IZRADA APLIKACIJE ZA VOĐENJE EVIDENCIJE O PČELINJACIMA

Severović, Željko

VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

IZRADA APLIKACIJE ZA VOĐENJE EVIDENCIJE O PČELINJACIMA

Željko Severović

Zagreb, siječanj 2020.

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, datum.

Ime Prezime

Predgovor

Zahvaljujem svim profesorima i asistentima Visokog učilišta Algebra koji su poticali moj napredak i širili granice mojih mogućnosti od prvih koraka u svijetu programiranja. Posebice zahvaljujem mentoru Danielu Beleu na njegovu prenesenom znanju i savjetima tijekom studija i prilikom izrade ovog rada.

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

Sažetak

Cilj je ovog završnoga rada prikazati koristi i načine primjene modernih tehnologija u tradicionalnim zanimanjima. Iako je fokus završnog rada orijentiran na pčelarstvo, slični su principi primjenjivi i u drugim tradicionalnim zanimanjima. Rezultat ovog završnog rada je sustav koji pčelarima omogućuje praćenje i planiranje njihovih aktivnosti, dok istovremeno osobama od interesa omogućuje uvid u stvarno stanje pčelinjaka i ključne podatke za planiranje radnji s ciljem smanjenja gubitaka pčelinjih zajednica.

Ključne riječi: sustav, moderne tehnologije, tradicionalna zanimanja, pčelarstvo

Abstract

The main goal of this thesis is to show benefits and application of modern technologies in traditional professions. The focus of this thesis is orientated on beekeeping but similar principles are applicable to other traditional professions. End result of this thesis is system that provides to beekeepers a way to monitor and plan their activities and at the same time provides to persons of interests insight into current conditions of apiaries and provide key data for planning future actions with goal on reducing bee losses.

Keywords: system, modern technologies, traditional professions, beekeeping

Sadržaj

1. Uvod 1

2. Arhitektura sustava i baza podataka 2

3. Azure okruženje i servis 4

3.1. Postavljanje baze podataka 4

3.2. Postavljanje web-servisa 5

4. Android 7

4.1. Povijest i verzije 7

4.2. Konfiguracija android aplikacija 8

4.3. Korištenje ugrađenih programskih sučelja Android ekosustava 9

5. Implementacija web-servisa kao serverskog dijela aplikacije 11

5.1. Autentikacija 11

5.2. Integracija s bazom podataka 12

5.3. Pružanje usluga u JSON formatu 13

5.4. Metrika o pčelinjacima 14

5.5. Upravljanje slanjem obavijesti 16

6. Implementacija klijentskog dijela aplikacije 18

6.1. Komunikacija android mobilne aplikacije sa serverom 18

6.2. Prijava korisnika 19

6.3. Glavno sučelje aplikacije 20

6.3.1. *NavigationDrawer* 21

6.3.2. *FragmentLoader* sučelje 22

6.4. Funkcionalnosti aplikacije 23

6.4.1. Popis pčelinjaka 23

6.4.2. Dodavanje pčelinjaka 26
6.4.3. Popis i dodavanje košnica 28
6.4.4. Pregled košnice 29
6.4.5. Povijest pregleda 30
6.4.6. Prikaz prinosa meda 32
6.4.7. Prikaz registriranih senzora i izmjena konfiguracije 33
6.4.8. Prikaz očitanja sa senzora 33
6.4.9. Primanje obavijesti 35
7. Automatsko prikupljanje podataka korištenjem senzora 37
7.1. Senzori temperature i vlage 37
7.2. Način rada i komunikacije s poslužiteljem 38
7.3. Aplikacija za prikupljanje podataka 38
Zaključak 41
Popis kratica 42
Popis slika 43
Popis tablica 44
Popis kôdova 45
Literatura 47

1. Uvod

HoneyComb sustav osmišljen je kao niz povezanih aplikacija pomoću kojih je pčelarima omogućeno bilježenje svakodnevnih radnji i pregledavanje stanja na njihovim pčelinjacima. Klijentska strana sustava sastoji se od android aplikacije namijenjene pčelarima kao sučelja za unos i praćenje stanja na vlastitim pčelinjacima, Java aplikacije za automatsko prikupljanje podataka sa senzora na pčelinjacima te web-aplikacije koja prikazuje statističku analizu na razini svih pčelinjaka za koje se vodi evidencija kroz sustav. Klijentske su aplikacije međusobno povezane kroz središnji dio sustava, web-servis kojem je namjena spremanje i obrada podataka iz središnje baze podataka.

Izvori ovog rada temelje se na službenim internet stranicama tehnologija i sustava koji su korišteni kroz izradu rada, *Wikipediji* i knjigama navedenim u poglavlju **Literatura**.

Početna su poglavlja posvećena arhitekturi sustava, platformi koja poslužuje sustav i opisu android operativnog sustava te njegovu razvoju kroz povijest.

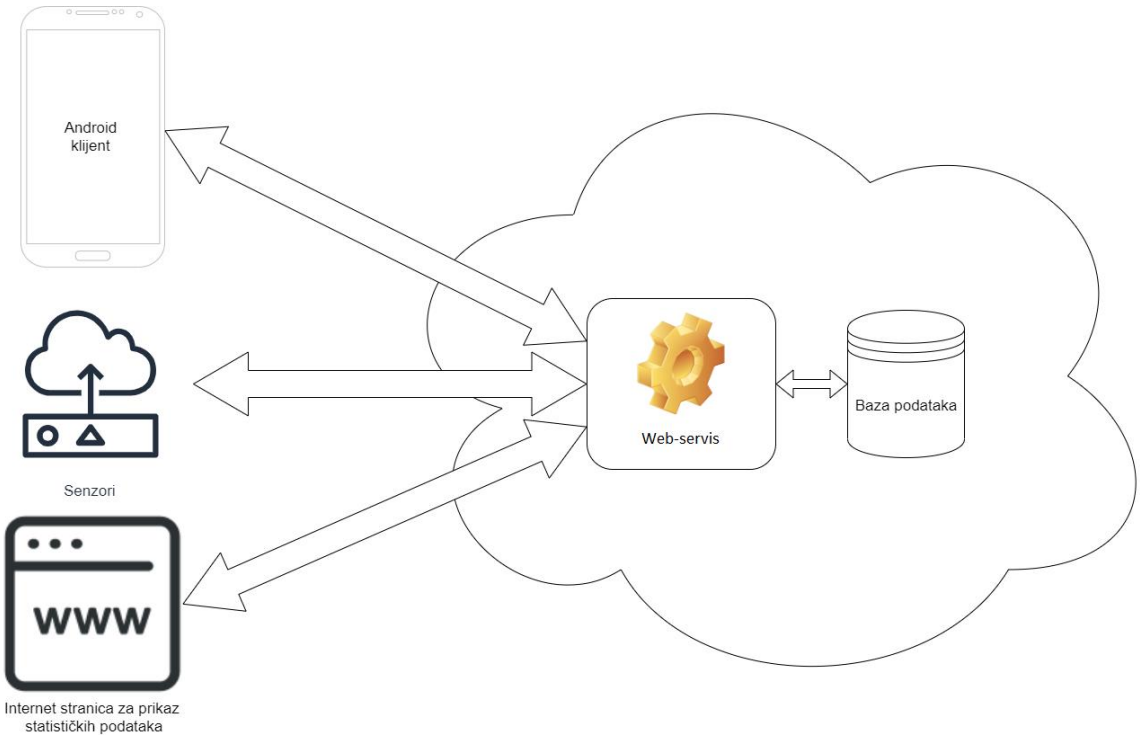
Peto je poglavlje ovog rada posvećeno središnjem djelu sustava, web-servisu i sustavu za slanje obavijesti.

Šesto poglavlje opisuje korisničko sučelje za unos podataka implementirano u obliku android aplikacije preko koje korisnici mogu unositi zapažanja tijekom pregleda te pregledavanje podataka o prijašnjem i trenutnom stanju na vlastitim pčelinjacima.

Sedmo je poglavlje posvećeno automatskom prikupljanju podataka s pčelinjaka upotrebom senzora.

2. Arhitektura sustava i baza podataka

HoneyComb sustav sastoji se od web-servisa i baze podataka preko kojih je povezano više klijentskih aplikacija te senzorskog sustava koji omogućuje prikupljanje podataka. Arhitektura sustava prikazana je na slici (Slika 2.1).



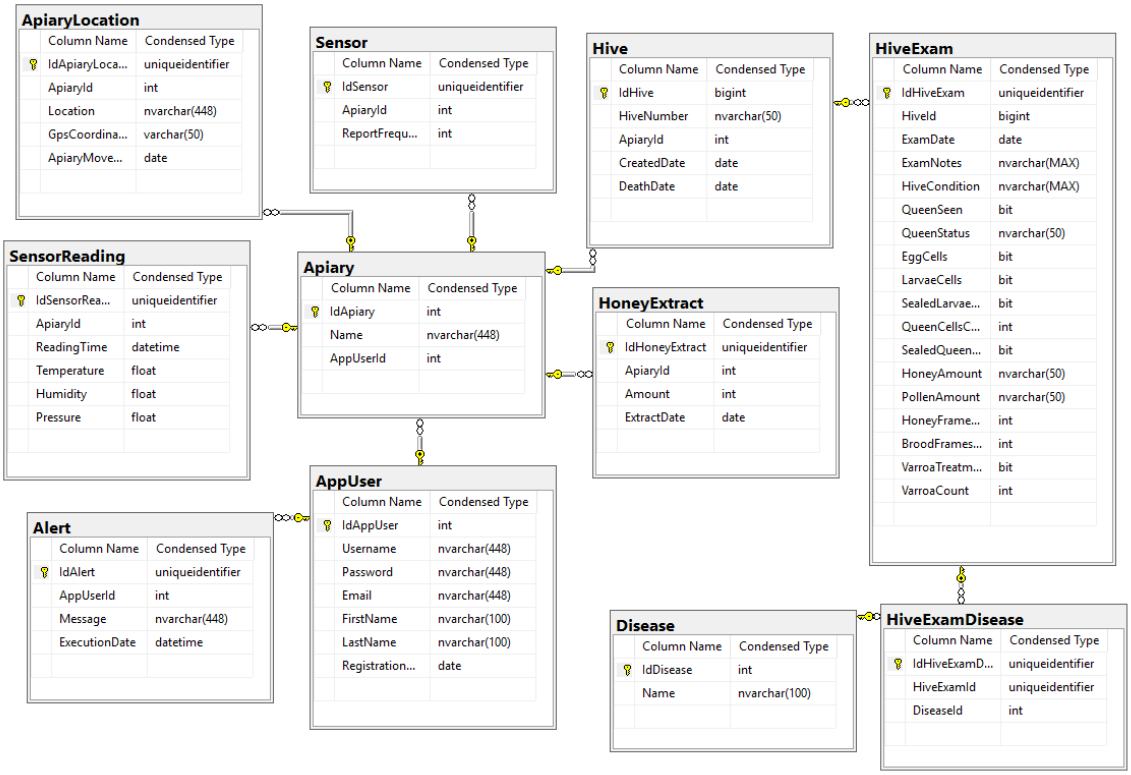
Slika 2.1 Arhitektura sustava

Svi podaci uneseni u sustav spremaju se u zajedničku bazu podataka pod korisničkim računima koje korisnici izrađuju pri registraciji, što im omogućuje pregledavanje i unos podataka o vlastitim pčelinjacima s više uređaja istovremeno. Korisnicima sustava omogućeno je automatsko prikupljanje podataka o temperaturi i vlazi zraka na pčelinjaku, koje je za potrebe ovog rada izvedeno korištenjem *Raspberry Pi* računala. Podaci, koje prikupe senzori, šalju se web-servisu koji ih sprema u bazu podataka. Nakon što su podaci spremljeni, korisnik ih može pregledavati prijavljivanjem na korisnički račun kroz android aplikaciju i na taj način imati uvid u vremenske prilike na pčelinjaku.

