# HACCAPP

**Vidaček, Mihovil**

**VISOKO UČILIŠTE ALGEBRA**

DIPLOMSKI RAD

# HACCAPP

Mihovil Vidaček

Zagreb, Svibanj 2018.

# Predgovor

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

# Sažetak

U radu se objašnjava problematika HACCP zahtjeva u ugostiteljskim objektima i opisuje se način na koji se generiranje HACCP dokumenata može ubrzati korištenjem HACCAPP mobilnom aplikacijom.

U sklopu rada je implementiran sustav koji olakšava vođenje i ispunu tih dokumenata. Sustav se sastoji od mobilne i web aplikacije i biti će dostupan širem krugu korisnika preko Google Play trgovine.

Mobilna aplikacija razvijena je za Android mobilne uređaje i povezana je s poslužiteljskom web API-em koji je razvijen u .NET okolini. Mobilna aplikacija razvijena je primjenom MVP dizajnerskog obrasca i koristi Googleove usluge u oblaku i Retrofit okvir za komunikaciju s poslužiteljem. Poslužitelj je razvijen u ASP.NET razvojnoj okolini i koristi Entity Framework i SQL bazu podataka.

Rad je podijeljen na dva djela, teoretski i praktični dio. U teoretskom djelu, opisane su tehnologije korištene pri izradi rada. U praktičnom djelu opisan je način implementacije i korištenja tih tehnologija.

**Ključne riječi**: HACCP, HACCAPP, SQL, .NET, entity framework, Android, MVP, GCM.

# Summary

This paper explains the issue of keeping in compliance with HACCP requests while running a catering facilites, and also how to accelerate the generation of HACCP documents by using HACCAPP mobile app.

Through this paper, a system is implemented that enables facilitation and filling of these documents. System consists of mobile and web application, and it will be made available to the wide circle of users through Google Play web store.

Mobile app is developed for Android mobile devices and it is linked to web API that is developed using .NET framework. Mobile app was developed by following MVP design patterns and it makes use of Google Cloud Messaging and Retrofit for communicating with server. Server is developed using ASP.NET framework and it uses Entity Framework and SQL server database.

Paper is divided into two parts, a theoretical and practical one. In the theoretical part, the technologies used are described. In the practical part, the way these technologies are implemented and used is described.

**Key words:** HACCP, HACCAPP, SQL, .NET, entity framework, Android, MVP, GCM.

# Sadržaj

# 1. Uvod

U današnje vrijeme gotovo svakodnevno se pojavljuju nove tehnologije koje ljudima omogućavaju lakše i efikasnije obavljanje posla. Stalni razvoj tehnologija omogućio je i programerima da brže i efikasnije no ikada razvijaju nove sustave i programe, primjenjive u svim područjima ljudskog djelovanja. Osim poslovnih prednosti, tehnologije sve više utječu i na kvalitetu života općenito.

Kako bi se dio iste kvalitete prenio u područja u kojima do sada nije jače zaživjela, razvijena je HACCAPP aplikacija. Ona omogućuje lakše i brže upravljanje HACCP dokumentima koje je obavezan voditi svaki prehrambeni objekt. Problem s vođenjem tih dokumenata je u tome što zaposlenici moraju trošiti dodatno vrijeme i resurse na izradu i ispunjavanje identičnih dokumenata svaki dan. Rješenje tog problema nalazi se u HACCAPP aplikaciji koja omogućuje zaposlenicima da troše manje vremena ispunjavajući te dokumente kroz aplikacijske forme. Aplikacija omogućuje i slanje tako generiranih dokumenata nadležnom voditelju objekta.

Rad je podijeljen na dvije glavne cjeline. Prvi dio rada daje pregled i daje osnovne informacije o korištenim tehnologijama u radu. Drugi dio rada opisuje samu aplikaciju i objašnjava kako navedene tehnologije tvore cjelinu. Zadnji dio rada je zaključak u kojem su dani osvrt na cijeli rad, kao i moguće buduće primjene i proširenja aplikacije.

# 2. HACCP sustav

## 2.1. Definicija HACCP sustava

HACCP[1] je sustav kontrole kvalitete hrane koji su obavezani slijediti svi prehrambeni objekti koji skladište ili poslužuju hranu. HACCP sustav sastoji se od niza dokumenata koji reguliraju načine postupanja s hranom i okolišem u kojem se hrana skladišti. Ispunjavanjem uvjeta koje HACCP sustav nalaže, osigurano je ispravno i kvalitetno skladištenje hrane.

Sustav postoji od 1959. godine i osmišljen je za potrebe NASA-e kako bi se osigurala kvalitetna hrana za astronaute. Od 1.1.2009. godine, uvedeno je obavezno korištenje HACCP sustava za sve: ugostiteljske objekte, slastičarnice i pekarnice, mesnice, trgovine prehranom i objekte društvene prehrane (menze, vrtići, domovi, škole...) u Hrvatskoj. Jedan od uvjeta za zadovoljavanje kriterija HACCP sustava je da svaka karika u lancu nabave hrane mora implementirati HACCP sustav. Ugostiteljski objekt koji se pridržava HACCP normi, ali nabavlja hranu od distributera koji se ne pridržava istih normi, ne ispunjava uvjete kontrole kvalitete hrane.

## 2.2. Dokumenti HACCP sustava

HACCP se sastoji od ukupno dvadeset dokumenata[2] koji pokrivaju dobavljanje, pohranu, distribuciju i pripremanje hrane i definiraju obavezna pravila kojih se trgovac mora pridržavati kako bi osigurao kvalitetu hrane. Od njih dvadeset, dvanaest dokumenata su obavezni i moraju ih popunjavati svi koji koriste HACCP sustav kontrole hrane. Preostalih osam dokumenata popunjava se ovisno o tehnološkim postupcima koji se primjenjuju u objektu. U mobilnoj aplikaciji HACCAPP bit će pokriveni svi dokumenti koji se ispunjavaju na dnevnoj i tjednoj razini, s obzirom na to da je cilj aplikacije skraćivanje vremena potrebnog za svakodnevno ispunjavanje tih dokumenata. Svi dokumenti imaju predefinirane vrijednosti koje govore u kakvom stanju bi se hrana trebala čuvati. Definirane su ciljne vrijednosti kojima bi trebalo težiti, kao i kritične granice. Ako neka od vrijednosti prelazi kritične vrijednosti obavezno je provesti korektivne mjere opisane na samom dokumentu. Slijedi pojašnjenje dokumenata koje je moguće popunjavati HACCAPP aplikacijom.

**Evidencija temperature u rashladnim uređajima**

Dokument za bilježenje stanja u rashladnim uređajima. Ispunjavanjem ovog dokumenta potvrđuje se ispravnost rashladnih uređaja. Radi ispravne evidencije, stanje u rashladnim

[1] MORTIMORE, C.WALLACE; HACCP Apractical approach; Springer (2013)
[2] U HACCP dokumentaciji navodi se koji se podaci moraju evidentirati i kada, ali ne definira se izgled samih dokumenata. Dok god su svi propisani podaci prikazani, izgled može varirati.

uređajima potrebno je provjeriti dva puta dnevno. Razne vrste hrane imaju različite ciljne temperature i kritične granice temperature kojih se mora držati.

Korektivna mjera je pohrana hrane u drugi rashladni uređaj ako unutar jednog sata nije moguće vratiti temperaturu na prihvatljivu razinu.

**Evidencija temperature u uređajima za skladištenje smrznute hrane**

Dokument za bilježenje temperature u zamrzivačima. Ispunjava se po istom principu kao i dokument za evidenciju temperature u rashladnim uređajima i ima iste korektivne mjere, ali s drugačijim iznosima za dozvoljenu temperaturu.

**Evidencija hlađenja hrane**

Dokument za bilježenje hlađenja termički obrađene hrane. Ovim se dokumentom provjeravaju uvjeti prostorije u kojoj se termički obrađena hrana hladi. Dokumentom su propisana pravila kojih se treba držati kako bi se hrana ohladila bez kvarenja. Mjerenje temperature hrane se provodi dva puta, inicijalno i nakon šest sati. Korektivne mjere, koje mora dozvoliti šef kuhinje, uključuju snižavanje temperature hlađenja ili smanjivanje sloja hlađene namirnice.

**Evidencija čuvanja hrane na toplom**

Dokument koji definira koliko dugo hrana smije stajati na toplom stolu i način provjere temperature. Korektivne mjere definiraju koliko se hrana puta smije podgrijavati i koje su kritične vrijednosti temperature hrane.

**Evidencija temperature hladnog stola**

Dokument koji definira koliko dugo i u kakvim uvjetima hrana smije stajati na hladnom stolu. Korektivna mjera je uklanjanje hrane koja je stajala na hladnom stolu čija je temperatura viša od kritične dozvoljene vrijednosti.

**Evidencija kontrola ulja u fritezama**

Dokument koji propisuje kakvo se ulje može koristiti u fritezama i određuje kritične vrijednosti temperature tog ulja prije termičke obrade hrane. Korektivna mjera je zamjena ulja i ponovljena provjera.

# 3. HACCAPP Android aplikacija

HACCAPP mobilna aplikacija omogućuje brže ispunjavanje HACCP dokumenata, pa tako i efikasnije korištenje vremena zaposlenika koji te dokumente inače popunjavaju ručno. Sustav se sastoji od Android mobilne aplikacije i web servisa. Putem mobilne aplikacije, moguće je ispunjavati prethodno definirane dokumente HACCP sustava i njihovo prosljeđivanje radi daljnje evidencije. U ovom će poglavlju biti objašnjene tehnologije korištene za izradu same aplikacije, dok će praktični dio biti objašnjen u poglavlju "klijentska strana sustava**".**

## 3.1. Arhitektura i pregled mogućnosti

Kod razvoja Android mobilnih aplikacija važno je razumjeti životni ciklus i komponente aplikacija[3]. To predstavlja i prvi izazov kod razvoja aplikacija za mobilne uređaje. Tijekom razvoja se potrebno prilagođavati životnom ciklusu aplikacije koji je propisan Android operacijskim sustavom, što u nekim slučajevima ograničava programera.

### Životni ciklus Android aplikacija

Životni ciklus aplikacije[4] (Slika 1), odnosno dretve aplikacije ograničen je resursima koje sustav raspodjeljuje i osigurava da uređaj neometano radi. Značajka Android aplikacija je da sama aplikacija ne kontrolira glavnu dretvu na kojoj se pokreće. Upravljanje dretvama je odgovornost Android OS-a, i on omogućuje aplikaciji korištenje dretve sve dok postoje aktivne komponente aplikacije. Kako ne bi došlo do nedostatka resursa, Android prioritizira dodjeljivanje resursa važnijim dretvama i moguće je da će sustav ugasiti aplikaciju kako bi uređaj nastavio stabilno raditi.

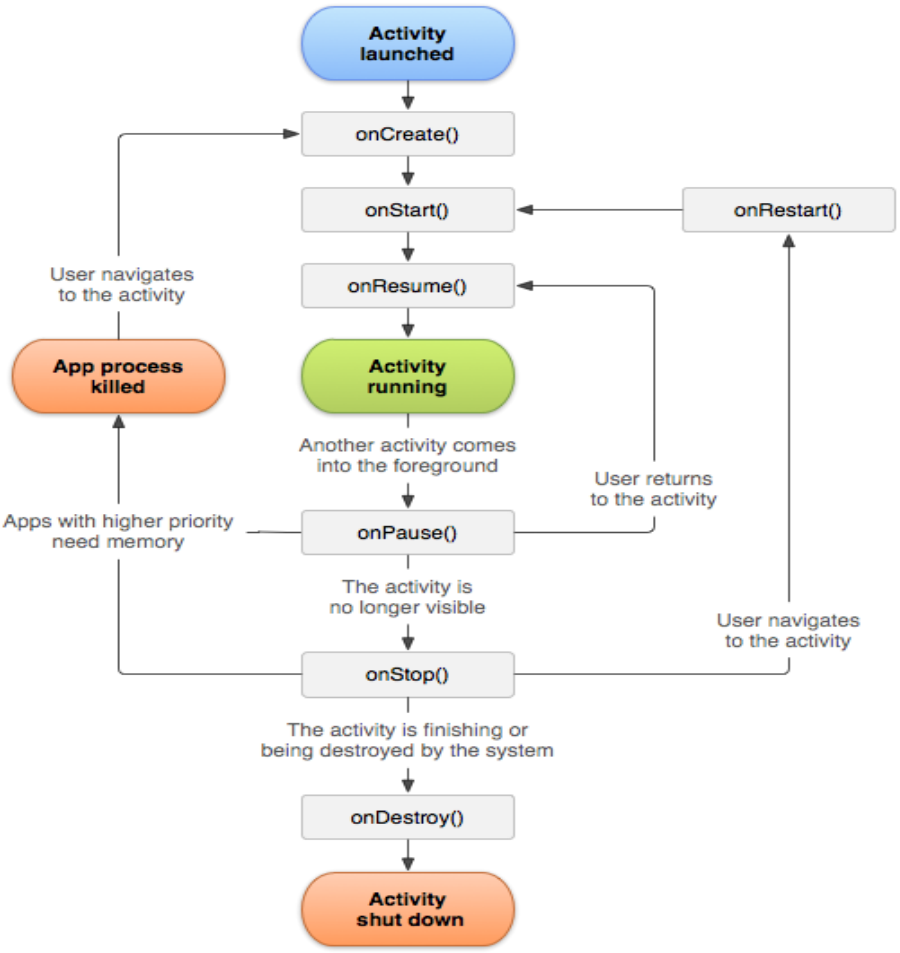
Ovo predstavlja važan dio razvoja Android aplikacija jer svaka aplikacija koja pristupa mreži ili vanjskim resursima mora tu funkcionalnost pokretati u zasebnim dretvama. Android OS neće dozvoliti da glavna dretva aplikacije ne radi dulje od pet sekundi. Nakon toga, ona se smatra troškom resursa i ukida se. S obzirom na to da dretve koje dohvaćaju vanjske resurse često moraju čekati odgovor s mreže dulje od pet sekundi, cijela ta funkcionalnost mora biti pokrenuta u zasebnoj dretvi. Zasebna dretva je zadužena za dohvat resursa, dok je glavna dretva zadužena za neometan rad aplikacije. U HACCAPP aplikaciji, i svim aplikacijama gdje dolazi do dohvata resursa s mreže, za dohvat se koristi posebna dretva.

Kako bi se razvojnim inženjerima olakšala komunikacija s Web API-jima, razvijeni su alati koji se slobodno mogu koristiti i bave se upravo ovim problemom. Jedan od tih alata je *Retrofit* i on se koristi za slanje zahtjeva WEB API-u.

---

[3] E.HELLMAN; Android programming: pushing the limits; Wiley (2013)

[4] B.PHILLIPS, C.STEWART, B.HARDY, K.MARSCIANO; Android Programming: The Big Nerd Ranch Guide (2nd Edition); Big Nerd Ranch Guides (2017)




Slika 1 Životni ciklus Android aktivnosti

## Retrofit

Retrofit[5] je razvio *Square* i važan je dio HACCAPP sustava jer omogućuje komunikaciju s WEB API-em. To je biblioteka za slanje i primanje REST zahtjeva i odgovora. Retrofit kao podlogu za rad s REST-om koristi drugu biblioteku, OkHttp[6] za generiranje HTTP zahtjeva. Uz pomoć Retrofita, moguće je definirati sve parametre REST zahtjeva koji će biti detaljnije obrađeni kasnije. U ovom sustavu, Retrofit se koristi za slanje i primanje zahtjeva u JSON obliku, ali to se može promijeniti korištenjem jednim od raznih drugih konvertera, npr. XML konvertera. Za korištenje samog Retrofita potrebna su tri elementa:

- Model klasa pomoću koje se podatci za slanje pretvaraju u JSON ili drugi format
- Sučelje koje implementira Retrofit gdje se definiraju moguće HTTP operacije
- Klasa Retrofit.Builder, koja nasljeđuje sučelje i koristi se za definiranje URL-a zahtjeva.

[5] http://square.github.io/retrofit; 29.3.2018
[6] http://square.github.io/okhttp/

Sučelje mora implementirati po jednu metodu za svaki mogući zahtjev prema WEB API-u. Sve je metode potrebno označiti (@GET, @DELETE...) kako bi se specificirao tip zahtjeva i dobio URL. U povratnom tipu metode definira se tip odgovora API-a. Odgovor poslužitelja će biti tipa *Call* s objektom povratnog tipa u njemu.

