Mobilne aplikacije za prijavu kvarova u hotelijerstvu

Parencan, Dejan

Sveučilište Jurja Dobrile u Puli
Fakultet informatike

DEJAN PARENCAN

MOBILNA APLIKACIJA ZA PRIJAVU KVAROVA U HOTELIJERSTVU

Diplomski rad

Pula, 2018. godine.

Sveučilište Jurja Dobrile u Puli
Fakultet informatike

DEJAN PARENCAN

MOBILNA APLIKACIJA ZA PRIJAVU KVAROVA U HOTELIJERSTVU

Diplomski rad

JMBAG: 2424013586, izvanredni student

Studijski smjer: Diplomski studij informatike

Predmet: Mobilne aplikacije

Znanstveno područje: Društvene znanosti
Znanstveno polje: Informacijske i komunikacijske znanosti
Znanstvena grana: Informacijski sustavi i informatologija

Mentor: Doc. dr. sc. Siniša Sovilj

Pula, 12.veljače 2018. godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Dejan Parencan, kandidat za magistra informatike ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Diplomskog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

Parencan Dejan

U Puli, 12. veljače 2018 godine

IZJAVA
o korištenju autorskog djela

Ja, Dejan Parencan dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom „Mobilna aplikacija za prijavu kvarova u hotelijerstvu“ koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, 12. veljače 2018 godine.

Potpis

Parencan Dejan

SVEUČILIŠTE JURJA DOBRILE U PULI
ODJEL ZA INFORMACIJSKO-KOMUNIKACIJSKE TEHNOLOGIJE

Pula, 1. ožujka 2017.

DIPLOMSKI ZADATAK

Pristupnik:	**Dejan Parencan (2424013586)**
Studij:	Sveučilišni diplomski studij Informatike
Naslov (hrv.):	**Mobilna aplikacija za prijavu kvarova u hotelijerstvu**
Naslov (eng.):	Mobile application for failures registration in hotel industry
Opis zadatka:	Izraditi Android aplikaciju za prijavu i zaprimanje prijavljenih kvarova unutar hotelskih smještajnih jedinica i drugim smještajnim kapacitetima. Cilj izrade Android aplikacije je unaprjeđenje dosadašnjeg procesa prijave-zaprimanja prijave te zadovoljstva gostiju brzinom otklonjenoga kvara. Na vremenski period koji je potreban za otklanjanje samog kvara ne može se utjecati, ali može se utjecati na vremenski period od prijave do primitka same prijave za otklanjanje kvarova koja ovom aplikacijom treba direktno stizati samom djelatniku ili više njih na pametni telefon. Potrebno je istražiti hoće li aplikacija olakšati posao samim zaposlenicima te zadovoljiti gosta kroz anketu kako zaposlenika koji direktno koriste aplikaciju tako i gostiju. U istraživačkom dijelu potrebno je utvrditi statističkom analizom rezultata ankete gostiju njihovo zadovoljstvo kompletnom percepcijom otklonjenoga kvara prije korištenja same Android aplikacije i nakon korištenja aplikacije na određenom broju ispitanika kroz određeni period.

Zadatak uručen pristupniku: 1. ožujka 2017.
Rok za predaju rada: 1. veljače 2018.

Mentor:

Siniša Sovilj

doc.dr.sc. Siniša Sovilj

SADRŽAJ

UVOD

Razvojem tehnologije dolazi do razvoja mobilnih uređaja. Oni se više ne koriste samo za telefoniranje i slanje poruka već postaju „pametniji“ i omogućuju korisnicima izvođenje nekih zadataka koje do tada nisu mogli. Ti iste uređaje više ne nazivamo mobitelima već pametnim uređajima. Data evolucija nije donijela korist samo korisnicima već je omogućila programerima i razvijateljima aplikacija razvijanje aplikacija za takve uređaje. Kao pobjednik iz mase operacijskih sustava izlazi Android. Android je stekao široku publiku na tržištu što zbog svoje cijene što zbog modifikacija operacijskog sustava koji je omogućio proizvođačima takvih uređaja da modificiraju operacijski sustav i prilagode svojim uređajima i potrebama.

Ovaj diplomski rad bavi se izradom mobilne aplikacije za prijavljivanje kvarova unutar hotelskih kuća na Android pametnim uređajima. Jedina restrikcija koja se postavljala na ovaj projekt bio je korištenje MySQL baze podataka radi posjedovanja na vlastitom poslužitelju ili nadograđivanja same aplikacije i prelaska na moguće web rješenje. Osim toga postojala je dvojba oko korištenja JDBC-a od strane JAVA programskog jezika i PHP-a za komunikaciju sa samom MySQL bazom podataka. Googleov-a Firebase baza podataka naizgled se činila kao najbolji odabir kod izrade ovakvih tipova klijent-poslužitelj aplikacije no zbog same restrikcije u korištenju MySQL baze podataka i zbog toga što je sama migracija podataka sa Firebase na neki drugu bazu podataka ponekad otežana, odustalo se od iste.

Zbog nemogućnosti provedbe diplomskoga zadatka, isti je uz dopuštenje mentora izmijenjen usporedbom između Firebase i MySQL baze podataka.

Usporedba MySQL baze podataka i Firebase baze podataka bila je jedna od temeljnih zadaća ovog diplomskog rada. No pošto Firebase koristi određene mogućnosti koji su za ovaj projekt neophodne a MySQL ne, potrebno je bilo približiti se radu Firebase baze podataka. Dali je pisac ovoga rada to uspio otkrit ćemo kroz nastavak ovoga rada.

1. ANDROID OPERACIJSKI SUSTAV

1.2. Povijest Android operacijskog sustava

Android[1] je operacijski sustav otvorenog koda dostupan svima na korištenje. Temelji se na Linux operacijskom sustavu. Namijenjen za uređaje sa zaslonom na dodir kao što su pametni telefoni i tablet uređaji. Povijest samog android operacijskog sustava vezana je uz tvrtku Android Inc. Koja je osnovana 2003. godine u Palo Alto (California,USA) od *strane* Andy Rubin*,* Rich Miner*,* Nick Sears i Chris White. Cilj osnivača bio je razvoj operacijskog sustava za mobilne uređaje koji bi omogućio vlasnicima veće mogućnosti od prijašnjih uređaja Iako je na početku uspjeh bio velik ubrzo je uslijedio nedostatak financijske potpore. U kolovozu 2005 godine stvar spašava Google[2] koji otkupljuje Android Inc. omogućivši vodećim ljudima Android Inc-a nastavak svojim dosadašnjim radom. Prednosti Android operacijskog sustava uvidjeli su i proizvođači mobilnih uređaja koji se međusobno udružuju i tako 2007 godine nastaje Open Handset Alliance[3] savez koji danas broji preko 80 tehnoloških kompanija kao što su T-Mobile, Intel, Nvidia, Qualcomm, HTC, LG, Samsung i mnoge druge.

[1] Dostupno na Android službenim stranicama

[2] Službena stranica Google-a

[3] Službena stranica Open Handset Alliance

Slika 1.Primjer Android operacijskog sustava

Izvor: Softpedia

1.3. Rasprostranjenost Android operacijskog sustava

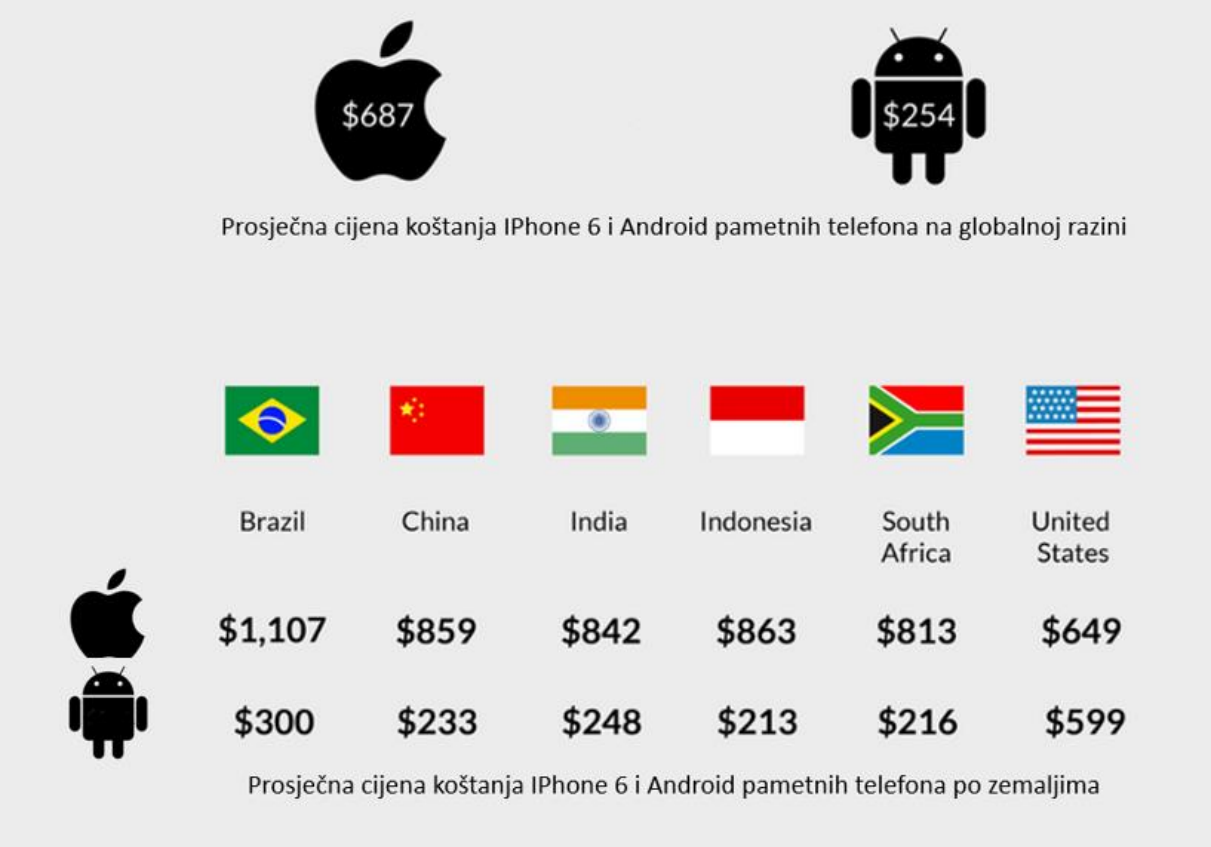
Na sljedećoj slici prikazani su udjeli(u postocima) između uređaja koji pokreću Android operacijski sustav od onih drugih proizvođača.

