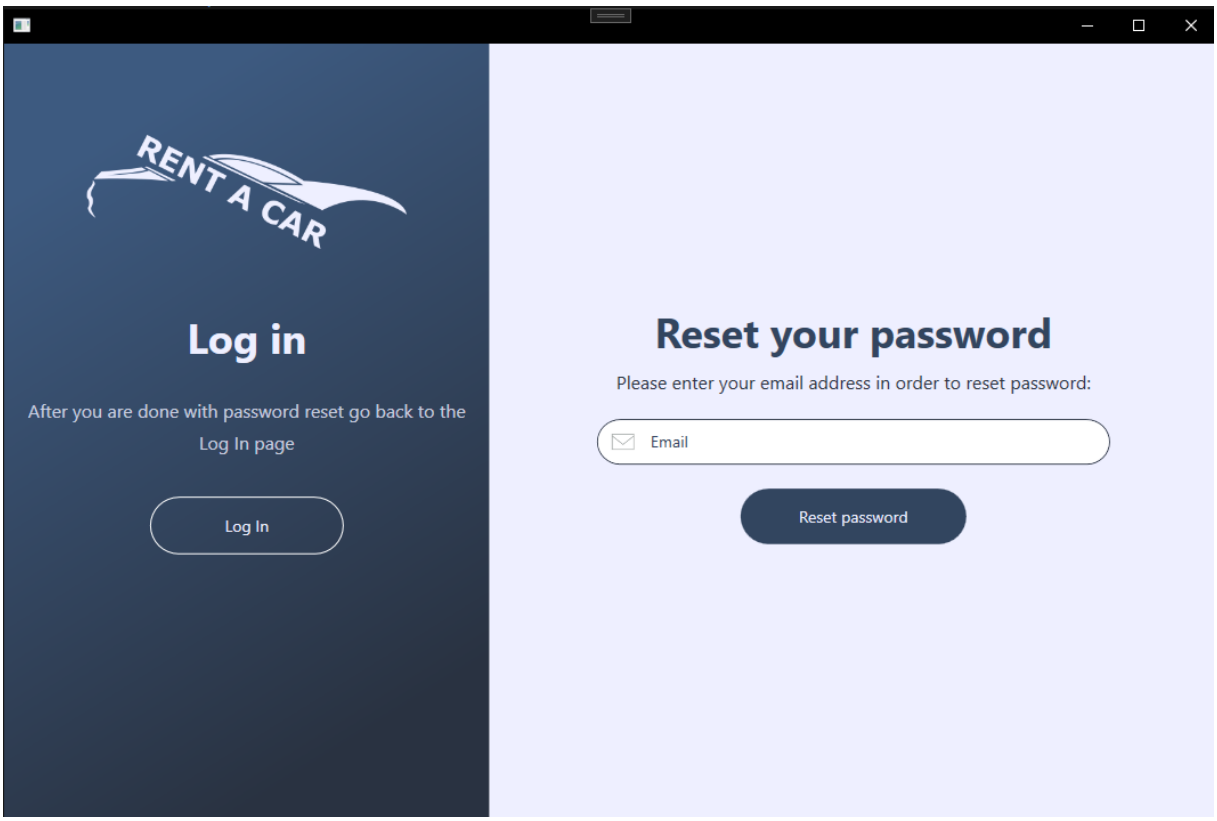
Kao prva cjelina objašnjen je proces registracije i autentikacije. Kao što je navedeno u funkcionalnostima aplikacije, svakom korisniku koji koristi aplikaciju je omogućen proces registracije. Nakon registracije korisnik je u mogućnosti ulogirati se u aplikaciju te ju koristiti nakon što administrator sustava odredi ovlasti korisnika. Za izradu ovog dijela unutar aplikacije postoje klase „AuthenticationWindow“ koja predstavlja pogled (eng. *View*) te „AuthenticationVM“ koja predstavlja „ViewModel“ za navedeni pogled. Sljedeći dio prikazuje vizualni dizajn ovog dijela aplikacije izrađen u jeziku oznaka „XAML“ nakon čega slijedi prilog najključnijeg dijela koda koji je bio zaslužan za realizaciju dijela te njegov opis.



Slika 4 Registracija



Slika 5 Autentikacija



Slika 6 Reset lozinke

Kako svaki pogled sadrži određene komande (eng. *Commands*) koji su povezani (eng. *Binding*) na odgovarajući gumb tako se i u ovom pogledu koriste jedne od njih. Prilikom inicijalizacije „ViewModel-a“ unutar konstruktora kreiramo te komande, a svaka komanda zapravo ima svoju klasu. Ovaj pogled koristi svega pet klasa koje nasljeđuju sučelje „ICommand“ , a to su redom „AuthenticationCommand“, „RegistrationCommand“, „ResetPasswordCommand“, „SwitchToRegistrationCommand“ te „SwitchToResetPasswordCommand“. Svaka od ovih klasa implementira sučelje „ICommand“, stoga implementira i iste metode koje određuje sučelje. Implementirajući ovo sučelje garantiramo da će klasa imati metode „CanExecute“, „Execute“ te event „CanExecuteChanged“. Najključnija metoda je „Execute“ u kojoj definiramo koja metoda iz „ViewModel-a“ će se koristiti prilikom korisnikove akcije, dok sam „ViewModel“ postavljamo unutar konstruktora klase. „CanExecute“ metoda određuje kada se akcija može izvršiti te nije obavezna za implementaciju. Ukoliko nema ograničenja nad akcijom, ova metoda će vraćati istinitu vrijednost (eng. *True*) dok u nekim slučajevima, kao npr. autentikacija, moramo ograničiti akciju da se može izvršiti samo u slučaju kada je korisnik unio svoje kredencije. Sljedeći kod prikazuje implementaciju klase za primjer autentikacije dok ostale nisu navedene jer sadrže sličnu logiku.

```
public class AuthenticationCommand : ICommand
{
    public AuthenticationVM VM { get; set; }

    public event EventHandler CanExecuteChanged
    {
        add
        {
            CommandManager.RequerySuggested += value;
        }
        remove
        {
            CommandManager.RequerySuggested -= value;
        }
    }

    public AuthenticationCommand(AuthenticationVM vm)
    {
        VM = vm;
    }

    public bool CanExecute(object parameter)
    {
        User user = parameter as User;

        if (user == null)
            return false;
        if (string.IsNullOrEmpty(user.Email))
            return false;
        if (string.IsNullOrEmpty(user.Password))
            return false;
```

```
            return true;
        }

        public void Execute(object parameter)
        {
            VM.Authenticate();
        }
    }
```

U navedenome slučaju, ograničenja nad ovom akcijom su da objekt korisnika, njegova adresa elektroničke pošte te lozinka ne smiju biti prazni (eng. *Empty*) ili ne imati vrijednost (eng. *Null*). Još jedna komanda vrijedna spomena je ona koja se izvršava prilikom registracije. Kod registracije se koristi ista logika, no ograničenja nad akcijom su da se moraju popuniti sve vrijednosti koje su definirane u formi prije izvršenja registracije. Koristeći ove komande držimo se nacrta koji definira „MVVM“ arhitektura. Svaka od prethodno navedenih komadi se inicijalizira unutar konstruktora „ViewModel-a“ te su na ovaj način napravljene i komande koje omogućuju kretanje između pogleda registracije, prijave i reset-a lozinke koji se svi nalaze unutar istog prozora. U kombinaciji s istim komandama ključnu ulogu prilikom promjene pogleda imaju svojstva koja predstavljaju vidljivost (eng. *Visibility Properties*). Ova svojstva su također inicijalizirana unutar konstruktora klase čija se deklaracija obavlja globalno. Na kraju konstruktora možemo vidjeti i inicijalizaciju korisnika.

```
public AuthenticationVM()
{
    AuthenticationVisibility = Visibility.Visible;
    RegisterVisibility = Visibility.Collapsed;
    ResetPasswordVisibility = Visibility.Collapsed;

    RegistrationCommand = new RegistrationCommand(this);
    AuthenticationCommand = new AuthenticationCommand(this);
    ResetPasswordCommand = new ResetPasswordCommand(this);
    SwitchToRegistrationCommand = new SwitchToRegistrationCommand(this);
    SwitchToResetPasswordCommand = new SwitchToResetPasswordCommand(this);

    this.User = new User();
}
```

U nastavku je navedena metoda koja omogućava promjenu između pogleda registracije i autentikacije, dok je na jako sličan način odrađeno i kretanje između autentikacije i pogleda za promjenu lozinke. Logika iza pogleda je da je prvotno ograničena vidljivost samo na autentikaciju, dok se koristeći određene gumbe i prethodno navedene komande za promjenu između pogleda pokreću sljedeće metode. Unutar ovih metoda provjerava se globalno deklarirana varijabla koja govori u kojem prozoru se sada nalazimo. U slučaju istinite vrijednosti vidljivost registracije postaje vidljiva i miče se vidljivost autentikacije dok je u suprotnom slučaju obrnuta situacija. Isto vrijedi i za poglede autentikacije i reset-a lozinke.

```
public void SwitchToRegistration()
{
    isShowingRegister = !isShowingRegister;

    if (isShowingRegister)
    {
        RegisterVisibility = Visibility.Visible;
        AuthenticationVisibility = Visibility.Collapsed;
    }
    else
    {
        RegisterVisibility = Visibility.Collapsed;
        AuthenticationVisibility = Visibility.Visible;
    }
    this.StatusMessage = "";
}


public void SwitchToResetPassword()
{
    isShowingResetPassword = !isShowingResetPassword;

    if (isShowingResetPassword)
    {
        ResetPasswordVisibility = Visibility.Visible;
        AuthenticationVisibility = Visibility.Collapsed;
    }
    else
    {
        ResetPasswordVisibility = Visibility.Collapsed;
        AuthenticationVisibility = Visibility.Visible;
    }
    this.StatusMessage = "";
}
```

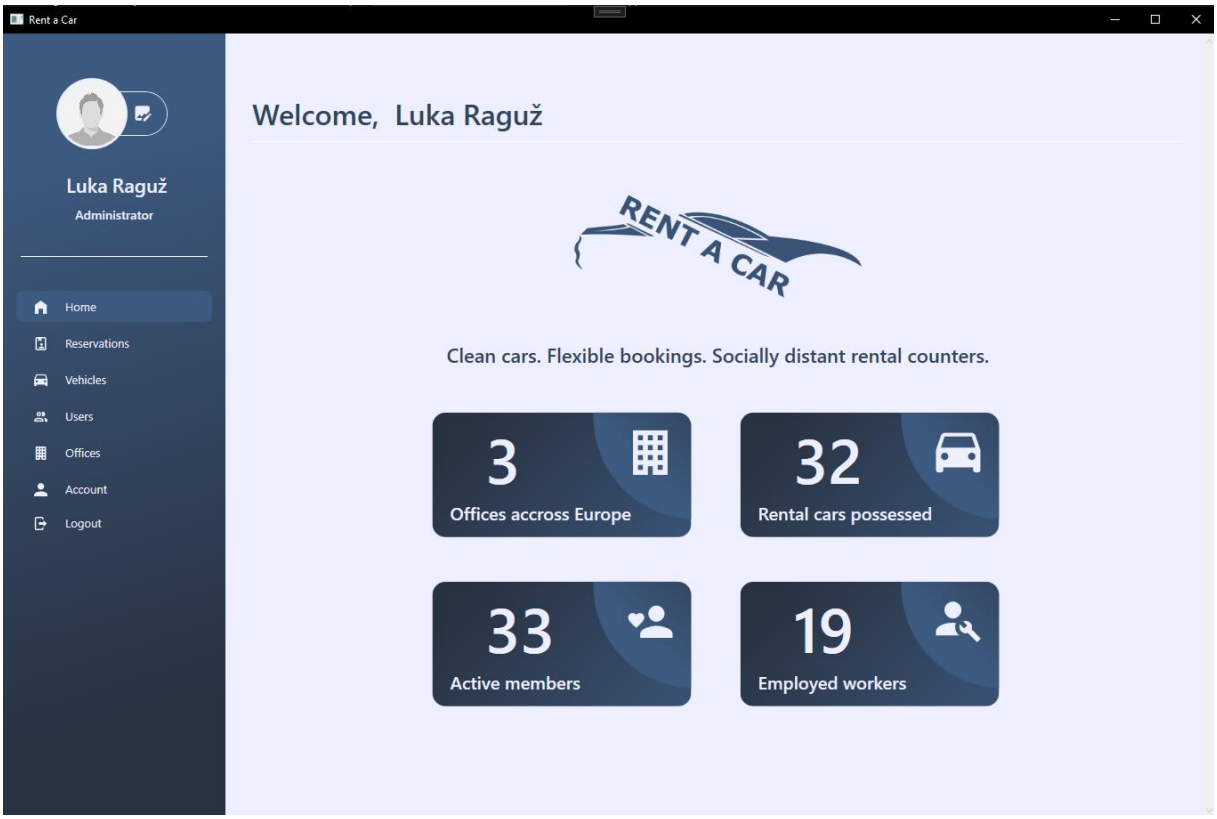
Radnje koje se izvršavaju na pritiske glavnih gumbi, a to su registracija, autentikacija i resetiranje lozinke koriste metode iz klase „FirebaseAuthHelper“ koja je navedena ranije. Iste te spomenute metode prilikom primanja odgovora od „Firebase Auth REST API-ja“ primaju poruku odgovora ili greške što se zabilježava u svojstvo „StatusMessage“ koje se korisniku u lijepom formatu prikazuje na ekran. Neke od ovih statusnih poruka su poruka slabe lozinke, poruka koja informira korisnika o zauzetosti adrese elektroničke pošte, poruka koja signalizira korisniku da je pogrešno unio svoje kredencije i slično. Statusna poruka koja je zapravo odgovor od korištenog „API-ja“ u svakom slučaju mora sadržavati vrijednost, bio to odgovor ili greška. Unutar metode za autentikaciju provjerava se da li to varijabla „statusMessage“ sadrži vrijednost te da li je korisnik blokiran. Ukoliko jedno od ovih uvjeta ne daje istinitosnu vrijednost te ukoliko korisnik nema verificiranu adresu elektroničke pošte, nije u mogućnosti ući unutar aplikacije. Ako su svi uvjeti zadovoljeni zabilježava se vrijeme kada je korisnik viđen unutar aplikacije te se isto vrijeme ažurira unutar baze podataka. Na sličan način je napravljena registracija gdje ključnu razliku ima pozivanje različitih metoda iz klase „FirebaseAuthHelper“ te postavljanje drugih vrijednosti koje se bilježe u bazu prilikom uspješne registracije.

```
public async void Authenticate()
{
    string statusMessage = await
FirebaseAuthHelper.Authenticate(this.User);
    this.StatusMessage = statusMessage;

    if (string.IsNullOrEmpty(statusMessage) && !this.User.Blocked)
    {
        App.User.LastSeenAt = DateTime.Now;
        await FirebaseDatabaseHelper.Update(App.User);
        Authenticated?.Invoke(this, new EventArgs());
    }
}
public async void Register()
{
    string statusMessage = await FirebaseAuthHelper.Register(this.User);
    this.StatusMessage = statusMessage;

    if (string.IsNullOrEmpty(statusMessage))
    {
        AuthenticationWindow authenticationWindow = new
AuthenticationWindow();
        authenticationWindow.Show();

        Authenticated?.Invoke(this, new EventArgs());

        this.User.UserId = App.User.UserId;
        this.User.IdToken = App.User.IdToken;
        this.User.RefreshToken = App.User.RefreshToken;
        this.User.EmailVerified = false;
        this.User.Blocked = false;
        this.User.Role = Role.Member.ToString();
        this.User.RegisteredAt = DateTime.Now;
        this.User.LastSeenAt = DateTime.Now;
        this.User.AvatarAzureLocation = string.Empty;
        this.User = SecurityHelper.HashPassword(this.User);
        await FirebaseAuthHelper.SendVerificationEmail(this.User);
        this.StatusMessage = statusMessage;
        await FirebaseDatabaseHelper.Insert(this.User);
    }
}
```