## Firebase

Firebase[7] je *BaaS (*engl. ***B**ackend **A**s **A** **S**ervice)* platforma koju je 2011. godine razvio Firebase, a 2014. godine kupio Google. Firebase sadrži niz servisa koji olakšavaju razvoj mobilnih i web aplikacija. Zbog servisa koje nudi, Firebase programerima omogućuje implementaciju različitih funkcionalnosti u mobilne aplikacije uz minimalan trošak vremena. Neki od servisa koje Firebase nudi su:

- Pohrana datoteka – Omogućuje pohranu datoteka korisnika. Najčešće se koristi za pohranu medijskih sadržaja u oblaku.
- Baza podataka – Omogućuje korištenje NoSQL baze podataka u oblaku. Radi s JSON podatcima. Pomoću sync funkcionalnosti omogućuje korisnicima pristup do podataka s bilo kojeg uređaja. Može se koristiti i kao offline baza podataka na uređaju.
- Autentikacija korisnika – Olakšava prijavu korisnika preko postojećih računa s društvenih mreža ili preko Google računa.
- Google analiza – Omogućuje generiranje izvještaja o raznim događajima u aplikaciji. Može se koristiti za praćenje prijava korisnika, ispada aplikacije ili za generiranje novo kreiranih izvještaja s posebnim svojstvima.
- Slanje poruka iz oblaka – Servis korišten u HACCAPP mobilnoj aplikaciji. FCM (engl. Firebase Cloud Messaging) pruža način za komunikaciju poslužitelja s klijentskim aplikacijama. Omogućuje slanje poruka na iOS, Android i web aplikacije. U HACCAPP aplikaciji koristi se za slanje PUSH notifikacija klijentima.

[7] https://firebase.google.com; 25.04.2018.

Prije samog korištenja, Firebase se mora dodati na Android projekt kroz Android studio. Prvi korak je stvaranje Firebase projekta kroz Firebase konzolu. Na konzoli se odabere opcija "*Add project*" gdje se unese naziv projekta. Nakon što se projekt stvori, dodaje se u Android projekt preko opcije "*Add Firebase to your Android App*". Ova opcija stvara *config* datoteku koja se dodaje u Android projekt. Nakon dodavanja *config* datoteke, ostalo je dodati Firebase SDK i SDK za Google servise u Gradle datoteke.

Kod novijih verzija, Firebase se može dodati direktno u Android aplikaciju kroz Android studio. Pod opcijama "*Tools*", odabire se Firebase opcija koja će otvoriti prozor Firebase asistenta. Ova opcija služi kao kratki vodič za instalaciju Firebasea na lokalni projekt i obično se koristi kod razvoja novih projekata.

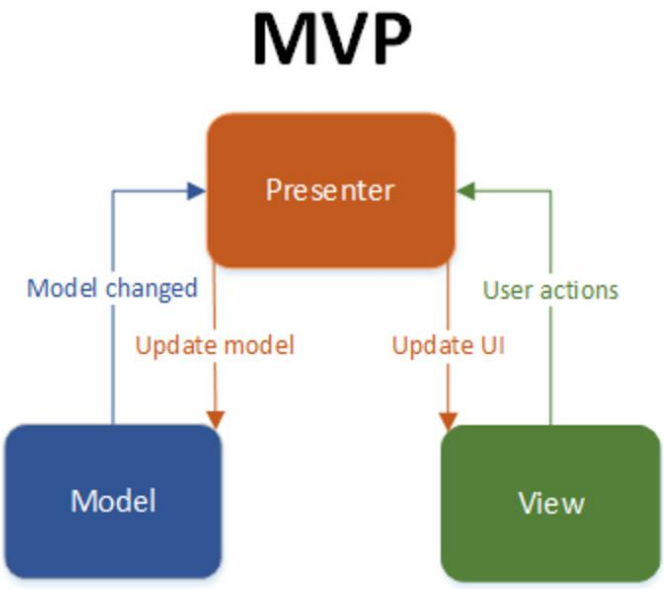
**Cloud Messaging**

FCM je Firebase servis koji se koristi u HACCAPP mobilnoj aplikaciji. Omogućuje slanje poruka i PUSH notifikacija na sve ili pojedine uređaje koji koriste određenu aplikaciju. U aplikaciji se PUSH notifikacije se koriste kao izvanredne obavijesti korisnicima koji generiraju dokumente. PUSH notifikacije se uglavnom koriste kod zahtjeva koji su završili neuspješno da se podsjeti korisnike da neke dokumente treba ponovno generirati.

**MVP arhitektura**

Kako bi aplikacija bila što jednostavnija za daljnje održavanje, u razvoju se koristi MPV(***M****odel* ***V****iew* ***P****resenter*)[8] arhitektura (Slika 2). MVP način razvoja mobilnih aplikacija predstavlja način programiranja tijekom kojeg se pazi na podjelu odgovornosti pojedinih slojeva aplikacije. Rezultat kod ovakvog razvoja je aplikacija koja više nije ovisna o aktivnostima (***A****ctivities*) i predstavlja moderan način programiranja mobilnih aplikacija. Zbog povećanog opsega posla, MVP se ne preporučuje za jednostavne aplikacije zbog velikog broja klasa i sučelja koji nastaju njegovim korištenjem. Prava snaga ovog pristupa leži u mogućnosti podjele briga (engl. Separation of concerns). Aplikacija se ovim pristupom dijeli u tri sloja: pogledi, prezenteri i modeli. Svaki sloj ima specifičnu namjenu i odgovornosti.

[8] http://www.ee.surrey.ac.uk/Projects/CAL/networks/Network-Transport_Application_Layers.htm; 12.4.2018




Slika 2 Android MVP

### 3.1.1. View

Ovaj sloj služi samo za prikaz mobilne aplikacije korisniku i ispunjava tradicionalnu ulogu prezentacijskog sloja (*Presentation layer*). View se sastoji od aktivnosti i fragmenata koji sačinjavaju aplikaciju. Ovaj sloj aplikacije ne može utjecati na podatke i poslovnu logiku. Jedina interakcija s ostatkom aplikacije je preko Presentera. Sve korisničke akcije na koje treba reagirati se preko Viewa prenose na Presenter. Svaki View mora imati implementiran odgovarajući Presenter preko kojega se komunicira s modelom.

### 3.1.2. Presenter

Ovaj sloj sadržava svu poslovnu logiku aplikacije. Služi kao vezivni sloj između prezentacijskog i podatkovnog sloja. Osim dohvaćanja podataka od modela, presenter definira i što će pojedini view prikazivati i odgovara na korisničke upite. View i odgovarajući Presenter stvara Android aktivnost. Za povezivanje Viewa i Presentera najčešće se koristi *dependency injection*. S obzirom na to da prezenter povezuje view s modelom i odrađuje velik dio posla, često se događa da presenter klase postanu prenapuhane i teške za održavanje, što poništava dobre stvari kod korištenja MVP-a. Jedini način da se to izbjegne je daljnje razdvajanje podjelom manjih zadataka presentera na manje klase.

### 3.1.3. Model

Ovaj sloj služi za dohvat podataka i radi kao i tradicionalni podatkovni sloj. Jedina svrha mu je dohvat potrebnih podataka u odgovarajuće vrijeme. Model odgovara na poziv Presentera i dohvaća podatke, te ih onda kroz Presenter šalje do Viewa gdje se podatci prikazuju. Baza podataka može biti lokalna, ali u većini slučajeva bit će vanjska i zato je u modelu obavezno

implementirati način za dohvat podataka bez zaustavljanja aplikacije. Za dohvat podataka iz vanjskog izvora, u HACCAPP aplikaciji koristi se Retrofit. Za razliku od viewa i presentera, model klasa je samo jedna (može biti i više klasa, ali MVP to ne definira) i svi ju presenteri koriste.

## 3.2. Komponente Android aplikacije

Komponente mobilne aplikacije su gradivni blokovi koji sačinjavaju tu aplikaciju[9][10]. Sve komponente aplikacije moraju biti navedene u *Manifest* dokumentu (*AndroidManifest.xml*). Ovaj dokument daju uvid u sve komponente koje se koriste u aplikaciji. Od osnovnih komponenata, u izradi aplikacije korišteni su:

- *Activities* – Svrha aktivnosti je prikaz grafičkog sučelja i interakcija s korisnikom. Svaka aktivnost predstavlja jedan prozor i omogućuje korisniku rad na tom prozoru. Tipično, svaka aktivnost odrađuje jednu funkcionalnost aplikacije. U manifestu se odabire koja aktivnost je glavna, odnosno ona koja će se prva pokretati s pokretanjem aplikacije. Za implementiranje aktivnosti obavezno je naslijediti *Activity* klasu.
- Services – Servisi služe za pozadinski rad s podatcima aplikacije. Uglavnom se koriste za dulje operacije kako ne bi blokirali korisničko sučelje aplikacije. Servisi omogućuju aplikaciji normalan rad dok korisnik koristi neku drugu aplikaciju. Iz tog razloga, servisi ne implementiraju korisničko sučelje. Servisi mogu neprestano raditi u pozadini nakon što ih neka aktivnost pokrene, ali i mogu biti vezani za tu aktivnost. Ako je vezan za aktivnost, servis prestaje raditi kada i aktivnost. Primjer servisa je aplikacija za sviranje glazbe koja radi u pozadini.
- *Broadcast receivers* – Komponenta koja omogućuje sustavu da aplikaciji dostavi informacije o nekom događaju u sustavu. Oni mogu dostavljati informacije i aplikacijama koje u tom trenutku nisu aktivne, što ih čini idealnima za prijenos obavijesti. Npr. obavijesti o slabom stanju baterije dostavljene su preko *broadcast receivera.*
- *Content providers* – Upravljači sadržajem su komponente koje upravljaju dijeljenim setom podataka pohranjenim u sustavu, lokalnoj bazi, webu ili bilo kojoj drugoj lokaciji za pohranu sadržaja kojoj aplikacija može pristupiti. Upravljač omogućuje drugim aplikacijama pristup i ažuriranje podataka. Primjer upravljača sadržajem je komponenta koja dozvoljava drugim aplikacijama pristup do .npr telefonskog imenika korisnika.

[9] J.BOŽIĆ, T.KADEŽABEK, F.TOMIĆ, T.KAŠTELAN; Izrada aplikacija za mobilne uređaje; Algebra (2013)
[10] D.GRIFFITHS, D.GRIFFITHS; Head First Android Development; O'reily Media (2015)

**Manifest dokument**

Manifest je dokument sadržan u glavnom dijelu aplikacije koji u sebi ima pobrojane sve komponente koje aplikacija posjeduje. Manifest omogućuje Android sustavu pokretanje komponente aplikacije i svaka komponenta koja nije definirana u tom dokumentu ne može se koristiti. U slučaju da se pokuša pokrenuti komponenta koja nije definirana u manifestu, bit će bačena iznimka.

Osim komponente, u manifestu su definirane i sve dozvole koje aplikacija mora imati za normalan rad. Pri instalaciji aplikacije, korisnik će morati dati svoje dopuštenje kako bi aplikacija mogla koristiti resurse na koje se dozvole odnose. Osim dozvola, definiraju se i svojstva hardvera i softvera koje aplikacija može koristiti, kao na primjer kamera.

U manifestu su definirane i minimalne API razine koje aplikacija zahtjeva za normalan rad. Pri pretraživanju aplikacija na trgovini, nije moguće pronaći aplikacije koje zahtijevaju API razine koje uređaj ne posjeduje.

Manifest deklarira i biblioteke s kojim aplikacija treba biti vezana.

U izradi HACCAPP aplikacije, korišteni su i sporedne komponente koje se ne navode u manifestu:

- *Views* – View je naziv za sve objekte koji se iscrtavaju na ekranu mobitela. Oni su elementi korisničkog sučelja.
- *Layouts* – *Layout* definira postavu grafičkih elemenata na ekranu.
- *Intents* – *Intent* služi za pokretanje novih komponenti i prijenos podataka među komponentama.
- *Resources* – Resursi su vanjski statički elementi korisničkog sučelja. Koriste se kao tekstovi na gumbima u aplikaciji i bilo kakvim znakovnim nizovima koji se ne mijenjaju, ali mogu biti i slike i bilo kakvi drugi elementi korisničkog sučelja koje korisnik ne mijenja.

## 3.3. Distribucija aplikacija

Aplikacija će u početku biti dana na korištenje manjem broju korisnika, radi testiranja aplikacije. Za potrebe testa, korisnicima će biti dostavljena instalacijska datoteka koja će instalirati aplikaciju izravno na mobitel. Nakon uspješnog testa, aplikacija će biti stavljena na Google Play Store, gdje će biti besplatna za preuzimanje.

**Zahtjevi aplikacije**

U ovom su segmentu opisani sistemski zahtjevi koje mobilni uređaj mora zadovoljavati kako bi mogao pokrenuti HACCAPP aplikaciju. Zahtjevi su skromni kako bi aplikaciju moglo koristiti što više ljudi.

Minimalna verzija Androida koju uređaj treba imati je Android 6.0, odnosno API razina 23. Skraćeno ime za Android verzije 6.x.x je Android Marshmallow.

Uređaj mora isto tako imati pristup Google Play uslugama. Osim za samu instalaciju aplikacije, Google Play usluge obavezne su i za korištenje Google servisa korištenih u HACCAPP aplikaciji.

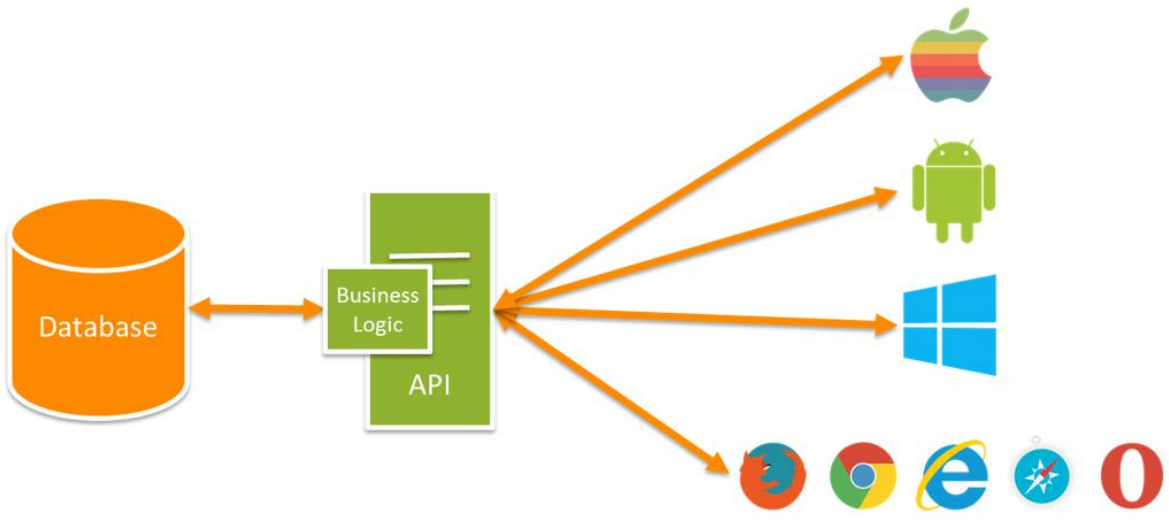
Zbog korištenja vanjskih biblioteka, minimalna API razina će rasti, kako bi se omogućio normalan rad aplikacije.

# 4. Web servisi

Kako bi se omogućilo kreiranje i slanje dokumenata nadležnim osobama pojedinog objekta, koristi se web API[11] (***A**pplication **P**rogramming **I**nterface*). On se koristi kako bi se taj dio logike mogao odvojiti od aplikacije, i služi kao baza podataka za podatke koji se ne mogu pohraniti na lokalnoj bazi aplikacije. Na taj način mobilna aplikacija nije prezatrpana dokumentima koji će biti pohranjeni na serverskom djelu aplikacije i ne zauzima previše mjesta na uređaju. Kako bi se osigurala povezanost zaposlenika i objekata u kojima rade, napravljena je baza podataka na poslužiteljskoj strani. U ovom će poglavlju biti objašnjene tehnologije korištene na poslužiteljskoj strani sustava.