Slika 2. Udijeli na tržištu među operacijskim sustavima za pametne uređaje

Period	Android	iOS	Windows Phone	Others
2016Q1	83.4%	15.4%	0.8%	0.4%
2016Q2	87.6%	11.7%	0.4%	0.3%
2016Q3	86.8%	12.5%	0.3%	0.4%
2016Q4	81.4%	18.2%	0.2%	0.2%
2017Q1	85.0%	14.7%	0.1%	0.1%

Izvor: wccftech

Slika 3. Prosječna cijena po zemljama i na globalnoj razini



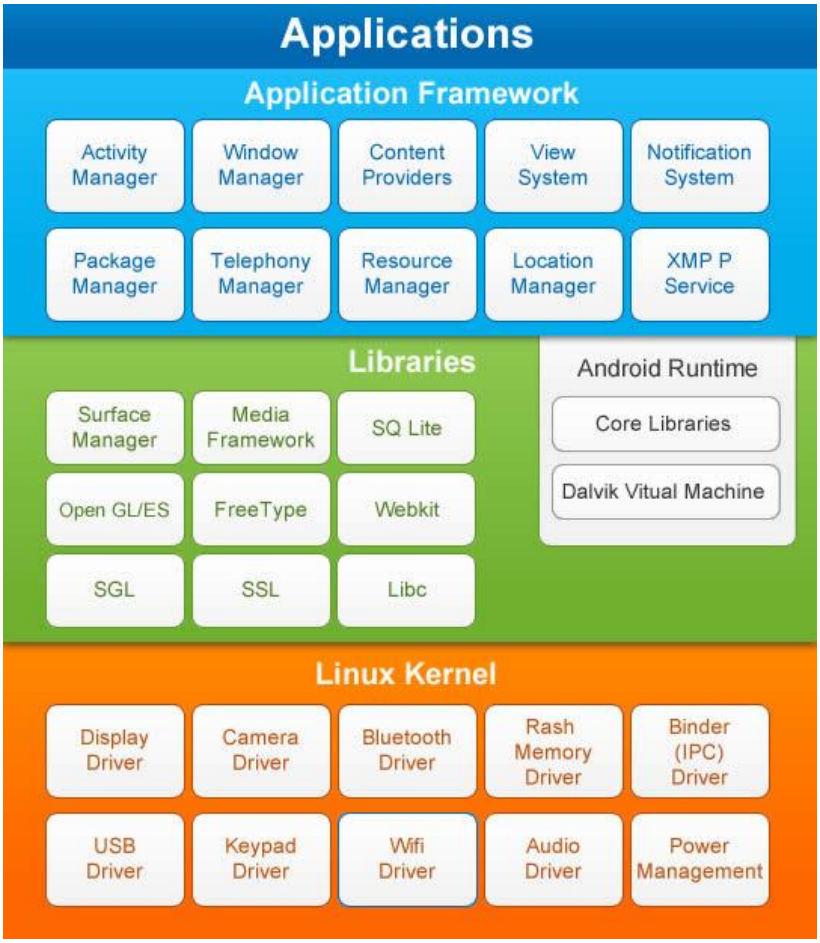
Izvor: modificirano prema: JANA

Slika iznad prikazuje prosječnu cijenu koštanja Android pametnih telefona nasuprot IPhone 6 po zemljama i na globalnoj razini. Iz date slike da se zaključiti da nije ni čudo sto je Android toliko rasprostranjen i cijena koštanja je daleko niza od IPhone-a a jedan od razloga leži u otvorenom kodu samog Android operacijskog sustava koji su prigrlili mnogi proizvođači pametnih telefona ali i drugih uređaja baziranih na Android operacijskom sustavu te to što omogućuje svakom proizvođaču modifikaciju istog.

1.4. Arhitektura Android operacijskog sustava

Android operacijski sustav se sastoji od različitih softverskih komponenti koji su podijeljeni unutar četiri sloja.

Slika 4. Arhitektura Android operacijskog sustava



Izvor: datayo

1.4.1. Linux kernel(jezgra)

Linux kernel odnosno jezgra nalazi se na dnu arhitekture Android operacijskog sustava i omogućuje osnovne funkcionalnosti sustava kao što su upravljanje memorijom, upravljanje procesima te upravljanjem ugrađenim uređajima kao što su tipkovnice, kamere i žiroskop. Važno je napomenuti da je programerima Android aplikacija sama jezgra nedostupna.

1.4.2. Libraries

Libraries ili u prijevodu biblioteke dolaze iznad sloja Linux jezgre te predstavlja skup biblioteka zadužene za funkcionira Android operacijskog sustava među kojima

SQLite, SSL biblioteke koje su odgovorne za sigurnost na Internetu, i open-source alat WebKit za web preglednik.

1.4.3. AndroidRuntime

Nalazi se na drugom sloju arhitekture koju čine biblioteke jezgre koje omogućuju programerima Android aplikacija pisanje istih koristeći standardni Java programski jezik i Dalvik virtualni stroj. Dalvik je vrsta virtualnog stroja nastala od Java virtualnog stroja(*eng. Java Virtual Mashine*) napravljen od strane Googleovog zaposlenika Dan-a Bornestein-a, posebno prilagođenog za Android operacijski sustav. Dalvik omogućuje svakoj android aplikaciji rad u vlastitom procesu s vlastitom instancom. Dalvik koristi osnovne značajke Linux operacijskog sustava a to su višedretveno izvođenje (*eng. Multithreading*) i upravljanje memorijom.

1.4.4. Application Framework

Application Framework pruža mnoge servise i biblioteke Android programerima prilikom izrade same android aplikacije.

1.4.5. Applications

Aplikacije(*eng.Applications*) nalaze se na najvišem sloju Android operacijskog sustava. Aplikacije same po sebi ne raspolažu administratorskim ovlastima potrebnima za mijenjanje samog operacijskog sustava i nemaju direktan pristup resursima Android uređaja već to čine preko Dalvik virtualnog stroja.

1.5. Java programski jezik

Java[4] je objektno orijentirani programski jezik razvijen od tvrtke Sun

[4] Službena stranica Java

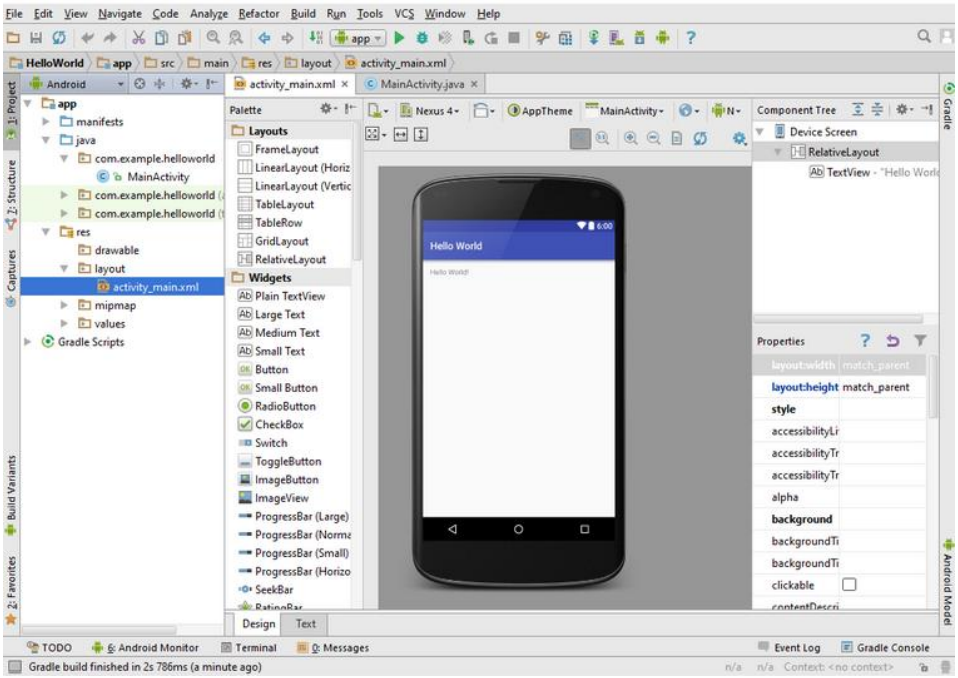
Microsystems. Cilj inženjera zaduženih za kreiranje jave bio je da isti mora biti jednostavan, objektno orijentiran, robustan i siguran, neovisan o arhitekturi. Glavni moto jave je da se jednom napisan kod može izvršavati na svakom računalu(WORA-write once run evrewhere) što je realizirano preko JVM(Java Virtual Mashine).

Java je danas jedan od najviše zastupljenijih i korištenih programskih jezika te se procjenjuje da ga koristi više od 9 milijuna ljudi širom svijeta. Do danas je obavljeno 9. verzija Jave od JDK 1.0 iz 1996. godine do JDK 9. iz 2017. godine.

1.6. Razvojno okruženje Android Studio

Android Studio je službeno slobodno razvojno okruženje*(eng. IDE*) za izradu Android aplikacija od strane Google-a. Valja napomenuti da Android studio nije jedino razvojno okruženje za android već je tu i Eclipse no isti je izgubio razvojnu podršku pojavom Android studija. Android studio temelji se na JetBrains' IntelliJ IDEA arhitekturi razvoja. Najavljen od strane Googlea na Google I/O 2013. konferenciji svjetlo dana svoje prve stabilne verzije(0.8) ugledao je 2014 godine. Danas se koristi verzija 3.0 koja je izašla u 10 mjesecu 2017. U ovom diplomskom radu korištena je verzija Android sustava je 2.2.

Slika 5. Razvojno okruženje Android Studio



Izvor: Autor

1.7. Android manifest

Android manifest je xml. datoteka unutar koje su definirani svi sastavni dijelovi projekta(aktivnosti), stilovi i dozvole pristupa. Kad govorimo o dozvolama pristupa prvenstveno mislim na dozvole pristupa same aplikacije hardveru Android uređaja(kamere i senzori). Ako aplikacija nema definirane dozvole pristupa unutar android manifesta ista neće moći koristiti sklopovlje samog android uređaja. Na sljedećoj slici dat je primjer Android manifesta aplikacije „Održavanje“. Pošto aplikacija ima mogućnost paljenja svjetiljke koju korisnik može koristiti za osvjetljavanje prostora ili nečega drugog, za istu je bilo potrebno tražiti dopuštenje za korištenje unutar samog manifesta.

Slika 6. Primjer Android manifesta

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.user.odrzavanje">

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <!-- Allows access to the flashlight -->
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.FLASHLIGHT" />
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    <uses-feature android:name="android.hardware.Camera" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/pocetna"
        android:label="Održavanje"
        android:largeHeap="true"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".Uvodna"
            android:theme="@style/AppTheme.NoActionBar"
            android:screenOrientation="portrait" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

Izvor: Autor

1.8. Aplikacijske komponente

Aplikacijske komponente su temelji svake Android aplikacije. One imaju svoj životni ciklus te omogućuju korisniku rad na njima definiran način.

Razlikujemo sljedeće tri osnovne aplikacijske komponente:

- Aktivnosti
- Fragmenti
- Usluge

1.8.1.Aktivnosti

Aktivnost(*eng.Activity*) je najbitnija komponenta unutar same Android aplikacije. Svaka aplikacija ima barem jednu aktivnost no može ih imati i više. Jedna aktivnost može pokretati drugu na zahtjev korisnika. Svaka je aktivnost unutar skupa aktivnosti nezavisna jedinka i sastoji se od XML. datoteke u kojoj je definiran izgled same aktivnosti. Najbolja usporedba aktivnosti bilo bi web sjedište(aplikacija) koje se sastoji od nekoliko web stranica(aktivnosti).Svako pokretanje aktivnosti iziskuje pokretanje novog procesa, alociranje memorije u samom operacijskom sustavu za sve objekte korisničkog sučelja, te drugih resursa potrebnih za pokretanje aktivnosti. Iz tog razloga svaka aktivnost ima definirani životni ciklus kojem upravlja Activity Menager.