Kod koji je zaslužan za registraciju sadrži neke slične stvari koje su već objašnjenje kod procesa autentikacije. Unutar bloka koji se izvršava ukoliko je registracija uspješna vraćamo se na inicijalni prozor koji prikazuje autentikaciju. Prilikom izvršenja ove metode spremamo sve inicijalne korisničke podatke koji se spremaju unutar baze podataka, a to su identifikatori koji su potrebni za korištene „API-ja“ koji su spremljeni prilikom odgovora unutar lokalni „storage“ aplikacije. Navedeni inicijalni podaci su redom neistinita vrijednost (eng. *False*) za status verificiranosti adrese elektroničke pošte te blokiranosti korisnika, inicijalna uloga član, vrijeme registracije i zadnje vidljivosti, prazna lokacija korisničke slike te heširana lozinka. Nakon spremanja ovih vrijednosti šalje se verifikacijski kod na korisnikovu adresu elektroničke pošte te se zapis zabilježava u bazu podataka.

8.3.2. Pogled početne stranice i navigacija

Početna stranica aplikacije daje korisnicima mali statistički osvrt na poslovanje auto kuća u minimalističkom modernom stilu. Sa lijeve strane aplikacije nalazi se korisnička kontrola koja prikazuje korisničku sliku, ime korisnika te njegovu ulogu dok u nastavku slijedi navigacija po cjelinama navedenim u prijašnjim poglavljima. Na početnoj stranici nalazi se pozdravna poruka korisniku uz logo i moto firme te korisničke kontrole (eng. *User controls*) koje iz baze podataka skupljaju podatke o brojkama koje su bitne i zanimljive korisniku. Sljedeća slika prikazuje dizajn navedenog dijela.



Slika 7 Početna stranica aplikacije

Pogled koji možemo vidjeti na priloženoj slici se dijeli na dva dijela, a to su lijevi fiksni dio koji se prikazuje uvijek, neovisno unutar cjeline koje se nalazimo te desni varijabilni dio koji se mijenja ovisno o cjelini koju smo odabrali unutar navigacije. Zbog navedene podjele, lijevi dio koristi „MainVM“,a varijabilni dio koji predstavlja poglede sadrži zaseban „ViewModel“ za svaki promijenjeni pogled. Korištene komade unutar glavnog pogleda (eng. *MainView*) su komade za odjavu iz aplikacije, za promjenu korisničke fotografije te za promjenu pogleda koje odabiremo unutar navigacije. Korištenje komandi je objašnjeno u prijašnjoj cjelini te se na isti način koristi u sljedećim cjelinama te su objašnjene samo ključne metode koje omogućuju rad funkcionalnosti. Unutar ove cjeline kao ključne dijelove možemo izdvojiti funkcionalnost

promjene korisničke fotografije, navigacije, odjave te jedan primjer izrade korisničke kontrole za prikaz statističkog podatka. Za funkcionalnost promjene korisničke fotografije koristi se „Azure storage“ na kojemu se čuvaju korisničke slike u formatu „.jpg“. Logika iza ove funkcionalnosti je da je na „Azure storage-u“ pohranjena inicijalna korisnička slika koja se dohvaća prilikom inicijalizacije „MainVM-a“ odnosno prilikom svakog logiranja korisnika. Inicijalna korisnička slika se prikazuje sve do prve korisničke promjene fotografije nakon čega se novoučitana fotografija sprema u oblak te se ista vuče prilikom sljedećeg logiranja. Sljedeći dio koda prikazuje konstruktor u kojemu se može vidjeti poziv metode za dohvaćanje korisničke fotografije iza čega se i nalazi sama implementacija dohvata i pohrane u oblak.

```
public MainVM()
{
    UpdateViewCommand = new UpdateViewCommand(this);
    LogoutCommand = new LogoutCommand(this);
    ChangeAvatarCommand = new ChangeAvatarCommand(this);

    this.User = App.User;
    GetAvatarPicture();
}
public async Task ChangeAvatar()
{
    OpenFileDialog dialog = new OpenFileDialog();
    dialog.Filter = "Image files (*.png; *.jpg)|*.png;*.jpg;*jpeg|All files
(*.*)|*.*";
    dialog.InitialDirectory =
Environment.GetFolderPath(Environment.SpecialFolder.MyPictures);

    string avatarFilepath = string.Empty;
    if (dialog.ShowDialog() == true)
    {
        avatarFilepath = dialog.FileName;
        this.ImageSource = new BitmapImage(new Uri(avatarFilepath));
    }

    string connectionString =
"DefaultEndpointsProtocol=https;AccountName=thesisappstoragelpa;AccountKey=
o2en7Tk69M/c6uEKcChJcxLQrrUZq6aRzkT/G+KroyA/ZreW9Fz+QuU2kpqdCg+d3lotNhhzl2m
k+ASteetfnQ==;EndpointSuffix=core.windows.net";
    string containerName = "avatars";
    string avatarFilePath = ImageSource.UriSource.LocalPath;

    var container = new BlobContainerClient(connectionString,
containerName);

    var blob = container.GetBlobClient(App.User.Id);
    await blob.UploadAsync(avatarFilePath, true);

    this.User.AvatarAzureLocation =
"https://thesisappstoragelpa.blob.core.windows.net/avatars/" + App.User.Id;
    await FirebaseDatabaseHelper.Update(this.User);

    this.StatusMessage = "Your avatar has been changed.";
}
```

```
public async void GetAvatarPicture()
{
    if (!string.IsNullOrEmpty(App.User.AvatarAzureLocation))
    {
        string downloadPath = $"{App.User.Id}.jpg";

        BlobClient blobClient = new BlobClient(new
Uri(App.User.AvatarAzureLocation));
        await blobClient.DownloadToAsync(downloadPath);

        using (FileStream fileStream = new FileStream(downloadPath,
FileMode.Open, FileAccess.Read))
        {
            ImageSource = new BitmapImage(new Uri(fileStream.Name));
        }
    }
    else
    {
        string downloadPath = "empty.jpg";

        BlobClient blobClient = new BlobClient(new
Uri("https://thesisappstoragelpa.blob.core.windows.net/avatars/empty"));
        await blobClient.DownloadToAsync(downloadPath);

        using (FileStream fileStream = new FileStream($"empty.jpg",
FileMode.Open, FileAccess.Read))
        {
            ImageSource = new BitmapImage(new Uri(fileStream.Name));
        }
    }
}
```

Priloženi kod prikazuje implementacije dvije metode koje su ključne za odrađivanje funkcionalnosti promjene korisničke fotografije. Prva metoda naziva „ChangeAvatar“ poziva se prilikom korisnikovog pritiska na gumb pored svoje fotografije pri čemu se otvara dijalog za odabir datoteka filtriranih po ekstenzijama koje simboliziraju fotografije. Prilikom odabira fotografije njena putanja se sprema unutar svojstva „ImageSource“ (eng. *Propertie*) te se uz pomoć „connectionStringa“ koji nas vodi do odgovarajućeg kontejnera (eng. *Container*) na „Azure storage-u“ pohranjuje u oblak. U kodu također možemo vidjeti definirano ime kontejnera koje je bitno da sadrži isti naziv kao i onaj gdje se fotografija postavlja. Nakon što su definirane sve potrebne stvari za slanje fotografije u oblak kreira se novi „BlobContainerClient“ s određenim parametrima te se pomoću gotove metode „UploadAsync“ fotografija postavlja na „Storage“ te se ažurira putanja do fotografije unutar baze podataka kako bi se u sljedećim korisničkim logiranjima mogla dohvaćati ista. Sljedeća metoda je „GetAvatarPicture“ koja se poziva u konstruktoru „MainVM-a“ te je zaslužnja za dohvaćanje fotografije iz oblaka. Unutar ove metode se provjerava postojanost putanje do „Storage-a“ te ukoliko je ona prazna dohvaća se inicijalna korisnička fotografija. U slučaju postojanja putanje kreira se novi „BlobClient“ s odgovarajućim „URI-jem“ te se poziva gotova metoda „DownloadToAsync“ koja kao parametar šalje ime fotografije što je zapravo korisnički identifikator, jer je tako prvotno spremljena iz

razloga da ne bi došlo do korištenja istog imena fotografije jer je identifikator za korisnika jedinstven. Nakon skidanja fotografije na korisničko računalo dohvaća se ista datoteka (slika) te se kao „BitmapImage“ postavlja za svojstvo „ImageSource“ koje je povezano (eng. *Binding*) kao resurs na element slike unutar jezika oznaka s kojim je izgrađeno korisničko sučelje.

Sljedeći bitan dio koda unutar ove cjeline je korištenje korisničke navigacije. Za ovu funkcionalnost ključnu ulogu ima implementacija metode „Execute“ unutar komade „UpdateViewCommand“. Sljedeći kod prikazuje kako se ovisno o proslijeđenom parametru odabire „ViewModel“ te se postavlja kao selektirani. Prilikom promjene selektiranog „ViewModel-a“ koristeći navigaciju korisnik može vidjeti izmjenu na varijabilnom dijelu prikaza.

```
public class UpdateViewCommand : ICommand
{
    private MainVM VM;

    public UpdateViewCommand(MainVM vm)
    {
        this.VM = vm;
    }

    public event EventHandler CanExecuteChanged;

    public bool CanExecute(object parameter)
    {
        return true;
    }

    public void Execute(object parameter)
    {
        if (parameter.ToString() == "Home")
        {
            VM.SelectedViewModel = new HomeVM();
        }
        else if (parameter.ToString() == "Reservations")
        {
            VM.SelectedViewModel = new ReservationsVM();
        }
        else if (parameter.ToString() == "Vehicles")
        {
            VM.SelectedViewModel = new VehiclesVM();
        }
        else if (parameter.ToString() == "Users")
        {
            VM.SelectedViewModel = new UsersVM();
        }
        else if (parameter.ToString() == "Offices")
        {
            VM.SelectedViewModel = new OfficesVM();
        }
        else if (parameter.ToString() == "Account")
        {
            VM.SelectedViewModel = new AccountVM();
        }
    }
}
```

Sljedeća objašnjena funkcionalnost je mogućnost odjave korisnika. Za ovu funkcionalnost se također koristi komada koja se veže na gumb objave te prilikom korisnikovog pritiska poziva metodu navedenu u nastavku. Unutar ove metode može se vidjeti ponovna inicijalizacija prozora za autentikaciju prilikom čega se zatvara glavni prozor koji je zapravo prikaz aplikacije. Nakon odjave postavlja se vrijeme zadnje pojave korisnika unutar aplikacije, isto vrijeme se ažurira unutar baze podataka te se objekt korisnika unutar lokalnog „storage-a“ aplikacije ponovno inicijalizira na prazan objekt korisnika. U nastavku se nalazi metoda zaslužna za izvršavanje ove radnje.