**Osnove web API-ja**

Web API je programsko sučelje predstavljeno kroz više krajnjih točaka na koje se klijenti mogu spajati. Najčešće se koriste kako bi se klijentima omogućio rad s podacima. Komunikacija web API-ja i klijenta odvija se na zahtjev / odgovor (engl. *Request-Response*) način[12]. Web API predstavlja mjesto gdje se odvija centralna logika rješenja koji može primati zahtjeve iz više sustava (Slika 3). Kod navedenog primjera, ako je aplikaciju potrebno razvijati kao web i mobilnu aplikaciju postoji velika razlika u rješenju kod web servisa i web API-ja. Bez tog centralnog mjesta koje upravlja poslovnom logikom, razvoj sustava postajao bi sve kompleksniji, što je više platformi za koje bi se trebalo razvijati. Web API predstavlja jednostavniju varijantu jer se u ovom slučaju implementira centralno rješenje na koje će se spajati svi klijenti. U ovom slučaju, web API se brine za pristup i ažuriranje podataka.




Slika 3 Web API Arhitektura

[11] https://www.asp.net/web-api; 18.4.2018.
[12] https://blogs.msdn.microsoft.com/martinkearn/2015/01/05/introduction-to-rest-and-net-web-api/; 18.4.2018

Ovaj način razvoja omogućuje lakši razvoj raznim klijentskim aplikacijama koje će se spajati na web API. Ako je potrebno napraviti promjenu kod dohvata ili ažuriranja podataka, promjena se radi samo na jednom mjestu, a klijentske aplikacije mogu biti fokusirane na prikaz podataka. Ovaj pristup će razvoj sustava učiniti i jeftinijim zbog smanjenja kompleksnosti na klijentskoj strani.

## REST

REST je arhitektonski stil koji definira svojstva i omogućuje razmjenu standarda između računala na internetu. Baziran je na HTTP-u. Web servisi i usluge koji koriste REST omogućuju interoperabilnost između računala na internetu, što olakšava međusobnu komunikaciju različitih sustava. REST sustavi su karakteristični po tome što ne pamte stanja (engl. stateless) i po odvajanju briga (engl. *separation of concerns*) klijenta i poslužitelja.

### Odvajanje briga klijenta i poslužitelja

Odvajanje briga znači da klijent i poslužitelj mogu biti razvijeni zasebno i biti nezavisni jedan o drugom. Ovaj način rada omogućuje pisanje i promjenu koda, i načina rada klijenta ili poslužitelja bez promjene drugog. Za ostvarivanje njihove komunikacije ipak je potrebno poznavati format poruka koje će se razmjenjivati. Dok god obje strane koriste ispravne formate poruka, nije važno kakva je njihova interna implementacija. Na taj se način osigurava odvajanje različitih slojeva u izradi aplikacije, npr. prezentacijski sloj je implementiran na klijentskoj strani, dok su podatkovni i sloj poslovne logike implementirani na poslužiteljskoj strani. To omogućuje različitim klijentima rad na istom poslužitelju bez potrebe posebnih implementacija.

### Statelessness

Ne praćenje stanja je svojstvo sustava koji prate REST paradigme. To znači da svaka klijentova poruka sadrži sve podatke koji su potrebni poslužitelju da je ispravno protumači. To omogućuje klijentu i poslužitelju da razumiju svaku poslanu poruku, bez obzira na prijašnje poruke. Ne praćenje stanja korisno je zbog iskorištenja resursa, koji imaju ulogu opisa svih objekata i dokumenata korištenih za spremanje i dohvat podataka. Zbog korištenja resursa, REST sustavi nisu ovisni o implementaciji istih sučelja, kao kod drugih rješenja.

Navedena dva svojstva omogućuju REST aplikacijama responzivnost, jednostavnost i interoperabilnost, kroz korištenje lako održivih elemenata koji se mogu ponovno primjenjivati.

## Komunikacija poslužitelja i klijenta

Kod REST arhitekture, klijent šalje zahtjeve za dohvat ili modifikaciju podataka poslužitelju, a poslužitelj odgovara na zahtjev.

Kod slanja zahtjeva, klijent mora definirati nekoliko parametara kako bi ga poslužitelj mogao pročitati:

- Tip HTTP zahtjeva – Definira koje akcije klijent želi raditi nad podacima
- Zaglavlje zahtjeva - Omogućuje klijentu slanje dodatnih informacija o zahtjevu
- Put do resursa
- Tijelo poruke s dodatnim podatcima o zahtjevu (opcionalni parametar)

### Tipovi HTTP zahtjeva

Postoje 4 osnovna tipa HTTP zahtjeva koji se najčešće koriste, i 4 dodatna tipa. Zahtjevi se koriste kako bi se definiralo koji tip CRUD[13] operacije će biti izvršen nad podacima[14] (Slika 4).

- GET – Koristi se za dohvat resursa. Omogućuje vraćanje kolekcije resursa, ili jednog resursa ako je u zaglavlju zahtjeva definiran identifikacijski broj traženog resursa.
- POST – Koristi se za stvaranje novog resursa. U zaglavlju zahtjeva definira se identifikacijski broj novog resursa, a u tijelu poruke se prenosi objekt koji će biti stvoren.
- PUT – Koristi se za ažuriranje specifičnog resursa čiji se identifikacijski broj nalazi u zaglavlju zahtjeva. Nove vrijednosti resursa se nalaze u tijelu zahtjeva.
- DELETE – Koristi se za brisanje resursa čiji je identifikacijski broj proslijeđen u zaglavlju zahtjeva.

Dodatna 4 HTTP zahtjeva su HEAD, CONNECT, OPTIONS i TRACE, ali oni nisu korišteni u ovom radu.

### Zaglavlje zahtjeva

Kroz zaglavlje zahtjeva, klijent definira koji tip sadržaja može primati i na taj način govori poslužitelju kako mora pripremiti odgovor zahtjeva. Polje u kojem se definira povratni tip naziva se *Accept* polje. Mogući tipovi *Accept* polja su MIME[15] (engl. ***M**ultipurpose **I**nternet **M**ail **E**xtensions*) tipovi. Polje sadržaja definira se kao par tip i podtip, odvojeni kosom crtom (*type/subtype*). Najčešća povratna polja su tipa *text* s podtipovima *plain, html, css* ili *xml*. Osim tekstualnih tipova, dostupni su i audio, slikovni, video i aplikacijski formati. U ovom radu, zahtjevi se šalju u JSON formatu, pa je povratni tip definiran kao *application/json*.

[13] Operacije nad podacima u bazi podataka. Create, Retrieve, Update i Delete
[14] https://www.getpostman.com/; 29.4.2018.
[15] Labela koja se koristi za raspoznavanje tipova podataka i pomaže programu da raspozna kako postupati s podacima označenim tom labelom.

### Putanja

Osim tipa operacije koje treba izvršiti, zahtjevi moraju definirati i put do resursa na kojima se treba izvršiti ta operacija. Tipično su nazivi putanja takvi da su ljudima lako čitljivi i pomažu u razumijevanju sustava. Tako je jasno da npr. putanja koja završava s .../Korisnici/5 dohvaća korisnika s identifikacijskim brojem 5 iz baze podataka. Moguće je i dobiti drugačije rezultate s istom putanjom, ovisno o tome koji je tip HTTP zahtjeva definiran. Na primjer, GET i POST zahtjevi mogu imati identične putanje (.../Korisnici/), ali GET zahtjev će vratiti sve korisnike iz tablice, dok će POST zahtjev unijeti novog korisnika sadržanog u tijelu zahtjeva.

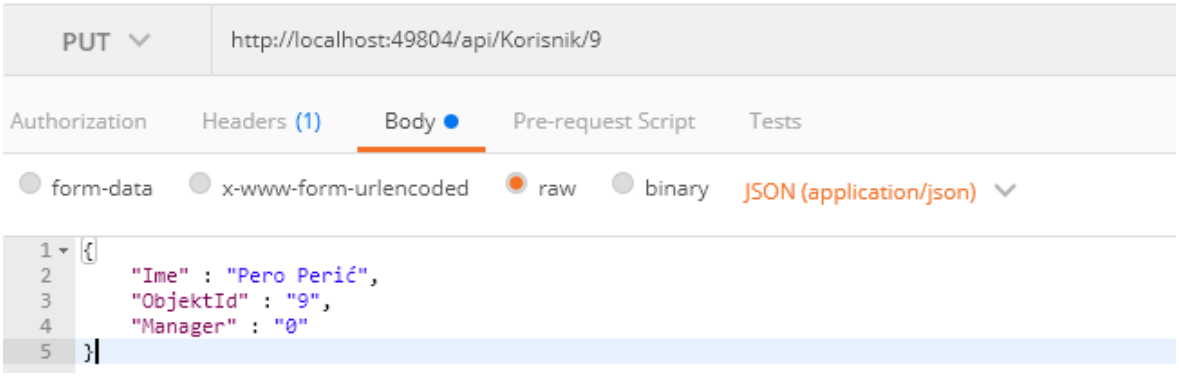
### Slanje odgovora klijentu

Slično kao i kod slanja zahtjeva, pri slanju odgovora klijentu, poslužitelj mora navesti povratni tip odgovora. Iako klijent u zahtjevu definira tip odgovora, moguće je navesti više različitih tipova, od kojih će jedan biti vraćen. Tip odgovora definiran od strane klijenta zapravo nabraja sve vrste povratnih tipova koje klijent može primiti. Zbog toga je polje tip sadržaja (engl. *Content type*) obavezno u odgovoru klijentu. Klijent može imati implementirano više načina tumačenja rezultata, ovisno o povratnom tipu koji je vraćen. Povratni tip je opet definiran kao par tip/podtip.

Osim povratnog tipa, poslužitelj s klijentom komunicira i preko kodova odgovora (engl, *Response codes*). To su često korišteni kodovi čije je značenje otprije poznato i zato se koristi kao univerzalni jezik kod slanja odgovora. Uvijek se šalju i služe za pojašnjenje rezultata operacije koju je klijent zatražio. Neki od tipičnih odgovora su:

- 200 – OK. Znači da je operacija uspješno izvršena. Često se koristi kod uspješne operacije ažuriranja resursa kod PUT naredbe.
- 201 – *Created*. Tipičan odgovor kod uspješne operacije dodavanje resursa u bazu podataka kod POST zahtjeva.
- 400 – *Bad request*. Status koji se šalje kada operacija nije uspješno izvršena, obično zbog greške s klijentske strane.
- 500 – *Internal server error*. Status koji se šalje kada operacija nije izvršena zbog greške na poslužitelju.

Osim potvrde je li operacija uspješna ili ne, kodovi odgovora govore i gdje je došlo do problema. Kodovi odgovora koji započinju s brojem 2 obično znače da je operacija uspješno izvedena. Kodovi koji počinju s brojem 4 označavaju pogrešku u zahtjevu odnosno grešku na klijentskoj strani. Kodovi koji počinju s 5 označavaju grešku na poslužitelju. Kod neuspješnog procesiranja zahtjeva, klijent koji je npr. dobio grešku 500 zna da je problem u poslužitelju i da vjerojatno ne treba mijenjati svoj zahtjev.

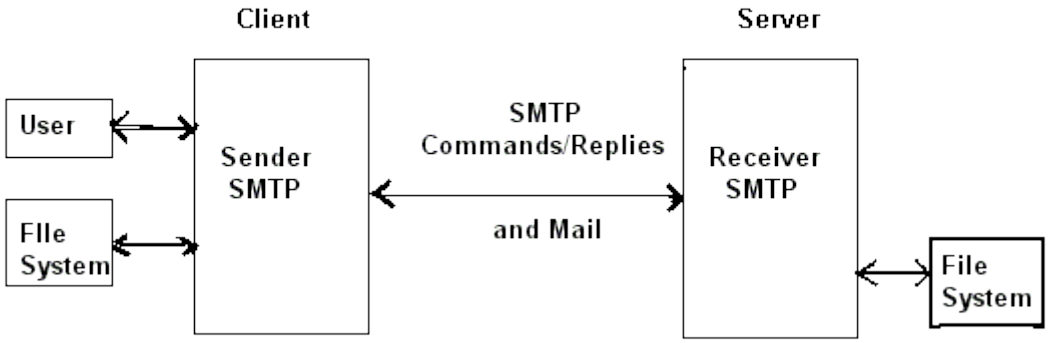



Slika 4 SMTP Protokol

## SMTP Protokol

SMTP[16] je internet standard za slanje email poruka, definiran u 1981. kroz RFC 788. Svrha mu je prenošenje emaila sigurno i efikasno.

Prijenos poruka se temelji na *end-to-end* arhitekturi gdje klijent i poslužitelj međusobno izmjenjuju poruke na zahtjev-odgovor način (Slika 5). Pri slanju poruke, klijent pošiljatelja komunicira s poslužiteljem primatelja poruke. Za komunikaciju se koristi port 25. Komunikacija između klijenta i poslužitelja odvija se kroz slanje naredaba s podacima koji su potrebni za izvršenje zahtjeva. Nakon zaprimanja zadnje naredbe koja označava kraj poruke, poslužitelj potvrđuje da je zaprimio poruku slanjem povratne naredbe.




Slika 5 Grafički prikaz SMTP protokola

Nakon uspješnog slanja poruke, klijent može:

1. Zaustaviti komunikaciju – Ako više nema potrebe za uspostavom komunikacije, klijent je može prekinuti slanjem naredbe koja će zatvoriti sesiju.

[16] http://www.ee.surrey.ac.uk/Projects/CAL/networks/Network-Transport_Application_Layers.htm; 1.5.2018.

2. Zamijeniti uloge – Ako je klijent poslao sve poruke koje je imao, ali može početi primati poruke od trenutnog poslužitelja, on šalje TURN naredbu. Nakon toga slijedi zamjena uloga gdje dosadašnji klijent postaje poslužitelji prima poruke po istom principu.
3. Poslati novu poruku – Ako klijent ima još poruka za slanje, može poslati MAIL naredbu i ponoviti cijeli proces od početka.

## 4.1. Arhitektura

Web servis se sastoji od pružatelja usluge, korisnika usluge i brokera usluge koji korisnicima omogućuje pronalazak web servisa.

## 4.2. WCF

WCF[17] je razvojno sučelje za razvoj servisno orijentiranih aplikacija. Koristeći WCF, moguće je asinkrono slati podatke između krajnjih točaka. Krajnje točke mogu biti klijenti ili server koji prima zahtjeve klijenata. Glavne značajke WCF-a su:

- Servisna orijentiranost - Ovisnost o web servisu za primanje i slanje podataka. Prednost ovakve arhitekture je to što jedan pružatelj usluga može uslugom opskrbljivati više različitih tipova korisnika, neovisno o platformi, što olakšava razvoj na klijentskoj strani.
- Više vrsta komunikacije – Omogućena je razmjena poruka na nekoliko načina. Osim najčešćeg, zahtjev / odgovor, moguće je i slanje podataka serveru bez očekivanja odgovora. Postoje i kompleksnije vrste komunikacije kod kojih se dvije krajnje točke povežu i međusobno razmjenjuju podatke.
- Metapodaci o servisu – WCF omogućuje objavljivanje metapodataka o servisu, zapisanih u nekim od industrijskih standarda. Pomoću njih je moguće generirati klijentski dio web servisa za pristupanje poslužitelju.
- Podatkovni ugovor – Kako bi se olakšao rad s podatcima iz baze, najčešće se stvaraju klase koje čiji model odgovara modelu podataka u bazi. Pomoću WCF-a, koji koristi .NET, moguće je generirati klase koje će predstavljati entitete iz baze podataka. Nakon generiranja klasa, servis automatski generira metapodatke koji omogućuju klijentima lakši pristup servisu.
- Sigurnost – Kako bi se komunikacija između klijenta i poslužitelja osigurala i zaštitila privatnost dijeljenih podataka, moguće je korištenje standarda kao ISS. Sigurnost podataka osigurava se enkripcijom ili autentikacijom koju klijenti moraju proći prije nego što mogu dobiti odgovor od poslužitelja usluge.
- AJAX i REST – Omogućuje procesiranje XML-a i JSON-a u običnom tekstu, odnosno bez SOAP omotnice.