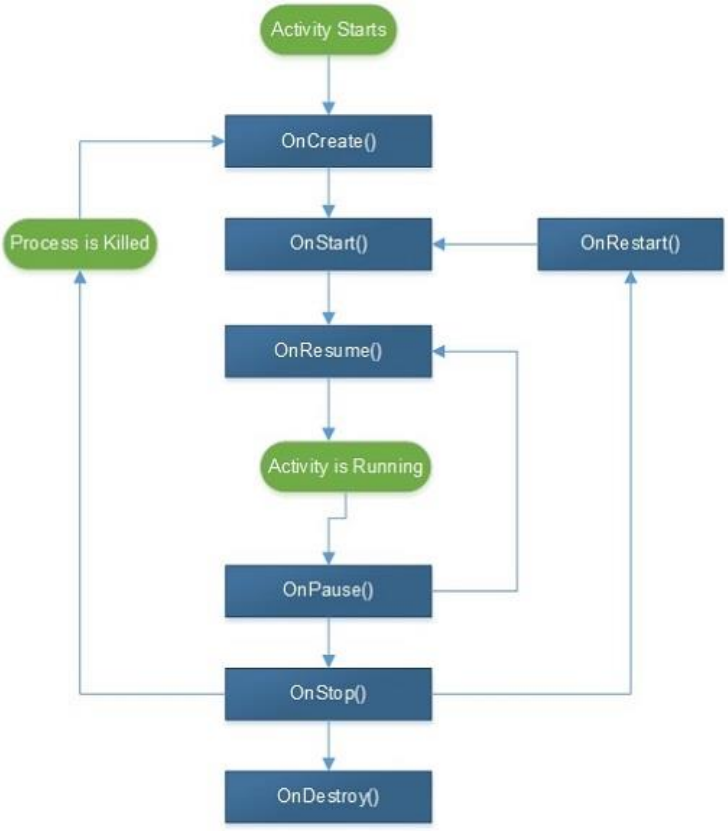
Životni ciklus aktivnosti potreban je iz razloga što kad korisnik napusti istu i nakon nekog vremena se vrati. bez životnog ciklusa, aktivnost bi se pokretala iz početka, no zbog Activity Menager-a ista se postavlja u stanje na čekanje te nakon ponovnog pokretanja se brze učitava. Ako korisnik ne pristupi aktivnosti kroz duži period ista se briše iz memorije i oslobađa se prostor za kreiranje novih aktivnosti

Svaka aktivnost od trenutka kreiranja prolazi kroz nekoliko životnih faza. Svaka se faza može upravljati sljedećim metodama:

- onCreate() - poziva se pri prvom kreiranju aktivnosti, koristi se za inicijalizaciju promjenjivih varijabli
- onStart() - poziva se kada aktivnost postane vidljiva korisniku

- onResume() - poziva se kada aktivnost počinje da se koristi od strane korisnika
- onPause() - poziva se kada je trenutna aktivnost privremeno zaustavi, a prethodna aktivnost nastavlja sa radom nastavlja.
- onStop() - poziva se kada aktivnost nije više vidljiva korisniku. Pozivom metode zaustavljaju se procesi koji nepotrebno troše memoriju ili procesor uređaja
- onDestroy() - poziva se pri uništavanju aktivnosti
- onRestart() - poziva se kada je aktivnost bila zaustavljena i ponovo se prekreće
- onSaveInstanceState(Bundle) - metoda koja se poziva u slučaju spremanja stanja aktivnosti(opcionalna)
- onRestoreInstanceState(Bundle) - poziva se prilikom ponovnog pokretanja pamteći prijašnje spremljeno stanje aktivnosti.

Slika 7. Životni ciklus aktivnosti



Izvor: Tutorialspoint

1.8.2. Fragmenti

Fragmenti su dijelovi aktivnosti. Pod dijelovima ne mislim suštinski na dio aktivnosti već kao dio proširene aktivnosti. Fragmenti su jako zgodni kada aktivnost moramo podijeliti na vise dijelova, prikaza korisničkog sučelja. Fragmenti ne mogu postojati sami za sebe već su uvijek vezani uz aktivnost iako imaju svoj zaseban životni ciklus. Fragmenti ne utječu na životni ciklus aktivnosti te se isto mogu dodavati i uklanjati dok se aktivnost izvodi. Valja napomenuti da prestankom aktivnosti tj. uništavanjem aktivnosti uništava se i fragment. Svaki fragment se kao i aktivnost sastoji od svoje XML datoteke u koje je definiran izgled korisničkog sučelja fragmenta.

1.8.3. Namjere

Namjere su poruke koje se šalju najčešće između dviju aktivnosti. Njihova glavna funkcija nije samo da šalju poruke već da pokreću aktivnost. Namjere su vrlo zgodne u slučaju da moramo proslijediti nekakav podatak iz jedne u drugu aktivnost prilikom pokretanja.

1.8.4. Servisi

Kada postoji potreba obavljanja određenih zadataka u pozadini aplikacije te nam za to ne treba nikakvo korisničko sučelje tada koristimo servise. Servisi omogućuju obavljanje zadataka u pozadini. Neovisne su o sadržaju prikazanom na zaslonu aplikacije. Imaju životni ciklus kao i aktivnosti no sa drastično reduciranim brojem stanja(pokrenute ili zaustavljene).

2. OSTALE TEHNOLOGIJE KORIŠTENE ZA IZRADU APLIKACIJE

2.1. Baza podataka

Baza podataka koriste se za čuvanje, manipulaciju i prikupljanje podataka. Baza je organizirana i uređena cjelina međusobno povezanih podataka spremljenih bez

nepotrebne redundancije (zalihosti). Baza podataka omogućuje pristup podacima višestrukom broju korisnika te pohranu i dohvat podataka spremljenih bazu preko poslužitelja. Sama povijest nastanka baze podataka vezana je uz sustav za automatsku obradu podataka popisa stanovništva u Americi. Patent je prijavio 1884 godine Herman Holerith. Njegov sustav zasnivao se na bušenim karticama koje su se ručno unosile u uređaj za očitavanje, dok je sustav automatski prebrojavao podatke. Sustavi upravljanja bazom podataka možemo podijeliti u 4 osnovna modela:

- Hijerarhijski model - kod ovog modela podaci su poredani unutar hijerarhijske strukture, isti je ujedno najstariji model baze podataka.
- Mrežni model - može se predstaviti usmjerenim grafovima gdje su čvorišta podaci a lukovi definiraju veze među samim podacima.
- Relacijski model - zasnovan na matematičkom modelu relacija, najčešće korišteni model, podaci i veze prikazani su dvodimenzionalnim tablicama određenog broja retka i stupaca.
- Objektni model - temelji se na konceptu objekta koji predstavlja skup podataka i operacija koji se nad tim podacima mogu izvršiti.

U ovom diplomskom radu korišten je relacijski model baze podataka tvrtke MySQL, a što je MySQL slijedi u sljedećem poglavlju.

2.1.1. MySQL

MySQL[5] je relacijska baza podataka tj. relacijski sustav upravljanja bazama podataka. MySQL je otvorenog koda što znači da isti dozvoljava korištenje svakome i prilagođavanje svojim potrebama. Distribuiran je pod GPL licencom(GNU General Public License-besplatno za kućnu upotrebu). Samo dohvaćanje podataka i pretraživanje baze podataka radi se pomoću SQL upita(*eng. Structured Query Language*- strukturni jezik za pretraživanje) koji je ujedno najčešći standardizirani jezik za pristupanje bazama podataka.

MySQL je brz, lagan za korištenje, pouzdan i fleksibilan. Koristi se godinama za

[5] Službena stranica MySQL

manipulaciju velikih baza podataka u najzahtjevnijim produktivnim uvjetima rada.

2.2. Skriptni jezik

2.2.1. PHP

PHP(*eng. Hypertext Preprocessor)*[6] je skriptni jezik koji se koristi na poslužiteljskoj strani(*eng. Server side*) za kreiranje dinamičkih i interaktivnih web rješenja. Razvijen je davne 1995. godine od strane Rasmus Ledorf-a koji je razvio skriptu za brojanje posjećenosti vlastite web stranice. PHP je besplatan i otvorenog koda te predstavlja odličnu alternativu Microsoftovom ASP. Podržava rad sa mnogim bazama podataka: MySQL, PostgreSQL, Oracle, Generic ODBC…

U ovom diplomskom radu PHP je korišten za izvršavanje upita u MySQL bazu podataka na poslužiteljskoj strani.

2.3. Format za razmjenu podataka

2.3.1. JSON

JSON (*eng. JavaScript Object Notation*)[7] je jednostavan format za strukturiranje i razmjenu podataka tvoren po principu ključ->vrijednost. Najčešće se koristi za razmjenu podataka između aplikacija i poslužitelja kao zamjena za XML. Ključ je uvijek tipa string dok vrijednost može poprimiti vrijednosti tipa:

- object
- string
- number
- array
- true
- false

[6] Službena stranica PHP

[7] Službena stranica JSON

- null

Strukturu samog JSON-a možemo organizirati kao što je već rečeno unutar skupa ključ-> vrijednost , ali i unutar niza koji je uređena lista vrijednosti od kojih je svaki član niza odvojen zarezom(„,“) i obilježen uglastim(„[,]“) zagradama.

JSON svakim danom sve više zamjenjuje svog suparnika XML iz razloga što JSON ne koristi tagove te stoga ima kraći kod koji je ujedno lakši programeru za samo pisanje te u konačnici razumljiviji za čitanje.

Slika 9. JSON

Izvor: Autor

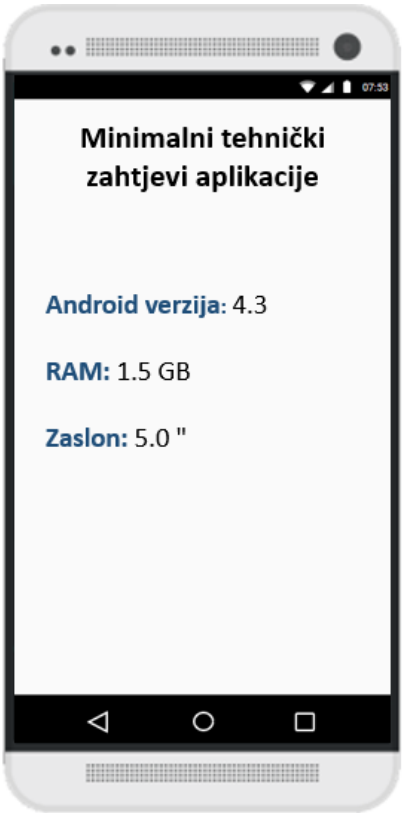
Prikazana slika prikazuje izgled JSON u kojemu imamo 4 JSON objekta sa njihovim vrijednostima od kojih je jedan niz.