Podaci spremljeni u bazu podataka služe i kao izvor za anonimnu analizu te realan prikaz stanja na pčelinjacima. Klijentske aplikacije u svakom trenu mogu dohvatiti podatke o

bolestima i prinosima meda s web-servisa, bez potrebe za prethodnom registracijom u sustav, koji su za potrebe ovog rada prikazani na internet stranici.

Središnja je baza podataka *HoneyComb* sustava izgrađena na relacijskom modelu baza podataka. U ovom modelu svaki entitet ima vlastitu tablicu čiji redci posjeduju ključeve (engl. *keys*). Ključ predstavlja jedinstveni identifikator retka kojim se prepoznaje pripadnost drugom entitetu prema relacijama definiranim u shemi baze podataka.



Slika 2.2 Shema baze podataka

Kao što je prikazano na slici (Slika 2.2), polazna je točka unutar baze podataka, a i samog sustava, *AppUser* entitet koji predstavlja korisnika. Korisnik može imati neograničen broj pčelinjaka, predstavljenih entitetom *Apiary*. Pčelinjaci se smatraju središtem sustava jer se na njih odnosi većina unesenih podataka od kojih su najvažniji: unos meda, entitet *HoneyExtract*, očitanja sa senzora, entitet *SensorReading* i košnice, entitet *Hive*, koje sadrže neograničen broj pregleda tijekom kojih se unose uočene bolesti, entitet *Disease.*

3. Azure okruženje i servis

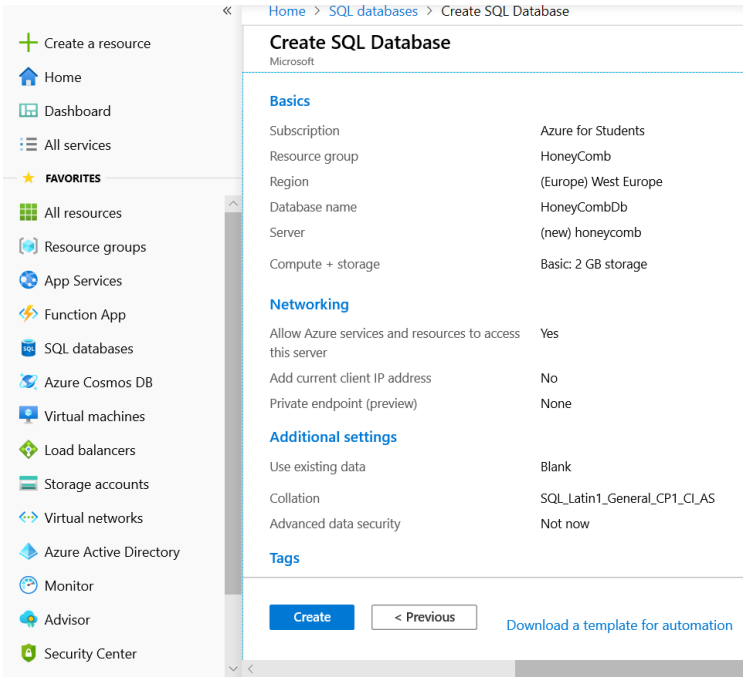
Računanje u oblaku (engl. *Cloud computing*) predstavlja način na koji računalna infrastruktura, aplikacije i poslovni procesi mogu biti isporučeni kao usluga, kada i gdje je potrebno, bez potrebe za izgradnjom vlastite IT infrastrukture. Jedan od glavnih razloga popularnosti ove usluge njezina je elastičnost, tj. mogućnost dodavanja i oduzimanje resursa prema potrebi, bez kupnje dodatnog komada hardvera [1].

Microsoft Azure je Microsoftova platforma za računanje u oblaku, dostupna u 140 zemalja. Namijenjena je izgradnji, testiranju i upravljanju aplikacijama i servisima u Microsoft podatkovnim centrima koji su grupirani u više od 50 regija diljem svijeta [2]. U izradi ovoga završnog rada korištene su Microsoft Azure usluge za posluživanje baze podataka i web-servisa.

3.1. Postavljanje baze podataka

Za pohranu podataka odabrana je Azure SQL baza podataka, relacijska baza podataka opće namjene. Navedena baza podataka temelji se na najnovijoj stabilnoj verziji Microsoft SQL servera i pruža visoku razinu dostupnosti i performansi za potrebe pohrane podataka u Azureu [3].

Azure SQL baza podataka postavlja se preko Azure portala. Koristeći čarobnjak za izradu nove baze, postavljaju se sve željene postavke, poput naziva virtualnog servera, odabir lokacije, tj. Azure regije, u kojem će se nalaziti baza podataka, postavke pristupa, procesorska snaga i količina željenog prostora. Procesorsku snagu i zakupljeni prostor za pohranu podataka moguće je, prema potrebi, jednostavno proširiti.

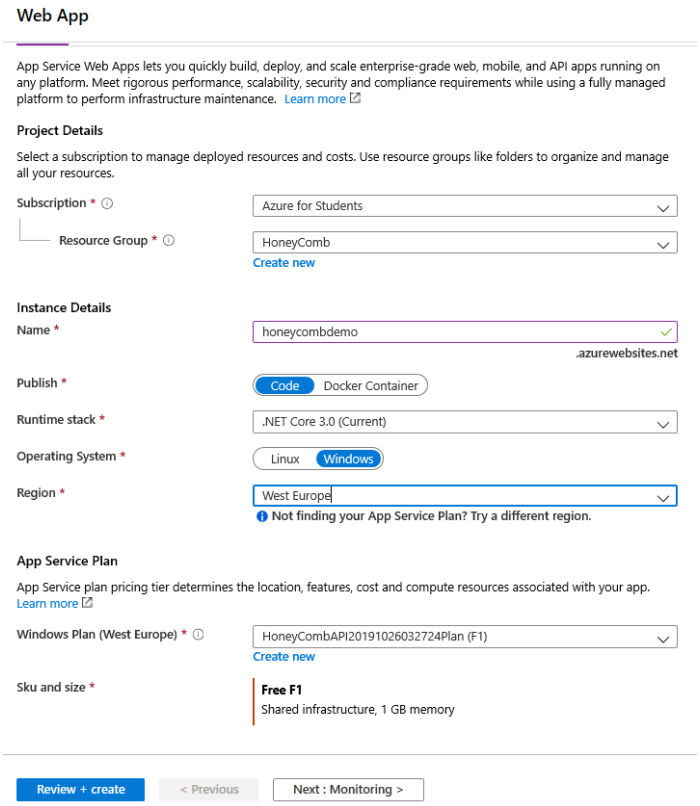


Slika 3.1 Postavljanje Azure SQL baze podataka

3.2. Postavljanje web-servisa

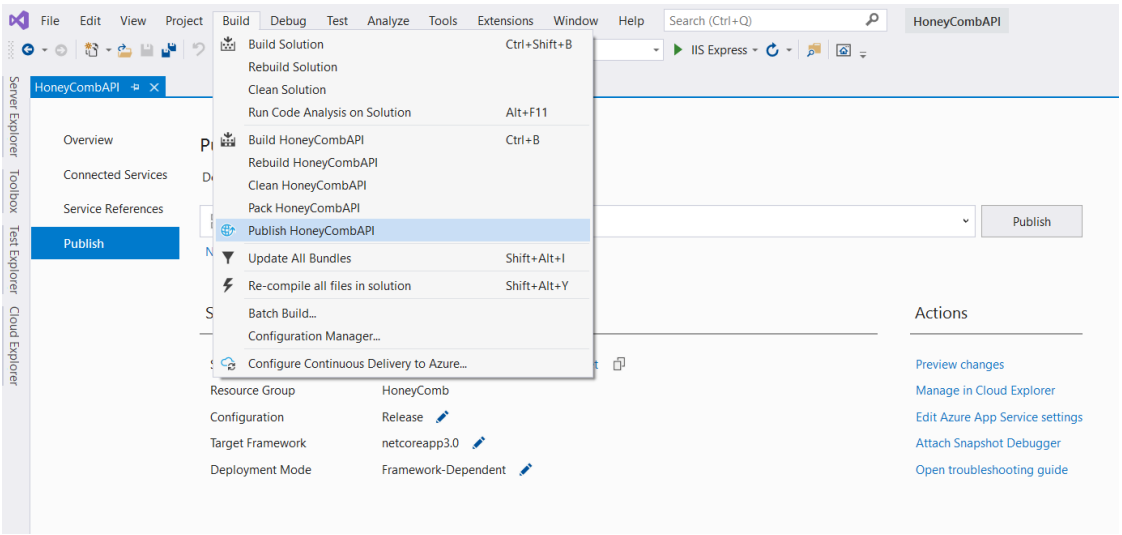
Microsoft Azure App servis omogućuje brzu i jednostavnu isporuku web-aplikacija izravno iz Microsoft Visual Studio programa.

Potrebno je pomoću čarobnjaka, a preko Azure portala, stvoriti novi servis za aplikacije, postaviti adresu lokacije na kojoj će web-aplikacija biti posluživana, odabrati platformu i verziju softverskog okvira (engl. *framework*) na kojem je izrađena aplikacija.



Slika 3.2 Postavljanje Web App servisa

Nakon što smo stvorili servis na kojem će naša aplikacija biti posluživana, potrebno je ulogirati se korisničkim podacima u Microsoft Visual Studio program te u čarobnjaku za isporuku odabrati postojeći servis koji smo stvorili kroz Azure portal. Nakon što smo pripremili čarobnjak za isporuku, izmjene na našem web-servisu možemo isporučivati brzo i jednostavno. Napokon, putanja do čarobnjaka za isporuku web-servisa Microsoft Visual Studio programa prikazana je na slici (Slika 3.1).



Slika 3.3 Isporuka web-servisa iz Microsoft Visual Studio programa

4. Android

Android je operativni sustav otvorenog koda, namijenjen prijenosnim uređajima poput pametnih telefona i tablet računala. Android pruža jedinstveni pristup razvoju aplikacija za prijenosne uređaje, što znači da programeri razvijaju aplikaciju za android, a njihova se aplikacija može pokrenuti na više različitih uređaja na kojima se nalazi operativni sustav Android [4]. Otvorenost ovoga operativnog sustava rezultirala je stvaranjem velike zajednice programera koji redovito rade, kako na samom unaprjeđenju sustava tako i na velikom broju besplatnih prilagođenih komponenti, što android, zajedno s njegovom rasprostranjenošću, čini odličnim kandidatom za razvoj mobilnih klijentskih aplikacija.

4.1. Povijest i verzije

Android operativni sustav je razvijen od strane *Open Handset Alliance* koju je predvodio *Google* i ostale kompanije. Prva inačica seta razvojnih alata (SDK) za android objavljena je od strane *Google LLC* kompanije 2007. godine, dok je prva komercijalna verzija, Android 1.0, objavljena u rujnu 2008. godine [4].

Značajnijim verzijama ovoga operativnog sustava smatraju se:

Android 3.0, objavljen 2011. godine, nosio je kodno ime *HoneyComb* i bio je namijenjen isključivo tablet računalima. Iako koncept ovoga operativnog sustava nije potrajao, Android 3.0 ostavio je značajni trag na izgled grafičkog sučelja budućih inačica sustava [5].

Android 4.0 osvježio je sve Android mobilne uređaje vizualnim konceptom predstavljenim na prijašnjoj inačici sustava i ponovno ujedinio pametne telefone i tablet računala zajedničkim grafičkim sučeljem [5].

Izlaskom verzije Android 5.0, kodnog imena *Lollipop*, 2014. godine, unesene su velike promjene na android mobilne uređaje. Predstavljeno je glasovno upravljanje, podrška za više korisnika te je *Google* predstavio svoj *Material Design standard,* što je pridonijelo sasvim novom izgledu sustava i aplikacija koji se zadržao do danas [5].