[17] J.LOWY, M.MONTGOMERY; Programming WCF Services, 4th Edition; O'reily Media (2015)

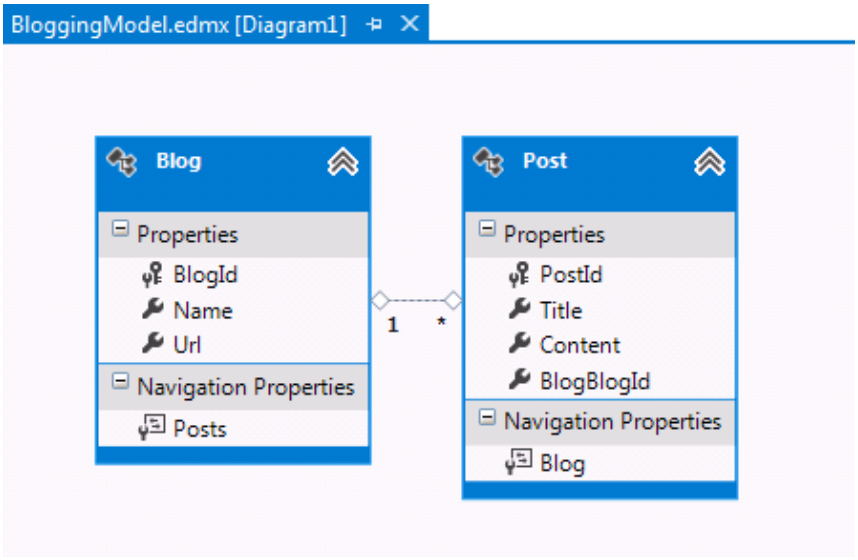
## 4.3. Entity Framework

Entity framework[18] je ORM razvojna okolina za ADO.NET. Predstavlja skup tehnologija koje omogućuju razvoj podatkovno orijentiranih aplikacija. Omogućuje pretvaranje podataka između nekompatibilnih sustava koristeći objektno orijentirane programske jezike. Bio je dio .Net razvojne okoline, ali od verzije 6 *Entity framework* je odvojen od .NET-a.

Razlog zašto se mnogo developera odlučuje za korištenje *Entity framework*-a je jednostavnost korištenja i dohvata podataka iz baze. Omogućuje kreiranje modela iz tablica baze kroz kod ili korištenjem *EF* dizajnera za grafički prikaz i stvaranje novih modela i tablica. Na taj je način moguće zamijeniti inače puno kompleksniji podatkovni sloj s modelima dobivenih izravno iz baze. Sve ovo omogućuje programerima da pišu jednostavniji i lakše održivi kod, što u konačnici vodi do boljih aplikacija. Pristup do baza podataka omogućen je na više načina:

- *Model first* – Koristi se kod spajanja na postojeću praznu bazu podataka, ili na potpuno novu bazu koja će biti stvorena. Omogućuje stvaranje modela baze u dizajneru uporabom pravokutnika koje predstavljaju tablice i linija koje predstavljaju veze između tablica (Slika 6). Iz tog modela će biti stvorene nove tablice ili potpuno nova baza. Klase za dohvat i rad s podatcima iz baze bit će automatski generirane iz napravljenih modela.
- *Database first* – Koristi se kod spajanja na postojeću bazu podataka s postojećim tablicama. Nakon spajanja na bazu, model je stvoren prema jednoj ili više tablica u toj bazi. Klase se opet automatski generiraju iz modela. Kod prva dva pristupa uobičajen je rad s dizajnerom.
- *Code first* – Koristi se kod stvaranja nove baze podataka. Kod ovog pristupa, prvo se pišu kod i klase. Na taj se način klase koje će biti kasnije predstavljene tablicama u bazi mogu prilagoditi potrebama razvoja u trenutku stvaranja. Model se stvara iz koda i baza podataka se onda stvara iz modela kao u *model first* pristupu. Ovaj način rada omogućuje korištenje migracija kako bi se promjene u modelu mogle odraziti na tablice u bazi.
- *Code first* s postojećom bazom – Kod ovog pristupa, model se opet generira iz koda, ali se spaja s postojećom bazom i tablicama.

[18] https://msdn.microsoft.com/en-us/library/ee712907(v=vs.113).aspx; 21.4.2018.




Slika 6 Primjer korištenja EF dizajnera u *Entity Frameworku*

**LINQ**

.NET LINQ[19] je dodatak .NET-u koji omogućava izvršavanje upita na bazu podataka preko podatkovnih modela. Prednost LINQ-a je što se može koristiti kroz kod i ne zahtijeva posebnu implementaciju za pristup do baze za dohvat podataka (Slika 7). To znači da će korištenje LINQ-a biti identično, bez obzira na izvor ili format podataka, što uvelike olakšava razvoj razvojnim inženjerima. Omogućuje rad s XML dokumentima, SQL bazama podataka, *DataSet* objektima ADO.NET-a i s mnogim drugim formatima koje podržava.

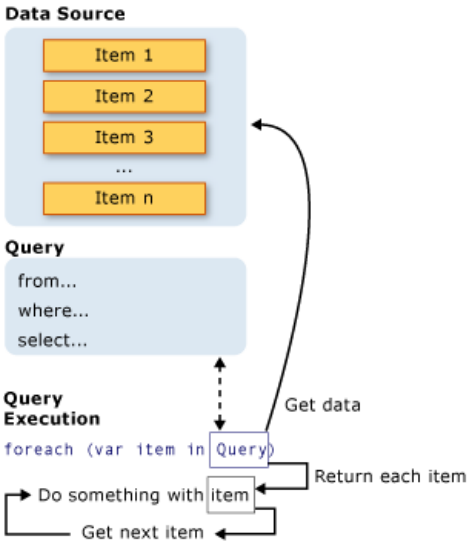
Rad s podatcima u LINQ-u uvijek je podijeljen na tri akcije:

1. Dohvat izvora podataka – Izvor podataka je jedan od ranije spomenutih, ali može biti i obična kolekcija primitivnih tipova podataka. Za LINQ je važno da navedeni izvor podataka implementira sučelje *IEnumerable* ili *IEnumerable<T>*. Razlog tome je što se sam upit izvršava u *foreach* bloku koda, koji radi samo s kolekcijama koje implementiraju jedan od ta dva sučelja. Za pristup podatcima u SQL bazi podataka, koristi se ORM (*Entity framework*), koji će se tijekom izvršavanja programa povezati sa SQL bazom i dohvaćati entitete iz baze. Za izvor podataka u tom slučaju, mora se koristiti kolekcija modela koji predstavlja tablicu iz koje se dohvaćaju podatci.
2. Izrada upita – U akciji izrada upita definira se koji će se podatci iz izvora podataka dohvaćati. Krajnji je rezultat moguće i sortirati ili grupirati prije nego što se vrati. Upis je sačuvan u *query* varijabli, a pokreće se kao izraz. Kako bi se olakšalo pisanje upita, u C#-u je definirana nova sintaksa upita. Svaki se upis sastoji od tri dijela: *from, where* i *select* dijela. Ova se sintaksa može usporediti sa SQL naredbama, ali raspored je drugačiji. *From* dio definira izvor podataka iz kojeg se dohvaćaju podatci, *where* dio predstavlja uvjete koje podatci trebaju zadovoljiti da bi bili dohvaćeni, i *select* dio

[19] https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/introduction-to-linq-queries; 15.4.2018.

definira koji tip podatka će biti vraćen. Važno je napomenuti da upit samo pohranjuje podatke koji su potrebni za generiranje rezultata i može biti izvršen kasnije. Po potrebi je omogućeno i ranije izvršavanje upita.

3. Izvršavanje upita – Predstavlja prolazak po varijabli upita s *foreach* petljom. Prije toga se upit ne smatra izvršenim. Prednost ovog pristupa je to da varijabla upita ne pohranjuje rezultate upita, što znači da se može koristiti po potrebi više puta. Moguće je i prisilno pokretanje upita. Ono se događa kod izvršenja naredbi koje već u sebi sadrže *foreach* petlju, jer moraju proći po svim elementima izvora podataka. To se događa kod pozivanja naredbi npr. *Count, Max ili First*, ali i kod završavanja upita s naredbom *ToList* ili *ToArray*, gdje se rezultat odmah sprema u vanjsku kolekciju.




Slika 7 Dijelovi LINQ poziva

## ADO.NET

ADO.NET je skup programskih tehnologija iz .NET *Framework*-a koji omogućuje komunikaciju i pristup do baza podataka. Najčešće se koristi za samo spajanje na bazu podataka i za rad s podacima u bazi. Najčešće se koristi za rad s relacijskim bazama podataka, ali nije ograničen na rad samo s njima. Smatra se da je razvijen da zamjeni ADO, ali je toliko razvijen da se smatra potpuno novim proizvodom.

Kako bi se olakšao rad s podatcima, ADO.NET razdvaja komponente za pristup do podataka od komponenata za manipulaciju podatcima. Za spajanje na bazu podataka, izvršavanja komandi i dohvat rezultata koriste se komponente .NET razvojne okoline. Rezultate je nakon toga moguće procesirati preko *DataSet* objekata ADO.NET-a ili tradicionalno preko implementacije podatkovnog sloja. *DataSet* objekti se mogu koristiti i zasebno, neovisno o opskrbljivačima podataka .NET-a. Pomoću tih objekata moguće je procesiranje lokalnih aplikacijskih podataka, ali i podataka iz XML-a.

### Microsoft SQL Server

MSSQL[20] je relacijska baza podataka korištena na poslužiteljskoj strani. Uz *Oracle* i *DB2* baze, MSSQL je jedan od glavnih tehnoloških rješenja na području baza podataka.

### DBFlow

DBFlow[21] je baza podataka korištena lokalno na korisničkom uređaju. DBFlow, za razliku od drugih relacijskih baza podataka, ne koristi *Java Reflection* za definiranje modela, shema i relacija među tablicama. DBFlow se koristi korištenjem posebnih anotacija koje zna procesirati i od njih generira Java kod. Kroz taj se način rada dobiva na responzivnosti aplikacije i omogućuje se brži rad r. inženjera zbog manje potrebe za pisanjem koda.

[20] https://www.microsoft.com/en-us/sql-server/sql-server-2016; 18.4.2018.
[21] https://guides.codepath.com/android/DBFlow-Guide; 9.4.2018.

# 5. Razvoj sustava

U prijašnjim poglavljima objašnjene su tehnologije korištene za izradu HACCAPP aplikacije. Ovo se poglavlje fokusira na svrhu, komponente i zahtjeve u razvoju aplikacije.

## 5.1. Cilj razvijanja sustava

Cilj razvijanja ovog sustava je olakšati ispunjavanje ranije navedenih HACCAP dokumenata. Iskustva zaposlenika su pokazala da se na ispunjavanje i upravljanje tim dokumentima troši puno vremena zbog velike količine dokumenata koje svaki dan treba popuniti. Sustav je podijeljen na mobilnu aplikaciju kroz koju je moguće upisati sve podatke o nekom dokumentu, i na web api koji prima te podatke, validira ih i šalje nadležnoj osobi za taj objekt.

## 5.2. Arhitektura i komponente

Dva glavna dijela ovog sustava su mobilna aplikacija koju koriste zaposlenici i web servis koji odrađuje zadatke koje nije moguće, ili nije praktično odraditi preko aplikacije. Uloga aplikacije je prikupljanje podataka i slanje tih podataka web servisu koji će prema potrebi ažurirati centralnu bazu podataka i generirati dokumente za voditelje objekata. Također, za čuvanje podataka koriste se dvije baze podataka. Jedna je centralna baza web servisa, a druga je lokalna baza na mobilnom uređaju koji koristi aplikaciju. Centralna baza brine o korisnicima, objektima kojima korisnici pripadaju i dokumentima koje je moguće generirati za pojedine objekte. Kroz tu se bazu osigurava da se prate pravila o korištenju koje nije moguće provjeravati samo s lokalnom bazom na mobilnom uređaju. Druga, lokalna baza na svakom uređaju služi za pohranu podataka o korisniku koji ju koristi. Na njoj se zapisuju podatci koje korisnik unese prilikom kreacije svog profila, ali se koristi i kao arhiva dokumenata koje je korisnik generirao, kao i vremena kada su generirani.

## 5.3. Zahtjevi i funkcionalnost

Kako bi generirani dokumenti bili ispravni za korištenje, obavezno je praćenje korisničkih zahtjeva. Ovi su zahtjevi definirani s budućim korisnicima kako bi se osiguralo što veće zadovoljstvo korisnika kod korištenja aplikacije. Važno je napomenuti da su korisnički zahtjevi podložni raznim promjenama u životnom vijeku aplikacije.

## Unos podataka o poslovnom objektu

Radi sigurnosti da svaki zaposlenik ispunjava dokumente objekta kojem pripada, kroz bazu podataka osiguravamo da svaki zaposlenik može pristupiti samo dokumentima objekta u kojem radi. To je osigurano kroz poveznicu stranog ključa u tablici sa zaposlenicima gdje se jedan zaposlenik povezuje s jednim objektom. Osim zaposlenika, i dokumenti su povezani s objektom, ali u zasebnoj tablici gdje se vodi evidencija o tome koji objekt ispunjava koje dokumente. Tablicu s dokumentima i objektima popunjava korisnik s rolom Voditelj određenog objekta. Svaki Voditelj može u tu tablicu dodavati dokumente, ali samo za objekt s kojim je povezan. Voditelj ne može biti povezan s više objekata odjednom. Kada je ta tablica popunjena za pojedini objekt, zaposleniku je omogućen pristup do aktivnosti za ispunjavanje dokumenata u mobilnoj aplikaciji.

## Validacija podataka u dokumentima

U prvom su poglavlju spomenuti svi dokumenti koje će biti moguće generirati kroz mobilnu aplikaciju. Svaki od tih dokumenata ima kritične vrijednosti koje se ne bi smjele prijeći, i korektivne radnje ako do toga ipak dođe. Te vrijednosti se šalju do web servisa zajedno s ostalim podatcima o dokumentu koji se ispunjava. Ako je došlo do premašivanja kritične vrijednosti, rezultat se na dokumentu mora napisati crvenom bojom i o tome se obavezno mora obavijestiti korisnik s rolom "Voditelj" tog objekta. Kako bi se osiguralo da se korektivna mjera provodi, obavezno je poslati "Push" notifikaciju kao podsjetnik da se dokument mora ponovno ispuniti. Dokument mora biti ponovno generiran za dva do tri sata, ovisno o definiciji na samom dokumentu. Osim ovih, korisničkih validacija, kroz web servis su implementirane i validacije tipova podataka koji će se slati preko mobilne aplikacije. Ako se takve greške pronađu, dokument neće biti generiran i korisnik će o tome biti obaviješten.

## Voditelj objekta

Voditelj je uloga dodijeljena korisniku aplikacije. Svaki registrirani objekt mora na sebi imati registriranog barem jednog korisnika. To je minimum za normalno funkcioniranje aplikacije za taj objekt. U svakom objektu, jedan od zaposlenika mora biti definiran kao "Voditelj". Taj zaposlenik predstavlja osobu koja ima mogućnost odabira koje će dokumente drugi zaposlenici tog objekta moći generirati. Ta se rola dodjeljuje korisniku pri njegovom kreiranju. Taj zaposlenik obavezno mora imati definiranu e-mail adresu na koju će se slati generirani dokumenti. Definirana e-mail adresa ne mora biti prava adresa korisnika koji ima rolu, nego samo predstavlja adresu na koju objekt želi primati dokumente generirane putem aplikacije. Isto tako, ne postoji validacija koji zaposlenik ima dodijeljenu rolu, ali za svaki objekt moguće je definirati samo jednog voditelja.

## Ažuriranje podataka o korisniku

Kako ne bi došlo do zagušivanja baze s novim korisnicima, kroz aplikaciju je omogućeno ažuriranje trenutno aktivnog korisnika. Ova je funkcionalnost dodana da se spriječi nepotrebno dodavanje novih korisnika. Svim se funkcionalnostima pristupa preko mobilne aplikacije, dok se većina podataka ažurira na bazi podataka web servisa. Kroz aktivnost u mobilnoj aplikaciji omogućeno je:

- Ažuriranje podataka o zaposleniku. Moguće je ažurirati sve podatke unesene prilikom stvaranja zaposlenika, obzirom da se zaposlenik prepoznaje po jedinstvenoj vrijednosti koja se dodjeljuje od strane web servisa prilikom stvaranja zaposlenika. Ta je vrijednost predstavljena identifikacijskim brojem i jedina je vrijednost koju nije moguće promijeniti. Zaposlenik može promijeniti svoje ime, lozinku i email adresu. Ako nove vrijednosti prođu validaciju, zaposlenik će ih odmah moći primjenjivati.

- Brisanje podataka o zaposleniku. Predstavlja prestanak rada tog zaposlenika u određenom objektu. Svi se podatci o zaposleniku brišu iz tablice aktivnih zaposlenika u centralnoj bazi podataka. Kako bi se očuvali podatci o korisniku, ovdje se radi o logičkom brisanju gdje se korisnički račun onemogućuje za daljnje korištenje. Ne postoji maksimalni broj zaposlenika koji objekt može imati vezanih na sebe, ali radi očuvanja stvarnog stanja u bazi podataka, preporučuje se da neaktivni zaposlenici izbrišu svoje profile.