3. ANDROID APLIKACIJA “ODRŽAVANJE“

3.1. Tehnički zahtjevi aplikacije i zastupljenost Android operacijskog sustava

Pošto se radi o klijent-poslužitelj arhitekturi aplikacije gdje klijent u ovom slučaju pametni uređaj sa instaliranim Android operacijskim sustavom izvršava upite poslužitelju na što poslužitelj pametnom uređaju(pametni telefon ili tablet) vrača rezultate upita. Fokus nije usmjeren na poslužitelja koliko na klijenta zbog ograničenja same aplikacije a taj je što je pisana za Android. Minimalne tehničke specifikacije koje bi trebao posjedovati Android uređaj su prikazane na sljedećoj slici.

Slika 10. Tehnički zahtjevi aplikacije



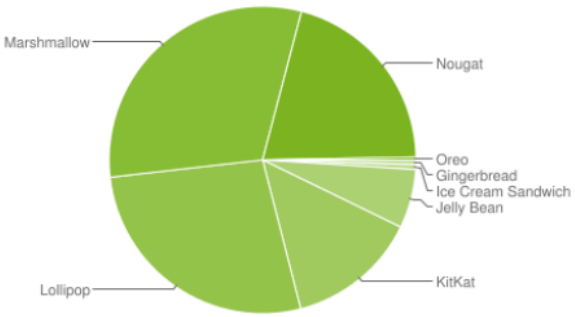
Izvor: Autor

Navedeni minimalni tehnički zahtjevi aplikacije glavni su kriterij za nesmetan rad aplikacije iako je minimalna verzija Androida najvažnija jer na uređajima sa starijim verzijama aplikacija neće zasigurno raditi. Za rad bljeskalice potrebno je da uređaj ima

ugrađenu bljeskalicu. Bljeskalica nije uvrštena u minimalne tehničke zahtjeve aplikacije iz razloga što nije neophodna za rad same aplikacije.

Slika 11. Tržišni udio pojedinih verzija Android operacijskog sustava

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.5%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.5%
4.1.x	Jelly Bean	16	2.2%
4.2.x		17	3.1%
4.3		18	0.9%
4.4	KitKat	19	13.8%
5.0	Lollipop	21	6.4%
5.1		22	20.8%
6.0	Marshmallow	23	30.9%
7.0	Nougat	24	17.6%
7.1		25	3.0%
8.0	Oreo	26	0.3%



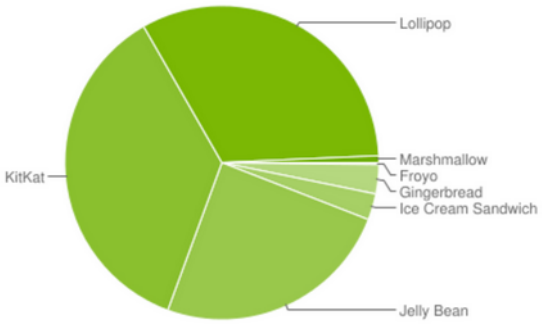
Data collected during a 7-day period ending on November 9, 2017.
Any versions with less than 0.1% distribution are not shown.

Izvor: Developer Android

Na prikazanoj slici može se vidjeti zastupljenost pojedine verzije Androida na tržištu(Slika 11.). Iz iste slike možemo iščitati da za period od 7 dana do 9 studenog 2017 najviše korištena verzija Android sustava bila je 6.0 Marshmalow. Ova aplikacija kao što je prije navedeno podržava od verzije 4.3 iz razloga što postoji mogućnost korištenja uređaja starije generacije Android sustava te se na taj način željelo obuhvatiti širem spektru Android uređaja jer još uvijek postoje na tržištu i prodaju se uređaji sa starijim Android operacijskim sustavom. Dodatna stavka tome u prilog je korisnik već posjeduje uređaj sa Android operacijskim sustavom postoji vjerojatna mogućnost da neće trebati zamjena uređaja sa novim samo zbog verzije samog Android operacijskog sustava u uređaju što potkrepljuje sljedeća slika dijagrama zastupljenosti Android verzije sustava za 1. mjesec. 2016.

Slika 12. Tržišni udio pojedinih verzija Androida(1. mjesec 2016. godine)

Version	Codename	API	Distribution
2.2	Froyo	8	0.2%
2.3.3 - 2.3.7	Gingerbread	10	3.0%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	2.7%
4.1.x	Jelly Bean	16	9.0%
4.2.x		17	12.2%
4.3		18	3.5%
4.4	KitKat	19	36.1%
5.0	Lollipop	21	16.9%
5.1		22	15.7%
6.0	Marshmallow	23	0.7%



Izvor: droidlife

Iz date ilustracije ne samo da je vidljivo da najzastupljenija ondašnja verzija Android sustava je 4.4 već, aplikacija održavanje podržava i verziju starije generacije iako u malom postotku od samo 3.5%.

3.2. Opis aplikacije

Mobilna aplikacija“ Održavanje“ namijenjena je za prijavu i zaprimanje kvarova unutar Hotelskih kuća ili drugih smještajnih jedinica od strane osoblja prvenstveno odjela domaćinstva i odjela održavanja. Arhitektura same aplikacije je klijent-poslužitelj koja je opisana tokom samog rada. Korisnost same aplikacije je velika iz razloga što određene hotelske kuće(kompanije) posjeduju programska rješenja za prijavu i evidenciju kvarova ali ona nisu mobilna tj. nisu namijenjena za rad na mobilnim uređajima. Ovdje valja naglasiti da postoje i one hotelske kompanije koji ovakav oblik evidencije kvarova dan danas rade ručno tj zapisuju se u dnevnik kvarova koji se periodički provjerava od strane odjela održavanja. One kao takve nisu pregledne i samo vrijeme proteklo od prijave kvara pa do rješavanja je veće iz razloga što svi sudionici unutar procesa od prijave pa najbitnije rješavanja kvarova nisu navrijeme

informirani o statusu samoga kvara tj jeli prijavljeni novi kvar ili riješeni stari. Valja napomenuti da je ista aplikacija jako zgodna za korištenje u kampovima gdje je osoblje stalno na terenu te se prijava istih izvodi usmenim putem preko telefona sto može rezultirati ne otklanjanju prijavljenoga kvara kao rezultat zaboravljanja od strane osoblja.

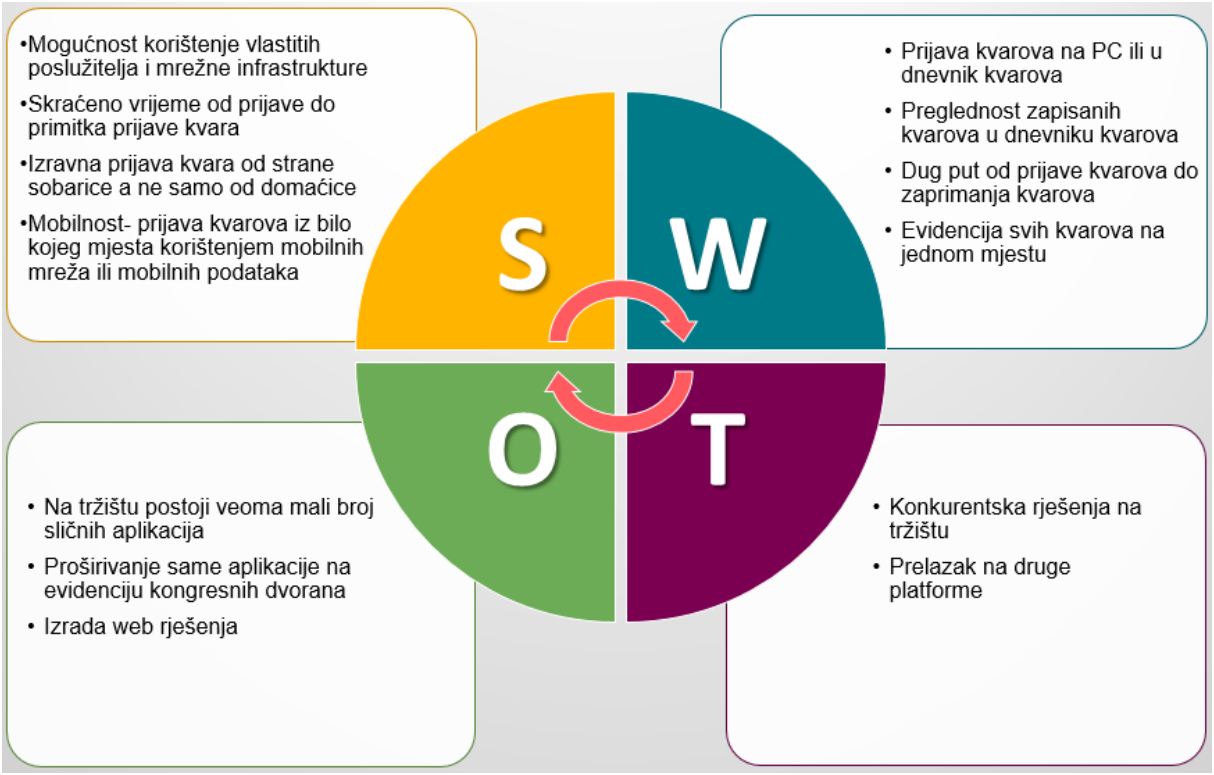
Aplikacija „Održavanje“ omogućava zaprimanje novoga kvara od strane domaćinstva te rješavanje najkraćem roku od odjela održavanja. Pošto je sama aplikacija mobilna ista omogućuje direktnu prijavu kvarova od strane sobarica odjelu održavanja. Česta je praksa da se svi kvarovi prijavljuju odjedanput a ne u trenutku nastajanja sto osim sto produžava vrijeme otklanjanja kvarova od nastajanja, povećava i obim posla odjelu održavanja jer se svi kvarovi prijavljuju odjednom. S druge strane smanjuje se vrijeme potrebno da se sobe koje su izgenerirane(očišćene) a imaju kvar prije stave u funkciju za iznajmljivanje novom gostu. Osim odjela domaćinstva i održavanja moguće je u cijelu priču uključiti odjel recepcije koji mogu isto tako prijavljivati kvarove od strane gostiju koristeći istu aplikaciju na tabletu.

Ako postoji kvar koje djelatnici odjela održavanje nisu u mogućnosti riješiti, isti se kvar može staviti ili na čekanje ili ako kvar zahtjeva intervenciju servisera istog staviti za vanjski servis.

Zašto baš Android aplikacija raspravljeno je i dokazano u prijašnjom poglavlju „Tehnički zahtjevi aplikacije i zastupljenost Android operacijskog sustava“.

3.3. SWOT analiza

Slika 13. SWOT analiza aplikacije „Održavanje“



Izvor: Autor

Iz prikazane SWOT analize zaključuje se opravdanost izrade same aplikacije. Mogućnost korištenja vlastite infrastrukture uz neizbježnu mobilnost koja čini razliku između konvencionalnih rješenja te mogućnost daljnje nadogradnje i transformacije aplikacije čine istu veoma poželjnom naspram drugih rješenja, iako je najveća prijetnja samoj aplikaciji prelazak na druge platforme tipa web.