4.2. Konfiguracija android aplikacija

HoneyComb android aplikacija izgrađuje se (engl. *build*) korištenjem *Gradle* alata za izgradnju. *Gradle* izgradnja je, u najjednostavnijem obliku, izvršavanje zadataka redoslijedom kojim su definirani u skripti za izgradnju [6]. Kako bi *Gradle* bio u mogućnosti izgraditi aplikaciju, potrebna mu je konfiguracijska datoteka *build.gradle,* u kojoj su od važnijih konfiguracijskih stavki navedene:

- putanje do repozitorija na kojima se nalaze ovisnosti (engl. *dependencies*) aplikacije
- ovisnosti koje se koriste u aplikaciji
- minimalna i ciljana verzija android operativnog sustava koja je potrebna
- trenutna verzija aplikacije, što omogućuje kasnija ažuriranja

Tijekom izrade aplikacije android projekt sadrži i *AndroidManifest.xml* konfiguracijsku datoteku, čiji sadržaj *Gradle*, kroz proces izgradnje, objedinjuje s *build.gradle* datotekom kako bi izgradio konačnu *AndroidManifest.xml* datoteku koja sadrži sve potrebne informacije o aplikaciji, važne za operativni sustav.

U *AndroidManifest.xml* naveden je:

- imenski prostor (engl. *namespace*)
- deklaracije komponenti
- dopuštenja, koja su aplikaciji potrebna za rad, poput pristupa internetu ili pristupa uslugama lokacije. U kontekstu android aplikacije ovog sustava, potrebni pristupi navedeni su kôdom (Kôd 4.1).

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"
/>
```

Kôd 4.1 Primjer potrebnih dozvola u *AndroidManifest.xml* datoteci

Važno je napomenuti da, kako bi operativni sustav znao za njihovo postojanje, potrebno je u *AndroidManifest.xml* datoteci deklarirati sve *activity, service, broadcast receiver, content provider* komponente te naglasiti *activity* koji će biti ulazna točka u aplikaciju.

```
<activity
	android:name=".LoginActivity"
	android:theme="@style/AppTheme.NoActionBar">
	<intent-filter>
		<action android:name="android.intent.action.MAIN" />
		<category android:name="android.intent.category.LAUNCHER" />
	</intent-filter>
</activity>
```

Kôd 4.2 Primjer deklariranja komponente u *AndroidManifest.xml* datoteci

4.3. Korištenje ugrađenih programskih sučelja Android ekosustava

Activity komponenta glavna je prezentacijska komponenta aplikacije, sadrži prezentacijski sloj, raspored (engl. *layout*) definiran u *xml* datoteci i poslovnu logiku. Predstavlja jedan prikaz ekrana s korisničkim sučeljem. Svaki android *Activity* ima svoj životni ciklus koji je popraćen pozivom određene metode u svakoj od faza životnog ciklusa [7], koji programeru otvaraju mogućnost lakog i efektivnog programiranja poslovne logike:

- *onCreate()* – poziva se kada je *Activity* stvorena
- *onStart()* – poziva se kada se *Activity* priprema za prikaz korisniku
- *onResume()* – poziva se kada *Activity* postane vidljiva i kada je korisniku omogućen unos
- *onPause()* – poziva se kada *Activity* izgubi fokus
- *onStop()* – poziva se kada *Activity* više nije vidljiva
- *onRestart()* – poziva se kada *Activity* ponovno postaje vidljiva
- *onDestroy()* – poziva se kod gašenja *Activity* komponente.

Service je komponenta koja je namijenjena izvođenju dugotrajnih operacija u pozadinskoj dretvi. Ako je potrebno, nakon pokretanja ova komponenta može nastaviti izvoditi operacije i nakon što se aplikacija, koja ju je pokrenula, ugasi. Ova komponenta najčešće se koristi za upravljanje mrežnim transakcijama i puštanje glazbe u pozadini [7].

BroadcastReceiver je komponenta zadužena za primanje *Broadcast Intent* objekata koje šalju druge aplikacije ili operativni sustav, npr. kada uređaj spojimo na punjač ili kada se

završi preuzimanje datoteke. Kada sustav zaprimi B*roadcast Intent* objekt, prosljeđuje ga dalje svim *BroadcastReceiver* komponentama koje su se pretplatile za primanje te vrste *Broadcast Intent* objekta [7].

Content Provider komponenta pomaže aplikaciji upravljanja vlastitim podacima, kao i pružanja podataka drugim, zainteresiranim aplikacijama. Ova je komponenta standardno sučelje za povezivanje podataka između procesa [7].

Fragment je modularni dio korisničkog sučelja na *Activity* komponenti. Ima vlastiti *layout*, poslovnu logiku i životni ciklus. Primjenom fragmenata omogućena nam je izrada dinamičnijeg sučelja te primjena jedne komponente na više različitih *Activitya* [7].

Od komponenti opisanih u ovom poglavlju za izradu *HoneyComb* Android aplikacije korišteni su:

- *Activity*, kao glavna prezentacijska komponenta
- *Fragment*, kao prezentacijske komponente koje se izmjenjuju na *Activityu* kako korisnik prolazi kroz opcije u izborniku aplikacije
- *Service*, za obradu primljenih obavijesti (engl. *notifications*).

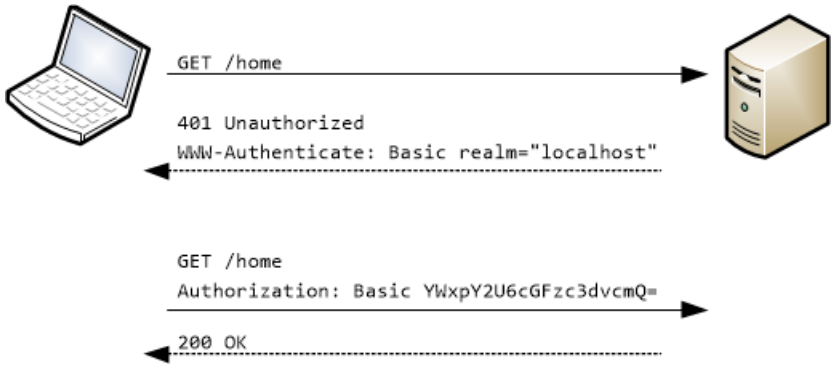
5. Implementacija web-servisa kao serverskog dijela aplikacije

Središnji dio sustava implementiran je kao web-servis. Njegova je svrha prikupljanje, obrada i isporuka podataka iz baze podataka svim klijentskim aplikacijama. Za izradu ovog servisa korišten je .NET Core programski okvir (engl. *framework*) koji se smatra nasljednikom .NET Frameworka. Za razliku od .NET Frameworka, ovaj je programski okvir predviđen za razvoj aplikacija namijenjenih za rad na više platformi (Windows, Linux, macOS). Ovakav nam pristup otvara mogućnost postavljanja aplikacije na platformu prema željama i mogućnostima korisnika, kao i promjenu platforme bez potrebe za razvijanjem nove aplikacije s istim funkcionalnostima.

5.1. Autentikacija

Za potrebe autentikacije na web-servisu implementirana je osnovna autentikacija (engl. *Basic Authentication*). Autentikacija je postupak utvrđivanja identiteta korisnika, za što se u slučaju *HoneyComb* sustava koristi korisničko ime i lozinka.

Prednost ovog načina autentikacije je jednostavnost pa je podržan na svim glavnim internet preglednicima. Istovremeno, glavni je nedostatak što se podaci kodiraju *base64* standardom i šalju u nekriptiranom obliku unutar svakog zahtjeva prema poslužitelju. Postupak odvijanja autentikacije prikazan je na slici (Slika 5.1) [8].



Slika 5.1 *Basic Authentication* [8]

Nakon što server zaprimi zahtjev s autentikacijskim zaglavljem, dekodirane podatke prosljeđuje funkcionalnom dijelu koji dohvaća korisnika prema korisničkom imenu iz baze podataka te uspoređuje lozinke. Ako se lozinke podudaraju, server vraća klijentu zatražene podatke. U protivnom, vraća HTTP statusni kod 401 kojim obavještava klijenta kako autentikacija nije uspjela. Funkcionalni je dio prikazan kôdom (Kôd 5.1).

```
public static AppUser AuthUser(string username, string password)
{
	AppUser user =
		RepoFactory.GetUserRepository().GetUserByUsername(username);
	if (user != null && user.Password == password)
	{
		return user;
	}
	return null;
}
```

Kôd 5.1 Provjera korisničkog imena i lozinke

5.2. Integracija s bazom podataka

Za komunikaciju s bazom odgovoran je podatkovni sloj aplikacije, razvijen korištenjem obrasca tvornice (engl. *Factory Pattern*). Poslovni sloj koristi statičku klasu *RepoFactory* za dohvaćanje odgovarajuće implementacije sučelja (engl. *Interface*) za rad s podacima. Ovakav nam pristup omogućuje promjenu načina rada s podacima bez potrebnih izmjena na poslovnom sloju aplikacije. Primjer sučelja za rad s korisničkim podacima prikazan je kôdom (Kôd 5.2).

```
public interface IUserRepository
{
   AppUser GetUserByUsername(string username);
   RegistrationStatus RegisterUser(AppUser user);
   int GetUsersCountAtDate(DateTime date);
}
```

Kôd 5.2 Sučelje za rad s korisničkim podacima

Kako bi se pojednostavio pristup podacima te izbjegao izravan rad s tablicama i relacijama unutar baze, podatkovni je sloj implementiran korištenjem *Entity Framework Core* ORM alata. Njegova je uloga mapiranje podataka dohvaćenih iz baze u .NET objekte prema

unaprijed definiranoj konfiguraciji. Primjer dohvaćanja podataka prikazan je kôdom (Kôd 5.3).

```
public AppUser GetUserByUsername(string username)
{
	AppUser user = null;
	using(HoneyCombDbContext dbContext = new HoneyCombDbContext())
	{
		try
		{
			user = dbContext.AppUser
			.Where(x => x.Username == username).FirstOrDefault();
		}
		catch (Exception ex)
		{
			RepoFactory.GetLogger().WriteLog(ex);
		}
	}
	return user;
}
```

Kôd 5.3 Dohvaćanje podataka iz baze koristeći se *Entity Framework Coreom*

5.3. Pružanje usluga u JSON formatu

Za razmjenu podataka između klijentskih aplikacija i web-servisa odabran je JSON format. JSON je javno dostupan, jezično nezavisan te ljudima čitljiv format za razmjenu podataka. Kod JSON formata podaci su strukturirani kao kolekcija naziv / vrijednost parova ili kao uređena lista vrijednosti, što ga čini dobrim izborom za razmjenu objekata između aplikacija razvijenih različitim programskim jezicima [9]. Primjer JSON-a definiran je sljedećim kôdom (Kôd 5.4)

```
{
  "id": 3,
  "username": "marko",
  "password": "1234",
  "email": "marko@mail.com",
  "firstName": "marko",
  "lastName": "maric"
}
```

Kôd 5.4 Korisnički podaci formatirani JSON formatom

Kako bi se smanjila količina podataka i kako bi klijentu slao samo potrebne podatke, web-servis objekte dohvaćene iz baze obrađuje i mapira u servisne objekte. Obrađeni i pročišćeni podaci pretvaraju se u JSON format i zatim vraćaju klijentu. Kao primjer naveo bih prikazivanje liste košnica na android klijentskoj aplikaciji. Navedena lista, prikazana kôdom (Kôd 5.5), prikazuje osnovne podatke poput identifikacijske oznake košnice i datum zadnjeg pregleda. Kako bi se izbjeglo slanje liste svih pregleda za pojedinu košnicu, što bi rezultiralo velikom količinom podataka za slanje, servis obrađuje podatke i mapira ih u novi servisni objekt, postavlja datum zadnjeg pregleda i zatim vraća klijentu.