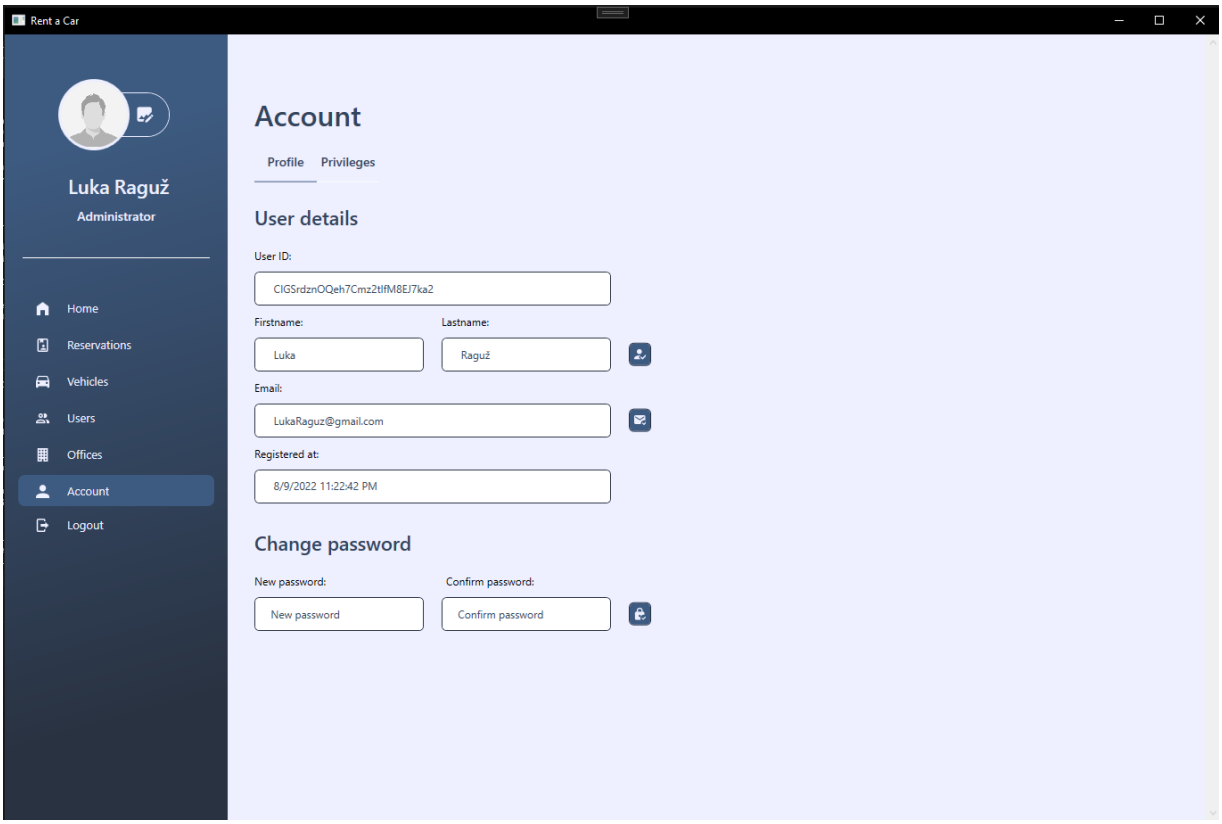
```
public async void Logout(Window currentWindow)
{
    AuthenticationWindow authenticationWindow = new AuthenticationWindow();
    authenticationWindow.Show();

    currentWindow.Close();

    App.User.LastSeenAt = DateTime.Now;
    await FirebaseDatabaseHelper.Update(App.User);

    App.User = new User();
}
```

Zadnja bitna stvar spomenuta unutar ove cjeline je kreiranje korisničkih kontroli na početnoj stranici aplikacije. Početna stranica aplikacije (desni varijabilni dio) je zapravo zaseban pogled koji sadrži svoj „ViewModel“ naziva „HomeVM“. Unutar ove klase nalaze se varijable koje predstavljaju sadržaj prikazan kroz brojke te se dohvaća korisnik kako bi bila realizirana pozdravna poruka. Na početnom pogledu se mogu vidjeti kartice realizirane kroz korisničke kontrole na koje se povezuju brojevi iz „ViewModel-a“, a kartice su implementirane jezikom oznaka „XAML“. Kako je kod napisan ovim jezikom unutar aplikacije jako velik na sljedećoj fotografiji je prikazan dizajn korisničke kontrole dok je način kreiranja i povezivanja elemenata sa „ViewModel-om“ te stilizacija aplikacije objašnjena u zasebnoj cjelini u nastavku.

Slika 8 Korisnička kontrola za prikaz broja zaposlenika

8.3.3. Pogled korisničkih postavki

Sljedeće poglavlje opisuje proces izrade korisničkih postavki. Kao što je navedeno u funkcionalnostima aplikacije, svakom korisniku je omogućen pregled i ažuriranje vlastitih postavki. Unutar glavnog pogleda omogućena ja sporedna navigacija između dva potpogleda, a to su korisničke postavke i korisničke privilegije. Sljedeća slika prikazuje dizajn korisničkih postavki dok su u nastavku objašnjeni ključni dijelovi koda ove cjeline.



Slika 9 Korisničke postavke

Kao što se na slici može vidjeti, osim pregleda, korisniku su omogućene tri druge akcije. Ove akcije unutar jezika oznaka definirane su kao gumbi te na sebi imaju povezane komade koje okidaju određene metode prilikom korisničke akcije. Korisnik je u mogućnosti promijeniti svoje ime i prezime, adresu elektroničke pošte te izmijeniti vlastitu lozinku. Izrada korisničkog sučelja sadrži veći dio koda što je objašnjeno u zasebnoj cjelini, dok su u nastavku objašnjene metode koje se izvršavaju prilikom okidanja gumbova. Prva akcija, odnosno promjena korisničkog imena je trivijalna te njen kod nije prikazan unutar rada, a funkcionira tako da se svojstvo unutar „AccountVM“ koje je povezano na element „TextBox“ zamjenjuje s imenom i prezimenom trenutno logiranog korisnika te se pomoću klase „FirebaseDatabaseHelper“ izvršava ažuriranje istog korisnika. Prilikom ove akcije korisniku se

u gornjem desnom ekranu pokazuje uspješna statusna poruka. U nastavku su navedene metode koje su implementirane u svrhu promjene adrese korisničke pošte te lozinke.

```
public async void ChangeEmail()
{
    this.User.Email = email;
    await FirebaseDatabaseHelper.Update(this.User);
    string statusMessage = await FirebaseAuthHelper.ChangeEmail(this.User);
    if (string.IsNullOrEmpty(statusMessage))
        this.StatusMessage = "Your email address has been changed.";
    else
        this.StatusMessage = statusMessage;
}

public async void ChangePassword()
{
    this.User.Password = password;
    this.User.ConfirmPassword = confirmPassword;
    string statusMessage = await
FirebaseAuthHelper.ChangePassword(this.User);
    if (string.IsNullOrEmpty(statusMessage))
        this.StatusMessage = "Your password has been changed.";
    else
        this.StatusMessage = statusMessage;
    this.User = SecurityHelper.HashPassword(this.User);
    await FirebaseDatabaseHelper.Update(this.User);
}
```

Ove dvije metode su lako shvatljive te glavna logika stoji unutar „FirebaseAuthHelper“ klase koja šalje zahtjev „Firebase Auth REST API-ju“ te zahtijeva promjenu korisnikovih kredencija. Prilikom promjene preko „API-ja“ još je potrebno promijeniti kredencije unutar baze podataka stoga se koristi i klasa „FirebaseDatabaseHelper“ za ostvarenje cilja. Sljedeći kod prikazuje zahtjeve koji se šalju na „REST API“ te je uz kod priloženo tijelo gdje se u oba slučaja unutar zahtjeva traži slanje identifikacijskog tokena dodijeljenog korisniku prilikom registracije. Osim identifikacijskog tokena šalju se podaci za promjenu te „returnSecureToken“ koji je uvijek istinite vrijednosti. Zahtjev je potrebno serijalizirati što je ostvareno s klasom „JsonConvert“ koja je ubačena u projekt kao „NuGet package“ te se zahtjev šalje na „REST API“.

```
.
public static async Task<string> ChangeEmail(User user)
{
    using (HttpClient client = new HttpClient())
    {
        var body = new
        {
            idToken = user.IdToken,
            email = user.Email,
            returnSecureToken = true
        };

        string bodyJson = JsonConvert.SerializeObject(body);
        var data = new StringContent(bodyJson, Encoding.UTF8,
"application/json");
```

```
        var response = await
client.PostAsync($"https://identitytoolkit.googleapis.com/v1/accounts:updat
e?key={api_key}", data);

        if (response.IsSuccessStatusCode)
        {
            string resultJson = await response.Content.ReadAsStringAsync();
            var result =
JsonConvert.DeserializeObject<FirebaseResult>(resultJson);

            return "";
        }
        else
        {
            string errorJson = await response.Content.ReadAsStringAsync();
            var error = JsonConvert.DeserializeObject<Error>(errorJson);

            return FirebaseErrorMessage.GetByKey(error.error.message);
        }
    }
}

public static async Task<string> ChangePassword(User user)
{
    using (HttpClient client = new HttpClient())
    {
        var body = new
        {
            idToken = user.IdToken,
            email = user.Email,
            password = user.Password,
            returnSecureToken = true
        };

        string bodyJson = JsonConvert.SerializeObject(body);
        var data = new StringContent(bodyJson, Encoding.UTF8,
"application/json");

        var response = await
client.PostAsync($"https://identitytoolkit.googleapis.com/v1/accounts:updat
e?&key={api_key}", data);

        if (response.IsSuccessStatusCode)
        {
            string resultJson = await response.Content.ReadAsStringAsync();
            var result =
JsonConvert.DeserializeObject<FirebaseResult>(resultJson);

            return "";
        }
        else
        {
            string errorJson = await response.Content.ReadAsStringAsync();
            var error = JsonConvert.DeserializeObject<Error>(errorJson);

            return FirebaseErrorMessage.GetByKey(error.error.message);
        }
    }
}
```

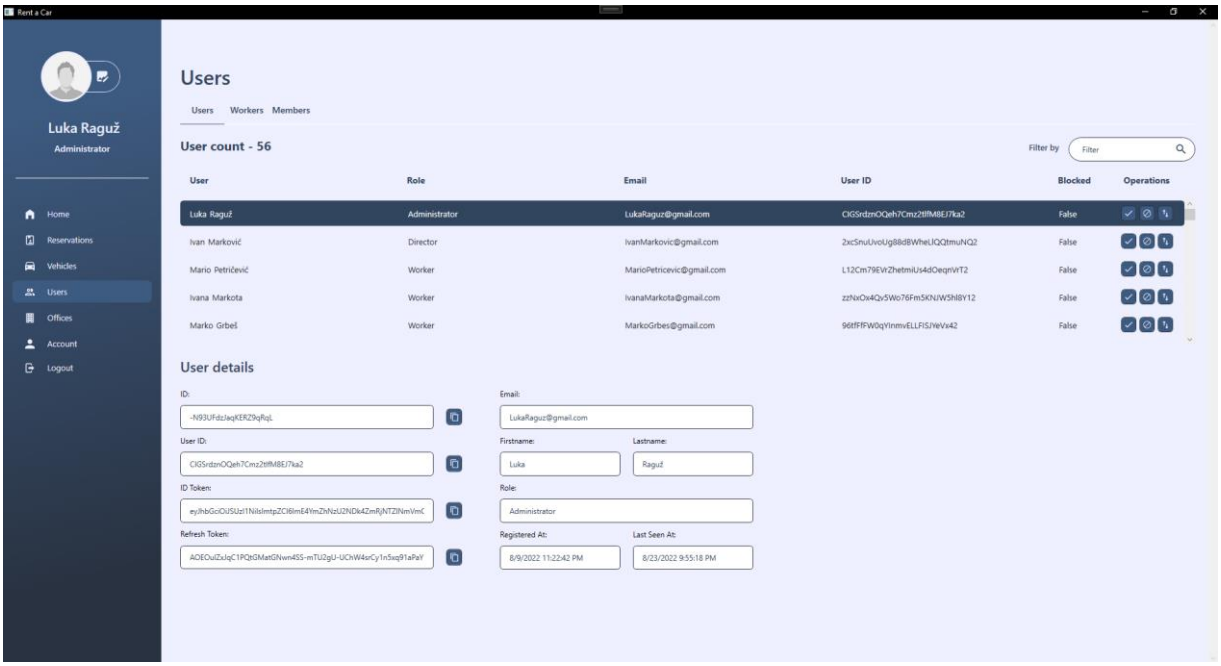
Navigacija između sporednih pogleda omogućena je na isti način kako je to napravljeno i kod „AuthenticationWindow-a“ gdje se prilikom pritiska na navigaciju, što su zapravo gumbovi, pokreće komanda koja mijenja vidljivost. Unutar „AccountVM“ klase definirane su dvije vidljivosti gdje se izmjenom tabova jedna aktivira i druga deaktivira, a to se ostvaruje metodom koja se okida na svaku korisnikov pritisak na gumb unutar navigacije. Nakon promjene na korisničke privilegije korisniku je omogućen pregled svih mogućih akcija gdje može pregledati svoje ovlasti na način da su mu iste označene kvačicom. Na popisu su uvlakom označene cjeline unutar aplikacije gdje se kao dijete svake prikazuju moguće akcije na istom. Iznad ovlasti je prikazan „TextBox“ koji se veže na ulogu trenutno logiranog korisnika. Sljedeća slika prikazuje dizajn korisničkih ovlasti koji je neovisno o ulozi omogućen svakom korisniku koji je ulogiran u aplikaciju.

Slika 10 Korisničke ovlasti

Ova funkcionalnost realizirana je na način tako što su postavljeni elementi „CheckBox“ prilikom izrade korisničkog sučelja sa „hardkodiranim“ vrijednostima akcija što je u ovom slučaju dozvoljeno jer podaci nisu varijabilni. Prilikom logiranja korisnika u sustav dolazi do očitavanja korisničke uloge te se „CheckBox-i“ popunjavaju ovisno o istoj. Na ovaj način se korisniku prikazuje jasan uvid što je sve moguće raditi u aplikaciji te je koristan radnicima, a pogotovo u slučaju nadogradnje sustava. Očitavanje korisničkih uloga te dodjeljivanje vrijednosti „CheckBox-ima“ ostvaruje se unutar konstruktora samog pogleda, a ne „ViewModel-a“.