3.4. Funkcionalnosti

Same funkcionalnosti možemo podijeliti ovisno o akterima ili sudionicima samog sustava. Na sljedećoj tablici prikazani su akteri zajedno sa funkcionalnostima:

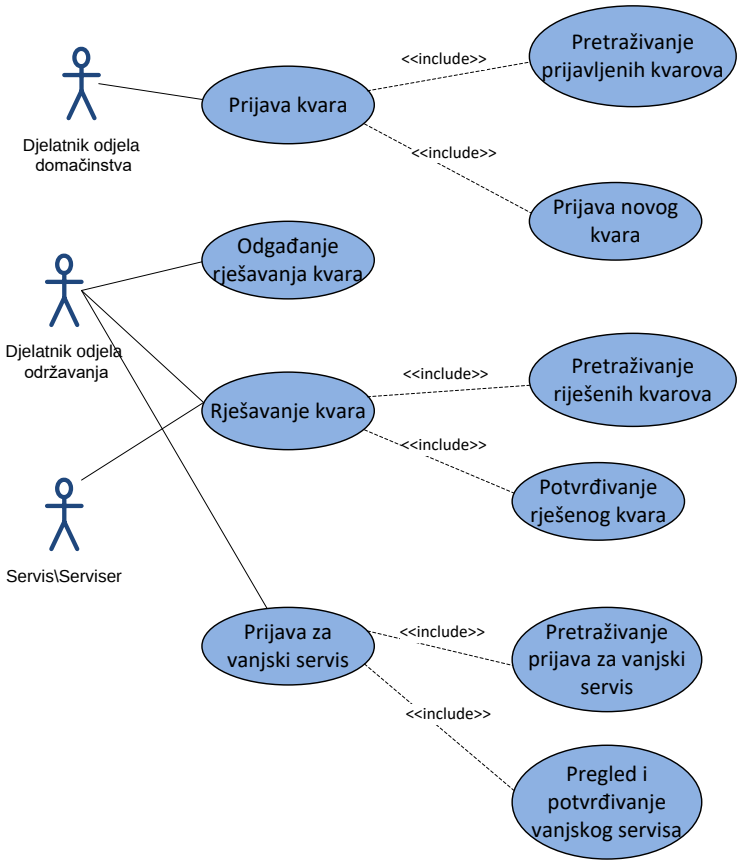
Tablica 1. Sudionici i funkcionalnosti

Sudionici	Funkcionalnost
Djelatnik domaćinstva	unos, pretraživanje, prijava kvarova
Djelatnik održavanja	pregled, potvrđivanje uspješno otklonjenih kvarova, prijava za vanjski servis, odgoda(na čekanje)

Izvor: Autor

Iako se unutar tablice spominju samo djelatnici odjela domaćinstva i održavanja, postoji još i jedan sudionik koji je direktno vezan za sustav i indirektno za aplikaciju a riječ je o servisu ili serviseru. Iako ne obavlja nikakve funkcije preko aplikacije, isti je važan u procesu rješavanja kvara kada odjel održavanja nije u mogućnosti sam otkloniti kvar. Date funkcionalnosti prikazane su dijagramom slučajeva korištenja

Slika 14. Dijagram slučajeva korištenja



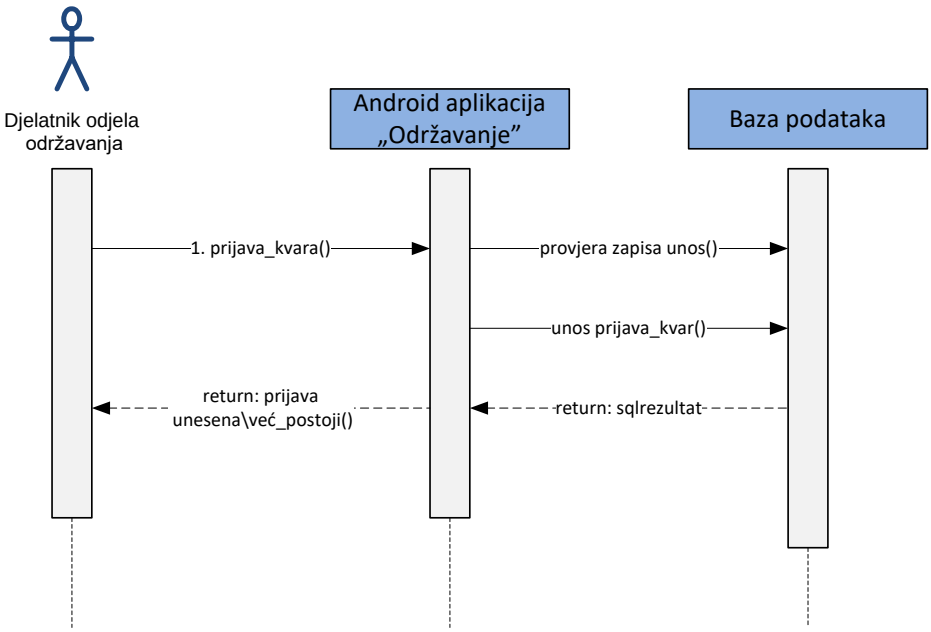
Izvor: Autor

3.4.1. Razrada funkcionalnosti

U sljedećom poglavlju razrađene su funkcionalnosti samog sustava a nakraju i aplikacije .Funkcionalnosti su grafički prikazane UML sekvencijalnim dijagramima.

3.4.1.1. Prijava kvara

Slika 15. Sekvencijski dijagram prijave kvara



Izvor: Autor

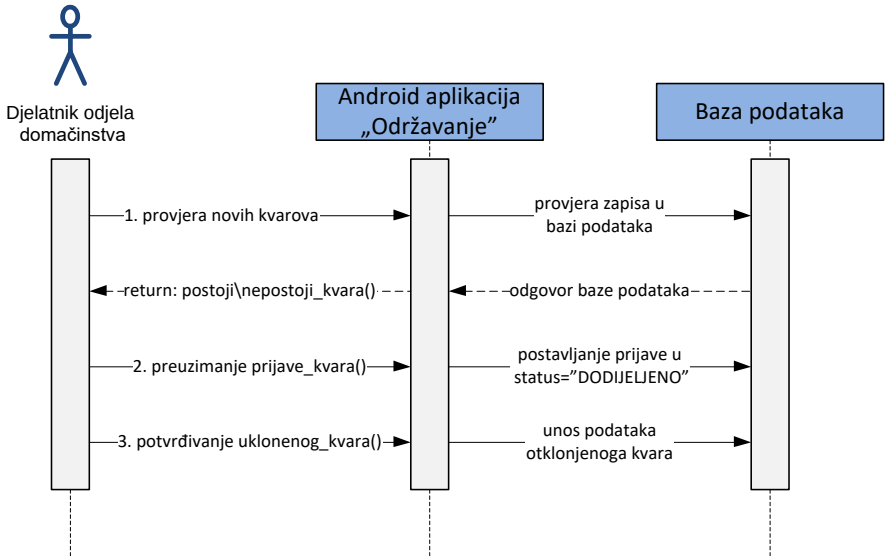
Prilikom prijave novoga kvara djelatnik domaćinstva preko aplikacije na svome Android uređaju prijavljuje novi kvar. Prilikom unosa, aplikacija provjerava u bazi podataka postoji li evidentirani zapis za dati kvar. Ako zapis postoji aplikacija obavještava djelatnika da navedena prijava već postoji u sustavu, u protivnom se zapis evidentira u bazi podataka.

U bazi podataka se evidentiraju sljedeći podaci prilikom prijave novoga kvara:

- Broj sobe ili apartmana
- Naziv kvara
- Opis kvara(po potrebi i želji)
- Korisnik koji je prijavio kvar
- Datum prijave
- Vrijeme prijave

3.4.1.2. Preuzimanje i potvrđivanje kvara

Slika 16. Sekvencijski dijagram preuzimanja i potvrđivanja kvara



Izvor: Autor

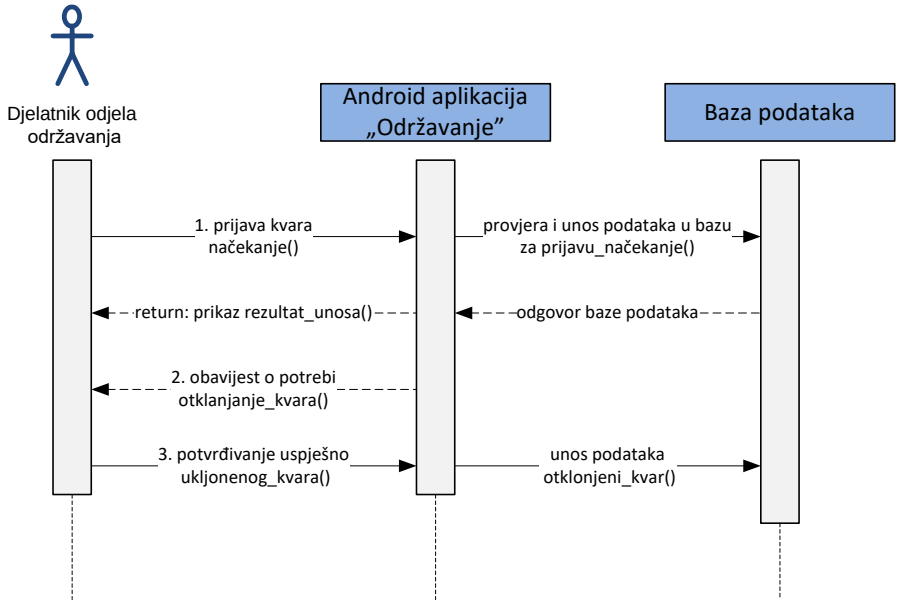
Djelatnik održavanja provjerava preko aplikacije postoji li koji novi prijavljeni kvar u bazi podataka. Ako postoji nova prijava istu preuzima. U sustavu(bazi podataka) se evidentira da je prijava preuzeta(„DODIJELJENO“). Nakon preuzimanja i uspješnog otklanjanja prijavljenog kvara djelatnik održavanja pristupa potvrđivanju uspješnog otklanjanja kvara.