```
[
  {
   "idHive": 1,
   "hiveNumber": "1",
   "latestExam": "2019-11-30T00:00:00",
   "createdDate": "2019-10-01T00:00:00",
   "deathDate": null,
   "apiaryId": 1
  },
  {
   "idHive": 2,
   "hiveNumber": "3",
   "latestExam": "2019-11-24T00:00:00",
   "createdDate": "2019-11-01T00:00:00",
   "deathDate": null,
   "apiaryId": 1
  }
 ]
```

Kôd 5.5 Obrađena lista košnica u JSON formatu

5.4. Metrika o pčelinjacima

Kako je pristup podacima ograničen na razini korisnika, što znači da svaki korisnik ima pristup samo podacima o svojim pčelinjacima, na web-servisu ugrađena je i funkcionalnost za dohvat i obradu statističkih podataka. Statistički su podaci bazirani na temelju podataka koje korisnici unose tijekom redovnog korištenja android klijentske aplikacije. Tako web-servis omogućuje dohvat podataka o broju košnica na mjesečnoj bazi kroz cijelu godinu, prosječan broj košnica po korisniku, prinose meda te podatke o količini i vrsti pčelinjih bolesti. Dohvaćanje prosječnog broja košnica po korisniku prikazano je kôdom (Kôd 5.6)

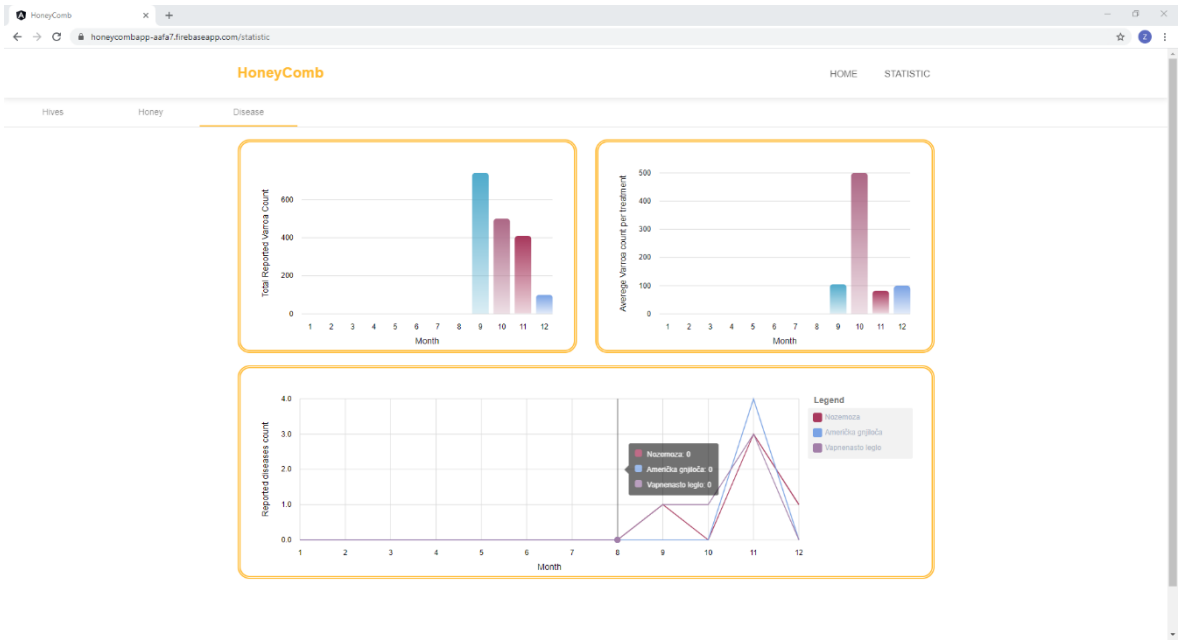
```
public List<StatisticsSO> HivesPerUser(int year)
{
	List<StatisticsSO> statisticsList = new List<StatisticsSO>();
	for (int i = 1; i <= 12; i++)
	{
		DateTime date = new DateTime(year, i, 1);
		int hivesCount =
		RepoFactory.GetHiveRepository().GetHivesCountAtDate(date);
		int usersCount =
		RepoFactory.GetUserRepository().GetUsersCountAtDate(date);
		StatisticsSO statisticsSO = new StatisticsSO();
		statisticsSO.Month = i;
		if (hivesCount > 0 && usersCount > 0)
		{
			statisticsSO.Amount = (float)hivesCount / usersCount;
		}
		statisticsList.Add(statisticsSO);
	}

	return statisticsList;
}
```

Kôd 5.6 Metoda za dohvat prosječnog broja košnica po korisniku na mjesečnoj bazi

U svrhu prezentacije ove funkcionalnosti web-servisa, razvijena je web-klijentska aplikacija koja prikazuje statističke podatke dohvaćene s web-servisa.

Aplikacija je razvijena koristeći *Angular framework*, softverski okvir za razvoj web-aplikacija baziran na *TypeScript* programskom jeziku, razvijenom većinskim dijelom od strane *Google* kompanije [10].

Na slici (Slika 5.2) prikazana je klijentska web-aplikacija za prikaz statističkih podataka u obliku grafikona koji se izmjenjuju odabirom opcija u navigacijskom izborniku. Vidljivi grafikoni prikazuju podatke o zabilježenim pčelinjim bolestima kroz cijelu godinu, kao i točnu vrijednost koja se prikazuje u skočnom prozoru postavljanjem kursora miša na željeni mjesec.



Slika 5.2 Web-stranica za prikaz statističkih podataka

5.5. Upravljanje slanjem obavijesti

HoneyComb sustav podržava slanje obavijesti korisnicima. Web-servis zadužen je za registraciju uređaja na koje želimo primati obavijesti, dok je za slanje obavijesti zadužena *HoneyCombAlertsWebJob* aplikacija.

HoneyComb sustav koristi *Azure Notification Hubs* servis za slanje obavijesti između različitih platformi. Ovaj servis omogućuje slanje obavijesti uređajima s različitim operativnim sustavima, dok je u *HoneyComb* sustavu implementirano slanje obavijesti namijenjenih uređajima s android operativnim sustavom. U tu je svrhu registrirana *Firebase Cloud Messaging* usluga na *Firebase* platformi, prije poznata kao *Google Cloud Messaging* (GCM), koja je razvijena od strane *Googlea*, a namijenjena je slanju obavijesti uređajima s android operativnim sustavom [11].

Web-servis omogućuje registraciju uređaja za primanje obavijesti koja se sastoji od sljedećih koraka:

1. Uređaj šalje zahtjev za registraciju s jedinstvenim identifikatorom uređaja.
2. Web-servis zatim provjerava u *Azure Notification Hubs* servisu postoje li već registrirani uređaji s navedenim identifikatorom i vraća *registrationId* uređaju.
3. Uređaj web-servisu šalje zahtjev u kojem se nalazi *registrationId* i oznaka tipa uređaja za koji su obavijesti namijenjene.

4. Web-servis registrira uređaj na *Azure Notification Hubs* servisu za primanje obavijesti namijenjenih korisniku koji je identificiran kao pošiljatelj zahtjeva tijekom autentikacije.

HoneyCombAlertsWebJob jednostavan je program koji dohvaća upozorenja spremljena u bazu podataka te ih preko *Azure Notification Hubs* servisa šalje korisnicima za koje su namijenjeni. *HoneyCombAlertsWebJob* postavljen je kao *Azure WebJob*, funkcionalnost *Azure App Servicea*, koja web-aplikacijama omogućuje izvođenje pozadinskih zadataka. *WebJob* podržava dva načina rada:

- Neprekidni – program koji se izvodi pod ovim načinom rada mora sadržavati beskonačnu petlju unutar koje odrađuje određene radnje. U protivnom, *WebJob* mora se ručno pokrenuti nakon završetka programa
- Okidani – *WebJob* pokreće program prema definiranom rasporedu.

Za potrebe *HoneyComb* sustava odabran je okidani način rada jer omogućuje jednostavnu izmjenu učestalosti izvršenja zadataka, bez potrebe za izmjenama samog kôda programa ili uvođenja dodatne razine kompleksnosti.

6. Implementacija klijentskog dijela aplikacije

Glavna je komponenta ovog sustava *HoneyComb* Android aplikacija. Kroz nju je korisnicima nakon registracije omogućeno bilježenje svakodnevnih radnji i pregledavanje stanja na njihovim pčelinjacima. Aplikacija je isključivo klijentske prirode pa joj je za rad potreban pristup internetu.

6.1. Komunikacija android mobilne aplikacije sa serverom

Komunikacija sa serverom izvedena je korištenjem *Retrofit* HTTP klijenta za android te Gson biblioteke za pretvorbu Java objekata u JSON i obratno. Za izgradnju *Retrofit* klijenta odgovorna je klasa *ServiceGenerator*, koja implementira generičku metodu *createService* koja izgrađuje *Retrofit* HTTP klijent, prikazanu u nastavku.

```
private static <S> S createService(
      Class<S> serviceClass, final String authToken) {
            if (!TextUtils.isEmpty(authToken)) {
                  AuthenticationInterceptor interceptor =
                        new AuthenticationInterceptor(authToken);

                  if (!httpClient.interceptors().contains(interceptor)) {
                        httpClient.addInterceptor(interceptor);

                        builder.client(httpClient.build());
                        retrofit = builder.build();
                  }
            }

            return retrofit.create(serviceClass);
}
```

Kôd 6.1 Metoda za izgradnju *Retrofit* HTTP klijenta

Metoda *createService* prima sučelje nad čijim su metodama, korištenjem anotacija, naznačene relativne putanje do željenog resursa na web-servisu i način na koji se šalju

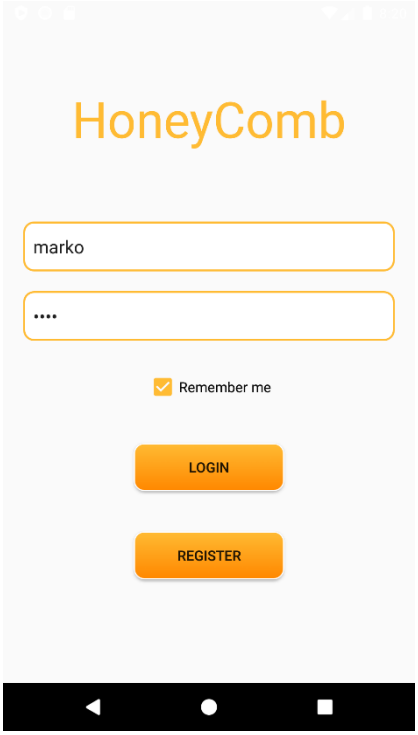
parametri, u putanji ili unutar tijela (engl. *body*) zahtjeva. Primjer sučelja za dohvaćanje podataka o pčelinjacima prikazan je kôdom (Kôd 6.2)