- Promjena role Voditelja objekta. Ako zaposlenik koji trenutno ima tu rolu iz nekog razloga mora obrisati svoj profil ili mora prestati biti Voditelj, obavezan je imenovati novog Managera. Ova je funkcionalnost omogućena kroz aktivnost na mobilnoj aplikaciji i pri tome trenutni Voditelj mora odabrati novog zaposlenika iz istog objekta kao i on. Novoodabrani zaposlenik bit će smatran Managerom i dobit će sve privilegije te role. Ukoliko se pokuša obrisati zaposlenik koji ima rolu Managera, obavezno je prije brisanja imenovati novog Managera.

## Stvaranje i ažuriranje objekata

U ovom sustavu, objektom se smatra trgovina, restoran ili hotel koji poslužuje ili skladišti hranu, u skladu s definicijom tih objekata na početku rada. novi je objekt moguće kreirati putem aktivnosti na mobilnoj aplikaciji. Kreiranje objekta predstavlja prvi korak u korištenju aplikacije, obzirom da se svaki kreirani zaposlenik mora vezati za neki objekt. Svaki kreirani objekt bit će sačuvan u lokalnoj bazi podataka uređaja, ali i u centralnoj bazi web servisa. Pri stvaranju objekta, objektu će automatski biti dodijeljen identifikacijski broj koji nije moguće naknadno mijenjati. Naziv objekta moguće je promijeniti naknadno. To može napraviti samo zaposlenik s Voditelj rolom. Objekt se može obrisati samo ako su svi ostali zaposlenici tog objekta obrisani. Kako bi se objekt obrisao, Voditelj tog objekta mora kao posljednji zaposlenik odabrati opciju “obriši objekt“ pri brisanju vlastitog profila.

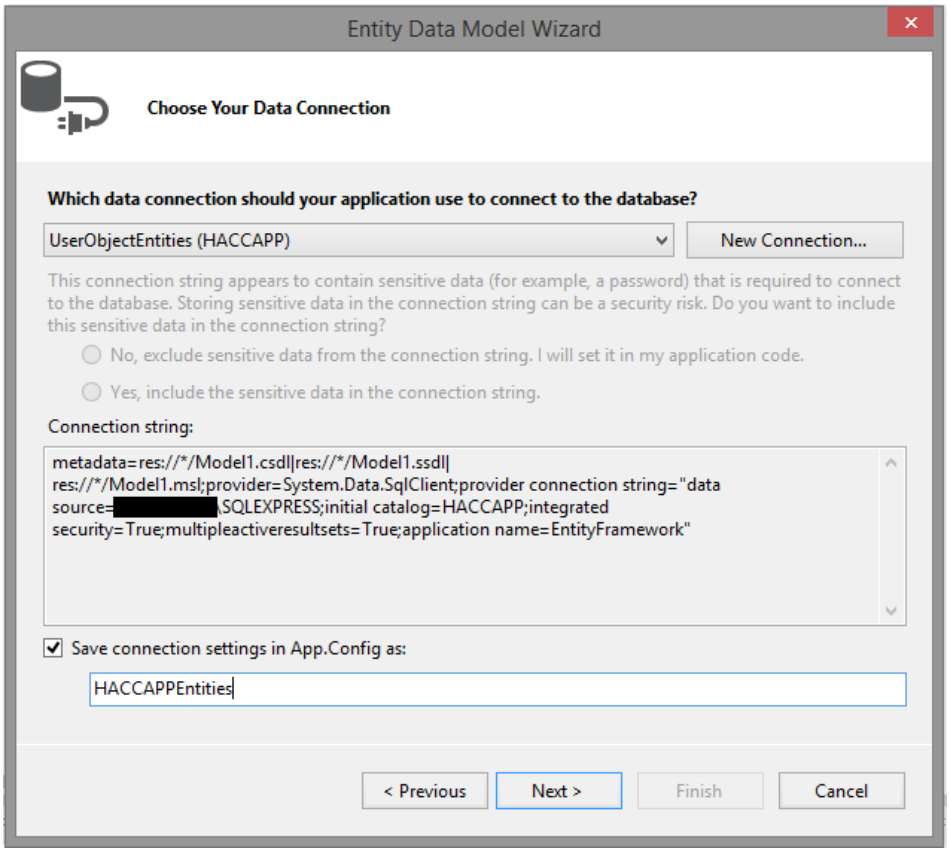
# 6. Serverski dio sustava

Serverski dio sustava predstavljen je ASP .NET Web API-em. Razlozi za korištenje su jednostavnost implementacije, primanja i slanja REST zahtjeva i korištenje podatkovnog modela.

## 6.1. Baza podataka

Na poslužiteljskoj strani, za pohranu podataka korišten je MSSQL, i kreirana je baza naziva HACCAPP. Za spajanje na bazi napravljen je novi projekt unutar istog *Solutiona* naziva *DatabaseAccess*. Taj je projekt tipa *ClassLibrary* i koristi se kako bi se preko njegovih klasa pristupalo do baze. Referenca na *DatabaseAccess* projekt dodana je u reference glavnog projekta HACCAPP.

*DatabaseAccess* projekt sadrži *DataModel* klase generirane tako da predstavljaju tablice iz baze (Slika 8). Te je klase moguće generirati putem *Entity Data Model* čarobnjaka.




Slika 8 *Entity Data Model Wizard*

Pomoću *Entity Framework* alata, generirane su sve klase koje će se koristiti u sustavu. Tablice iz kojih su generirane klase su (Slika 9):

**Employee**

Tablica Employee sadrži podatke o svim korisnicima kreiranih preko HACCAPP mobilne aplikacije. Koristi se kod autentikacije korisnika.

**DBDocument**

Tablica za pohranu podataka o dokumentima koje je moguće generirati kroz aplikaciju. Ova tablica sadrži podatke o svim mogućim dokumentima, iako korisnik koji koristi aplikaciju neće nužno moći vidjeti sve dokumente.

**DBCompany**

Tablica DBCompany sadrži podatke o poslovnom objektu kojem korisnik pripada. Nakon što korisnik pri registraciji poslovnog objekta odabere PAN[22] tog objekta svi će ga ostali korisnici morati koristiti.

**DBDocumentCompanyLink**

Tablica za povezivanje poslovnih objekata i dokumenata koje je moguće ispuniti za njih. S obzirom na to da se radi o vezi M:N, za povezivanje se koristi nova tablica.

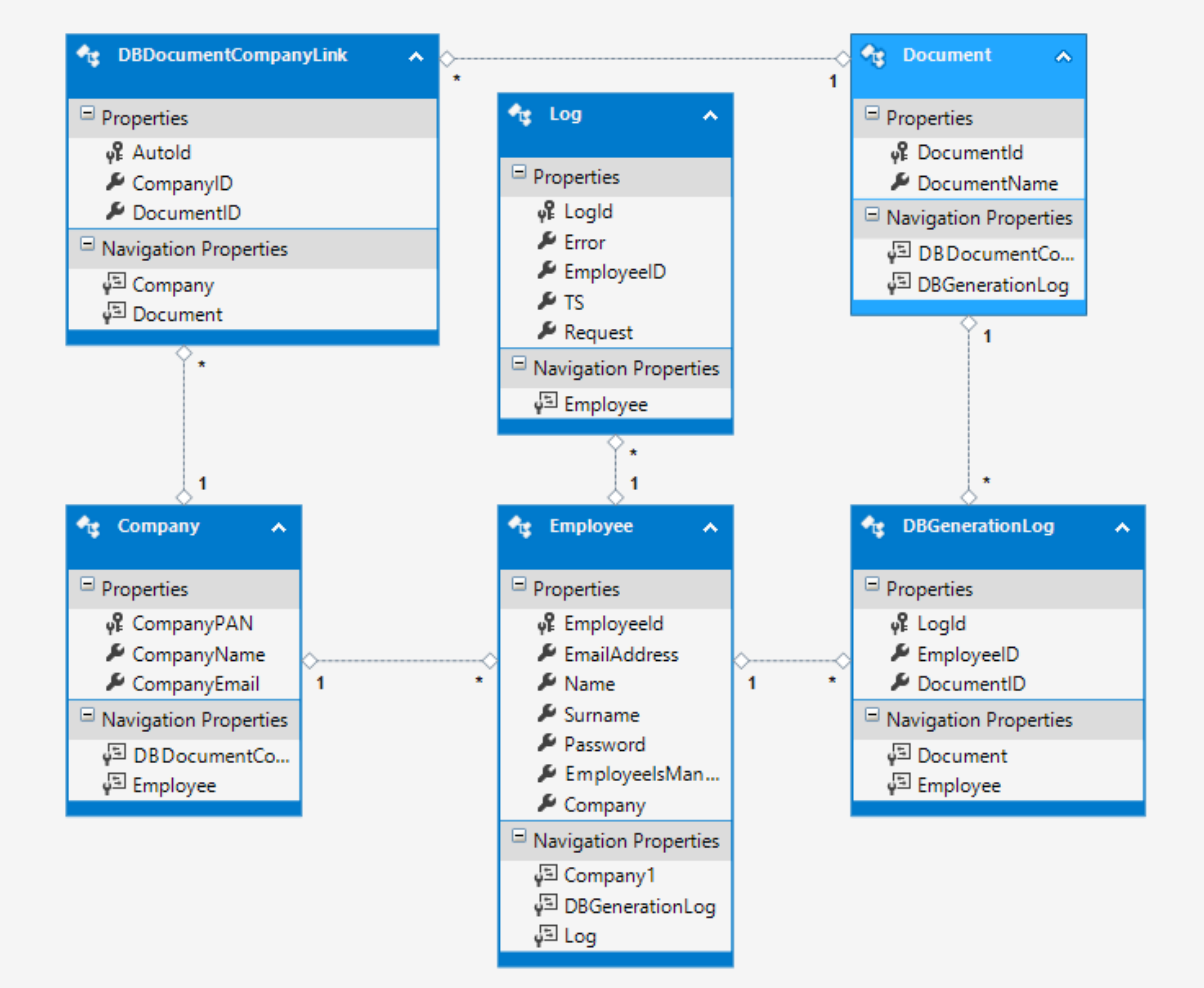
**DBGenerationLog**

Ova tablica služi za arhiviranje podataka o korištenju aplikacije. Ova će tablica omogućiti generiranje izvještaja o tome koji je korisnik i kada generirao pojedine dokumente. U tablici se čuvaju podaci o tome kada je zahtjev došao do poslužitelja, identifikacijski broj korisnika koji je kreirao zahtjev i koji je dokument generirao. Ako tijekom obrade zahtjeva dođe do greške, zahtjev će biti zapisan u Log tablicu, a ne u DBGenerationLog. Ova tablica sadrži samo popis ispravnih zahtjeva.

**Log**

Log tablica služi za zapisivanje grešaka do kojih je dolazilo u radu poslužitelja. U njoj će biti pohranjena poruka o grešci, zajedno s datumom i vremenom kada je greška nastala, te identifikacijskim brojem korisnika čiji je zahtjev doveo do greške i sam zahtjev koji je uzrokovao grešku.

[22] Primary Account number. Jedinstvena vrijednost koju ima svaki poslovni objekt




Slika 9 Shema HACCAPP baze podataka

## 6.2. Web servis

HACCAPP WEB API je zadužen za komunikaciju s klijentima na poslužiteljskoj strani. Zvanje HACCAPP-a omogućeno je s klijentske strane korištenjem Retrofita. Ovisno o akcijama korisnika, Retrofit će generirati različite vrste upita na poslužitelju. Svaki će upit biti upućen na određenu tablicu u poslužiteljskoj bazi podataka.

**ApiController**

Za svaku tablicu s kojom je moguće raditi preko klijenta napravljen je kontroler. Kontroler je klasa koja nasljeđuje klasu ApiController i omogućuje primanje zahtjeva koji su namijenjeni određenoj tablici u poslužiteljskoj bazi. Klasa ApiController je dio imeničkog prostora *System.Web.Http*. To nazivlje sadrži klase za rad s HTTP-om. Osim nasljeđivanja klase ApiController, svaka controller klasa mora imati naziv koji završava s "*Controller*". Sve javne metode implementirane u kontroleru zovu se akcijske metode jer će biti pozvane kao odgovor na zahtjev s klijenta. Nazivi akcijskih metoda u controller klasama odgovaraju nazivu HTTP metoda na čije zahtjeve odgovaraju (Get, Post...).

## Arhitektura HACCAPP WEB API-a

HACCAPP WEB API podijeljen je na četiri dijela kako bi bili pokriveni svi korisnički zahtjevi i kako bi se olakšalo razvijanje aplikacije:

- *DatabaseAccess* – Predstavlja podatkovni sloj aplikacije. Sadrži model klase preko kojih je moguće raditi s podatcima u bazi.
- *Controllers* – Klase koje implementiraju ApiController i prihvaćaju zahtjeve od klijenata.
- *DocumentWorker* – Poseban dio aplikacije u kojem se generiraju dokumenti prije slanja.
- *Sender* – Klasa za prosljeđivanje dokumenata nadležnim korisnicima i za slanje PUSH notifikacija korisnicima ako dođe do greške ili upozorenja pri generiranju dokumenata.

## DatabaseAccess

DatabaseAccess je *ClassLibrary*[23] projekt unutar HACCAPP softverskog rješenja koji služi za dohvat podataka iz baze. Za rad s podatcima iz baze koristi se s model klasama koje se generirane preko *EntityFramework* alata. Prikaz tablica i njihovih veza prikazani su u slici 8.

*EntityFramework* je generirao klase s istim imenom kao i tablice iz baze. Te će klase biti korištene zajedno s kontrolerima. Kontroleri će odgovarati na REST zahtjeve poslane s uređaja i raditi direktno s podatcima u bazi.

## Kontroleri

Kontroleri su klase za primanje zahtjeva generiranih od strane HACCAPP mobilne aplikacije. Obzirom da svi odgovaraju na iste tipove zahtjeva (GET, PUT...), napravljeno je više kontrolera. Po jedan za svaku tablicu na poslužitelju. Značajka ovih klasa je da mogu raditi s ulaznim HTTP zahtjevima. Za rad s podatcima u bazi, stvoreni su kontroleri:

## EmployeeController

Kontroler klasa za rad s podatcima o korisnicima aplikacije. Na ovaj se kontroler primaju REST zahtjevi vezani za tablicu Employee iz baze podataka. Implementirane su metode:

### Get

Metoda koja ima povratni tip *IEnumerable<Employee>*. Ova će metoda vratiti sve korisnike koji se trenutno nalaze u bazi podataka.

Get metoda koja kao ulazni parametar ima identifikacijski broj ima povratni tip Employee. Metoda s definiranim brojem korisnika vratit će samo korisnika s tim identifikacijskim

[23] Biblioteka klasa, sučelja i vrijednosnih tipova koji se koriste za pristup do funkcionalnosti sustava

brojem. Ako je kao parametar proslijeđen id broj koji ne postoji u bazi podataka, bit će vraćen odgovor *BadRequest*. Svi će uspješni zahtjevi biti vraćeni u JSON formatu.

**Put**

Put je metoda koja u tijelu poruke dostavlja podatke o novom korisniku. Ova metoda će biti pozvana nakon registracije korisnika preko mobilne aplikacije. Ova metoda ima povratni tip *IHttpActionResult*. Ovisno o ispravnosti poslanih podataka, moguće je poslati natrag nekoliko odgovora. Prvo se radi provjera valjanosti modela. Za to postoji već implementirana metoda *ModelState.IsValid.* To je metoda koja vraća *bool* vrijednost. Ako je moguće mapirati ulazne vrijednosti iz zahtjeva u model iz baze podataka metoda će vratiti *true.* Ako metoda vrati *false*, na zahtjev će biti vraćen odgovor *BadRequestResult*. To je odgovor s kodom 400 koji obavještava pošiljatelja zahtjeva da zahtjev nije bilo moguće procesirati.

Ako je zahtjev prošao prvu validaciju, slijedi dodatna provjera ulaznih podataka. U prvoj je validaciji samo potvrđeno da zahtjev odgovara modelu, ali nije potvrđeno da su poslani podatci iskoristivi u sustavu. Zbog toga se i na poslužiteljskoj strani radi validacija podataka koje je korisnik unio. Obavezno je provjeriti email adresu korisnika *Regex* validatorom za email adrese.

Osim email adrese, provjerava se i lozinka koju je korisnik unio. Korisnik je pri registraciji obavezan unijeti lozinku dva puta, što se provjerava i na klijentskoj i na poslužiteljskoj strani.