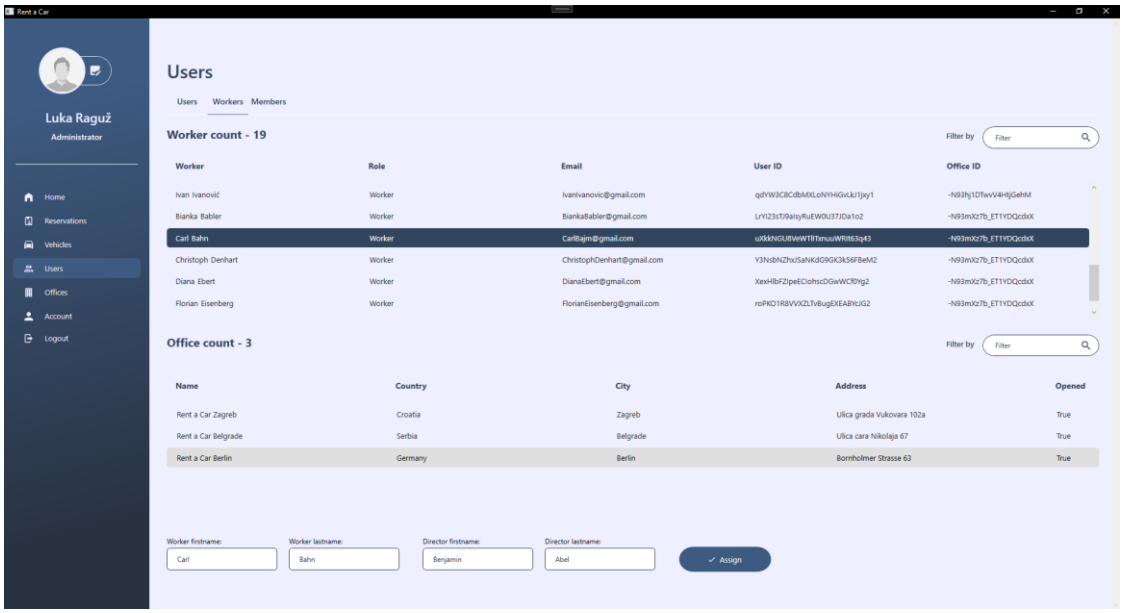
8.3.4. Pogled administracije korisnika

Naredno poglavlje opisuje funkcionalnost administracije korisnika čiju vidljivost, kao što je opisano u funkcionalnostima aplikacije, ima samo administrator sustava. Ova cjelina administratoru omogućuje pregled svih korisnika te različite akcije i operacije nad istima. U ovoj cjelini prikazan je dizajn sučelja te glavni dijelovi koda koji su zaslužni za implementaciju ovog dijela. Ova cjelina, osim pregleda korisnika, sadrži još dvije pod cjeline koje omogućuju dodjeljivanje radnika i članova po različitim auto kućama. Sljedeća slika prikazuje dizajn pogleda za administraciju korisnika kroz sva tri moguća pogleda.

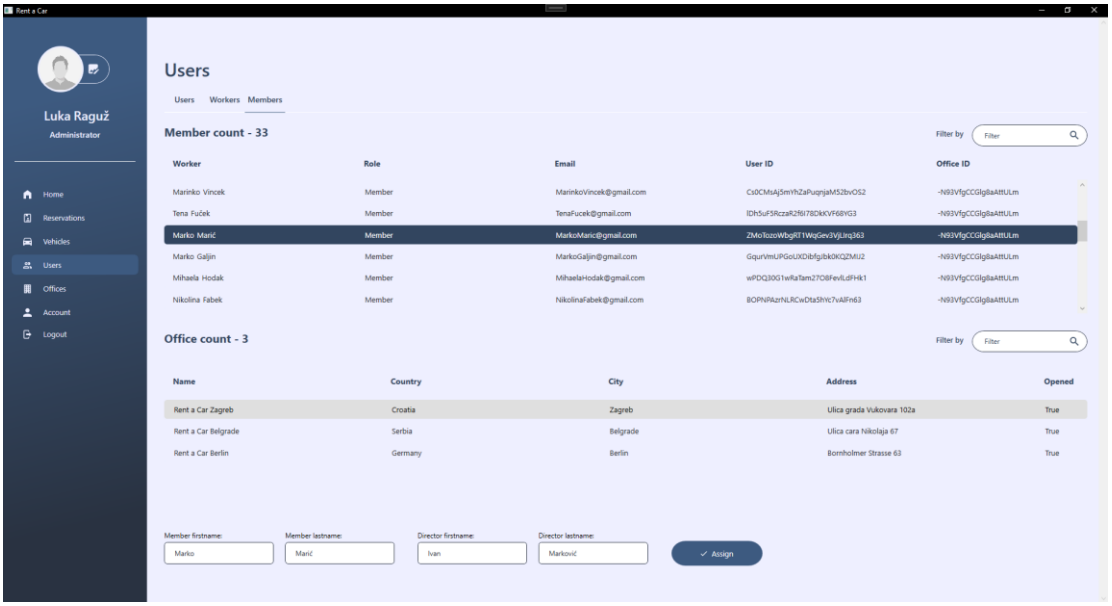


Slika 11 Pogled administracije korisnika

Slika 11 prikazuje sve korisnike unutar tablice gdje se na desnoj strani mogu vidjeti akcije koje su dostupne za korištenje. Svaka od ovih akcija predstavljena je gumbom koji na sebe ima povezanu komadu koja izvršava metodu iz „ViewModel-a“ te metoda obavlja svu daljnju potrebnu logiku. Prilikom odabira jednog od korisnika ispod tablice se automatski popunjavaju „TextBox-ovi“ njegovim detaljima. Kod detalja vezanih za identifikatore korisnika i njegove tokene korisnik ima opciju kopiranja identifikatora kako bi ga dalje mogao koristiti ukoliko je to potrebno. S desne strane se vide neki od osnovnih korisničkih podataka. Slika 12 i slika 13 imaju sličnu funkcionalnost, a služe dodjeljivanju radnika i članova auto kućama te dodjeljivanju nadređene osobe radniku. Prilikom izvršavanja ove radnje tim korisnicima se automatski omogućava korištenje aplikacije s čime im se generiraju i ovlasti. Nakon slika je prikazan jedan dio koda iz „ViewModel-a“ dok je jezik oznaka zaslužan za izradu korisničkog sučelja prikazan u zadnjoj cjelini koja pokriva korištenje „XAML-a“.



Slika 12 Pogled dodjeljivanja radnika auto kući



Slika 13 Pogled dodjeljivanja članova auto kući

Dio koda u nastavku pokazuje konstruktor „UsersVM“ klase u kojemu možemo vidjeti slične stvari spomenute u prijašnjim cjelinama. Iz razloga što su te stvari već objašnjenje, a ovdje se koriste na isti način preskočene su te su objašnjene neki drugi elementi koji se koriste u ovom „ViewModel-u“. Unutar konstruktora možemo vidjeti inicijalizaciju komandi, elemente koji ograničavaju vidljivost (koji su zaslužni su za izmjenu između podcjelina), inicijalizaciju nekih objekata te pozive metoda koji popunjavaju podatkovne tablice (eng. *datagrid*) prilikom ulaska u sam pogled odnosno kreiranja „ViewModel-a“ koji se odvija istovremeno.

```
public UsersVM()
{
    UnblockUserCommand = new UnblockUserCommand(this);
    BlockUserCommand = new BlockUserCommand(this);
    ChangeRoleCommand = new ChangeRoleCommand(this);
    ChangeToWorkersCommand = new ChangeToWorkersCommand(this);
    ChangeToUsersCommand = new ChangeToUsersCommand(this);
    ChangeToMembersCommand = new ChangeToMembersCommand(this);
    AssignWorkerToOfficeCommand = new AssignWorkerToOfficeCommand(this);
    AssignMemberToOfficeCommand = new AssignMemberToOfficeCommand(this);

    this.UsersVisibility = Visibility.Visible;
    this.WorkersVisibility = Visibility.Collapsed;
    this.MembersVisibility = Visibility.Collapsed;

    this.Users = new ObservableCollection<User>();
    this.Workers = new ObservableCollection<User>();
    this.Members = new ObservableCollection<User>();
    this.Offices = new ObservableCollection<Office>();

    App.Worker = new User();
    App.Member = new User();
    App.Office = new Office();

    GetUsers();
    GetWorkers();
    GetMembers();
    GetOffices();
}
```

U nastavku se nalazi dio koda koji objašnjava popunjavanje jedne od tablica, filtriranje tablice, objašnjenje komande koja omogućuje promjenu korisničke uloge te finalno akciju dodjeljivanja direktora odnosno poslovnice u kojoj će radnik raditi. Kao prva metoda navedeno je dohvaćanje svih korisnika (na isti način je napravljeno i dohvaćanje radnika, članova te ureda) gdje se koristi pomoćna klasa „FirebaseDatabaseHelper“ kako bi se dohvatili korisnici iz baze podataka. Nakon dohvata korisnika prolazi se po globalno deklariranoj kolekciji te se za svakog dohvaćenog korisnika kolekcija popunjava s istim. Ako unutar dohvata želimo postaviti uvjet te na taj način ograničiti dohvat koristimo metodu „.Where(uvjet)“ u nastavku.

```
private async void GetUsers()
{
    var users = await FirebaseDatabaseHelper.Read<User>();

    this.Users.Clear();
    foreach (var user in users)
    {
        this.Users.Add(user);
    }
}
```

Kako bi omogućili filtriranje korisnika potrebno je korištenje ranije spomenute metode za ograničavanje dohvata. Kao demonstracija u nastavku je prikazan kod filtriranja tablice po bilo kojoj od ključnih riječi koje se nalaze unutar iste. Ova metoda kao parametar prima niz

znakova koje predstavljaju filter koji se prvotno pretvara u mala tiskana slova kako bi se razlika u velikim i malim slovima mogla zanemariti prilikom pretrage. Nakon provjere da li je unesena vrijednost unutar filtera dohvaćaju se svi korisnicima kojima jedan od atributa iz tablice sadrži znakove unutar filtera. Kolekcija korisnika se čisti te se se novi filtrirani korisnici dodaju unutar kolekcije. Ova metoda se poziva na svaku izmjenu slova unutar „TextBox-a“ za filtriranje te je omogućena pomoću funkcije postavljanja događaja (eng. *Event*) unutar jezika oznaka „XAML“.

```
public async void FilterUsersGrid(string filter)
{
    filter = filter.ToLower();
    if (!string.IsNullOrEmpty(filter))
    {
        var filteredUsers = (await
FirebaseDatabaseHelper.Read<User>()).Where(u =>
        u.Firstname.ToLower().Contains(filter) ||
        u.Lastname.ToLower().Contains(filter) ||
        u.Role.ToLower().Contains(filter) ||
        u.Email.ToLower().Contains(filter) ||
        u.UserId.ToString().ToLower().Contains(filter) ||
        u.Blocked.ToString().ToLower().Contains(filter)).ToList();

        Users.Clear();
        foreach (var user in filteredUsers)
        {
            Users.Add(user);
        }
    }
    else
    {
        GetUsers();
    }
}
```

Kao jedna od glavnih i bitnijih operacija administratora odabrana je mogućnost izmjene korisničkih uloga. Ova mogućnost nalazi se unutar tablice pod stupcem operacija te prilikom korištenja podiže korisničku ulogu po redoslijedu. Za ulogu odabranog korisnika provjerava se postojeća uloga te mu se prilikom izmjene dodjeljuje jedan stupanj veća uloga i tako u krug. Prilikom izmjene uloge u gornjem desnom kutu administratoru sustava se ispisuje statusna poruka koja ga obavještava o podnesenoj akciji. Nakon izmjene uloge osvježavaju se sve tablice kako bi napravljene izmjene odmah bile vidljive. U slučaju ako administrator sam sebi pokuša promijeniti ulogu neće uspjeti te mu se ispisuje poruka kako ta akcija nije moguća. Osim ove operacije još su napravljene operacije blokiranja i odblokiranja korisnika što je trivijalna radnja te nije objašnjeno unutar ovoga rada. U nastavku se nalazi implementacija koja omogućava promjenu korisničkih uloga.

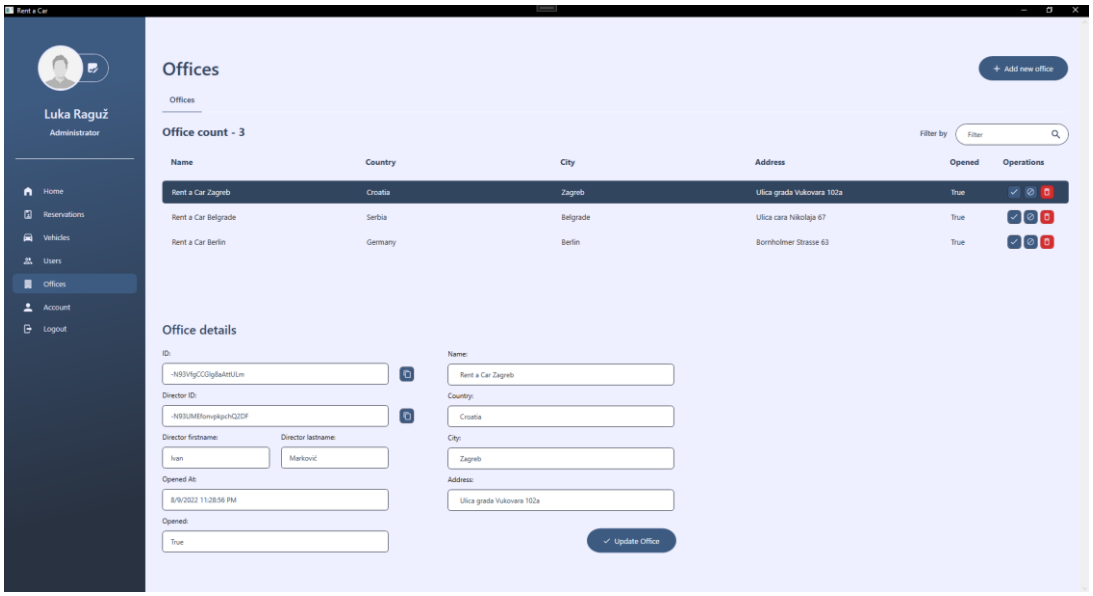
```
public async void ChangeRole()
{
    if (this.SelectedUser.Role == Role.Member.ToString())
    {
        this.SelectedUser.Role = Role.Worker.ToString();
        this.StatusMessage = $"Role for user {this.SelectedUser.Firstname}
{this.SelectedUser.Lastname} successfully changed to Worker.";
    }
    else if (this.SelectedUser.Role == Role.Worker.ToString())
    {
        this.SelectedUser.Role = Role.Director.ToString();
        this.StatusMessage = $"Role for user {this.SelectedUser.Firstname}
{this.SelectedUser.Lastname} successfully changed to Director.";
    }
    else if (this.SelectedUser.Role == Role.Director.ToString())
    {
        this.SelectedUser.Role = Role.Member.ToString();
        this.StatusMessage = $"Role for user {this.SelectedUser.Firstname}
{this.SelectedUser.Lastname} successfully changed to Member.";
    }
    else
    {
        this.SelectedUser.Role = Role.Administrator.ToString();
        this.StatusMessage = "You cannot change Administrator role.";
    }
    await FirebaseDatabaseHelper.Update(this.SelectedUser);
    GetUsers();
    GetMembers();
    GetWorkers();
}
```

Sljedeća metoda unutar „UsersVM“ klase objašnjava način dodjeljivanja korisnika odgovarajućoj poslovnici, a realizirana je na način da se dodjeljivanje omogućuje samo u slučaju ako su odabrani i korisnik i ured prilikom čega se ispod tablice popunjavaju radnikovo ime i ime direktora poslovnice. Nakon odabira obje stavke ažurira se atribut „OfficeId“ unutar tablice korisnika koji ukazuje na mjesto u kojemu korisnik radi/iznajmljuje aute (ovisno o ulozi).