```
public interface ApiaryService {
  @GET("api/apiary")
  public Call<List<Apiary>> getAllApiariesGet();
  @POST("api/apiary/Save")
  public Call<String> saveApiary(@Body Apiary data);
}
```

Kôd 6.2 Primjer sučelja za dohvaćanje podataka o pčelinjacima s web-servisa

6.2. Prijava korisnika

LoginActivity je ulazna točka u aplikaciju i prilikom pokretanja *Activitya* na njoj se prikazuje *LoginFragment* čiji je *layout* definiran u *fragment_login.xml* datoteci. Prilikom pritiska na gumb *Login*, upućuje se zahtjev za autorizacijom prema web-servisu. Nakon uspješne autorizacije, ako je odabrana opcija *Remember me,* korisničko ime i lozinka spremaju se u *SharedPreferences,* kolekciju ključ / vrijednost parova. Spremljeni podaci koriste se za autorizaciju kod sljedećeg pokretanja aplikacije, što znatno poboljšava korisnički dojam, olakšava i ubrzava rad korisniku.



Slika 6.1 *Login* forma

Podaci za registraciju korisnika unose se na *RegistrationFragment* komponenti, koja se prikazuje na *LoginActivityu*, a prilikom registracije web-servis provjerava jedinstvenost korisničkog imena. Ako već postoji korisnik s unesenim korisničkim imenom, aplikacija obavještava korisnika prikazom maloga skočnog prozora (engl. *popup*), prikazano sljedećim kôdom.

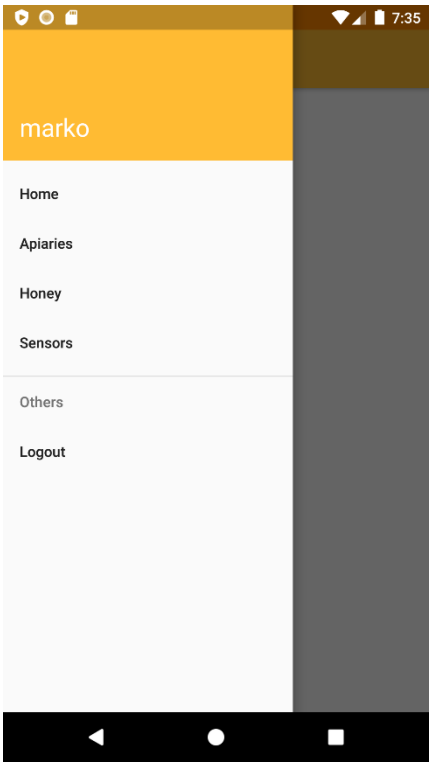
```
Toast.makeText(context, R.string.userExists,
Toast.LENGTH_SHORT).show();
```

Kôd 6.3 Kreiranje *popup* prozora za prikaz poruka

6.3. Glavno sučelje aplikacije

Aplikacija je izgrađena korištenjem dviju *Activity* komponenti, *LoginActivity* i *MainActivity*.

MainActivity je glavno sučelje na kojem se nalazi glavni izbornik za navigaciju kroz aplikaciju. Glavni izbornik, preko kojeg korisnici mogu pristupiti popisu pčelinjaka, prikazu prinosa meda, registriranim senzorima te se odjaviti iz sustava, prikazan je u *NavigationDrawer* panelu. Odabirom stavke u izborniku, na *MainActivityu* se izmjenjuju fragmenti za prikaz korisničkog sučelja.



Slika 6.2 Navigacijski izbornik

6.3.1. *NavigationDrawer*

NavigationDrawer je skriveni panel koji prikazuje navigacijski izbornik. Panel se prikazuje pritiskom na ikonu na alatnoj traci ili povlačenjem prsta od lijevog ruba zaslona prema desnome rubu.

Kao korijenski element ovog *layouta*, postavlja se *DrawerLayout* koji kao podelement (engl. *child element*) sadrži *LinearLayout*, na kojem se prikazuje vidljivi dio sučelja i *NavigationView* koji se prikazuje kao navigacijski izbornik. *NavigationView* kao parametre prima *layout*, koji će se prikazati kao zaglavlje (engl. *header*), i listu stavki (engl. *items*) koju prikazuje kao stavke izbornika, prikazano kôdom (Kôd 6.4)

```
<?xml version="1.0" encoding="utf-8"?>
<menu
xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  tools:showIn="navigation_view">
  <group android:checkableBehavior="single">
   <item
     android:id="@+id/nav_home"
     android:title="@string/menu_home" />
```

```
    <item
      android:id="@+id/nav_apiaries"
      android:title="@string/menu_apiaries" />
    <item
      android:id="@+id/nav_honey"
      android:title="@string/menu_honey" />
    <item
      android:id="@+id/nav_sensors"
      android:title="@string/menu_sensors" />
   </group>
   <item android:title="Others">
    <menu>
      <item
       android:id="@+id/nav_logout"
       android:title="@string/menu_logout" />
    </menu>
   </item>
</menu>
```

Kôd 6.4 Stavke navigacijskog izbornika

6.3.2. *FragmentLoader* sučelje

Sučelje *FragmentLoader* deklarira metodu za izmjenu fragmenata koju implementira *MainActivity* kao glavno sučelje aplikacije. Svrha je ovog sučelja omogućiti fragmentima način kako bi, umjesto sebe, na zaslonu prikazali neki drugi fragment.

```
@Override
public void loadFragment(Fragment fragment) {
 getSupportFragmentManager()
     .beginTransaction()
           .replace(R.id.flContent, fragment)
           .addToBackStack(fragment.getClass().getName())
           .commit();
}
```

Kôd 6.5 Implementacija *FragmentLoader* sučelja

.

6.4. Funkcionalnosti aplikacije

Glavne funkcionalnosti aplikacije su praćenje stanja na pčelinjacima i evidencija pregleda pčelinjih zajednica. Aplikacija podržava unos više pčelinjaka i košnica.

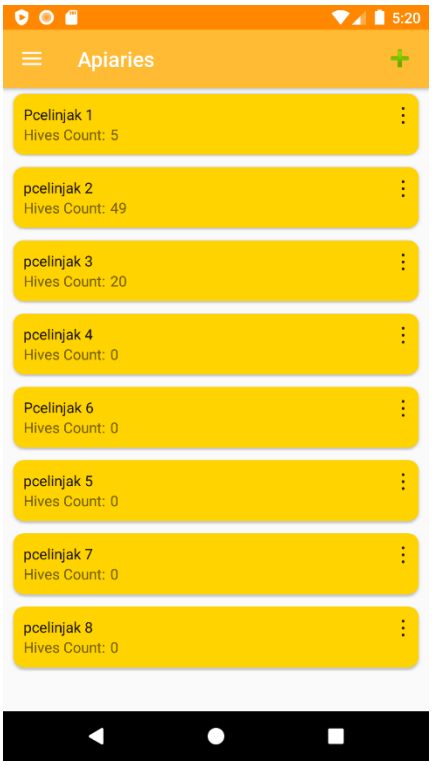
6.4.1. Popis pčelinjaka

Za potrebe prikazivanja popisa pčelinjaka koristi se *ApiaryListFragment*. Ovaj fragment implementira *ApiaryChangeListener* te se prilikom inicijalizacije registrira kao slušatelj promjena na popisu pčelinjaka za čije dohvaćanje se brine *ApiaryDataService* klasa slanjem asinkronih zahtjeva prema web-servisu.

```
private void fetchApiaries() {
	ApiaryService service =
	ServiceGenerator.createService(ApiaryService.class,
	CredentialUtils.getUser().getUsername(),
	CredentialUtils.getUser().getPassword());
	Call<List<Apiary>> allApiariesGet = service.getAllApiariesGet();
	allApiariesGet.enqueue(new Callback<List<Apiary>>() {
		@Override
		public void onResponse(Call<List<Apiary>> call,
		Response<List<Apiary>> response) {
			if (response.isSuccessful()) {
				apiaries = response.body();
				notifyListeners(true, "Success");
			} else {
				notifyListeners(false, "Fail");
			}
		}
		@Override
		public void onFailure(Call<List<Apiary>> call, Throwable t) {
			notifyListeners(false, "Server not responding");
			}
		});
}
```

Kôd 6.6 Dohvaćanje popisa pčelinjaka s web-servisa

Nakon što *ApiaryDataService* zaprimi odgovor s web-servisa, obavještava sve pretplaćene slušatelje o promjenama, među kojima je i *ApiaryListFragment* koji će promjene prikazati u *RecyclerView* grafičkoj komponenti.



Slika 6.3 Sučelje s popisom pčelinjaka u *RecyclerView* komponenti

Elementi u listi koju prikazuje *RecyclerView* imaju vlastiti izbornik s opcijama koje omogućuju dodatne radnje za pčelinjak, poput promjene lokacije, dodavanja prinosa meda i prikaz očitanja sa senzora postavljenog na pčelinjaku.

6.4.1.1 RecyclerView

RecyclerView grafička komponenta koristi se za prikazivanje liste *ViewHolder* objekata koji prilikom izgradnje napuhuju (engl. *Inflate*) svoj *layout* i prikazuje listu entiteta koju smo putem adaptera postavili na *RecyclerView*. Svaki *ViewHolder* može prikazivati vrijednosti samo jednog entiteta [7].

```
@Override
public void apiaryListChanged(List<Apiary> apiaryList) {
      recyclerView.setAdapter(new
      ApiariesRecyclerViewAdapter(apiaryList, context,
      fragmentLoader));
}
```

Kôd 6.7 Postavljanje adaptera na *RecyclerView*

Posebnost *RecyclerView* komponente je način upravljanja prikazom te izgrađuje samo potreban broj *ViewHoldera* kako bi popunila prostor na zaslonu. Kako povlačenjem prsta po ekranu prolazimo kroz listu, *RecyclerView* izgrađuje nove *ViewHoldere,* a one koji su

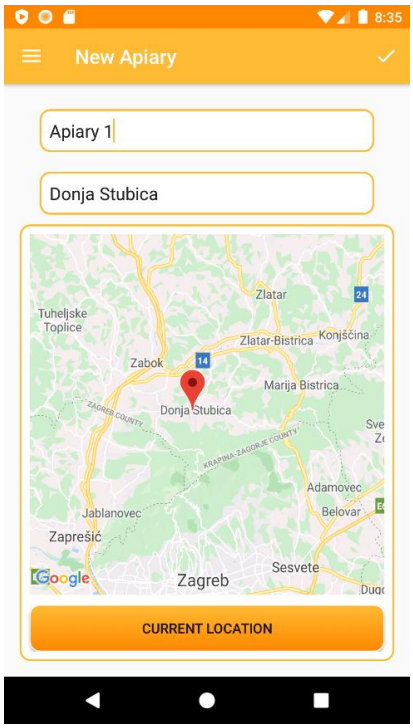
izgrađeni sprema za slučaj da korisnik promijeni smjer prolaska kroz listu. Ako korisnik nastavi prolaziti kroz listu u istom smjeru, *RecyclerView* zamjenjuje *ViewHoldere* koji su najduže spremljeni i povezuje ih s podacima drugog entiteta, čime štedi memoriju uređaja i ubrzava rad aplikacije [7]. Popunjavanje *layouta ViewHoldera* atributima entiteta iz liste prikazano je kôdom u nastavku (Kôd 6.8).

```
@Override
public void onBindViewHolder(final ViewHolder holder, int
position) {
  final Disease disease = diseaseList.get(position);
  holder.tvName.setText(disease.getName());
  holder.btnDelete.setOnClickListener(new
  View.OnClickListener() {
   @Override
   public void onClick(View view) {
     diseaseList.remove(position);
     notifyDataSetChanged();
   }
  });
}
```

Kôd 6.8 Postavljanje vrijednosti *ViewHoldera*

6.4.2. Dodavanje pčelinjaka

Za dodavanje i uređivanje pčelinjaka koristi se *EditApiaryFragment* koji sadrži polje za unos naziva, polje za prikaz naziva trenutne lokacije na kojoj se pčelinjak nalazi i *MapFragment* za prikazivanje i postavljanje lokacije pčelinjaka. Implementira *LocationChangeListener* sučelje kojim *MapFragment* obavještava slušatelje o odabiru nove lokacije na karti.



Slika 6.4 Sučelje za dodavanje novog pčelinjaka

6.4.2.1 MapFragment

MapFragment se koristi za prikazivanje lokacija pčelinjaka na karti korištenjem *GoogleMaps* APIa koji omogućuje pristup *Google* kartama iz aplikacije. Za korištenje *GoogleMaps* APIa potrebno je kreirati ključ (engl. *key*) na *Google Cloud Platform* web-stranici, koji se koristi za autorizaciju zahtjeva upućenih *Google* poslužiteljima te ga deklarirati u *AndroidManifest.xml* datoteci.

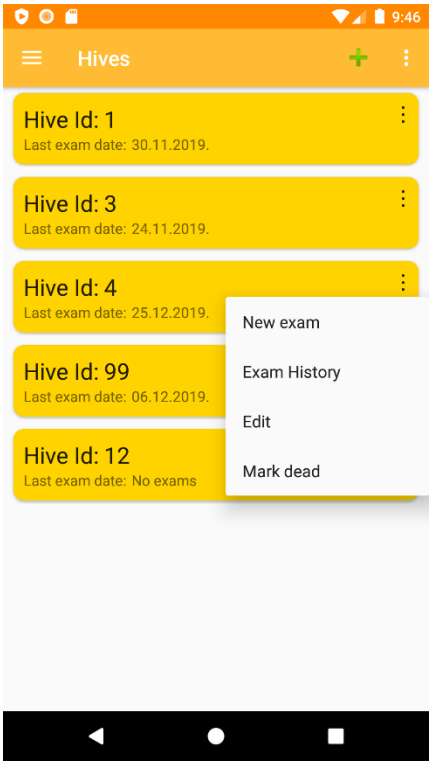
Layout MapFragmenta sadrži dvije grafičke komponente: *SupportMapFragment*, koja prikazuje kartu, i gumb za označavanje trenutne lokacije uređaja na karti. Prilikom prvog korištenja ove opcije potrebno je od korisnika zatražiti dopuštenje za korištenje lokacijskih usluga uređaja, kao što prikazuje kôdni odsječak (Kôd 6.9).