Nakon lozinke, provjerava se poslovni objekt koji je korisnik unio. Da bi se zadovoljila validacija, korisnik mora biti povezan s već postojećim poslovnim objektom. Osim samog povezivanja s poslovnim objektom, provjerava se i rola korisnika s tim objektom. Ako je korisnik definiran kao Voditelj objekta kroz zahtjev za registraciju, obavezno je provjeriti postoji li već korisnik s takvom rolom na tom poslovnom objektu. Ako postoji, zahtjev neće proći validaciju. Ako zahtjev ne prođe sve dodatne validacije, bit će vraćen *BadRequest* odgovor. Ako validacija nije prošla, u poruci o grešci bit će poruka koja objašnjava koji dio zahtjeva nije prošao validaciju.

Uspješnim prolaskom svih validacija, novi je korisnik unesen u tablicu Employee. Pri uspješnom završetku, generira se odgovor s kodom 200, OK.

**Post**

Post metoda služi za ažuriranje podataka o korisniku. Kao ulazni parametar u tijelu poruke, prosljeđuje se korisnik kojeg čije podatke treba ažurirati na bazi. Metoda kao povratni tip ima *IHttpActionResult*. Pri zaprimanju zahtjeva, provodi se ista validacija kao i kod unosa korisnika kod Put metode. Kod neispravnih zahtjeva, vraća se odgovor tipa *BadRequestResult,* kao i kod Put metode.

Kroz ovu metodu nije moguće promijeniti ulogu korisnika. Promjena managera poslovnog objekta izvodi se preko posebne metode. Isto tako, nije moguća promjena poslovnog objekta

kojem korisnik pripada. Za promjenu poslovnog objekta, obavezno je stvoriti novi korisnički račun.

Uspješnim ažuriranjem korisnika, vraća se rezultat s kodom 200, OK.

**Delete**

Delete metoda služi za deaktivaciju korisničkog računa. Ulazni parametar poruke je korisnik tipa Employee kojeg treba deaktivirati. Povratni tip metode je *IHttpActionResult*. Pri zaprimanju, zahtjev prolazi kroz validaciju modela radi utvrđivanja ispravnosti. Neispravni zahtjev generira *BadRequest* odgovor.

Ako zahtjev prođe validaciju ispravnosti modela, radi se dodatna validacija. Dodatna validacija provjerava ulogu korisnika u njegovom poslovnom objektu. Ako se radi o korisniku koji je Voditelj svog poslovnog objekta, zahtjev se odbija s odgovorom *BadRequest*. Ako takav korisnički račun mora biti izbrisan, prvo je potrebno proslijediti ulogu managera na nekog drugog korisnika istog poslovnog objekta.

Kako bi se sačuvalo stanje u bazi, korisnika nije moguće obrisati iz baze podataka. Glavni razlog tome su arhivske tablice koje pamte koji je korisnik generirao koji dokument i kada. Te tablice sadržavaju strani ključ na redak u tablici Employee i brisanjem korisnika, morali bi biti obrisani i svi zapisi o njemu. Ova metoda omogućuje ažuriranje statusa korisnika u bazi podataka. Svi aktivni korisnici u bazi podataka imaju kolonu Status u stanju “A“. Kroz metodu Delete, ova se kolona ažurira na stanje “N“, neaktivan. Ukoliko je korisnik u statusu “N“, on se više neće moći prijavljivati u mobilnu aplikaciju, niti će moći izvršavati bilo kakve radnje koje se dozvoljene korisnicima aplikacije, ako se nekom greškom uspije prijaviti. Jednom kada je korisnički račun postao neaktivan, nije ga više moguće aktivirati. Na taj je način riješen problem čuvanja povijesnog stanja u bazi podataka i omogućena je trajna zabrana pristupu korisničkom računu, što kao rezultat ima privid brisanja korisnika.

Uspješnim izvršenjem zahtjeva, vraća se odgovor s kodom 200, OK.

**MigrateManager**

Metoda MigrateManager omogućuje migraciju uloge managera s jednog na drugog korisnika. Kao ulazne parametre, metoda prima dva identifikatora korinsika. Kao odgovor, metoda vraća *IHttpActionResult*.

Bez modifikacija, kontroleri ne mogu prihvatiti metode koje nemaju naziv Get, Put, Post ili delete. Kako bi se omogućilo zvanje metode MigrateManager, u global.asax datoteci projekta mora se promijeniti definicija rute. To se radi preko klase *Routes.MapHttpRoute.* Kroz tu se klasu definira ime rute, predložak rute i regularni poziv. Bez te modifikacije, poslužitelj će metodu posebnog naziva pokušati protumačiti kao jednu od Http metoda i vjerojatno će doći do greške.

Nakon uspješno pozvane metode, provodi se validacija nad dva identifikatora korisnika koji su proslijeđeni. Prvi id pripada korisniku s ulogom managera, a drugi pripada korisniku iz istog poslovnog objekta kao i Voditelj. To se provjerava validacijom na bazi. Ako je

validacija uspješna, korisnici će biti ažurirani tako da se prvom korisniku ukine uloga managera, a drugom dodjeli.

Ako je validacija zadovoljena, vraća se odgovor tipa OK. Ako validacija ne prođe generira se odgovor tipa *BadRequest*.

## CompanyController

Kontroler klasa za zaprimanje zahtjeva vezanih za Company tablicu u HACCAPP bazi podataka. CompanyController omogućuje ažuriranje podataka o poslovnim objektima kroz metode:

**Get**

Metoda s povratnim tipom *IEnumerable<Company>*. Omogućuje dohvat podataka o svim poslovnim objektima u bazi. Ako je metoda pozvana s parametrom id, dohvaća se samo odgovarajući objekt tipa *Company*. Ako se kroz parametar pokuša dohvatiti nepostojeći objekt, vraća se odgovor tipa *BadRequest*. U suprotnom, obje Get metode vraćaju dohvaćene objekte u JSON formatu.

**Put**

Metoda Put služi za dodavanje novog poslovnog objekta u bazu u tablicu Company. Metoda ima povratni tip *IHttpActionResult*. Put zahtjev generira se pri registraciji novog poslovnog objekta kroz aplikaciju.

Kod zaprimanja zahtjeva, radi se validacija modela. Ako objekt dostavljen u tijelu zahtjeva ne odgovara modelu iz baze podataka, dodavanje objekta se prekida i generira se odgovor *BadRequest*. Nakon validacije modela, slijedi validacija podataka koji se unose u tablicu. Prvo slijedi provjera PAN-a. Pan je brojčana vrijednost jedinstvena za pojedini objekt i predstavlja primarni ključ u tablici Company. Validacija je uspješna ako vrijednost se vrijednost PAN-a već ne nalazi u tablici. Ista se validacija provodi i za kolonu CompanyName. Svaki poslovni objekt mora imati jedinstveno ime u tablici Company. Zadnja validacija je provjera email adrese. Kroz aktivnost za registraciju poslovnog objekta, email adresa se unosi dva puta. To je obavezno kako ne bi došlo do slanja dokumenata na pogrešnu adresu. Ta se provjera nalazi i na klijentskoj strani, u aplikaciji. Osim toga, email adresa se validira i preko *Regex* validatora, slično kako i kod email adrese korisnika.

Ako jedna od validacija ne prođe, *BadRequest* odgovor će sadržavati poruku o tome koji dio validacije je izazvao grešku. Ako je validacija uspješna, novi poslovni objekt je dodan u tablicu Company i generiran je odgovor OK.

**Post**

Post metoda omogućuje ažuriranje podataka o poslovnim objektima. Povratni tip metode je *IHttpActionResult*. Ulazni parametar u tijelu zahtjeva je objekt tipa Company kojeg se ažurira. Validacija pri zaprimanju zahtjeva je identična onoj za kod Put metode.

Kroz ovu se metodu može ažurirati email adresa ili naziv poslovnog objekta. Parametar pri slanju zahtjeva je id broj korisnika koji šalje zahtjev. Poslovni objekt može ažurirati samo

Voditelj tog poslovnog objekta. Preko id broja u zahtjevu dohvaća se korisnik i provjerava se je li on zaista Voditelj tog objekta. Ako je provjera uspješna, poslovni objekt se ažurira i šalje se odgovor tipa OK.

**Delete**

Delete metoda omogućuje deaktivaciju poslovnog subjekta. Kao ulazni parametar, metoda prima poslovni objekt koji se treba deaktivirati i id korisnika koji je zatražio deaktivaciju. Povratni tip metode je *IHttpActionResult*.

Prije deaktivacije poslovnog objekta, provodi se validacija modela iz zahtjeva. Ako ulazni poslovni objekt odgovara modelu, prelazi se na sljedeću provjeru. Poslovni objekt nije moguće deaktivirati dok god na njemu ima aktivnih korisnika. Ako postoje aktivni korisnici, zahtjev će generirati *BadRequest* odgovor. Jedini način da se poslovni objekt trajno deaktivira je taj da zahtjev za deaktivacijom napravi Voditelj objekta kao zadnji aktivni korisnik. U tom slučaju i Voditelj i poslovni objekt su stavljeni u status "N", neaktivan. Isto kao i kod korisnika, deaktivirane poslovne objekte nije moguće ponovno aktivirati. Razlog zašto se brisanje provodi deaktivacijom je, kao i kod korisnika, čuvanje povijesnih podataka. Nakon uspješne deaktivacije, vraća se odgovor OK.

## DocumentController

DocumentController je poseban kontroler na koji se šalju zahtjevi sa klijentske strane kada se stvara novi dokument. Služi za provjeru i pripremu podataka prije stvaranja novog dokumenta.

**ReceiveDocument**

Metoda koja se poziva pri slanju zahtjeva na ovaj kontroler. Metoda ima povratni tip IHttpActionResult. Nakon inicijalne provjere tipova poslanih podataka, svi se podatci prosljeđuju DocumentWorker-u koji stvara i popunjuje novi dokument.

## DocumentWorker

DocumentWorker je poseban *ClassLibrary* unutar projekta, kao i DatabaseAccess. DocumentWorker služi za pohranu dokumenata u njihovom izvornom obliku. Kada poslužitelj zaprimi zahtjev za generiranje novog dokumenta, jedan se od postojećih predložaka kopira i ispuni podacima iz zahtjeva.

DocumentWorker sadrži sve dokumente koje je moguće generirati kroz HACCAPP aplikaciju. Svi su dokumenti unaprijed ispunjeni s *placeholder* vrijednostima. Kada se podatci o dokumentu ispune preko HACCAPP aplikacije, unesene vrijednosti se šalju u strukturi sličnoj rječniku. Kroz aplikaciju se prenose unesene vrijednosti, datum i vrijeme kada je zahtjev nastao, ime i prezime korisnika koji je poslao zahtjev i naziv dokumenta koji se treba ispuniti. Vrijednosti koje se šalju organizirane su tako da tvore ključ – vrijednost strukturu. Ključ naziva ImeKorisnika kao vrijednost će imati ime i prezime korisnika koji je poslao zahtjev. Za datum i vrijeme prosljeđuju se trenutne vrijednosti na mobilnom uređaju.

Ostale vrijednosti unešene preko aplikacije imat će dodijeljene ključeve ovisno o njihovom budućem položaju u dokumentu. Svaki dokument na poslužitelju ispunjen je *placeholder* vrijednostima. Kao ključevi za slanje podataka iz aplikacije koristit će se te *placeholder* vrijednosti.

Jednom kada se zahtjev zaprimi, iz poslane se strukture čita naziv dokumenta koji se treba popuniti poslanim podatcima. Ovisno o toj vrijednosti, bira se način i dokument koji se ispunjava. Slični dokumenti se ispunjavaju na isti način, ali postoji nekolicina posebnih dokumenata. Oni su zbog svoje strukture kompleksniji i zahtijevaju posebnu programsku logiku. Nakon odabira načina ispune dokumenata, kroz poslanu se strukturu iterira i traži iste vrijednosti ključa i oznake u odabranom dokumentu. Za svaki poslani podatak, pročita se njegov ključ, koji se zatim potraži u dokumentu. Ako je ključ pronađen, zamijeni vrijednost u dokumentu s poslanom vrijednošću iz zahtjeva. Za ovu se funkcionalnost koristi *Microsoft.Office.Interop.Word*. To je nazivlje koje omogućuje rad s *Office* aplikacijama. U ovom slučaju, koristi se za pozivanje *Word Application* objekta i otvaranje dokumenta koji se treba popuniti.

Nakon što je objekt otvoren, na njemu se poziva funkcija *FindAndReplace*. Funkcija pri pozivu prima parametre koji određuju na koji će se način izvršiti zamjena podataka. Uz ostale parametre, prosljeđuju se ključ pod kojim je pojedina vrijednost čuvana i sama vrijednost. Nakon izvršavanja, željena vrijednost će biti upisana u dokument. Isti se proces ponavlja za svaku poslanu vrijednost. Kada se sve vrijednosti iz zahtjeva prepišu u dokument, brišu se sve preostale *placeholder* vrijednosti i dokument je spreman za slanje.

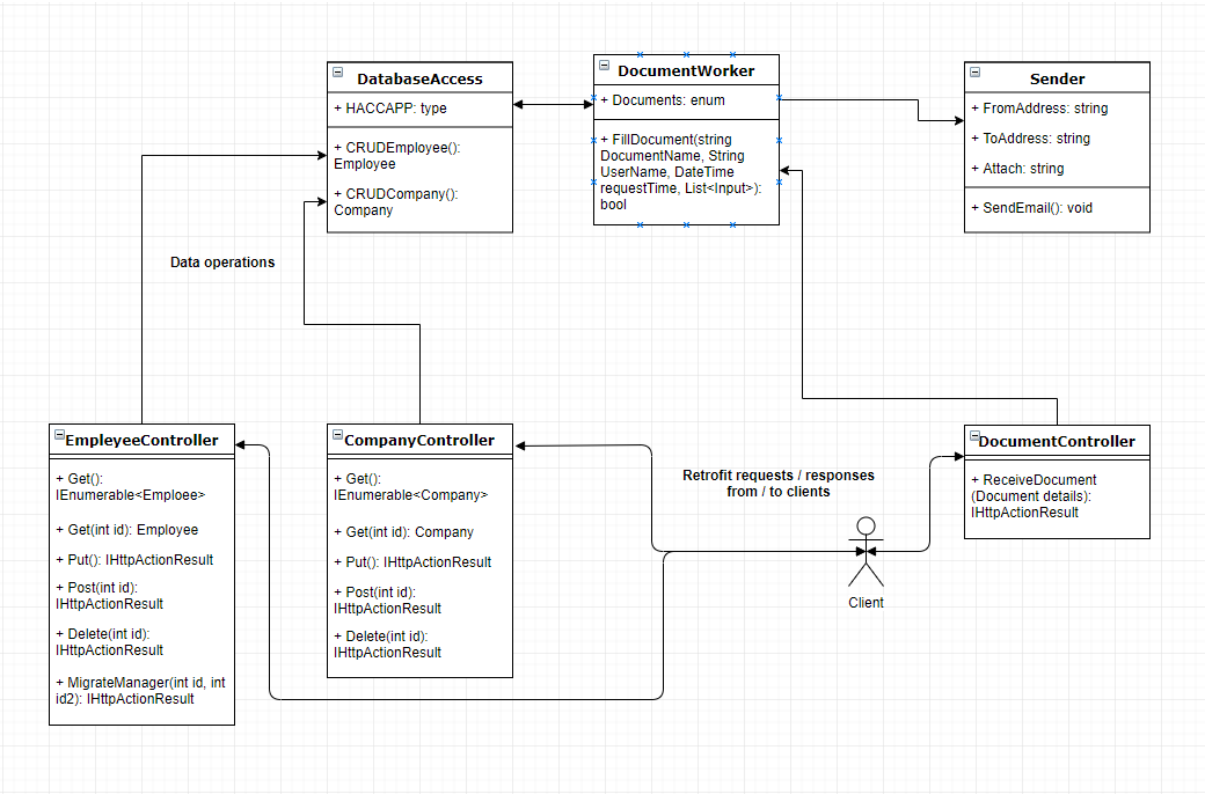
## Sender

Sender je *ClassLibrary* za prosljeđivanje popunjenih dokumenata na odgovarajuću email adresu. Email adresa na koju se dokument prosljeđuje ovisi o tome koji je korisnik pokrenuo zahtjev. Preko korisnika se može saznati kojem poslovnom objektu pripada i dokument će biti poslan na email adresu tog poslovnog objekta. Za slanje emaila koristi se *System.Net* nazivlje. Ono omogućuje slanje email poruka s dodatkom koristeći klasu MailMessage. Za slanje poruke koristi se *SMTP* protokol. Ovisno o tome koji email klijent poslovni objekt koristi, poruka se šalje preko drugačijeg SMTPServera i porta.