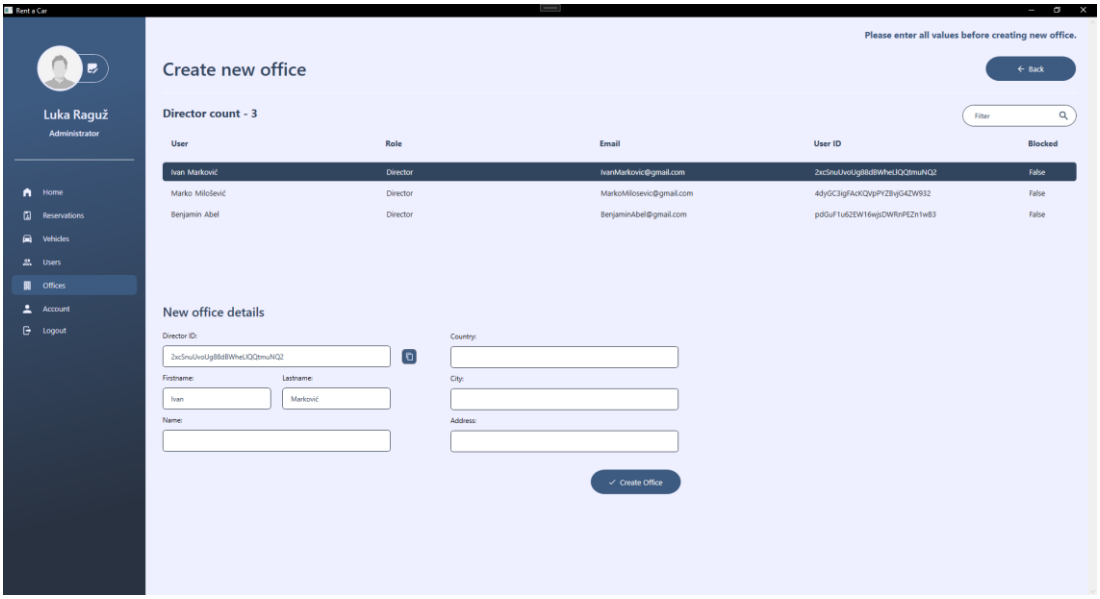
```
public async void AssignWorkerToOffice()
{
    if (App.Office.Id == null || App.Worker.Firstname == null ||
App.Worker.Lastname == null)
    {
        this.StatusMessage = "Please select worker and office to apply
assignment.";
    }
    else
    {
        App.Worker.OfficeId = App.Office.Id;
        await FirebaseDatabaseHelper.Update(App.Worker);
        this.StatusMessage = App.Worker.Firstname + " " +
App.Worker.Lastname + " is now working at " + App.Office.Name;
        GetWorkers();
        GetUsers();
    }
}
```

8.3.5. Pogled administracije poslovnica

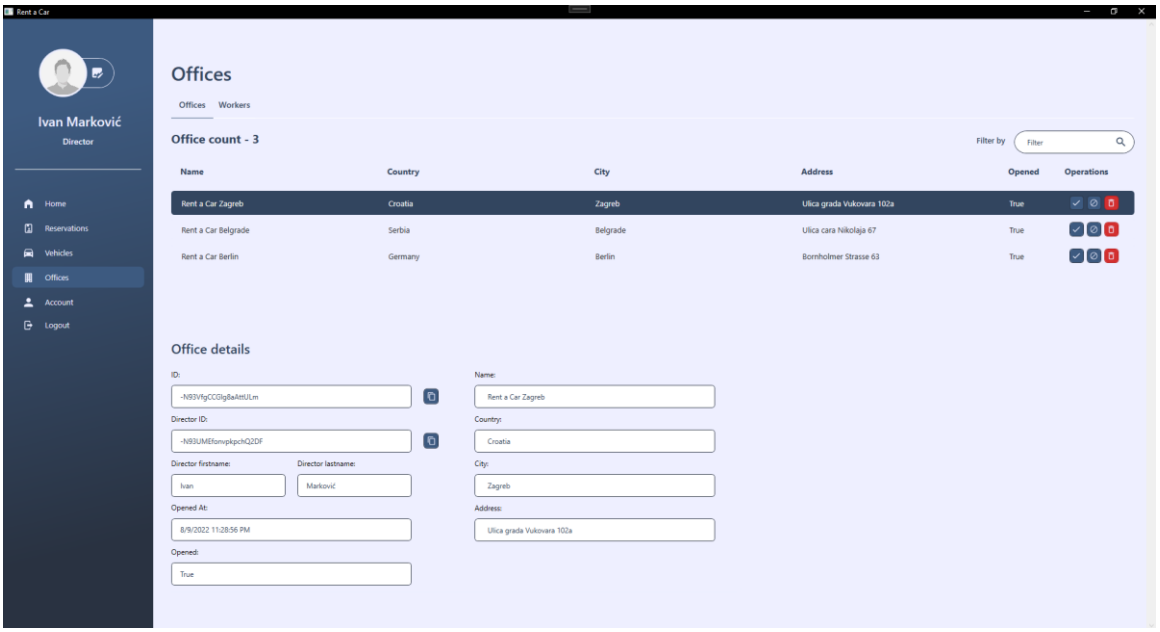
Sljedeća cjelina prikazuje pogled administracije poslovnica u kojoj su omogućene radnje dodavanja, ažuriranja, pregleda i brisanja poslovnica te pregled radnika poslovnice. Ovi pogledi se razlikuju iz različitih uloga gdje administrator može vidjeti i modificirati sve poslovnice dok direktor i radnici mogu vidjeti samo svoju poslovnicu i radnike unutar svoje poslovnice te ne mogu dodavati novu. Kako su neki od elemenata već spomenuti u prijašnjim poglavljima unutar ove cjeline objašnjena je dosad nespomenuta mogućnost dodavanja novih zapisa te brisanja istih iz tablice poslovnica. Sljedeće slike prikazuju dizajn ove cjeline iz uloga administratora, a zatim iz uloge direktora.



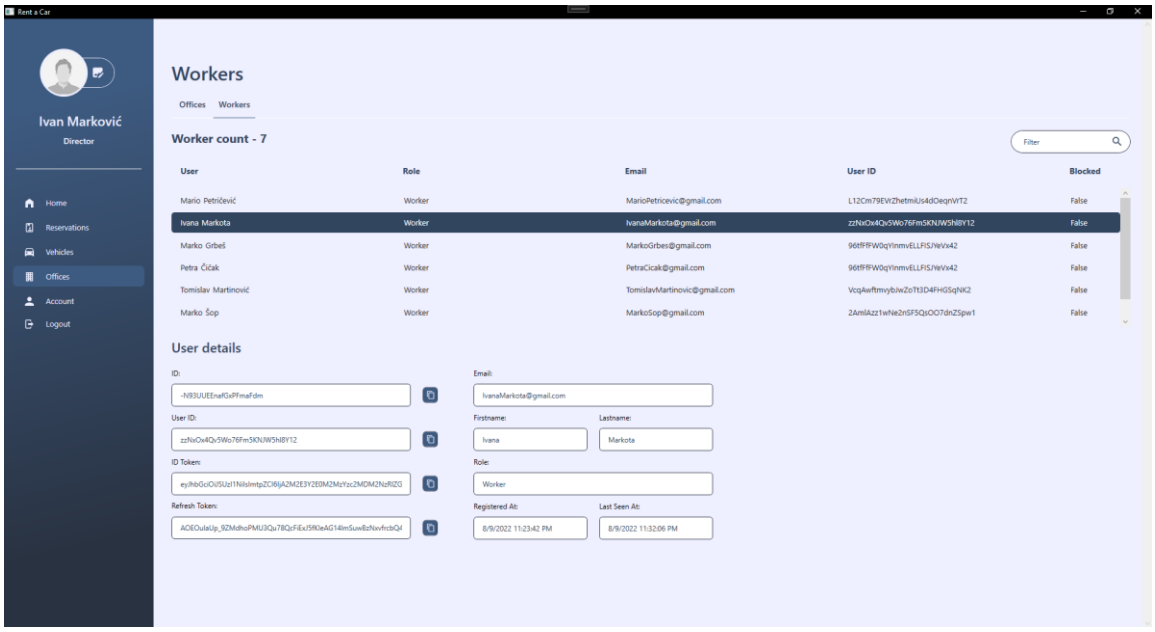
Slika 14 Administracija poslovnica - Uloga administrator



Slika 15 Dodavanje nove poslovnice - Uloga administrator



Slika 16 Pregled ureda - Uloga direktor



Slika 17 Pregled radnika poslovnice - Uloga direktor

Kao reprezentativni dio koda unutar ove cjeline u nastavku je prikazana implementacija kreiranja novog zapisa te brisanja postojećih. Brisanje zapisa iz tablice omogućeno je korištenjem crvenog gumba unutar same tablice te je ova akcija dostupna samo ulozi administratora. Kao što je spomenuto i za druge radnje, na ovaj gumb je povezana komanda koja poziva metodu za brisanje zapisa koja se nalazi u nastavku. Unutar ove metode koristi se ranije objašnjena klasa „FirebaseDatabaseHelper“ gdje pozivamo metodu „Delete“ i unutar parametra pružamo odabrani zapis za brisanje. Nakon brisanja se korisniku prikazuje statusna poruka te se poslovnice ponovno dohvaćaju kako bi se omogućilo automatsko ažuriranje.

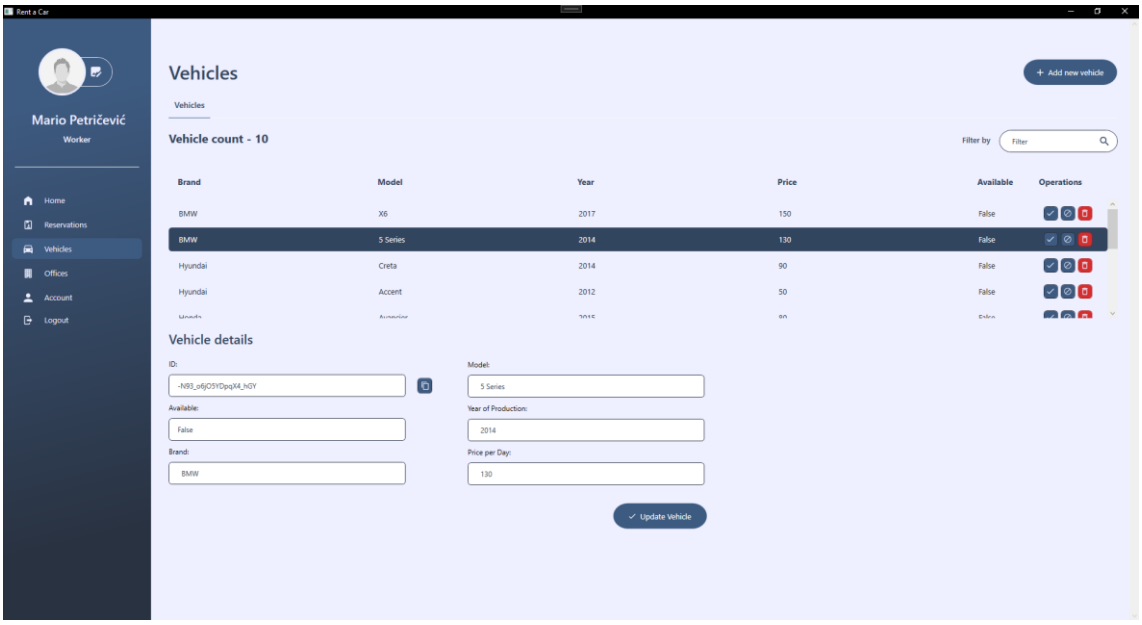
```
public async void DeleteOffice()
{
    await FirebaseDatabaseHelper.Delete(this.SelectedOffice);
    this.StatusMessage = this.SelectedOffice.Name + " successfully
deleted.";
    GetOffices();
}
```

Akcija dodavanja poslovnica je ipak nešto malo kompliciranija te zahtijeva cijeli novi pogled koji je vidljiv na slici 15. Prilikom kreiranja novog ureda potrebno je postaviti i odgovornu osobu koja će taj ured voditi stoga se u pogledu za dodavanje nalazi i tablica direktora gdje se odabirom jednog od zapisa automatski popunjavaju neki od elemenata forme za unos. Ukoliko direktor nije odabran akcija je onemogućena te se administratoru ispisuje statusna poruka u gornjem desnom kutu. Nakon odabira direktora potrebno je ručno popuniti ostale podatke o poslovnici te se omogućuje akcije kreiranja poslovnice. Nakon što korisnik pritisne gumb izvršava se metoda zaslužna za kreiranje poslovnice koja se nalazi u nastavku. Unutar ove metode prvo se provjerava popunjenost svih elemenata te se u slučaju valjanosti novog zapisa popunjavaju neki od inicijalnih atributa. Nakon što su inicijalni atributi dohvaćeni zapis se unosi u bazu podataka te se ažurira zapis unutar tablice poslovnica s novim direktorom.