```
if (getActivity().checkSelfPermission(ACCESS_FINE_LOCATION)
!= PackageManager.PERMISSION_GRANTED &&
getActivity().checkSelfPermission(Manifest.permission.ACCESS_
COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
      ActivityCompat.requestPermissions(getActivity(),
      new String[]{ACCESS_FINE_LOCATION}, 1);
      return;
}
```

Kôd 6.9 Traženje dopuštenja za korištenje usluga lokacije uređaja

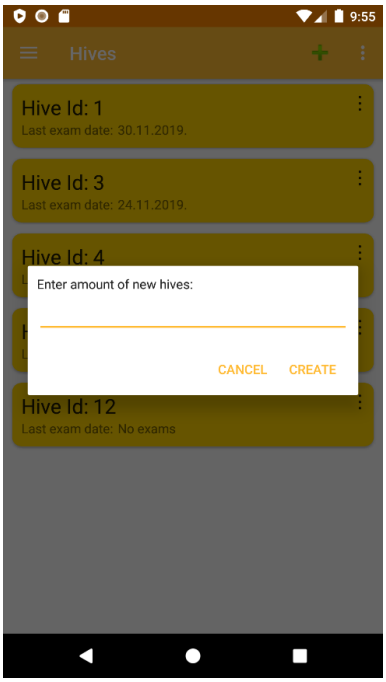
6.4.3. Popis i dodavanje košnica

Popis košnica prikazuje se u *HiveListFragment* fragmentu koji za prikazivanje koristi *RecyclerView* grafičku komponentu. Kroz izbornik s opcijama, koji je postavljen na *ViewHolder* objektima unutar *RecyclerViewa*, omogućeno je započeti novi pregled košnice, povijest pregleda i označiti uginuće košnice. Označavanjem uginuća košnice, košnica se više ne prikazuje na popisu te daljnje radnje u vezi s njom nisu moguće, ali za potrebe statističke obrade ne briše se iz sustava.



Slika 6.5 Sučelje za prikaz popisa košnica s izbornikom opcija

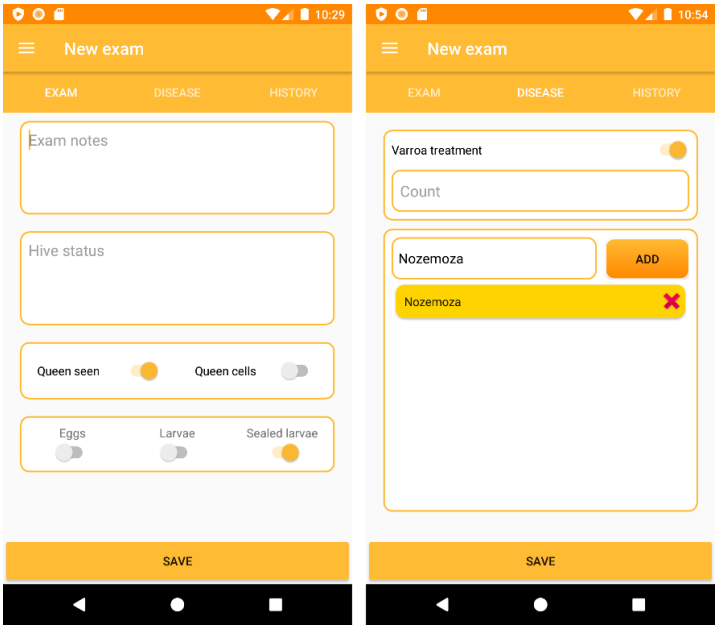
Za potrebe dodavanja košnica, implementirana su dva rješenja, pojedinačno i grupno, kako bi se olakšalo početno postavljanje. Odabirom željene opcije prikazuje se skočni (engl. *popup*) prozor za unos identifikacijske oznake u slučaju pojedinačnog dodavanja ili za unos željenog broja novokreiranih košnica u slučaju grupnog dodavanja, kao što je prikazano na slici (Slika 6.6).



Slika 6.6 Skočni prozor za grupno dodavanje košnica

6.4.4. Pregled košnice

HiveExamFragment se koristi tijekom unosa pregleda košnice. Fragment sadrži *ViewPager* komponentu koja omogućuje stranični prikaz fragmenata koje se izmjenjuju pritiskom na naslov željenog fragmenta u zaglavlju komponente ili pomicanjem prsta po zaslonu ulijevo ili udesno. Izgled sučelja s *ViewPager* komponentom, koja prikazuje više fragmenata, prikazan je slikom (Slika 6.7).



Slika 6.7 Sučelja za unos podataka o pregledu košnice

ViewPager koristi *FragmentStatePagerAdapter* za promjenu fragmenata koji se trenutno prikazuje. Adapter tijekom kreiranja, koristeći refleksiju, učitava fragmente prema imenima klasa spremljenim u resursima. Refleksija je postupak dinamičkog učitavanja klasa tijekom izvođenja programa, korištenjem imena klasa, što omogućuje veliku fleksibilnost.

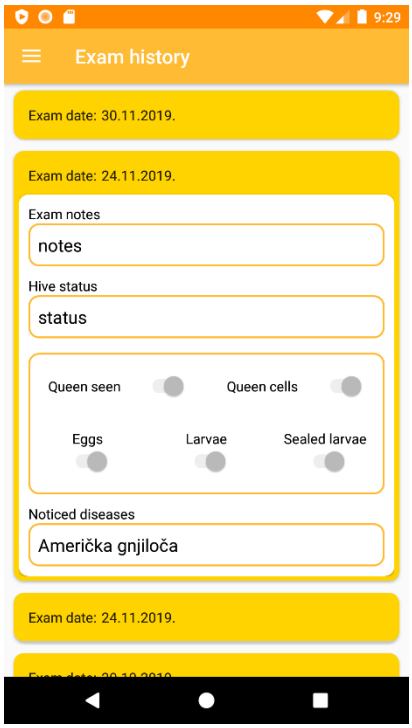
```
private void initConfig() {
	examFragments = new ArrayList<Fragment>();
	fragmentTitles = context.getResources()
	.getStringArray(R.array.fragment_hive_exam_titles);
	fragmentClasses = context.getResources()
	.getStringArray(R.array.fragment_hive_exam_classses);
	for (int i = 0; i < fragmentClasses.length ; i++) {
		examFragments
		.add(Fragment.instantiate(context, fragmentClasses[i]));
	}
}
```

Kôd 6.10 Kreiranje fragmenata pomoću refleksije

6.4.5. Povijest pregleda

HiveExamHistoryFragment prikazuje listu prijašnjih pregleda za određenu košnicu u *RecyclerView* grafičkoj komponenti čiji *ViewHolderi* imaju mogućnost proširivanja za potrebe prikazivanja detalja o pregledu.

Na slici (Slika 6.8) prikazana je lista prijašnjih pregleda košnice s jednim proširenim *ViewHolderom* koji prikazuje detalje o pregledu. U polju *Exam notes* prikazane su bilješke o radnjama koje su obavljene prilikom pregleda. Polje *Hive status* prikazuje stanje košnice koje je korisnik opisao prilikom pregleda. Treće polje prikazuje uočene pojedinosti tijekom osnovne provjere košnice, poput prisustva matice, matičnjaka, te stanje pčelinjeg legla. Napokon, u polju *Noticed diseases* navedene su bolesti uočene prilikom pregleda.



Slika 6.8 Sučelje za prikaz povijesti pregleda s proširivim *ViewHolderom*

Za prikazivanje povijesti pregleda koriste se dvije vrste entiteta, *BaseHiveExam* i *HiveExam*. *BaseHiveExam* je klasa koja sadrži samo osnovne podatke o pregledu, dok je *HiveExam* nasljeđuje i sadrži sve detaljne podatke.

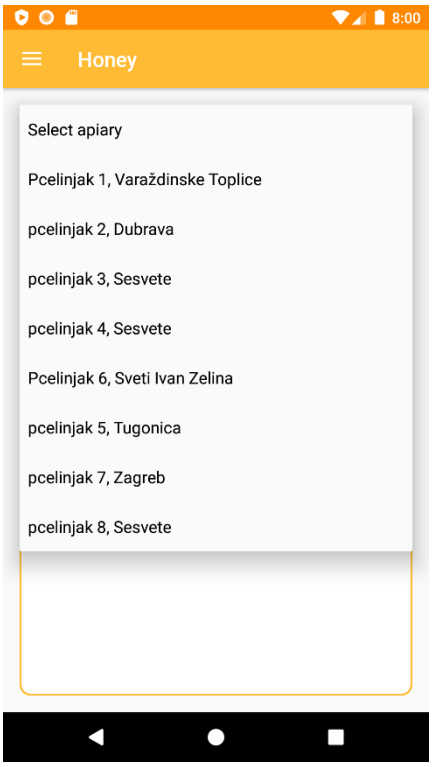
Adapter *RecyclerViewa* sadrži listu *BaseHiveExam* objekata koju dohvaća s web-servisa. Pritiskom prsta na *ViewHolder* aktivira se proširivanje tijekom kojeg se provjerava tip objekta koji prikazuje. Ako je objekt tipa *BaseHiveExam*, dohvaća se *HiveExam* s web-servisa te zauzima njegovo mjesto u listi adaptera kako bi *ViewHolder* mogao prikazati detalje o pregledu, kao što je prikazano sljedećim kôdom (Kôd 6.11).

```
if (holder.cvDetails.getVisibility() == View.GONE){
  if (!(hiveExam instanceof HiveExam)) {
   fetchDetailedExam(hiveExam.getIdHiveExam(),
       position);
  }
  expandedExams.add(hiveExam.getIdHiveExam());
  holder.cvDetails.setVisibility(View.VISIBLE);
} else {
  expandedExams.remove(hiveExam.getIdHiveExam());
  holder.cvDetails.setVisibility(View.GONE);
}
```

Kôd 6.11 Provjera i dohvaćanje detalja o pregledu

6.4.6. Prikaz prinosa meda

HoneyFragment namijenjen je prikazivanju unesenih prinosa meda po pčelinjaku. *Layout* ovog fragmenta ima dvije grafičke komponente, *Spinner* i *RecyclerView*. *Spinner* je grafička komponenta koja omogućuje odabir jednog objekta u listi.



Slika 6.9 *Spinner* s prikazanim mogućim opcijama

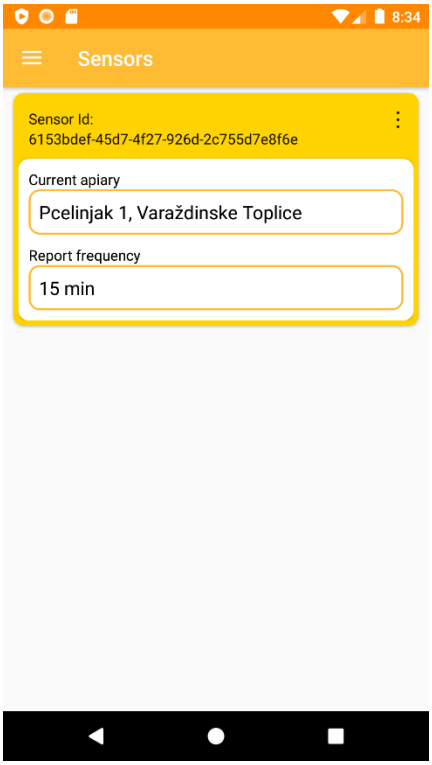
Spinner na *HoneyFragmentu* sadrži popis pčelinjaka, a promjenom odabranog pčelinjaka dohvaća se lista unosa meda za odabrani pčelinjak koji se prikazuje u *RecyclerViewu.*