Kroz MailMessage objekt definira se adresa pošiljatelja (poslužitelja), adresa primatelja kao definirana email adresa na poslovnom objektu i generirani dokument kao dodatak.

## Slanje PUSH notifikacija

Kako bi se omogućilo slanje PUSH notifikacija na klijentsku mobilnu aplikaciju, prvo je potrebno registrirati Android projekt preko *Firebase* konzole. Ovaj je korak objašnjen u klijentskom dijelu aplikacije i omogućuje primanje PUSH notifikacija. Poslužitelj služi, između ostaloga, i za dojavu korisnicima da je došlo do problema s generiranjem dokumenata. Kako bi ih se o tome što brže obavjestilo, i podsjetilo na prepravke, koriste se PUSH notifikacije.




Slika 10  Veze između komponenti i način rada HACCAPP web API-ja

# 7. Klijentski dio sustava

Klijentski dio sustava predstavljen je Android mobilnom aplikacijom. Aplikacija je razvijena za Android Oreo, SDK 26. Razlog korištenja te verzije Androida je to da je u vrijeme početka razvoja ove aplikacije, to bio najraširenij Android SDK među ciljanom publikom koja će koristiti aplikaciju.

## 7.1. Komponente aplikacije

U ovom će poglavlju detaljnije biti razrađene razne komponente HACCAPP aplikacije koje korisnicima omogućuju brigu o dokumentima. U svim aktivnostima u aplikaciji, za povezivanje komponenata iz klase Activity s komponentama iz resursa, koristi se *ButterKnife*.

### AppController

AppController je klasa koja nasljeđuje klasu *Application*. Ona služi za čuvanje stanja kroz aktivnosti aplikacije. Kroz ovu se klasu čuvaju podatci o trenutno aktivnom korisniku, i sadrži podatke o Retrofit-u i omogućuje njegovo pozivanje iz bilo koje aktivnosti. Klasa koja nasljeđuje Application klasu koristi se slično kao klasa implementirana kao *Singleton* obrazac. Iako su slični, AppController je upravljan od strane programskog okvira i ima definiran životni vijek.

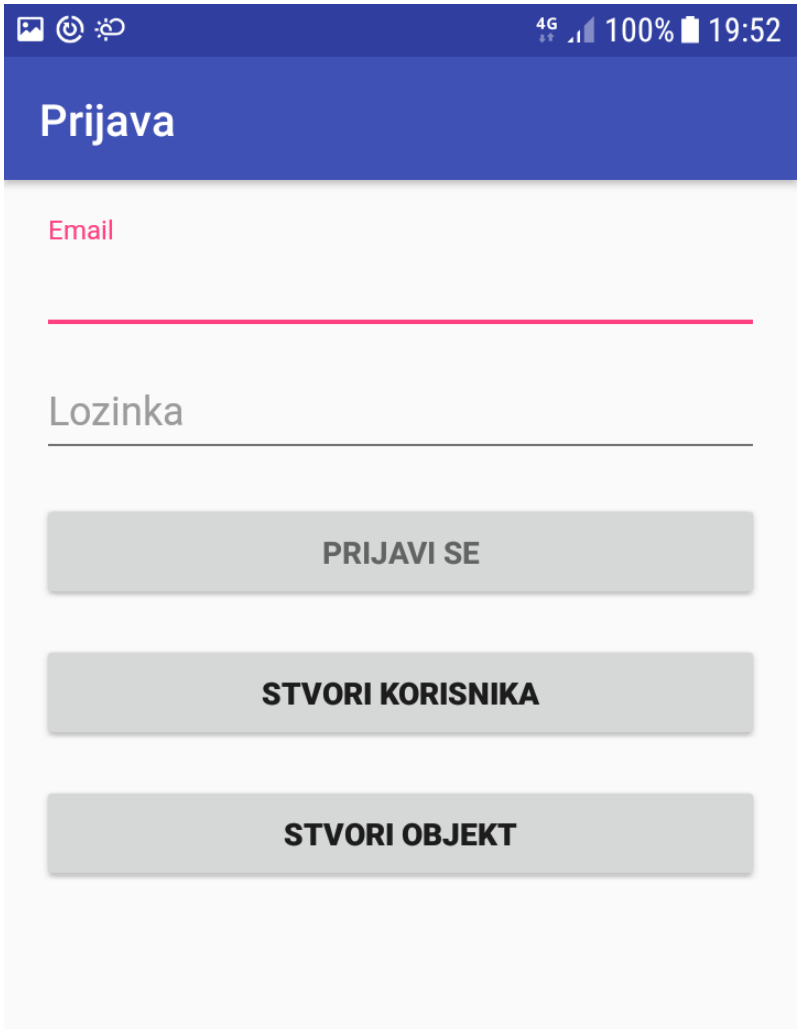
AppController je stvoren pri prvom pokretanju aplikacije i pamti stanja o aplikaciji dok god je ona aktivna. Kako bi se mogla koristiti, klasa mora biti definirana u Android Manifestu. Ona olakšava upravljanje potrošnjom resursa aplikacije omogućavanjem dohvata aplikacijskog konteksta kod ponovnih pokretanja aktivnosti aplikacije ako korisnik u nekom trenutku “pauzira“ aplikaciju otvaranjem neke druge aplikacije. Osim toga, AppController u sebi sadrži podatke o trenutno aktivnom korisniku kako bi se olakšao prijenos informacija između aktivnosti aplikacije. Omogućava i manje dohvata podataka iz baze podataka čuvanjem najčešće potrebnih vrijednosti.

### DBContext

DBContext predstavlja klasu za rad s lokalnom bazom podataka uređaja. Lokalna baza pamti korisnike, poslovne objekte i dokumente koji su generirani preko uređaja. Lokalna baza je implementirana kako bi se omogućio rad aplikacije u *offline* načinu rada. DBContext omogućuje rad s svim tablicama stvorenima preko *DBFlow*-a.

## LogInActivity

Ovo je početna aktivnost prikazana korisniku pri pokretanju aplikacije (Slika 11). Prozor se sastoji od dva polja za unos teksta i 3 gumba. Kućice za unos teksta predstavljaju prostor za unos podataka za prijavljivanje, korisnikove email adrese i lozinke. Gumbi služe za prijavljivanje korisnika u sustav, ili za registraciju novog korisnika ili poslovnog objekta. Ako je korisnik nov i nema kreiran korisnički račun prvo mora kreirati novi poslovni objekt, ili ako objekt već postoji, može kreirati korisnički račun.

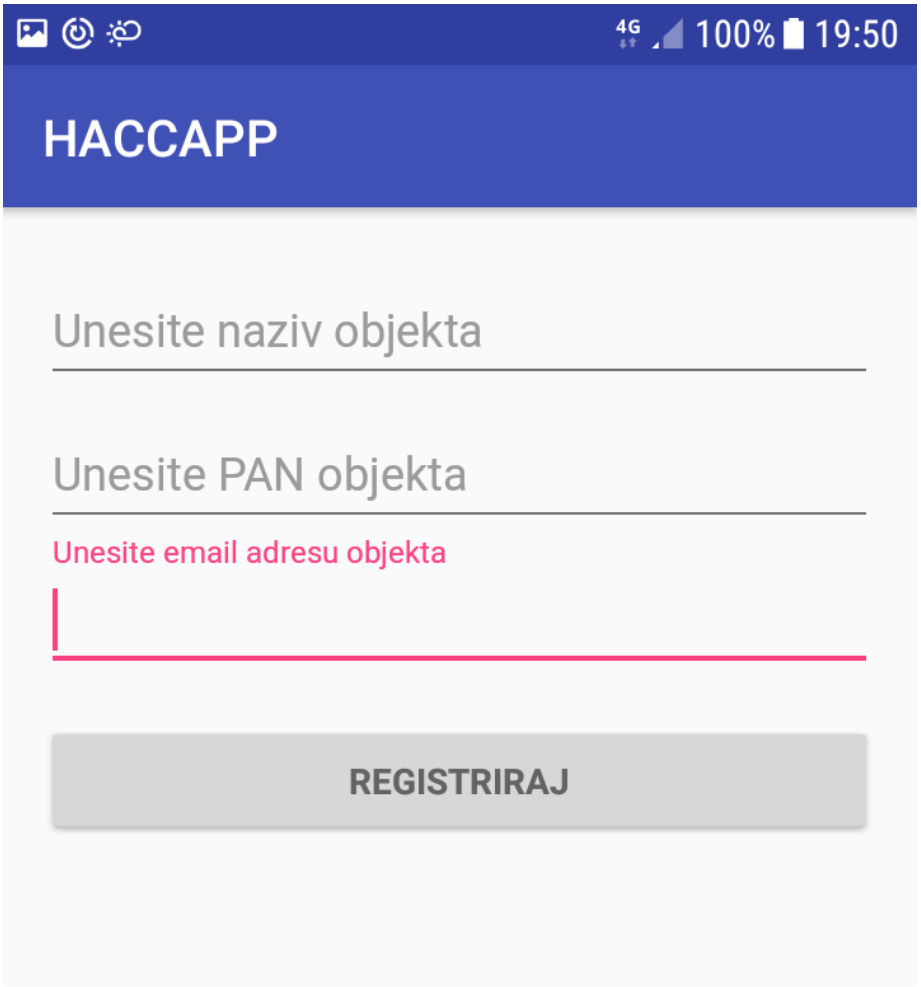



Slika 11 Aktivnost za prijavu

Korisnik je o neuspjehu prijave obaviješten Toast porukom. Ako je prijava uspješna, otvorit će se jedan od dva ekrana, ovisno o ulozi korisnika. Ako korisnik ima Voditelj rolu, otvorit će se prozor za upravljanje dokumentima poslovnog objekta. To je aktivnost kroz koju Voditelj odlučuje koje će dokumente korisnici istog poslovnog objekta moći koristiti. Ako korisnik nema rolu Voditelj, otvorit će se prozor s popisom omogućenih dokumenata za korisnikov poslovni objekt.

### RegisterCompanyActivity

Pritiskom na gumb "*Register new company*" u početnoj aktivnosti za prijavu, otvara se nova aktivnost za registraciju novog poslovnog objekta (Slika 12). Ova je aktivnost jednostavna i sadrži dva polja za unos teksta i jedan gumb. U poljima za unos teksta unose se naziv novog poslovnog objekta i vrijednost naziva PAN. PAN će biti korišten kao primarni ključ u poslužiteljskoj bazi podataka i mora biti jedinstvena vrijednost duljine minimalno 8 brojeva. Pritiskom na gumb "*Register*" poziva se metoda "*RegisterNewCompany*" na AppControlleru, koja će podatke o novom poslovnom objektu prenijeti sučelju koje implementira Retrofit. Preko Retrofita se šalje zahtjev koji sadrži vrijednosti za unos novog poslovnog objekta u poslužiteljsku bazu podataka. Osim u poslužiteljsku bazu, novi se poslovni objekt kreira i na lokalnoj bazi, što znači da korisnik može nastaviti s radom čak i kada nema pristup internetu ili kada WEB Api ne radi. Nakon uspjeha ili neuspjeha unosa podataka u bazu podataka, generira se Toast poruka koja obavještava korisnika o stanju unosa.




Slika 12 Aktivnost za registraciju novog poslovnog objekta

## RegisterUserActivity

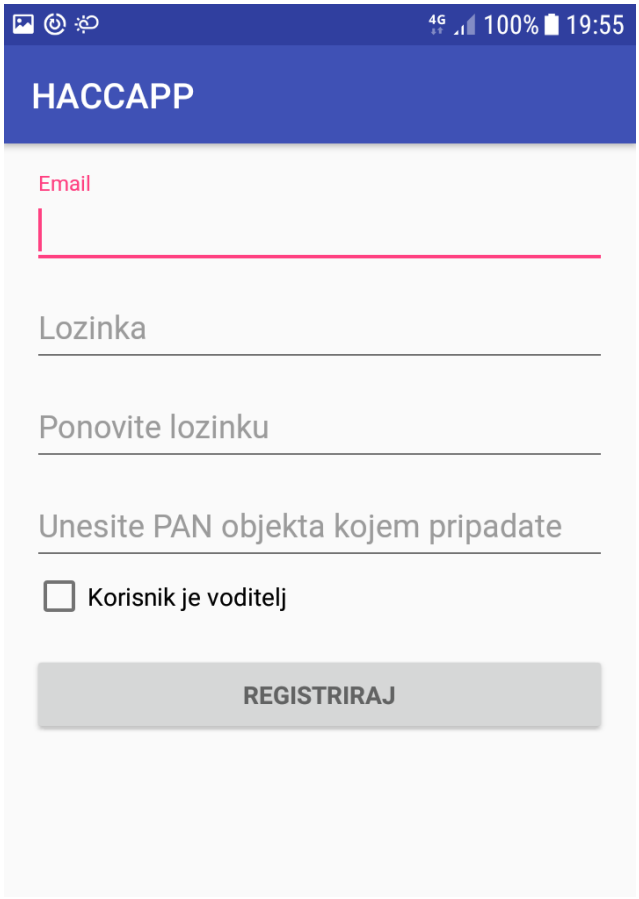
S aktivnosti za prijavu, gumb "*Register new user"* vodi na aktivnost RegisterUserActivity (Slika 13). Ova aktivnost sadrži četiri polja za unos, jednu kućicu za označavanje i jedan gumb. Kroz tekstualna polja unose se parametri korisničkog računa. Jedno je polje predviđeno za unos email adrese. Dva polja služe za unos lozinke i provjeru lozinke ponovnim unosom. Zadnje tekstualno polje služi za unos poslovnog objekta pod koji će novi korisnik spadati. Za povezivanje je potrebno unijeti PAN poslovnog objekta. Kroz kućicu za provjeru korisnik se označava kao Voditelj poslovnog objekta kojem se pridodaje. Kako bi bile unesene smislene vrijednosti, dodan je sustav validacije unosa.

Za validaciju email adrese koristi se regularni izraz koji prepoznaje email adresu. Android ima implementiran sustav provjere jednostavnijih predložaka (engl. *Patterns*). Za to se koristi biblioteka android.util.Patterns. Ona, između ostalog, sadrži i provjeru valjanosti email adrese.

Za provjeru lozinke koriste se dva tekstualna polja. Lozinka mora biti dulja od 8 znakova i unos u oba tekstualna polja mora biti isti za zadovoljavanje provjere. Lozinku je moguće promijeniti naknadno.

Polje za povezivanje s poslovnim subjektom ne validira se na klijentskoj strani. Ono će biti provjereno na poslužiteljskoj strani i unos mora odgovarati već postojećem PAN-u. Poslovni objekt nema ograničenje broja korisnika koji mogu biti povezani na njega. Ako je PAN krivo unesen, korisnik neće proći validaciju i neće biti stvoren. Dodatna provjera koja se vrši na poslužiteljskoj strani je provjera uloge korisnika. Ako je novi korisnik označen kao Voditelj, ali odabrani objekt već ima definiranog managera, korisnik neće proći validaciju.

Uspješnim prolaskom validacije, korisnik će biti dodan u lokalnu u poslužiteljsku bazu podataka. O uspjehu registracije bit će obaviješten generiranjem Toast poruke. Nakon potvrde da je registracija uspješno izvršena, korisnik se može prijaviti u aplikaciju na prozoru za prijavu.




Slika 13 Aktivnost za registraciju novog korisnika

## AvailableDocumentsActivity

AvailableDocumentsActivity je aktivnost koju mogu vidjeti samo korisnici koji imaju rolu Voditelj. Preko ovog prozora Voditelj odabire dokumente koje će ostali korisnici istog poslovnog objekta moći ispunjavati (Slika 14). Aktivnost je implementirana pomoću widgeta RecyclerView. To je naprednija verzija ListView-a i služi za prikaz kolekcije podataka u listi. Za korištenje ListView-a, obavezno je definirati resurs datoteku koja predstavlja jedan element u listi. Svaki element iz kolekcije koja puni RecicylerView bit će predstavljen jednim retkom u listi. Za prikaz elementa u RecyclerView obavezno je stvoriti Adapter klasu koja popunjava redak iz resurs datoteke s podatcima iz baze. RecyclerView se puni podatcima o dokumentima iz lokalne baze uređaja. Lokalna baza sadrži podatke o svim dokumentima koje je moguće ispuniti na poslužitelju. S obzirom na to da se rijetko dodaju novi dokumenti, i da korisnici ne mogu dodavati nove dokumente, podatci o dokumentima se čuvaju lokalno. Na taj je način izbjegnuto čekanje dohvata podataka o dokumentima s poslužitelja svaki put kad se pokrene ova aktivnost.