Tom prilikom u bazi podataka evidentiraju sljedeći podaci:

- Korisničko ime osobe koja je riješila kvar
- Datum rješavanja kvara
- Vrijeme rješavanja kvara
- Čišćenje sobe

3.4.1.3. Odgoda rješavanja kvara

Slika 17. Sekvencijski dijagram odgode rješavanja kvara



Izvor: Autor

Kvar se stavlja na čekanje(odgoda) ako djelatnik održavanja nije u mogućnosti kvar otkloniti odmah, a za isti nije potreban vanjski servis. Prilikom postavljanja kvara na čekanje u sustav se evidentiraju sljedeći podaci:

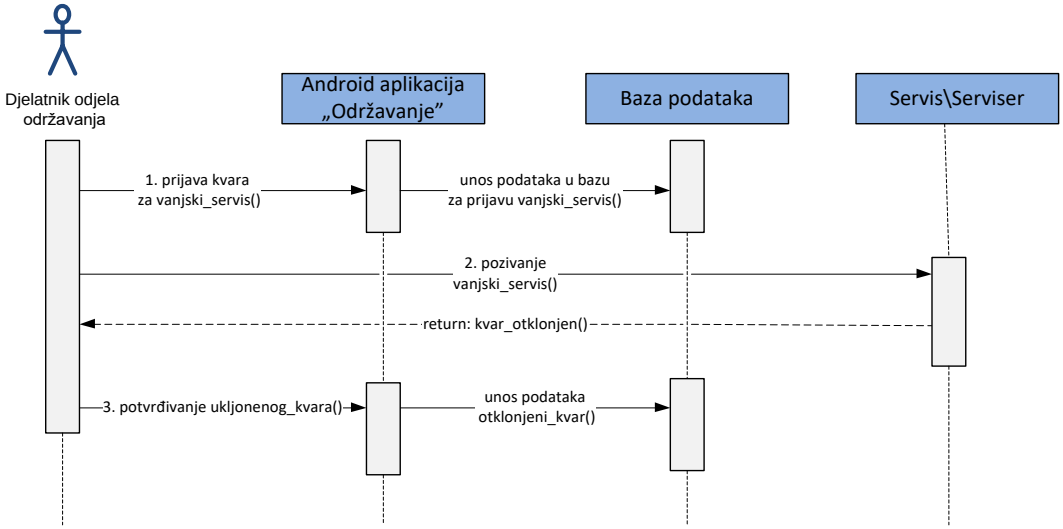
- Podaci prijave kvara
- Razlog zbog kojeg se kvar stavlja na čekanje
- Datum do\kada treba otkloniti kvar
- Sat do\kada treba otkloniti kvar

Sustav upozorava djelatnika da je došlo vrijeme za otklanjanje kvara. Nakon

otklanjanja kvara, djelatnik potvrđuje kvar.

3.4.1.4. Vanjski servis

Slika 18. Sekvencijski dijagram vanjski servis



Izvor: Autor

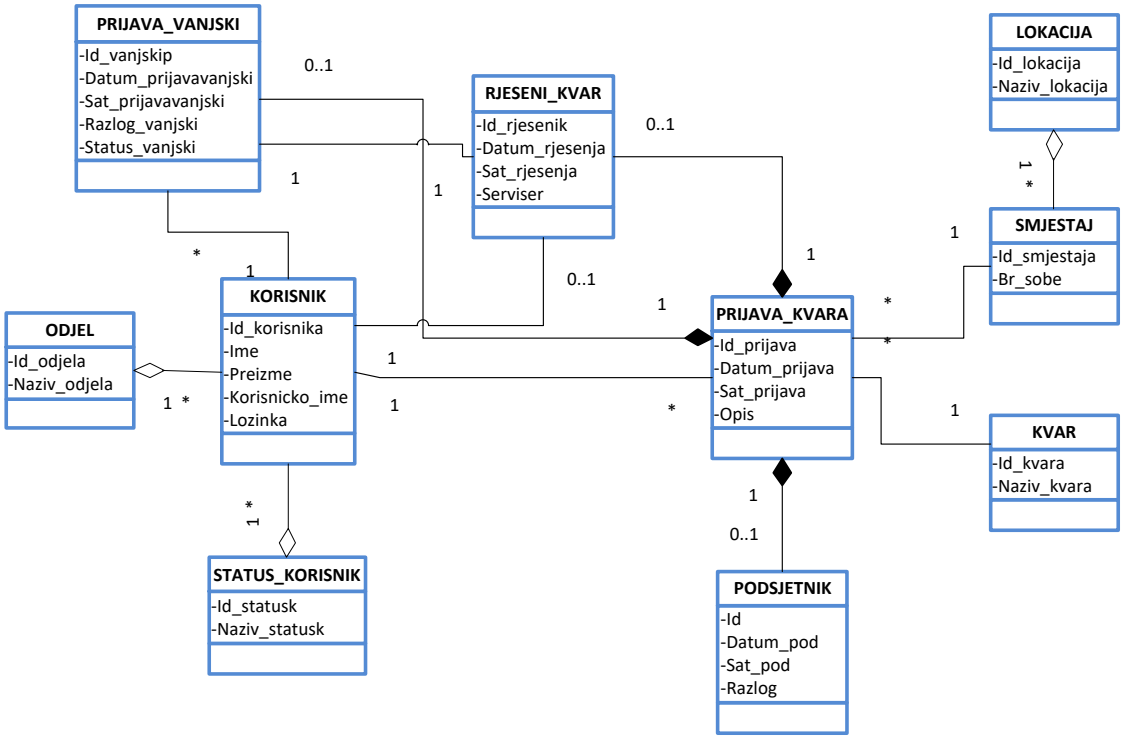
Ako djelatnik održavanja nije u mogućnosti otkloniti kvar, te je za isti potrebno pozivanje vanjskog servisa, kvar se prijavljuje za vanjski servis u mobilnu aplikaciju. Prilikom prijave za vanjski servis u bazi podataka aplikacije evidentiraju se sljedeći podaci:

- Podaci prijave kvara
- Razlog pozivanja vanjskog servisera
- Korisnik koji je prijavio vanjski servis
- Datum prijavljivanja vanjskog servisa
- Vrijeme prijavljivanja vanjskog servisa

Nakon pozivanja servisera i otklanjanja kvara, slijedi pregled kvara od strane djelatnika održavanja. Nakon pregleda otklonjenog kvara, djelatnik održavanja potvrđuje uspješno otklonjenog kvara od strane servisa/serviser. Tom se prilikom u sustav evidentira korisničko ime djelatnika odjela održavanja koji je izvršio pregled.

3.5. Klasni dijagram aplikacije

Slika 19. Klasni dijagram



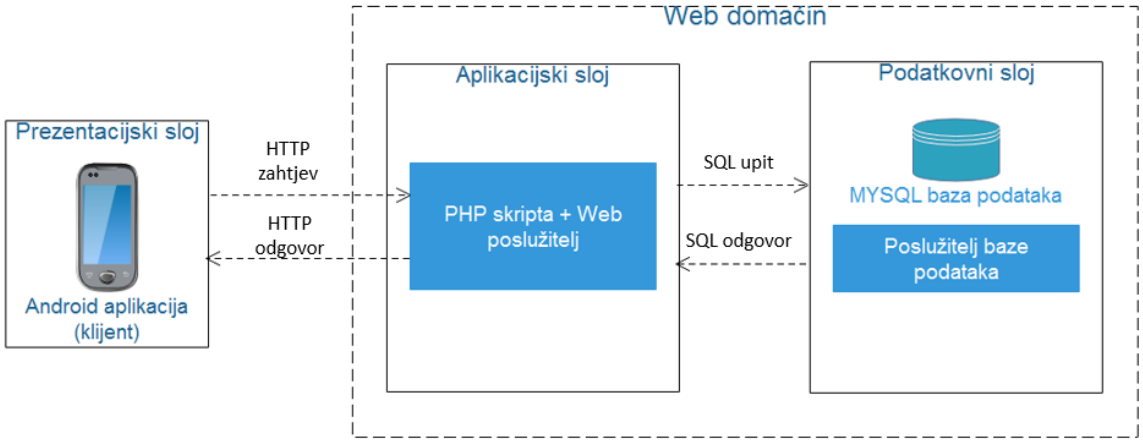
Izvor: Autor

3.7. Arhitektura sustava

Arhitektura sustava korištenja u ovome diplomskom radu je klijent-poslužitelj. Klijent poslužitelj je arhitektura kod koje s jedne strane imamo klijenta koji šalje zahtjev za izvođenjem određenih zadataka od poslužitelja koji ih izvršava. Model klijent poslužitelj najrasprostranjeniji i najkorišteniji je model današnjice i susrećemo ga kod dijeljenja različitih resursa od elektronske pošte, datoteka, pristupa internetu, dijeljenje prostora na disku, bazama podataka.

Konkretno u ovom radu korisnik preko aplikacije na svom Android uređaju šalje zahtjev web poslužitelju tj SQL upit bazi podataka preko PHP skripte. Baza na upit odgovara SQL odgovorom koji se prosljeđuje korisničkoj aplikaciji. Preko korisničke aplikacije prikazuju se dobiveni podaci samom korisniku.

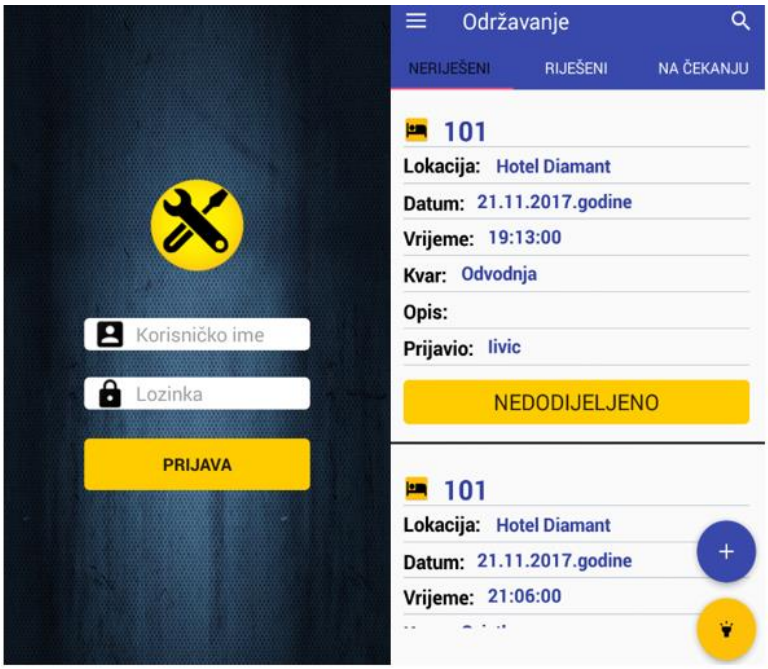
Slika 20. Model klijent-poslužitelj



Izvor: Autor

3.8. Grafički izgled aplikacije

Slika 21. Grafički izgled aplikacije



Izvor: Autor

Grafički izgled same aplikacije osmišljen je da bude intuitivan i što jednostavniji za koristiti. Do te spoznaje došlo se analiziranjem korisnika same aplikacije. Imajući

na umu da će istu koristiti korisnici različite dobi i profila sa više ili manje iskustva u korištenju mobilnih aplikacija a u konačnici samog Android operacijskoga sustava ista je morala biti što preglednija i jednostavnija za korištenje. Aplikacija zbog svog jednostavnog grafičkog sučelja ne gubi na vrijednosti iz razloga što je najbitnije čitko prikazivanje podataka(kvarova).