```
@Override
public void onItemSelected(AdapterView<?> adapterView, View view, int i,
long l) {
            Object item = spinnerApiary.getAdapter().getItem(i);
            if (item instanceof Apiary) {
                  int idApiary = ((Apiary) item).getIdApiary();
              if (idApiary != 0){
                        fetchHoneyExtractsForApiary(idApiary);
              } else{
                  recyclerView.setAdapter(null);
              }
            }}
```

Kôd 6.12 Implementacija promjene odabranog objekta na *Spinneru*

6.4.7. Prikaz registriranih senzora i izmjena konfiguracije

Za prikaz svih registriranih senzora, postavljenih na pčelinjake, koristi se *SensorsListFragment*. Senzori se prikazuju u *RecyclerView* komponenti koja prikazuje informacije o identifikacijskoj oznaci senzora, trenutnom pčelinjaku na kojem se senzor nalazi, i vremenski raspon između očitanja senzora.

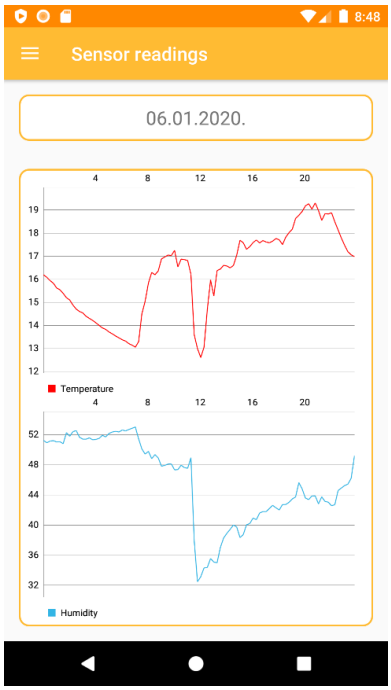


Slika 6.10 Sučelje za prikaz informacija o senzorima

Konfiguracija senzora izmjenjuje se odabirom opcije za uređivanje. Moguće je izmijeniti pčelinjak na kojem se senzor nalazi te vremenski raspon između očitanja senzora u minutama.

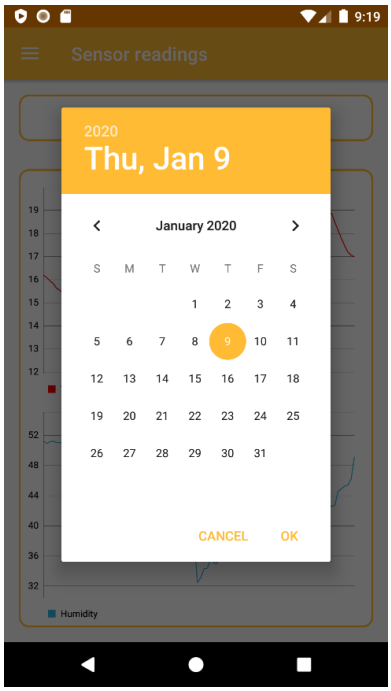
6.4.8. Prikaz očitanja sa senzora

SensorReadingsFragment fragment prikazuje očitanja o temperaturi i vlazi zraka sa senzora za određeni datum u linijskom dijagramu, vrijeme očitanja nalazi se na X-osi dijagrama, dok se na Y-osi nalazi očitana vrijednost.



Slika 6.11 Sučelje za prikaz očitanja sa senzora na pčelinjacima

Željeni datum, za prikaz očitanja, odabire se koristeći *DatePickerDialog* komponentu koja aktivacijom prikazuje skočni prozor za odabir datuma.



Slika 6.12 Skočni prozor za odabir datuma

Dijagrami se izgrađuju koristeći *MPAndroidChart* biblioteku [12]. *LineChart* objekt, koji predstavlja linijski dijagram, prima listu *Entry* objekata. Ovi objekti predstavljaju točke na dijagramu na temelju kojih *LineChart* iscrtava liniju za prikaz vrijednosti.

Odabirom željenog datuma na *DatePickerDialog* komponenti s web-servisa dohvaćaju se sva očitanja na određeni datum te se izgrađuju *Entry* objekti za prikazivanje na linijskim dijagramima.

```
for (SensorReading reading:sensorReadings) {
      float time = getTimeValue(reading.getReadingTime());
      temperatureEntries
      .add(new Entry(time,(float)reading.getTemperature()));
      humidityEntries.add(new Entry(time,(float)reading.getHumidity()));
}
```

Kôd 6.13 Izrada točaka za prikaz na linijskom dijagramu

Kako *LineChart* ne podržava rad s jedinicama vremena, potrebno je vrijeme pretvoriti u odgovarajući broj kako bi se podaci ispravno prikazali na dijagramu, kao što je prikazano u nastavku.

```
private float getTimeValue(LocalDateTime readingTime) {
   float f = (((readingTime.getHour() * 60) +
readingTime.getMinute()) / 1440f) * 24;
   return f;
}
```

Kôd 6.14 Metoda za pretvorbu jedinice vremena u broj

6.4.9. Primanje obavijesti

Prilikom korisničke prijave u aplikaciju, aplikacija se registrira za primanje obavijesti, za čiju je provedbu odgovorna klasa *NotificationRegisterService*. Tijekom procesa registracije, aplikacija dohvaća *FirebaseInstanceId*, jedinstveni identifikator instalirane instance aplikacije na uređaju s *Firebase* servisa.

```
FirebaseInstanceId.getInstance().getInstanceId().addOnSuccessListener(new
OnSuccessListener<InstanceIdResult>() {
      @Override
      public void onSuccess(InstanceIdResult instanceIdResult) {
            FCM_token = instanceIdResult.getToken();
            register(FCM_token, new HashSet<String>());
      }
});
```

Kôd 6.15 Dohvaćanje identifikatora instalirane aplikacije s *Firebase* servisa

Dodijeljeni *FirebaseInstanceId* aplikacija šalje *HoneyComb* web-servisu koji je odgovaran za upravljanje procesom slanja obavijesti.

Primanje i prezentacija zaprimljenih obavijesti implementirano je u *FirebaseService* klasi koja nasljeđuje *FirebaseMessagingService* te je deklarirana kao pozadinski servis u *AndroidManifest.xml* datoteci:

```
<service
      android:name=".services.FirebaseService"
      android:exported="false">
      <intent-filter>
            <action
      android:name="com.google.firebase.MESSAGING_EVENT" />
      intent-filter>
</service>
```

Kôd 6.16 Deklaracija *service* komponente za primanje obavijesti

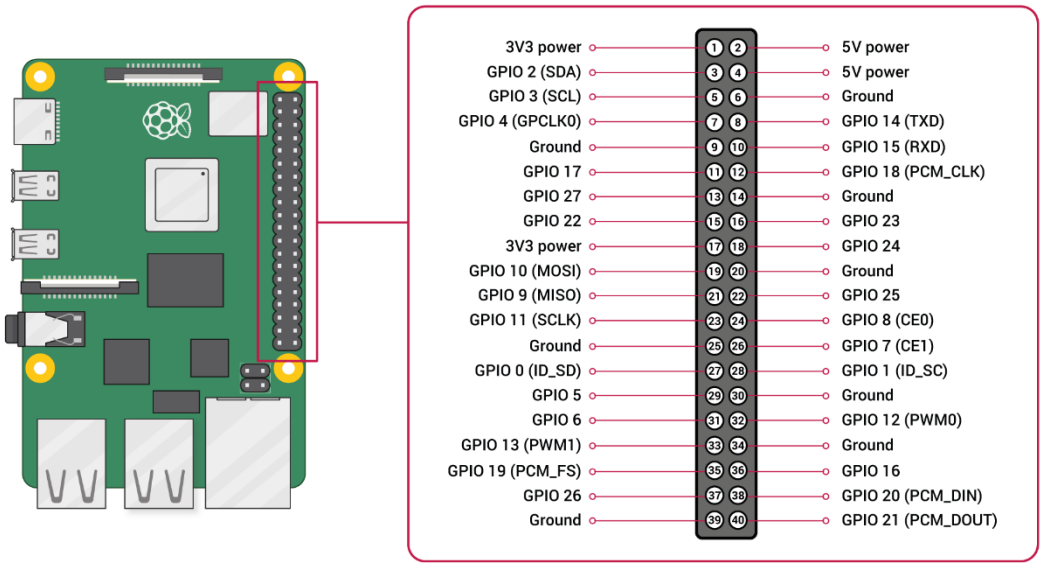
Kada operativni sustav primi poruku namijenjenu *HoneyComb* aplikaciji, prosljeđuje ju *FirebaseService* klasi. Poruka se obrađuje i vraća sustavu kao obavijest.

```
@Override
public void onMessageReceived(RemoteMessage remoteMessage) {
            String nhMessage;
      if (remoteMessage.getData().size() > 0) {
            if (remoteMessage.getData().containsKey("message")){
        nhMessage = remoteMessage.getData().get("message");
        sendNotification(nhMessage);
            }
      }
}
```

Kôd 6.17 Metoda za obradu obavijesti

7. Automatsko prikupljanje podataka korištenjem senzora

Automatsko prikupljanje podataka s pčelinjaka izvedeno je na *Raspberry Pi,* niskobudžetnom računalu, veličine bankovne kartice, izgrađenom na jednoj elektroničkoj pločici. Sadrži sve potrebne dijelove modernog računala, mikroprocesor, radnu memoriju i ulazno-izlazne komponente za povezivanje vanjskih uređaja [13]. Jedna od glavnih značajki *Raspberry Pi* računala je GPIO sučelje, ulazno-izlazno sučelje, koje nam omogućuje povezivanje i upravljanje raznim uređajima kroz programski kôd, što je bio glavni razlog odabira *Raspberry Pi* računala za potrebe automatskog prikupljanja podataka korištenjem senzora.



Slika 7.1 *Raspberry Pi* računalo s GPIO sučeljem [14]

7.1. Senzori temperature i vlage

BME280 senzor vlage zraka, proizvođača Bosch, posebno je razvijen za razvoj aplikacija na mobilnim uređajima, gdje su veličina i niska potrošnja energije ključni čimbenici. Ovaj senzor pruža visoku razinu preciznosti i nisko vrijeme odaziva te omogućuje mjerenje

relativne vlažnosti, barometarski pritisak i temperaturu zraka [15]. Specifikacija BME280 senzora prikazana je tablicom (Tablica 7.1).

Tablica 7.1 Specifikacija BME280 senzora [16]

	Najniža vrijednost mjerenja	Najviša vrijednost mjerenja	Preciznost mjerenja
Relativna vlaga zraka	0%	100 %	±3%
Barometarski pritisak	300 hPa	1100 hPa	~ ±1 hPa
Temperatura	-40 °C	85 °C	~ ±1 °C

7.2. Način rada i komunikacije s poslužiteljem

BME280 senzor povezan je putem I^2C sabirnice podržanoj na GPIO sučelju *Raspberry Pi* računala.

I^2C sabirnica izumljena je 1982. godine od strane Philips Semiconductor kompanije za povezivanje sporijih perifernih uređaja s mikrokontrolerom. Za komunikaciju koristi dvije linije: *Serial Data Line* (SDA) i *Serial Clock Line* (SCL). Dizajn I^2C sabirnice predviđa adresni prostor od 7 bita, koji omogućuje raspoznavanje više paralelno povezanih uređaja na sabirnici [17].

7.3. Aplikacija za prikupljanje podataka

Aplikacija za prikupljanje podataka razvijena je u Java programskom jeziku. Prilikom pokretanja aplikacija dohvaća konfiguraciju s web-servisa koja definira učestalost slanja podataka sa senzora, a kao jedinstveni identifikator koristi konstantu u formatu GUIDa, čija je vrijednost unaprijed definirana u bazi podataka web-servisa:

```
private static String idSensor = "6153bdef-45d7-4f27-926d-2c755d7e8f6e";
```

Kôd 7.1 Jedinstveni identifikator senzora

Za obavljanje ponavljajuće radnje očitanja u vremenu zadanom u konfiguraciji brine se *ScheduledExecutorService* objekt, koji pokreće izvršavanje objekta tipa *Runnable* s definiranim vremenskim odmakom.