Definiranje dokumenata poslovnog objekta radi se preko dva gumba stvorena za svaki dokument iz lokalne baze. Jedan gumb dodaje dokument poslovnom objektu managera, a drugi ga uklanja. Veza između poslovnog objekta i dokumenta predstavljena je poveznom tablicom na poslužitelju. Ako je neki dokument već povezan s poslovnim objektom,

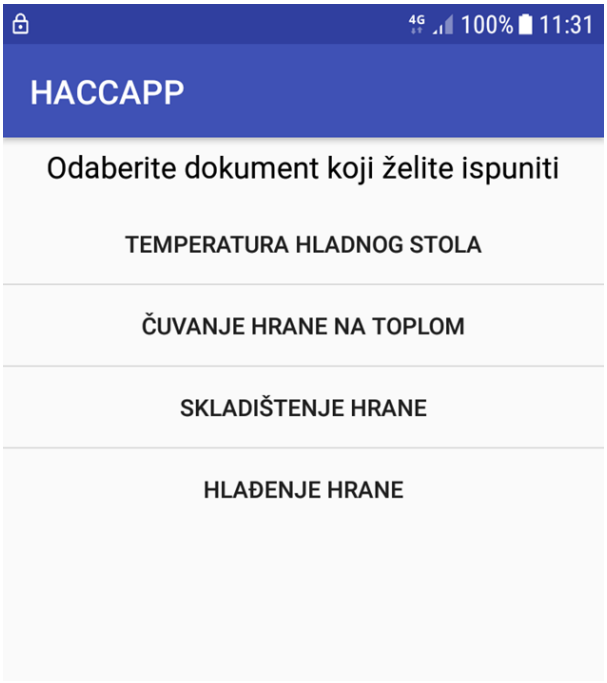
ponovnim dodavanjem ništa se neće dogoditi. Pritiskom gumba za uklanjanje dokumenta, poslužitelju se šalje zahtjev za brisanje poveznice dokumenta i poslovnog objekta iz baze. Svi pozivi WEB API-u napravljeni su odrađeni su preko Retrofita.

Osim RecyclerViewa, aktivnost sadrži i gumb koji pokreće aktivnost za ispunu dokumenata.


Slika 14 Izgled AvailableDocuments akitvnosti

## DocumentListActivity

Ovo je aktivnost u kojoj se prikazuju svi dokumenti koje korisnik pod nekim poslovnim objektom može ispunjavati (Slika 15). Te je dokumente definirao korisnik s rolom Voditelj. Dokumenti se prikazuju u ListView objektu. Lista omogućenih dokumenata dostavljena je preko WEB API-a. Pomoću te liste se puni kolekcija koja postaje izvor podataka za ListView. Kao i kod RecyclerViewa, za korištenje ListViewa obavezno je definirati resurs datoteku koja predstavlja jedan redak u listi. Dokumenti se ispunjavaju u zasebnoj aktivnosti, do koje se dolazi pritiskom na jedan od dokumenata.




Slika 15 Izgled DocumentList aktivnosti

## FillDocumentsActivity

FillDocuments aktivnost omogućuje korisnicima ispunjavanje formi koje će se koristiti za generiranje dokumenata. Ovisno o tome koji dokument korisnik odabere, prikazat će mu se odgovarajući prozor za ispunu dokumenata. Određeni dokumenti mogu koristiti jednake forme za ispunu zbog sličnog sadržaja, ali oni zahtjevniji imaju vlastitu formu za ispunu.

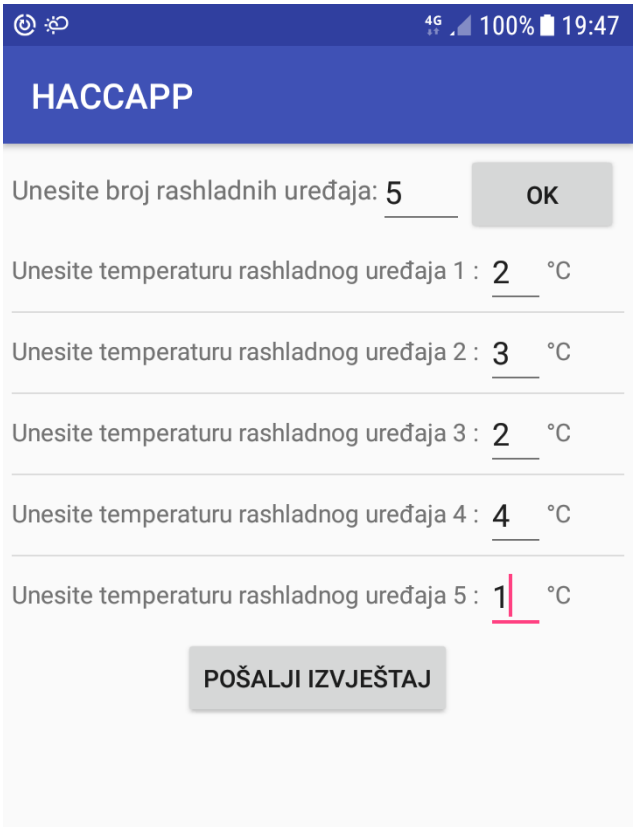
Kako bi se uvijek prikazala ispravna forma za dokument, proširena je lokalna tablica koja čuva podatke o svim dokumentima. Dodano je polje koje sadrži ime klase koja se koristi za ispunu te vrste dokumenata. Kada korisnik odabere koji dokument želi ispuniti, iz lokalne baze se dohvaća trenutna vrijednost tog polja. Vrijednost polja odgovara punom imenu jedne od klasa koja predstavlja prozor za popunjavanje dokumenta. Pri pozivanju nove aktivnosti kroz Intent objekt, polja forme za ispunu bit će dohvaćena preko refleksijskog okvira, ovisno o tome koji je dokument korisnik odabrao.

Aktivnost koja prikazuje forme za unos podataka ima tri važna elementa, EditText polje u koje korisnik upisuje koliko redaka u dokumentu treba popuniti. Nakon pritiska OK gumba, popunjava se ListView element. Ovisno o tome koji dokument je korisnik odlučio ispunjavati, elementi ListView-a će biti drugačiji. Ta je funkcionalnost omogućena korištenjem prilagođenog ListView-a koji može prikazati više različitih tipova redaka.

Nakon pronalaska određenog tipa retka kroz refleksijski okvir, ListView se popunjava. U primjeru (Slika 16), korisnik je odabrao da želi ispuniti dokument "Evidencija temperature u rashladnim uređajima". Nakon unosa broja rashladnih uređaja, prikazan je uneseni broj redaka za popunjavanje temperature rashladnih uređaja. Da bi se izvještaj mogao poslati pritiskom na gumb, svi elementi u retku moraju biti ispunjeni, u suprotnom, korisniku će biti prikazana Toast poruka o grešci. Poruka o grešci se ispisuje i ako korisnik unese krivi tip podatka u polja za unos.

Pritiskom na gumb generira se zahtjev prema poslužitelju. Tipovi zahtjeva su različiti, ovisno o tome koji se dokument ispunjava. Podaci koji se šalju sadržani su u klasi stvorenoj za čuvanje podataka određenog tipa dokumenta. Svaki zahtjev sadrži identifikacijski broj korisnika koji je napravio zahtjev, točno vrijeme i datum kada je poslao zahtjev i listu elemenata koji predstavljaju njegov unos u formi za ispunjavanje dokumenata. Svaki od tih elemenata čuva se u strukturi ključ – vrijednost. Kada se popunjeni elementi iz liste na formi pretvaraju u listu koja se šalju u zahtjevu, dodijeljen mu je ključ, ovisno o njegovoj poziciji u ListView elementu. Taj ključ govori na koje mjesto u generiranom dokumentu poslana vrijednost treba doći. Neke se vrijednosti u listi iz zahtjeva šalju prazne, ali s ključem. One će biti popunjene na poslužiteljskoj strani kako bi se olakšao rad korisniku.

Nakon uređivanja svih elemenata, zahtjev se šalje pomoću Retrofita kroz AppController. ulazni se zahtjev pretvara i šalje u JSON formatu prema poslužitelju. Primjer generiranog dokumenta dan je među prilozima rada, prilog broj 1.




Slika 16 Ispuna vrijednosti za dokument o evidenciji rashladne temperature

## 7.2. Baza podataka

Na lokalnom uređaju koristi se DBFlow baza podataka. Svi podatci pohranjeni na uređaju, osim dokumenata koje je moguće ispunjavati, vezani su za korisnika koji koristi aplikaciju. Preko lokalne baze nije moguće doći do podataka o drugim korisnicima niti poslovnim objektima. DBFlow baza podataka na uređaju sadrži iste tablice za čuvanje podataka o korisnicima kao i baza podataka na poslužitelju.

Iznimke su tablice Employee i Document, koje su na mobilnoj verziji proširene kako bi se sačuvali podaci o korisnicima ili kako bi se omogućile posebne funkcionalnosti dostupne samo na mobilnom uređaju.

Sve CRUD operacije na mobilnoj bazi podataka odvijaju se preko DBContext klase.

## Document

Tablica Document proširena je da sadrži polje "*Template*". To polje služi za definiranje predloška koji će se generirati kod mapiranja podataka iz aktivnosti *FillDocumentsActivity*. *Template* radi tako da definira ključ vrijednost svake ćelije koja će biti popunjena u tablici u pojedinom dokumentu. Osim podataka o tablicama, template definira i mjesto za datum generiranja dokumenta, ime i prezime zaposlenika koji je generirao zahtjev i ime voditelja poslovnog objekta kojem pripada korisnik koji je generirao dokument. Dio predloška koji generira podatke u tablicama poseban je za svaki dokument koji se može generirati.

## Employee

U tablicu Employee dodana je nova kolona "PUSHKey". Ova kolona služi za čuvanje podataka o tokenu koji se koristi kod slanja PUSH notifikacija pojedinom uređaju. Radi se o token vrijednosti koja se generira na mobilnoj aplikaciji. Tu je vrijednost moguće dobiti pozivanjem *getAccessToken* metode prema FCM poslužitelju. Taj zahtjev vraća vrijednost tokena koja će se dalje pohraniti u bazi i koristiti kod slanja PUSH notifikacija pojedinom korisniku.

# Zaključak

Jedan od preduvjeta za povećanje profitabilnosti u svakom poslu je pravilno upravljanje resursima, koji su naravno, ograničeni. U mnogim prehrambenim objektima koji obavezno moraju voditi HACCP dokumente, to nije riješeno efikasno. Zbog neorganiziranosti se često događa da ispunjavanje i dostavljanje tih dokumenata nadležnima uzima više vremena nego što je potrebno.

HACCAPP aplikacijom to se pokušava promijeniti. Kroz aplikaciju je omogućen pristup samo do onih dokumenata koji su relevantni zaposleniku. Samo ispunjavanje dokumenata napravljeno je preko forme koja zahtjeva unos samo najosnovnijih podataka za generiranje dokumenta. Svi podatci koji se mogu sami upisati ne traže unos od korisnika, a dokument se nakon popunjavanja preko poslužitelja šalje direktno voditelju poslovnog objekta. Sustav je prilagođen zaposleniku prehrambenog objekta i zamišljen je kako bi osigurao što veću uštedu vremena. Taj je cilj uspješno realiziran, obzirom da omogućuje korisniku više vremena za bavljenje njegovom primarnom aktivnosti, i obzirom da aplikacija omogućuje neupućenoj osobi da samostalno stvori HACCP dokumente.

Kao i sve aplikacije, HACCAPP će se mjenjati ovisno o zahtjevima i željama klijenata. Iako će uvijek biti aplikacija za popunjavanje HACCP dokumenata, način pristupa tom problemu može se u budućnosti promijeniti, kao i broj dokumenata koji će se ispunjavati.

Student vlastoručno potpisuje Završni rad na prvoj stranici ispred Predgovora s datumom i oznakom mjesta završetka rada te naznakom:

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

*U Zagrebu, 19.5.2018..*

# Popis kratica

| | | |
|---|---|---|
| HACCP | *Hazzard Analysis and Critical Control Point* | Analiza opasnosti i kontrola kritičnih točaka |
| MSSQL | *Microsoft Sql server* | Microsoft SQL poslužitelj |
| REST | *Representational State Transfer* | Reprezentacijski prijenos stanja |
| JSON | *JavaScript Object Notation* | Javascript objektna notacija |
| XML | *EXtensible Markup Language* | Prošitivi označni jezik |
| URL | *Uniform Resource Locator* | Ujednačen lokator sadržaja |
| MVP | *Model View Presenter* | Model, pogled, predlagač |
| API | *Application Programming Interface* | Aplikacijsko programsko sučelje |
| HTTP | *HyperText Transfer Protocol* | Protokol za prijenos hiperteksta |
| CRUD | *Create, Read, Update, Delete* | Stvori, čitaj, ažuriraj, obriši |
| WCF | *Windows Communication Foundation* | Windows komunikacijska osnova |
| ISS | *Information Security Standard* | Informacijski sigurnosni standard |
| SOAP | *Simple Object Access Protocol* | Jednostavan protokol za pristupanje objektima |
| ORM | *Object Relational Mapping* | Povezno mapiranje objekata |
| EF | *Entity Framework* | Entitetski okvir |
| SMTP | *Simple Mail Transfer Protocol* | Jednostavan protokol za prijenos maila |
| LINQ | *Language-Integrated Query* | Jezično-Integriran upit |
| ADO | *ActiveX Data Objects* | ActiveX podatkovni objekti |
| SDK | *Software Development Kit* | Programska razvojna oprema |

# Popis slika

# Literatura


[1] Android MVP; http://www.ee.surrey.ac.uk/Projects/CAL/networks/Network-Transport_Application_Layers.htm; 12.4.2018

[2] B.PHILLIPS, C.STEWART, B.HARDY, K.MARSCIANO; Android Programming: The Big Nerd Ranch Guide (2nd Edition); Big Nerd Ranch Guides (2017)

[3] DBFLOW; https://guides.codepath.com/android/DBFlow-Guide; 9.4.2018.

[4] D.GRIFFITHS, D.GRIFFITHS; Head First Android Development; O'reily Media (2015)

[5] E.HELLMAN; Android programming: pushing the limits; Wiley (2013)

[6] Entity Framework; https://msdn.microsoft.com/en-us/library/ee712907(v=vs.113).aspx; 21.4.2018.

[7] Firebase; https://firebase.google.com; 25.04.2018.

[8] J.BOŽIĆ, T.KADEŽABEK, F.TOMIĆ, T.KAŠTELAN; Izrada aplikacija za mobilne uređaje; Algebra (2013)

[9] J.LOWY, M.MONTGOMERY; Programming WCF Services, 4th Edition; O'reily Media (2015)

[10] LINQ; https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/introduction-to-linq-queries; 15.4.2018.

[11] MORTIMORE, C.WALLACE; HACCP Apractical approach; Springer (2013)

[12] MSSQL; https://www.microsoft.com/en-us/sql-server/sql-server-2016; 18.4.2018.

[13] POSTMAN; https://www.getpostman.com/; 29.4.2018.

[14] Retrofit; http://square.github.io/retrofit; 29.3.2018

[15] SMTP Network layer; http://www.ee.surrey.ac.uk/Projects/CAL/networks/Network-Transport_Application_Layers.htm; 1.5.2018.

[16] Web API infrastructure; https://blogs.msdn.microsoft.com/martinkearn/2015/01/05/introduction-to-rest-and-net-web-api/; 18.4.2018

[17] WEB API https://www.asp.net/web-api; 18.4.2018.

# Prilog

Diplomski rad može imati priloge, ali se oni ne prilažu uz pisanu verziju diplomskog rada već se mogu priložiti na diplomskom ispitu ukoliko povjerenstvo na diplomskom ispitu tako odluči. Važno je čuvati svu povratnu dokumentaciju koja je nastala pri izradi diplomskog rada.

S unutarnje strane na zadnjim koricama originala, kao i svake kopije diplomskog rada, pričvršćuje se CD s kompletnim diplomskim radom u izvornom formatu (npr. .docx) i .pdf formatu sa svom popratnom dokumentacijom i programima. Pri čemu je obvezno da na tom CD- u postoji i dokument koji opisuje kako se rezultat njegova diplomskog rada (softver ili hardver) koristi (ili kako se npr. izvode mjerenja koja je opisao u radu). Ako se radi o softveru nužno je opisati i kako se programska podrška instalira.