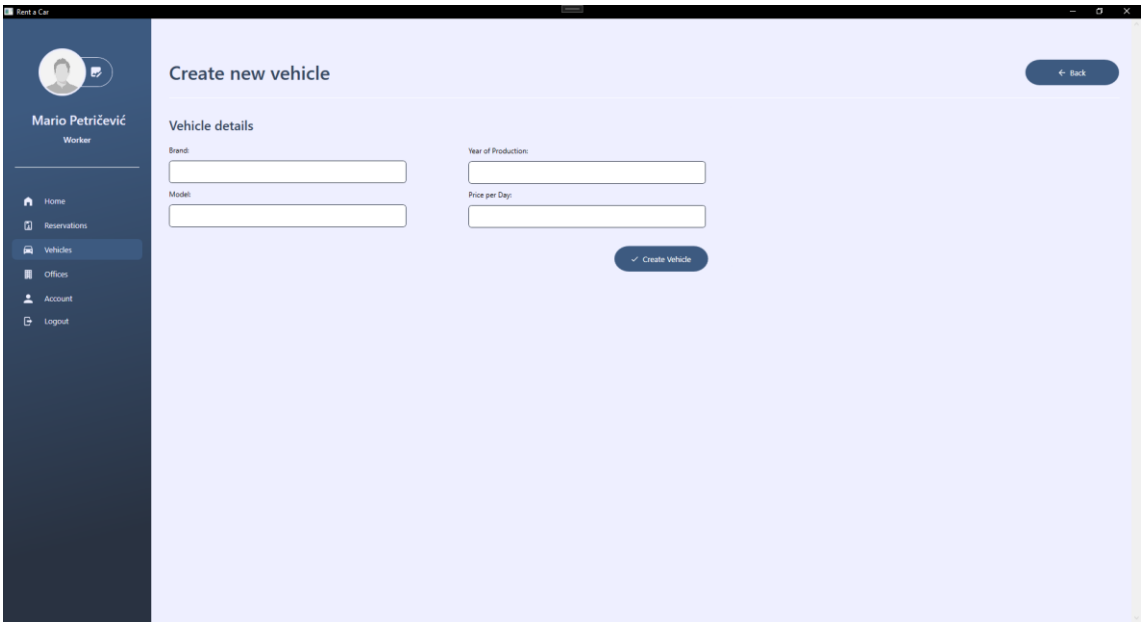
```
public async void CreateOffice()
{
    if (string.IsNullOrEmpty(App.Office.UserId) ||
string.IsNullOrEmpty(App.Office.Name) ||
string.IsNullOrEmpty(App.Office.Country) ||
string.IsNullOrEmpty(App.Office.City) ||
string.IsNullOrEmpty(App.Office.Address))
    {
        this.StatusMessage = "Please enter all values before creating new
office.";
    }
    else
    {
        App.Office.OpenedAt = DateTime.Now;
        App.Office.Opened = true;
        App.Office.UserId = App.Director.Id;
        await FirebaseDatabaseHelper.Insert(App.Office);
        var Offices = (await FirebaseDatabaseHelper.Read<Office>()).Where(o
=> o.UserId.Equals(App.Director.Id)).ToList();
        Office office = Offices.FirstOrDefault();
        App.Director.OfficeId = office.Id;
        await FirebaseDatabaseHelper.Update(App.Director);
        this.StatusMessage = "New office successfully created.";
        this.AddName = string.Empty;
        this.AddCountry = string.Empty;
        this.AddCity = string.Empty;
        this.AddAddress = string.Empty;
        AddVisibility = Visibility.Collapsed;
        UpdateVisibility = Visibility.Visible;
        GetOffices();
    }
}
```

8.3.6. Pogled administracije vozila

Pogled administracije vozila je ključan za ulogu radnika jer se ovdje obavlja zapisnik novih vozila unutar poslovnice. Kada ovaj pogled gledamo iz uloge radnika ne razlikuje se mnogo od prijašnjeg pogleda po korištenim elementima. Iz ovog razloga sljedeća cjelina prikazuje izgled i dizajn ove cjeline uz objašnjenje dijelova aplikacije koji do sada nisu spomenuti. Kada ovaj pogled gledamo iz uloge administratora razlika je u tome što su administratoru omogućene akcije za bilo koju od poslovnica. Na sljedećoj slici prikazan je pogled administracije vozila iz uloge radnika.



Slika 18 Pogled administracije vozila - Uloga radnik



Slika 19 Pogled dodavanja vozila - Uloga radnik

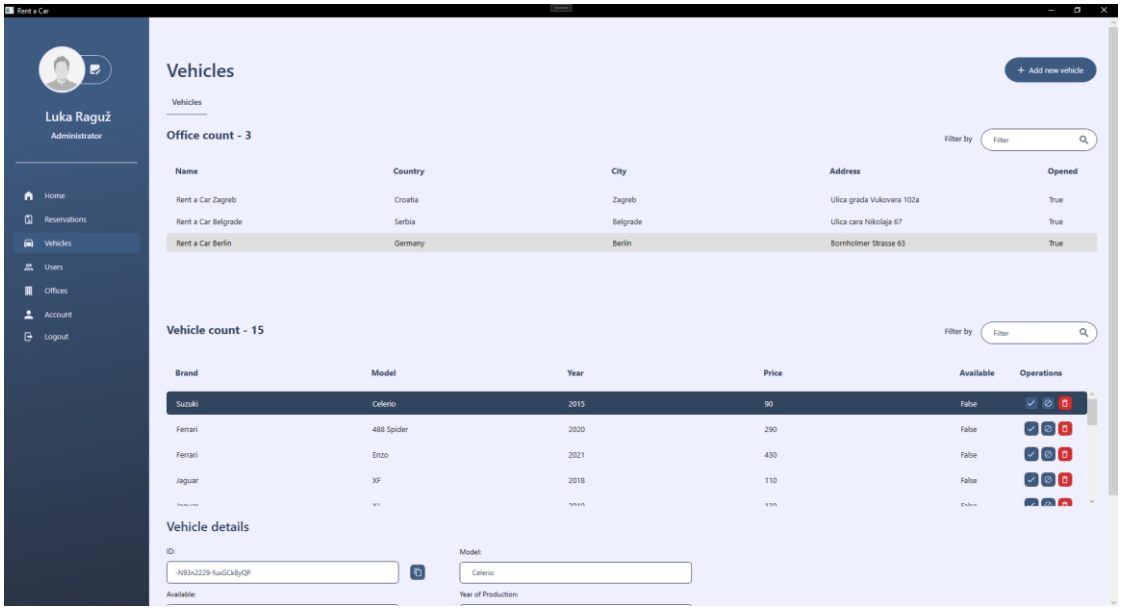
Kako još do sada nije bilo spomenuto, u ovoj cjelini obrađena je mogućnost odabira zapisa unutar podatkovne tablice (eng. datagrid) te klasa koju nasljeđuju svi „ViewModel-i“ unutar aplikacije. Ovaj pogled (kao i svaki drugi) nasljeđuje klasu „BaseVM“ unutar koje je definirano svojstvo „StatusMessage“ koje je zaslužno za ispis statusne poruke u slučaju potrebe za obavještenjem korisnika ili u slučaju greške. Ovo svojstvo, kao i svako drugo korišteno unutar aplikacije, ima korespondirajući „getter“ i „setter“ gdje se u „setter-u“ poziva metoda „OnPropertyChanged“ kojoj se kao parametar pruža ima svojstva te se na ovaj način detektira da li je došlo do promjene svojstva što se može iskoristiti za razne akcije. Kako bi se unutar „setter-a“ dopustilo korištenje „OnPropertyChanged“ metoda potrebno je napraviti i deklarirati „PropertyChanged“ varijablu koja je tipa podataka „PropertyChangedEventHandler“ te predstavlja događaj. Finalno, unutar koda možemo vidjeti ranije spomenuti „OnPropertyChanged“ metodu s njenom implementacijom. Sljedeći dio koda prikazuje opisanu „BaseVM“ klasu.

```
public class BaseVM : INotifyPropertyChanged
{
    private string statusMessage;
    public string StatusMessage
    {
        get { return statusMessage; }
        set
        {
            statusMessage = value;
            OnPropertyChanged("StatusMessage");
        }
    }

    public event PropertyChangedEventHandler PropertyChanged;

    protected void OnPropertyChanged(string propertyName)
    {
        PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
    }
}
```

Prethodni dio je spomenut iz razloga što se nadovezuje na sljedeći dio koda, a to je odabir zapisa unutar tablice te mogućnost rada nad istim zapisom. Kako bi se ova mogućnost ostvarila unutar „ViewModel-a“ je potrebno dodati „EventHandler“ koji će kontrolirati odabir nad tablicom. U ovom primjeru objašnjava se tablica vozila stoga sljedeći dio koda prikazuje deklaraciju „SelectedVehicleChanged“ varijable. Svojstvo koje je prisutno unutar klase naziva se „SelectedVehicle“ te se njegov „setter“ pokreće svaki puta kada se detektira promjena vrijednosti toga svojstva. Prilikom promjene ovaj objekt spremamo u lokalni „storage“ aplikacije te se omogućava njegovo daljnje korištenje. Sljedeći dio koda prikazuje implementaciju te se ovaj način koristi unutar svakog drugog „ViewModel-a“.

```
public event EventHandler SelectedVehicleChanged;

private Vehicle selectedVehicle;
public Vehicle SelectedVehicle
{
    get { return selectedVehicle; }
    set
    {
        selectedVehicle = value;
        App.Vehicle = selectedVehicle;
        OnPropertyChanged("SelectedVehicle");
        SelectedVehicleChanged?.Invoke(this, new EventArgs());
    }
}
```

U nastavku je prikaz istog pogleda iz uloge administratora koji ima dodatnu podatkovnu tablicu s poslovnicama. Odabirom jedne od poslovnica unutar tablice pokreće radnju filtriranja tablice vozila te se unutar te tablice prikazuju samo vozila iz odabrane poslovnice. Ova mogućnost je također usko povezana s prijašnje navedenim kodom stoga je ovdje opisana uz prikaz pogleda iz uloge administratora.



Slika 20 Pogled administracije vozila - Uloga administrator

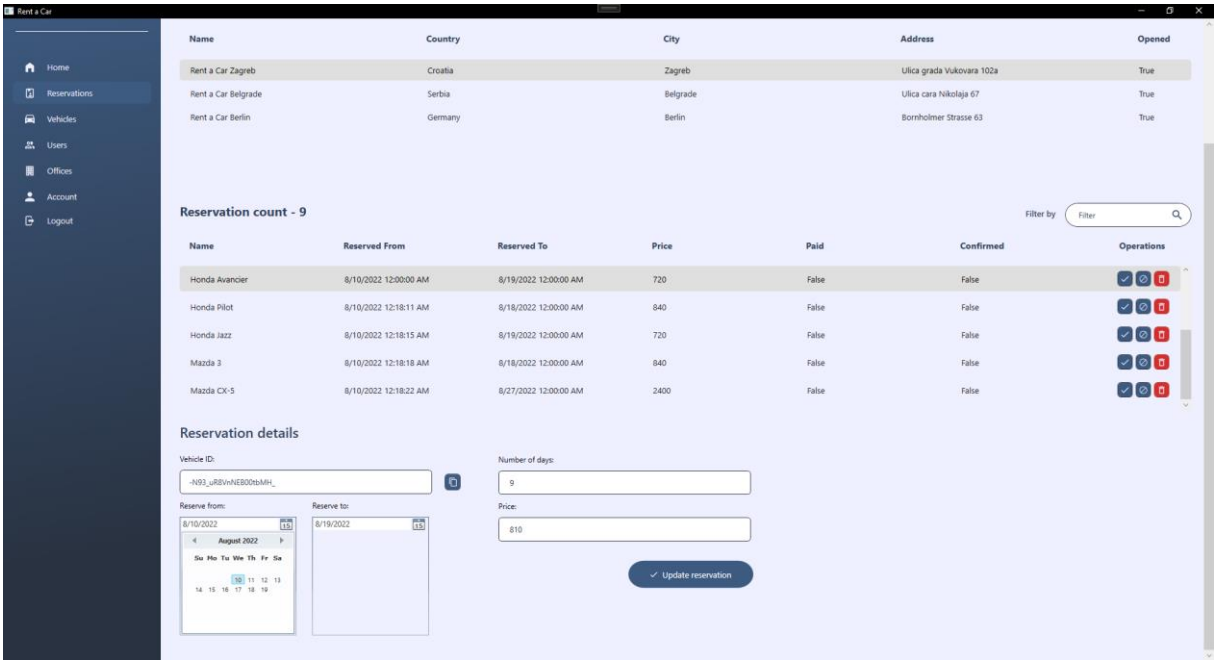
Unutar elementa tablice u jeziku oznaka postavljeno je svojstvo „SelectedOffice“ kao vrijednost za „SelectedItem“ te se svojstvo izmjenjuje prilikom svakog klika na drugi zapis tablice te se prilikom promjene zapisa okida i filtriranje same tablice vozila. Ovaj način čini korištenje ove stavke jednostavnom i preglednom gdje se istovremeno mogu vidjeti i najbitniji detalji poslovnice kao i njena vozila. Kao i u prethodnom primjeru sve je napravljeno na isti način osim što se u „setter-u“ ovog svojstva poziva metoda „GetVehiclesByOffice“ te se u parametru pruža odabrani zapis.

```
public event EventHandler SelectedOfficeChanged;
private Office selectedOffice;
public Office SelectedOffice
{
    get { return selectedOffice; }
    set
    {
        selectedOffice = value;
        App.Office = selectedOffice;
        GetVehiclesByOffice(selectedOffice.Id);
        OnPropertyChanged("SelectedOffice");
        SelectedOfficeChanged?.Invoke(this, new EventArgs());
    }
}


private async void GetVehiclesByOffice(string OfficeId)
{
    var vehicles = (await FirebaseDatabaseHelper.Read<Vehicle>()).Where(v
=>
        v.OfficeId.Equals(OfficeId)).ToList();

    this.Vehicles.Clear();
    foreach (var vehicle in vehicles)
    {
        this.Vehicles.Add(vehicle);
    }
}
```