```
public static ScheduledExecutorService scheduler;

public static void main(String[] args) {
      scheduler = Executors.newSingleThreadScheduledExecutor();

      try {
            initConfiguration();
            scheduler.schedule(new SensorTask(), 0, TimeUnit.SECONDS);
            scheduler.awaitTermination(Long.MAX_VALUE, TimeUnit.DAYS);
      } catch (InterruptedException ex) {
            Logger.getLogger(HoneyCombSensors.class.getName()).log(Level.
            SEVERE, null, ex);
      }
}
```

Kôd 7.2 Program za slanje očitanja sa senzora

Klasa *SensorTask* implementira sučelje *Runnable* unutar čije metode *run* se prikupljaju podaci i šalju prema web-servisu, u pozadinskoj dretvi.

Klasa BME280 koristi *pi4j* biblioteku za komunikaciju s uređajima na I^2C sabirnici *Raspberry Pi* računala te implementira metodu *getReading* u kojoj se dohvaćaju podaci pomoću adrese uređaja na sabirnici. Kôd za obradu podataka preuzet je s javno dostupnog repozitorija [18].

```
// Create I2C bus
I2CBus bus = I2CFactory.getInstance(I2CBus.BUS_1);
// Get I2C device, BME280 I2C address is 0x76(108)
I2CDevice device = bus.getDevice(0x76);

// Read 24 bytes of data from address 0x88(136)
byte[] b1 = new byte[24];
device.read(0x88, b1, 0, 24);
```

Kôd 7.3 Dohvaćanje podataka sa senzora na I^2C sabirnici

Klasa *HoneyCombWebService* implementira metode za dohvaćanje konfiguracije i slanje očitanja prema web-servisu, koristeći *HttpURLConnection*. Za pretvorbu objekata u JSON format koristi Gson biblioteku.

```
Gson g = new GsonBuilder()
      .registerTypeAdapter(LocalDateTime.class, new LocalDateAdapter())
      .create();
SensorReading sendingReading =
                  new SensorReading(Configuration.getApiaryId(),
                                    reading.getTemperature(),
                                    reading.getHumidity(),
                                    reading.getPressure(),
                                    LocalDateTime.now());
String input = g.toJson(sendingReading, SensorReading.class);
```

Kôd 7.4 Pretvorba Java objekta u JSON format pomoću GSON biblioteke

Kako bi premostili zadani način na koji GSON radi pretvorbu *LocalDateTime* objekta, koji je nerazumljiv *HoneyComb* web-servisu, potrebno je postaviti adapter tijekom izgradnje GSON-a.

```
private static final class LocalDateAdapter extends
TypeAdapter<LocalDateTime> {
      @Override
      public void write(final JsonWriter jsonWriter, final
      LocalDateTime localDate) throws IOException {
     jsonWriter.value(localDate.toString());
  }

      @Override
      public LocalDateTime read(final JsonReader jsonReader)
      throws IOException {
            return
            LocalDateTime.parse(jsonReader.nextString());
  }
}
```

Kôd 7.5 GSON *TypeAdapter* za pretvorbu *LocalDateTime* objekta

Zaključak

HoneyComb sustav, iako na trenutnoj razini implementacije sadrži samo osnovni dio funkcionalnosti primjenjivih u svakodnevnom životu pčelara, jasno prikazuje način na koji se moderne tehnologije mogu korisno primijeniti u tradicionalnim zanimanjima. Korištenje ovog sustava olakšava pčelarima vođenje evidencije o pojavljivanjima bolesti, prijašnjim radnjama na košnicama te praćenje razvoja pojedinih pčelinjih zajednica.

Središnji je dio *HoneyComb* sustava zamišljen kao nezavisna komponenta koja nije usko vezana uz klijentske aplikacije, čime je omogućeno povezivanje drugih klijentskih aplikacija u sustav ili povezivanje s drugim sustavima i bazama podataka za prikazivanje vjerodostojnijih statističkih analiza.

Web-aplikacija, koja na trenutnoj razini implementacije omogućuje statistički prikaz stanja na pčelinjacima, može poslužiti kao osnova za razvoj aplikacije kroz koju pčelari prijavom u svoj korisnički račun mogu detaljnije pregledavati stanje te pratiti napredak pčelinjih zajednica ili za prikazivanje rasprostranjenosti bolesti i količinu prinosa meda na određenome geografskom području.

Korištenje senzora u samom sustavu otvara mnoge mogućnosti, primjerice automatsko prikupljanje dnevnih prinosa meda korištenjem pčelarskih vaga ili kao sustav za upozorenje korisnika o neovlaštenom pristupu pčelinjaku.

Popis kratica

SDK	*Software development kit*	set razvojnih alata
XML	*Extensible Markup Language*	jezik za označavanje podataka
API	*Application programming interface*	sučelje za programiranje aplikacija
HTTP	*HyperText Transfer Protocol*	protokol za prijenos dokumenata
ORM	*object-relational mapper*	alat za pretvorbu podataka u objekte
JSON	*JavaScript Object Notation*	standard za razmjenu podataka
GPIO	*general-purpose input/output*	sučelje na *Raspberry Pi* računalu
I^2C	*Inter-Integrated Circuit*	sabirnica za povezivanje uređaja
GUID	*Globally Unique Identifier*	broj za identifikaciju resursa

Popis slika

Slika 2.1 Arhitektura sustava 2

Slika 2.2 Shema baze podataka 3

Slika 3.1 Postavljanje Azure SQL baze podataka 5

Slika 3.2 Postavljanje Web App servisa 6

Slika 3.3 Isporuka web-servisa iz Microsoft Visual Studio programa 6

Slika 5.1 *Basic Authentication* [8] 11

Slika 5.2 Web-stranica za prikaz statističkih podataka 16

Slika 6.1 *Login* forma 20

Slika 6.2 Navigacijski izbornik 21

Slika 6.3 Sučelje s popisom pčelinjaka u *RecyclerView* komponenti 25

Slika 6.4 Sučelje za dodavanje novog pčelinjaka 27

Slika 6.5 Sučelje za prikaz popisa košnica s izbornikom opcija 28

Slika 6.6 Skočni prozor za grupno dodavanje košnica 29

Slika 6.7 Sučelja za unos podataka o pregledu košnice 29

Slika 6.8 Sučelje za prikaz povijesti pregleda s proširivim *ViewHolderom* 31

Slika 6.9 *Spinner* s prikazanim mogućim opcijama 32

Slika 6.10 Sučelje za prikaz informacija o senzorima 33

Slika 6.11 Sučelje za prikaz očitanja sa senzora na pčelinjacima 34

Slika 6.12 Skočni prozor za odabir datuma 34

Slika 7.1 *Raspberry Pi* računalo s GPIO sučeljem [14] 37

Popis tablica

Tablica 6.1 Specifikacija BME280 senzora [16] 38

Popis kôdova

Kôd 4.1 Primjer potrebnih dozvola u *AndroidManifest.xml* datoteci 8

Kôd 4.2 Primjer deklariranja komponente u *AndroidManifest.xml* datoteci 9

Kôd 5.1 Provjera korisničkog imena i lozinke 12

Kôd 5.2 Sučelje za rad s korisničkim podacima 12

Kôd 5.3 Dohvaćanje podataka iz baze koristeći se *Entity Framework Coreom* 13

Kôd 5.4 Korisnički podaci formatirani JSON formatom 13

Kôd 5.5 Obrađena lista košnica u JSON formatu 14

Kôd 5.6 Metoda za dohvat prosječnog broja košnica po korisniku na mjesečnoj bazi 15

Kôd 6.1 Metoda za izgradnju *Retrofit* HTTP klijenta 18

Kôd 6.2 Primjer sučelja za dohvaćanje podataka o pčelinjacima s web-servisa 19

Kôd 6.3 Kreiranje *popup* prozora za prikaz poruka 20

Kôd 6.4 Stavke navigacijskog izbornika 22

Kôd 6.5 Implementacija *FragmentLoader* sučelja 22

Kôd 6.6 Dohvaćanje popisa pčelinjaka s web-servisa 24

Kôd 6.7 Postavljanje adaptera na *RecyclerView* 25

Kôd 6.8 Postavljanje vrijednosti *ViewHoldera* 26

Kôd 6.9 Traženje dopuštenja za korištenje usluga lokacije uređaja 27

Kôd 6.10 Kreiranje fragmenata pomoću refleksije 30

Kôd 6.11 Provjera i dohvaćanje detalja o pregledu 31

Kôd 6.12 Implementacija promjene odabranog objekta na *Spinneru* 32

Kôd 6.13 Izrada točaka za prikaz na linijskom dijagramu 35

Kôd 6.14 Metoda za pretvorbu jedinice vremena u broj 35

Kôd 6.15 Dohvaćanje identifikatora instalirane aplikacije s *Firebase* servisa 35

Kôd 6.16 Deklaracija *service* komponente za primanje obavijesti 36

Kôd 6.17 Metoda za obradu obavijesti 36

Kôd 7.1 Jedinstveni identifikator senzora 38

Kôd 7.2 Program za slanje očitanja sa senzora 39

Kôd 7.3 Dohvaćanje podataka sa senzora na I^2C sabirnici 39

Kôd 7.4 Pretvorba Java objekta u JSON format pomoću GSON biblioteke 40

Kôd 7.5 GSON *TypeAdapter* za pretvorbu *LocalDateTime* objekta 40

Literatura

[1] HURWITZ, J., BLOOR, R., KAUFMAN, M., HALPER, F. *Cloud Computing For Dummies*, Wiley Publishing, Inc., 2010.

[2] Microsoft Azure, What is Azure? https://azure.microsoft.com/en-us/overview/what-is-azure/, siječanj 2020.

[3] Microsoft Azure, What is the Azure SQL Database service? https://docs.microsoft.com/en-us/azure/sql-database/sql-database-technical-overview, siječanj 2020.

[4] TUTORIALS POINT, Android Tutorial, 2014.

[5] Computerworld, Android versions: A living history from 1.0 to 10 https://www.computerworld.com/article/3235946/android-versions-a-living-history-from-1-0-to-today.html, siječanj 2020.

[6] PELGRIMS, K., *Gradle for Android*, Packt Publishing, srpanj, 2015.

[7] Android developers, https://developer.android.com/guide, siječanj 2020.

[8] Microsoft, Basic Authentication in ASP.NET Web API https://docs.microsoft.com/en-us/aspnet/web-api/overview/security/basic-authentication, siječanj 2020.

[9] Json.org, Introducing JSON https://www.json.org/json-en.html, siječanj 2020.

[10] Wikipedia, Angular (web framework) https://en.wikipedia.org/wiki/Angular_(web_framework), siječanj 2020.

[11] Firebase, Firebase Cloud Messaging https://firebase.google.com/docs/cloud-messaging, siječanj 2020.

[12] GitHub, MPAndroidChart https://github.com/PhilJay/MPAndroidChart/wiki, siječanj 2020.

[13] RaspberryPi.org, What is a Raspberry Pi? https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/, siječanj 2020.

[14] RaspberryPi.org, GPIO https://www.raspberrypi.org/documentation/usage/gpio/, siječanj 2020.

[15] Bosch Sensortec, BME280 https://www.bosch-sensortec.com/products/environmental-sensors/humidity-sensors-bme280/, siječanj 2020.

[16] Bosch Sensortec GmbH, BME280 Datasheet

[17] Wikipedia, I^2C https://en.wikipedia.org/wiki/I%C2%B2C, siječanj, 2020.

[18] GitHub, ControlEverythingCommunity/BME280 https://github.com/ControlEverythingCommunity/BME280, siječanj, 2020.

ALGEBRA
VISOKO
UČILIŠTE

IZRADA APLIKACIJE ZA VOĐENJE EVIDENCIJE O PČELINJACIMA

Pristupnik: Željko Severović, 0321006709

Mentor: struč.spec.ing.comp Daniel Bele