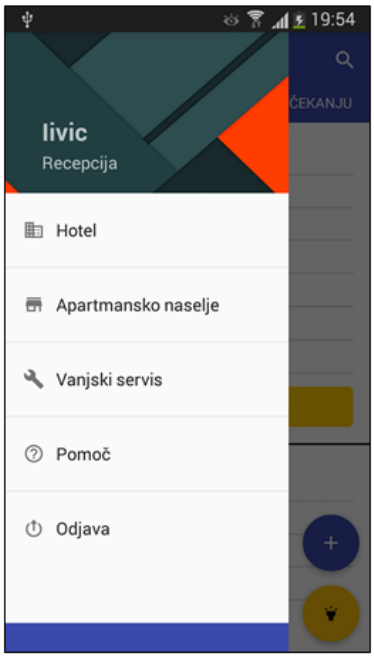
Iz istoga razloga autor ove aplikacije odlučio se na implementaciju pomoći unutar same aplikacija kao zasebne aktivnosti i još jedan vid pomoći korisniku u kojem će se na jedan ilustrativan način razumljiv širem spektru korisnika ako ne i svima prikazati pojedine funkcije same aplikacije.

3.8.1. Izbornici

3.8.1.1. Glavni izbornik aplikacije

Glavni izbornik aplikacije „Održavanje“ zamišljen je u obliku kliznog izbornika koji se aktivira povlačenjem dodirom s lijeva na desno ili klikom na gumb samog zaglavlja(Slika 22.).

Slika 22. Glavni izbornik aplikacije



Izvor: Autor

Na glavnom izborniku korisniku su dostupne sljedeće opcije(aktivnosti):

- Hotel
- Apartmansko naselje
- Vanjski servis
- Pomoć
- Odjava

Na samom zaglavlju glavnog izbornika prikazano je korisničko ime i odjel kojemu pripada korisnik.

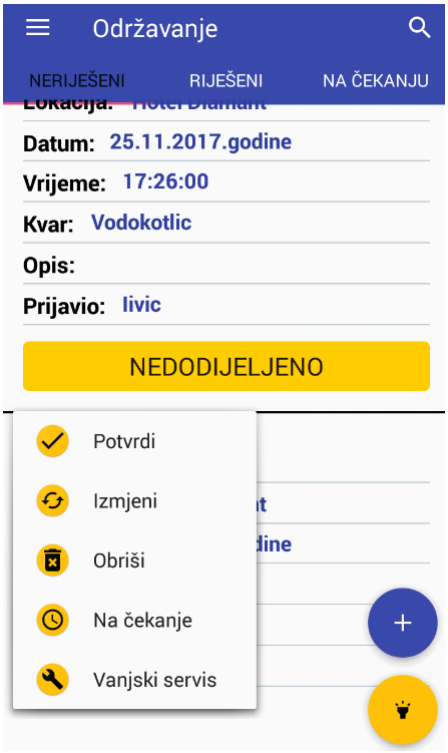
3.8.1.2. Skočni izbornik aplikacije

Osim glavnog izbornika aplikacija se sastoji od skočnog izbornika koji se aktivira odabirom kvara sa liste kvarova. Dodirom na izabrani kvar pojavljuje se skočni izbornik sa sljedećim opcijama:

- Potvrdi - za potvrđivanje kvara.
- Izmjeni - za izmjenu prijave kvara.
- Obriši - za brisanje kvara sa liste kvarova.
- Na čekanje - za odgodu rješavanja kvara i stavljanje istog na čekanje.
- Vanjski servis - za prijavu kvara za vanjski servis.

Isti izbornik odskače od svoje implementacije ovisno o kojoj se aktivnosti ili fragmentu nalazimo.

Slika 23. skočni izbornik



Izvor: Autor

3.8.1.3. Pregledi

Korisniku aplikacije dostupni su različiti pregledi kvarova. Prijave kvarova sortirane su u sljedeće kategorije:

- Svi kvarovi(hotel + apartmansko naselje).
- Samo hotel.
- Samo apartmani.
- Vanjski servis(pregled).

Slika 24. Izgled popisa neriješenih vanjskih servisa

Izvor: Autor

Osim navedenih pregleda prijava kvarova koji su ujedno i glavne aktivnosti aplikacije održavanje, svaka kategorija sastoji se od sljedećih podjela:

- Neriješeni kvarovi.
- Riješeni kvarovi.
- Kvarovi stavljeni na čekanje.

Slika 25. Primjer popisa riješenih kvarova

Izvor: Autor

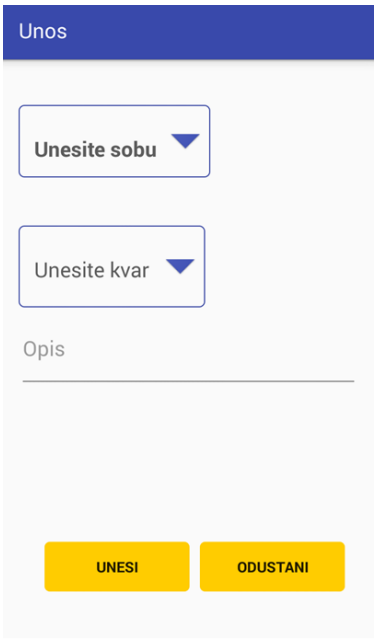
Slika 26. Primjer popisa kvarova koji su na čekanju

Izvor: Autor

3.8.3. Grafičko sučelja funkcija aplikacije

3.8.3.1. Unos prijave kvara

Slika 27. Dijalog unosa prijave kvara

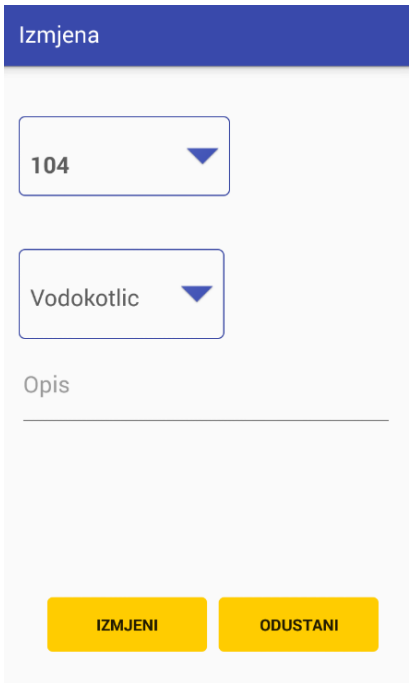


Izvor: Autor

Broj sobe i naziv kvara korisnik bira iz padajućeg izbornika. Ako kojim slučajem na listi kvarova nema kvara kojeg korisnik traži, korisnik odabire stavku „Razno“ i u opis opisuje koji je to kvar i gdje se nalazi.Za datum i vrijeme prijave uzima se sistemski datum i sistemsko vrijeme, dok se za korisnika uzima korisničko ime prijavljenog korisnika.

3.8.3.2. Izmjena prijave kvara

Slika 28. Dijalog izmjene prijave kvara

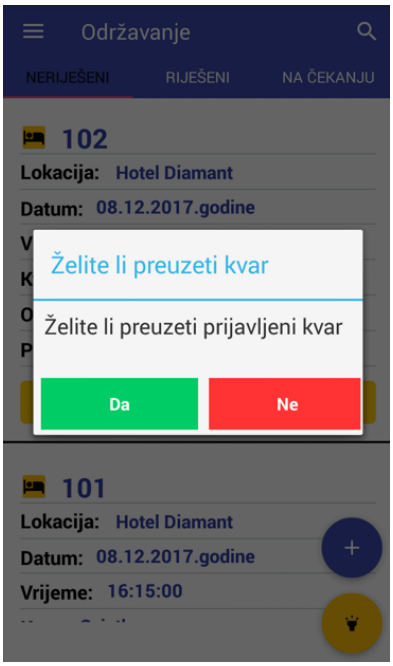


Izvor: Autor

Slika prikazuje izgled dijaloga za izmjenu kvara u kojoj korisnik može izmijeniti neke od stavka samoga kvara. Prihvati li korisnik(klikom na gumb “Izmjeni“) izmjenu, u sustav se bilježe novi podaci zajedno sa novim vremenom i datumom te korisničkim imenom korisnika koji je izvršio izmjenu prijave kvara.

3.8.3.2. Preuzimanje kvara

Slika 29. Dijalog preuzimanja kvara“

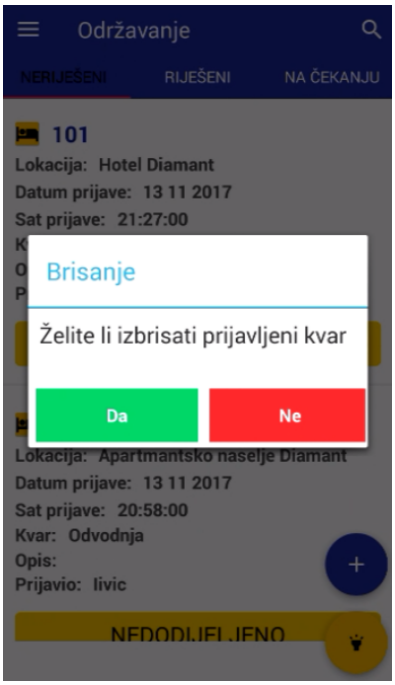


Izvor: Autor

Na slici je prikazan dijalog za preuzimanje kvara. Do samog dijaloga se dolazi klikom na gumb(“NEDODIJELJENO“). Jednom preuzeti kvar promijenit će svoj status unutar aplikacije(“DODIJELJENO“), a isti će biti vidljiv i drugim korisnicima

3.8.3.3. Brisanje kvara

Slika 30. Dijalog brisanja kvara

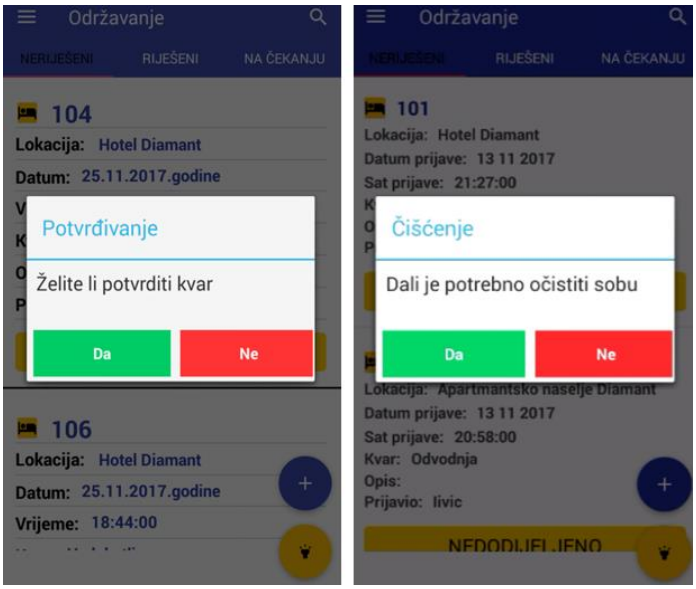


Izvor: Autor

Na slici je prikazan dijalog za brisanje kvara. Brisanje kvara svodi se na doslovno klasično brisanje kvara u kojemu korisnik odabire kvar koji želi izbrisati i izvršava akciju brisanje. Valja napomenuti da se isti kvar ne briše samo iz baze podataka već se programski isti briše iz zaslona korisnika koji ga je izbrisao iz razloga da bi promjena na listi prijavljenih kvarova bila odmah vidljiva a ne tek nakon ažuriranja liste prijavljenih kvarova.