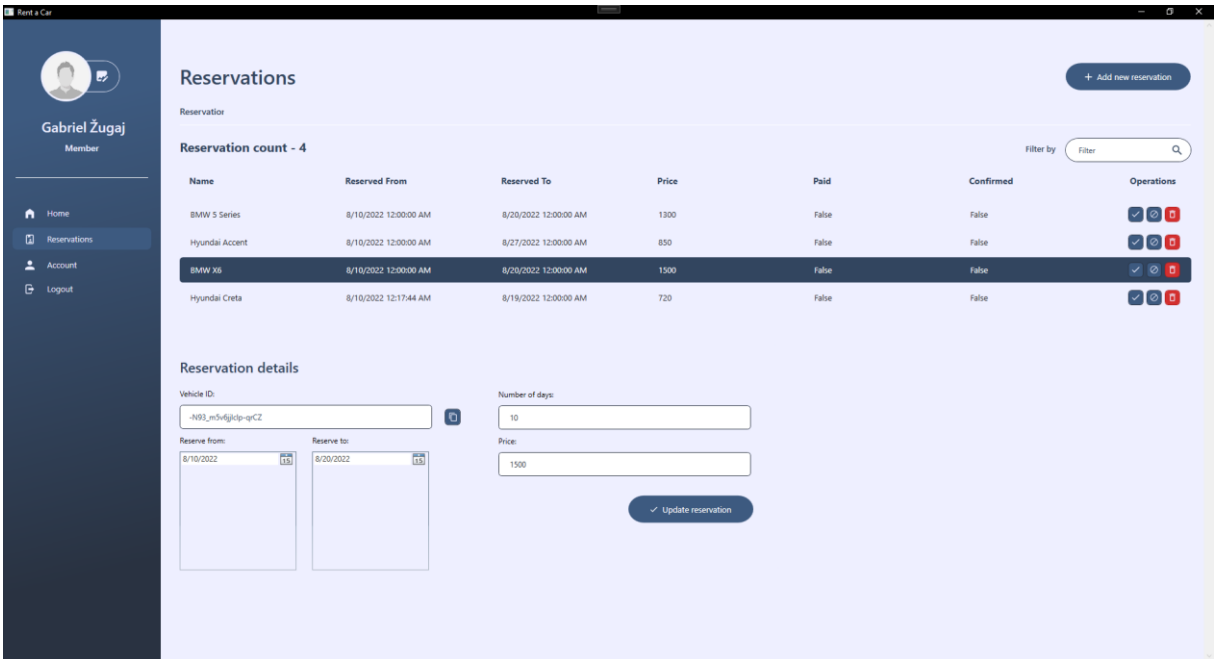
Metoda „GetVehiclesByOffice“ je jednostavna za shvaćanje te je slična metodi filtriranja kroz „TextBox“ dok se razlikuje u tome što se poziva u drugim slučajevima te unutar uvjeta postoji provjera da li vozilo pripada odabranom uredu.

8.3.7. Pogled rezervacija vozila

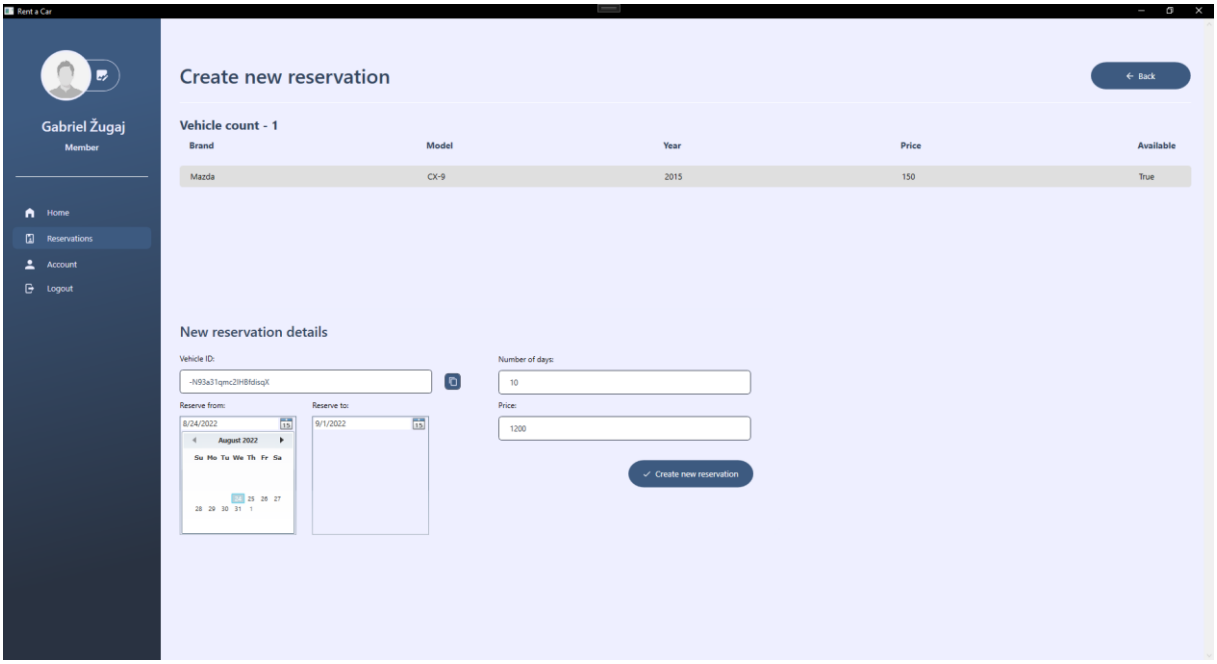
Zadnji pogled objašnjen unutar ovoga rada je pogled rezervacije vozila koji je glavni pogled korišten od strane korisnika uloge član. Ovaj pogled omogućuje različite mogućnosti iz različitih uloga. Kao administrator, korisnik je u mogućnosti odabrati poslovnicu te pregledati sve dosadašnje rezervacije unutar te poslovnice i vidjeti njihove detalje. Kada se član ulogira u aplikaciju i otvori ovaj pogled vidjet će samo rezervacije koje je on kreirao te prilikom izrade nove rezervacije vidi pregled vozila koja su dostupna unutar auto kuće u kojoj je zabilježen. Prilikom iznajme vozila ono se postavlja na nedostupno te zapis nije dostupan za odabir prilikom drugih rezervacija dok se dostupnost ne odobri od strane zaposlenika. Član prilikom pregleda svojih rezervacija može vidjeti izračun dana rezervacije zajedno sa ukupnom cijenom troška koja se računa kao broj dana puta cijena najma po danu. Nakon pregleda svoje rezervacije te uplate novca mora potvrditi plaćanje nakon čega zaposlenik ili direktor moraju provjeriti plaćanje te potvrditi sa svoje strane. Nakon plaćanja i provedene provjere korisnik može preuzeti svoje vozilo unutar auto kuće. Prilikom odabira raspona datuma rezervacije korisnik ne može odabrati dane koji su prošli te dane koji su veći od krajnjeg dana rezervacije što se može vidjeti na jednoj od sljedećih slika. Kao istaknuti dio mogućnosti unutar ove cjeline objašnjen je odabir raspona datuma zajedno s izračunom cijene najma, dok su ostali bitni dijelovi već barem jednom spomenuti u prijašnjim cjelinama. Sljedeće slike prikazuju pogled rezervacije iz uloge administratora, a zatim iz uloge člana.



Slika 21 Pregled rezervacija vozila - Uloga administrator



Slika 22 Pregled vlastitih rezervacija - Uloga člana



Slika 23 Dodavanje nove rezervacije - Uloga člana

U nastavku se nalazi dio koda koji je zaslužan za ažuriranje datuma koji se nalaze ispod podatkovne tablice te izračun cijene najma. Sljedeći dio koda okida se na promjenu zapisa unutar tablice rezervacije te za početak provjerava da li je rezervacija odabrana. Ukoliko je rezervacija odabrana postavljaju se početni datum najma te datum prekida najma. U nastavku se radi izračun vremena između zadnjeg i prvog dana najma te se tako dobiva

vremenski period predstavljen u danima. Kako je prilikom kreiranja vozila dodana cijena najma po danu za svako vozilo te smo na ovaj način dobili broj dana najma dalje se lako može izračunati ukupna cijena tako što se ove dvije vrijednosti pomnože. Sljedeći kod prikazuje ključni dio implementacije ove funkcionalnosti.

```
public event EventHandler SelectedReservationChanged;
private Reservation selectedReservation;
public Reservation SelectedReservation
{
    get { return selectedReservation; }
    set
    {
        selectedReservation = value;
        if (selectedReservation != null)
            App.Reservation = selectedReservation;
        if (selectedReservation != null)
        {
            this.UpdateReservedFrom = selectedReservation.ReservedFrom;
            this.UpdateReservedTo = selectedReservation.ReservedTo;
        }
        if (this.UpdateNumberOfDays != null && this.UpdatePrice != null)
        {
            TimeSpan timespan =
UpdateReservedFrom.Subtract(UpdateReservedTo);
            this.AddNumberOfDays = Math.Abs((int)timespan.TotalDays);
            if (App.Reservation != null)
            {
                CalculatePrice();
            }
        }
        OnPropertyChanged("SelectedReservation");
        SelectedReservationChanged?.Invoke(this, new EventArgs());
    }
}


public async void CalculatePrice()
{
    if (App.Vehicle != null)
        this.UpdatePrice = this.UpdateNumberOfDays *
App.Vehicle.PricePerDay;
}
```

Osim ovog dijela, unutar pogleda se koriste i mnogo drugih elemenata koji su barem u jednom dijelu rada već bili spomenuti stoga su objašnjeni neki ključni, dosad nespomenuti dijelovi.

8.3.8. Korisničko sučelje

U sljedećem poglavlju opisana je izrada korisničkog sučelja za jezikom oznaka „XAML“. Kako implementacija korisničkog sučelja sadrži jako veliku količinu koda isti nije priložen u cjelini unutar rada te su opisani korišteni elementi i metode s kojima se radilo prilikom izrade aplikacije. Unutar ove cjeline opisana je i stilizacija aplikacije koja je napravljena u odvojenoj datoteci čiji dijelovi sadrže ime stila te se vežu na određene elemente.

Kao glavni pogled može se istaknuti „MainWindow“ koje se dijeli na statički i dinamički dio. Ova velika cjelina predstavljena je kao element „ScrollViewer“ što omogućuje vertikalnu kretnju po aplikaciji ukoliko je kontent veći od visine vidljivog prozora. Kao glavni elementi korišten u svakom pogledu mogu se istaknuti „Border“, „Grid“ i „StackedPanel“ element koji služe za organizaciju ostalih elemenata. U većini slučajeva „Border“ se koristi kao glavni element unutar kojeg se nalazi manji elementi za organizaciju, a to su „Grid“ i „StackedPanel“. U nastavku je naveden primjer korištenja ovih elemenata zajedno jedan s drugim.

```
<ScrollViewer>
  <Border Background="{StaticResource GrayColorBrush}">
    <Grid>
      <Grid.ColumnDefinitions>
        <ColumnDefinition Width="250"/>
        <ColumnDefinition Width="*"/>
      </Grid.ColumnDefinitions>

      <ContentControl Grid.Column="1" Content="{Binding
SelectedViewModel}"/>

      <Grid Grid.Column="0">
        … [Ostali elementi] …
        <DockPanel>
          <StackPanel DockPanel.Dock="Top">
            <uc:UserView User="{Binding}" Width="250" Height="250"
VerticalAlignment="Top"/>

            <Separator Height="1" Background="{StaticResource
WhiteColorBrush}" Margin="20 0 20 35"/>

            <Button x:Name="homeButton" Style="{StaticResource menuButton}"
Command="{Binding UpdateViewCommand}" CommandParameter="Home"
Click="Button_Click">
              <StackPanel Orientation="Horizontal">
                <Icon:PackIconMaterial Kind="HomeVariant"
Style="{StaticResource menuButtonIcon}"/>
                <TextBlock Text="Home"/>
              </StackPanel>
            </Button>
          … [Ostali elementi] …
          </StackPanel>
        </DockPanel>
      </Grid>
    </Grid>
  </Border>
</ScrollViewer>
```

Ovaj veoma mali isječak koda iz aplikacije perfektno prikazuje ugniježđeno korištenje elemenata jedan unutar drugog kao i što sadrži povezivanje na svojstva iz „ViewModel-a“, korištenje događaja, korištenje „NuGet“ paketa za ikone, korištenje ranije spomenutih komandi te sadrži neke elemente stilizacije stoga je prikazan kao reprezentativni dio. Unutar koda možemo vidjeti već spomenuti „ScrollViewer“ element koji predstavlja roditelja ostalih elemenata. Prvo i jedino dijete unutar ovog elementa je „Border“ koji je ključni element i za svaki drugi pogled. Element „Border“ nam nudi mogućnost postavljanja boje pozadine koja će se prikazivati unutar aplikacije te je na ovom primjeru kao boja postavljen statički resurs koji sadrži heksadekadski zapis za određivanje boja unutar druge klase te se ovdje samo poziva. Sljedeći ugniježđeni element je „Grid“ koji nam omogućava raspored elemenata po pogledu u obliku tablice čija se pravila određuju unutar „ColumnDefinitions“ i „RowDefinitions“ na početku korištenja elementa. U ovom primjeru raspored je podijeljen na lijevi i desni dio gdje lijevi dio zauzima 250 piksela prozora dok desni dio zauzima sav preostali ostatak. „ContentControl“ je element koji popunjava dinamički dio aplikacije (desni dio) te za svoj kontent ima povezan „SelectedViewModel“ koji određuje prikaz cjeline te se mijenja ovisno o odabranom elementu iz navigacije. Navigacija je također napravljena unutar elementa „Grid“ gdje se na vrhu nalazi korisnička kontrola (eng. *User Control*) koja prikazuje podatke o korisniku te njegovu fotografiju. U nastavku se nalazi roditelj „DockPanel“ zajedno sa svojim djetetom „StackPanel“. Ovaj element omogućava vertikalno postavljanje elemenata jedan na drugi od čega dolazi i njegov naziv. Na ovaj način je napravljena navigacija gdje je u priloženom kodu prikazano kako je napravljen jedan gumb. Ovaj gumb unutar elementa ima određeno ime kako bi se mogao koristit unutar parcijalne klase, primijenjen stil koji se vuče iz klase u kojemu su određeni stilovi elemenata, na sebe veže ranije objašnjenu komandu „UpdateViewModel“ koja omogućava pokretanje metode iz „ViewModel-a“ te kao parametar šalje ime cjeline kako bi se znalo koji „ViewModel“ se treba inicijalizirati. Na kraju možemo vidjeti element „Icon“ te „TextBox“ koji u kombinaciji daju jedan izgled gumba na navigaciji te predstavljaju ikonu i tekst uz ikonu. Ikone koje se mogu vidjeti unutar navigacije i gumbova omogućene su korištenjem „NuGet“ paketa za ikone te su ubačene unutar pogleda na samom vrhu svakog „XAML“ dokumenta.

Na ranijem dijagramu klasa se mogu vidjeti imena svih pogleda te raniji prilog slika pokazuje raspored svih elemenata unutar aplikacije. Kao jedan od najviše korištenijih elemenata objašnjen je element „DataGrid“ koji se koristi skoro u svakom pogledu te je njegova implementacija uz naglasak na ključne dijelove priložena u nastavku. Ovaj element omogućuje prikaz zapisa iz baze podataka na prilagođen način te korisniku omogućuje interakciju s istim zapisima. Pod interakciju se podrazumijevaju operacije kreiranja, ažuriranja, brisanja i čitanja zapisa prilikom njihovog odabira te sporedne radnje kao što su otvaranje ureda, blokiranje korisnika, plaćanje i potvrđivanje računa omogućene unutar posljednjeg stupca tabele.

```
<DataGrid Visibility="{Binding UsersVisibility}" HorizontalAlignment="Left"
Height="280" ItemsSource="{Binding Users, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged, Source={StaticResource vm}}"
SelectedItem="{Binding SelectedUser, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged, Source={StaticResource vm}}"
RowStyle="{DynamicResource DataGridRowStyle1}"
ColumnHeaderStyle="{DynamicResource DataGridColumnHeaderStyle1}"
CellStyle="{DynamicResource DataGridCellStyle1}" x:Name="usersDataGrid"
Style="{DynamicResource DataGridStyle1}">
    <DataGrid.Columns>
        <DataGridTemplateColumn Header="User" IsReadOnly="True" Width="*">
            <DataGridTemplateColumn.CellTemplate>
                <DataTemplate>
                    <StackPanel Orientation="Horizontal">
                        <TextBlock Text="{Binding Firstname}"
VerticalAlignment="Center" Margin="0 0 5 0"/>
                        <TextBlock Text="{Binding Lastname}"
VerticalAlignment="Center"/>
                    </StackPanel>
                </DataTemplate>
            </DataGridTemplateColumn.CellTemplate>
        </DataGridTemplateColumn>

        <DataGridTextColumn Header="Role" Binding="{Binding Role}"
IsReadOnly="True" MinWidth="120" Width="*"/>

        <DataGridTextColumn Header="Email" Binding="{Binding Email}"
IsReadOnly="True" MinWidth="220" Width="*"/>

        <DataGridTextColumn Header="User ID" Binding="{Binding UserId}"
IsReadOnly="True" MinWidth="260" Width="*"/>

        <DataGridTextColumn Header="Blocked" Binding="{Binding Blocked}"
IsReadOnly="True" Width="100"/>

        <DataGridTemplateColumn Header=" Operations" IsReadOnly="True"
Width="115">
            <DataGridTemplateColumn.CellTemplate>
                <DataTemplate>
                    <StackPanel HorizontalAlignment="Left"
Orientation="Horizontal">
                        <Button Style="{StaticResource gridBlueButton}"
Command="{Binding UnblockUserCommand, Source={StaticResource vm}}">
                            <Icon:PackIconMaterial Kind="Check"
Style="{StaticResource gridButtonIcon}"/>
                        </Button>
                        <Button Margin="5 0 0 0" Style="{StaticResource
gridBlueButton}" Command="{Binding BlockUserCommand, Source={StaticResource
vm}}">
                            <Icon:PackIconMaterial Kind="BlockHelper"
Style="{StaticResource gridButtonIcon}"/>
                        </Button>
                        <Button Margin="5 0 0 0" Style="{StaticResource
gridBlueButton}" Command="{Binding ChangeRoleCommand,
Source={StaticResource vm}}">
                            <Icon:PackIconMaterial Kind="SwapVertical"
Style="{StaticResource gridButtonIcon}"/>
                        </Button>
                    </StackPanel>
                </DataTemplate>
            </DataGridTemplateColumn.CellTemplate>
```

```
        </DataGridTemplateColumn>
    </DataGrid.Columns>
</DataGrid>
```

Unutar ovog elementa možemo vidjeti razne stvari od kojih su neke već spomenute ranije prilikom opisivanja „ViewModel-a“. Za početak, kod prikazuje implementaciju podatkovne tablice koja prikazuje korisnike te je dostupna administratoru sustava. Kako se unutar pogleda za administraciju korisnika nalazi više pod pogleda unutar tablice možemo vidjeti povezivanje sa svojstvom „UsersVisibility“ koje ograničava da je ova tablica vidljiva samo u slučaju kada se korisnik nalazi na tabu „Users“. Još neke od bitnih postavki su „SelectedItem“ na koje je povezano svojstvo „SelectedUser“ iz „ViewModel-a“ te izvor podataka na što se veže kolekcija korisnika. Unutar elementa „DataGrid“ možemo vidjeti više elemenata „DataGridTemplateColumn“ koji predstavljaju stupac unutar tablice te se unutar svakog stupca za ćeliju vežu pojedini atributi objekta koji se prikazuje u tablici. Ove ćelije se mogu modificirati raznim stilskim elementima te je u ovom primjeru postavljena širina elementa te njegova minimalna širina kako bi se onemogućilo preveliko sužavanje ćelije prilikom smanjivanja prozora aplikacije. U zadnjem stupcu „DataGrid-a“ nalaze se operacije koje su prikazane s gumbovima te se na svaki gumb vežu komandne klase (eng. *Command Class*, klase koje nasljeđuju sučelje *ICommand*) koje pokreću određene metode iz „ViewModel-a“. Sljedeća slika prikazuje izgled tablice definirane ranijim prilogom koda.

Slika 24 Prikaz podatkovne tablice (eng. DataGrid)

9. „Setup“ aplikacije

Kao zadnje poglavlje objašnjena je izrada „setup-a“ aplikacije što se podrazumijeva na instalaciju aplikacije odnosno izradu „.exe“ datoteke. Mogućnost instalacije se radi kako bi se aplikacija dalje mogla distribuirati korisnicima. Kako bi se omogućila instalacija aplikacije potrebno je skinuti „Microsoft Visual Studio Installer Projects“ te se odabire verzija ovisno o verziji okruženja koja se koristi. Nakon instalacije ekstenzije potrebno je kreirati novi projekt tipa „Setup Project“ nakon čega se dobiva pogled gdje se unutar „Application folder-a“ može napraviti publiciranje aplikacije za koju želimo napraviti instalaciju. Unutar iste datoteke potrebno je napraviti prečac aplikacije preko koje će se aplikacija pokretati uz što se preporučuje i postavljanje ikone. Finalna stvar koju je potrebno napraviti je postaviti prečac unutar „User's Desktop“ datoteke te napraviti gradnju (eng. *Build*) aplikacije. Unutar projekta se kreira datoteka „Setup“ gdje je moguće pokrenuti aplikaciju čija se ikona pojavljuje na pozadini. [17]

Na ovaj način omogućeno je instaliranje aplikacije za bilo kojeg korisnika te njeno korištenje, a ukoliko je to u planu aplikaciju je bitno napraviti dostupnom svim korisnicima te publicirati istu na određenoj platformi.

10. Zaključak

Kako u suvremeno doba manje više svi koriste računala ideja za aplikaciju proizašla je iz istog razloga kao i želje za napretkom u ovom području IT industrije. Cilj ovog rada bio je izraditi aplikaciju koja na određeni način može olakšati rad kako zaposlenicima, tako i njezinim korisnicima te je isti ispunjen. Odabir korištene tehnologije proizašao je iz dosadašnjeg iskustva na kolegijima na fakultetu te samom popularnošću jezika uz cilj unaprijeđenja znanja iz tog područja. Osim što sam unaprijedio dosadašnja znanja, korištenjem navedenih alata i metoda susreo sam se sa novim tehnologijama i iskustvima koje će mi pomoći u daljnjem razvoju. Nakon izrade praktičnog rješenja puno toga je sjelo na svoje mjesto te smatram da bi svaki idući proizvod bio za nijansu bolji od prethodnoga. Kroz ovaj rad naučeni su osnovni koncepti i pokriveni su neki elementi koje bi, manje više, svaka aplikacija trebala imati. Iako je dosta pokriveno unutar rada, okruženje „Microsoft Visual Studio“ nam pruža puno drugih mogućnosti te se uvijek može otkriti nešto novo.

Popis literature

[1] *„Visual Studio: IDE and Code Editor for Software Developers and Teams“* [Na internetu] Dostupno: https://visualstudio.microsoft.com/ Pristupano: 11.08.2022.

[2] *„Visual Studio Documentation“* [Na internetu] Dostupno: https://docs.microsoft.com/ Pristupano: 11.08.2022.

[3] *„.NET Framework Documentation“* [Na internetu] Dostupno: https://dotnet.microsoft.com/ Pristupano: 11.08.2022.

[4] *„What is WPF?“* [Na internetu] Dostupno: https://www.wpf-tutorial.com/ Pristupano: 11.08.2022.

[5] *„What is C#?“* [Na internetu] Dostupno: https://www.techtarget.com/whatis/definition/C-Sharp Pristupano: 12.08.2022.

[6] *„NuGet packages gallery“* Dostupno: https://www.nuget.org/packages/ Pristupano: 14.08.2022.

[7] *„Firebase documentation“* Dostupno: https://firebase.google.com/docs Pristupano: 14.08.2022.

[8] *„Firebase introduction“* Dostupno: https://www.geeksforgeeks.org/firebase-introduction/ Pristupano: 14.08.2022.

[9] *„Inkscape“* Dostupno: https://inkscape.org/ Pristupano: 14.08.2022.

[10] *„Requirement specifications“* Dostupno: http://www.requirements-specification.com/ Pristupano: 16.08.2022.

[11] *„Non-functional requiremets“* Dostupno: https://www.altexsoft.com/blog/non-functional-requirements/ Pristupano:18.08.2022.

[12] *„MVVM Arhitecture“* Dostupno: https://www.educba.com/mvvm-architecture/ Pristupano: 18.08.2022.

[13] „What is class diagram?“ Dostupno: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/ Pristupano: 20.08.2022.

[14] „ER Diagram“ Dostupno: https://www.lucidchart.com/pages/er-diagrams Pristupano: 20.08.2022.

[15] *„Functional Fun: WPF PasswordBox and Databinding“* Dostupno: http://blog.functionalfun.net/2008/06/wpf-passwordbox-and-data-binding.html Pristupano: 21.08.2022.

[16] „How to Create Setup .exe in Visual Studio 2022 Step by Step“ Dostupno: https://www.youtube.com/watch?v=bJw0eGfM1ZU Pristupano: 25.08.2022.

Popis literature za učenje

[1] „Windows Presentation Foundation Masterclass“ Dostupno: https://www.udemy.com/course/windows-presentation-foundation-masterclass/ Pristupano: 03.06.2022.

Popis korištenih alata

[1] *Razvojno okruženje (eng. Integrated Development Environment, skraćeno IDE): Microsoft Visual Studio* Dostupno: https://visualstudio.microsoft.com/ Pristupano: 03.06.2022.

[2] *Platforma Firebase* Dostupno: https://firebase.google.com/ Pristupano: 05.06.2022.

[3] *Program za izradu ilustracija: Inkscape*, Dostupno: https://inkscape.org/ Pristupano: 11.07.2022.

[4] *Alat za izradu UML dijagrama: Draw.io*, Dostupno: https://app.diagrams.net/ Pristupano: 20.08.2022.

[5] *Alat za izradu dijagrama klasa unutar okruženja Microsoft Visual Studija: Class Designer* Dostupno: https://docs.microsoft.com/en-us/visualstudio/ide/class-designer/designing-and-viewing-classes-and-types?view=vs-2022 Pristupano: 20.08.2022.

Popis slika

Slika 1: *Komponente „MVVM“ arhitekture sustava* Dostupno: https://cdn.educba.com/academy/wp-content/uploads/2020/04/MVVM-Architecture-2.jpg.webp Pristupano 18.08.2022.

Popis tablica

Prilozi

Izvorni kod: https://github.com/LukaRaguz/ThesisApp (Potreban pristup)

Dijagram klasa (Priložena slika na sustavu „Foi Radovi“)