3.8.3.4. Potvrđivanje kvarova

Slika 31. Dijalog potvrđivanja kvara

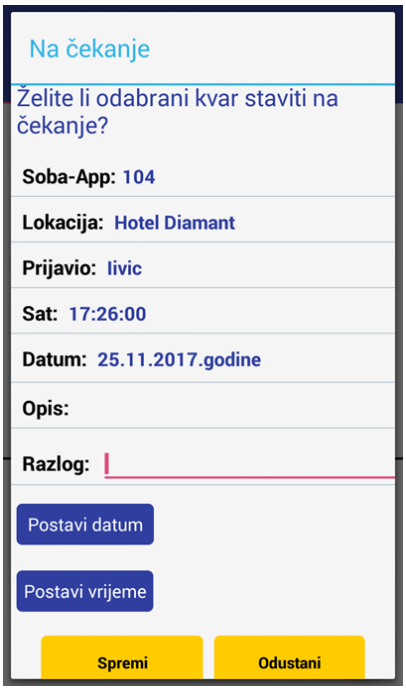


Izvor: Autor

Na slici su prikazana dva dijaloga potvrđivanja kvara. Prvi dijalog(“Potvrđivanje“) je standardni dijalog u kojemu se traži od korisnika da potvrdi kvar ili ne. Automatski nakon potvrde prvog dijaloga korisniku se prikazuje drugi dijalog sa zahtjevom o potrebi čišćenja sobe ili apartmana. Sam dijalog „Čišćenje“ je vrlo zahvalan iz razloga što daje informaciju o tome jeli soba prljava nakon otklanjanja kvara ili nije. Ako je prljava isti će status sobarica vidjeti pod kategorijom riješeni kvarovi. Na taj način izbjegava se mogućnost ulaska gosta bilo novog ili starog u prljavu sobu.

3.8.3.4. Odgoda rješavanja kvara(na čekanje)

Slika 32. Dijalog prijave kvara na čekanje

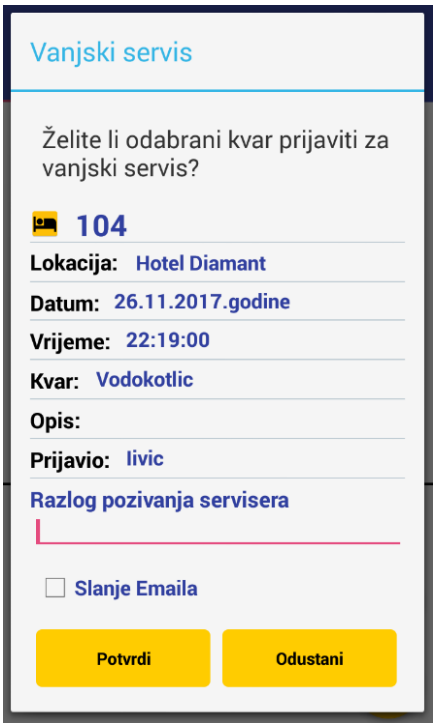


Izvor: Autor

Na slici je prikazan dijalog na čekanje aplikacije “Održavanje“. Djelatnik prilikom stavljanje kvara na čekanje treba unijeti razlog zbog kojeg se kvar stavlja na čekanje sa datumom i satom obavijesti, ostali podaci preuzeti su iz prijave kvara. Mogućnost stavljanja kvara na čekanje je vrlo korisna iz razloga što kad određeni kvar nismo u mogućnosti odmah riješiti bilo zbog gosta koji boravi u toj sobi ili apartmanu ili iz nekog drugog razloga, za isti je moguće postaviti podsjetnik za kasnije.

3.8.3.5. Vanjski servis

Slika 33. Izgled prijave kvara za vanjski servis

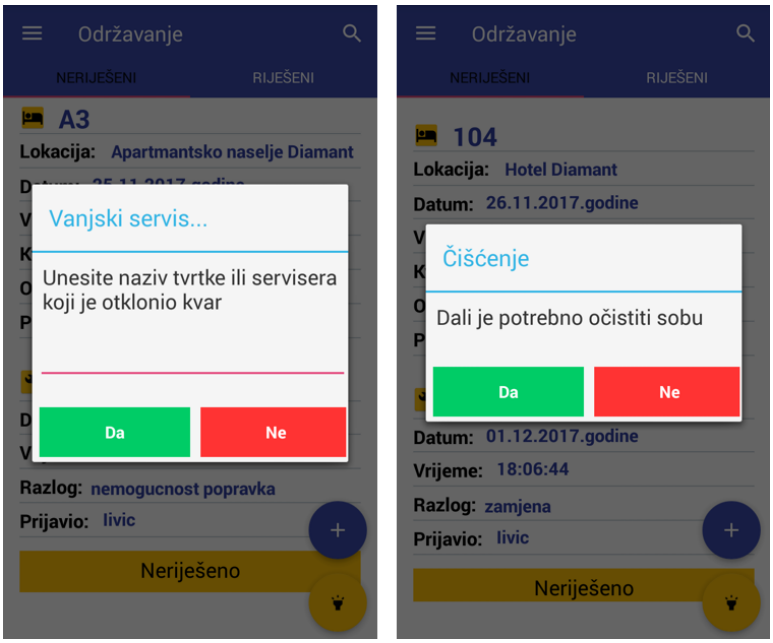


Izvor: Autor

Na slici je prikazan dijalog prijave kvara vanjskom servisu unutar same aplikacije. Svi podaci unutar samog dijaloga preuzeti su od same prijave kvara. Jedini podatak kojeg djelatnik održavanja mora unijeti je „Razlog pozivanja servisera“ koji je obavezan. Prilikom prijave vanjskom servisu, korisniku aplikacije omogućeno je slanje prijave na E-mail odgovornoj osobi ili više njima(voditelj odjela održavanja, direktor hotela…) radi informiranja o pozivanju vanjskog servisa. Nerijetko se dešavaju kvarovi koji nisu baš jeftini te iziskuju veća financijska sredstva za svoje otklanjanje. Na taj način svi su sudionici u procesu odlučivanja vanjskog servisa pravovremeno(odmah) obavješteni o potrebi za pozivanje vanjskog servisera.

3.8.3.6. Potvrđivanje vanjskog servisa

Slika 34. Dijalog potvrđivanja vanjskog servisa



Izvor: Autor

Slika prikazuje izgled dijaloga za potvrđivanje vanjskog servisa i dijaloga za čišćenje. Kod potvrđivanja vanjskog servisa od korisnika se traži unos naziva tvrtke ili servisera koji je kvar popravio. Pritiskom na gumb “Da“ korisnik aplikacije potvrđuje uspješno otklonjeni kvar od strane vanjskog servisa\servisera. Nakon dijaloga potvrđivanja vanjskog servisa, korisniku aplikacije se pojavljuje dijalog „Čišćenja“ ako sobu slučajno treba očistiti.

3.8.3.6 Dodatak: Pomoć

Slika 35. Aktivnost „Pomoć“

Izvor: Autor

Pomoć je implementirana kao zasebna funkcija(aktivnost). Iako ne obavlja neke zadatke kao druge aktivnosti itekako je bitna. Bitna je u pogledu pomoći korisniku jer aktivnost “Pomoć“ prikazuje, na jedan veoma slikovit način mogućnosti same aplikacije, te korisnik može u bilo kojem trenutku kad je nesiguran kako napraviti određeni zadatak unutar aplikacije, pogledati unutar aktivnosti Pomoć(HelpActivity).

3.9. BAZA PODATAKA APLIKACIJE

U ovom diplomskom radu korištena je kao što je već i spomenuto MySQL baza podataka. Sami upiti izvršavaju se pomoću PHP skriptnog jezika. Primjer strukture baze podataka aplikacije “Održavanje“ prikazano je na sljedećoj slici

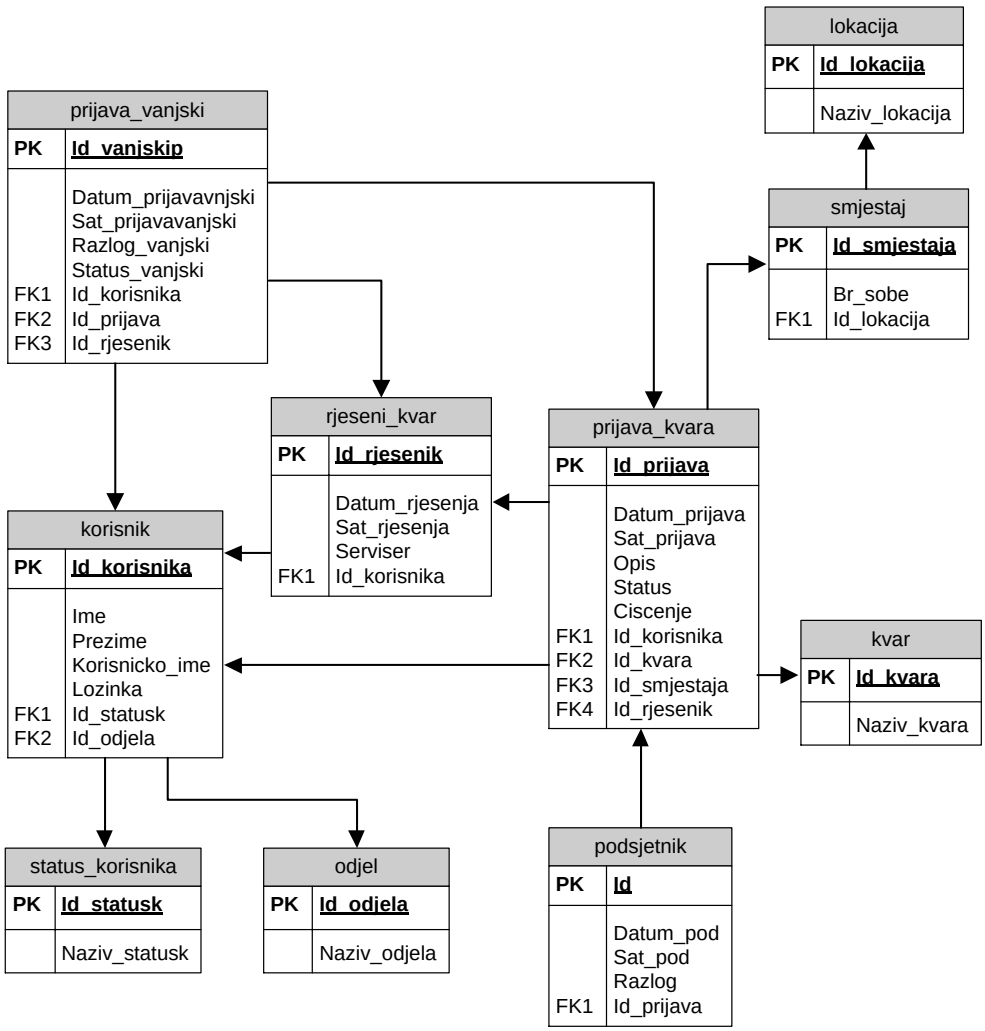
Slika 36. Primjer tablice prijava_kvara u phpMyAdmin-u

	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
☐	1	**Id_prijava**	int(100)			No	*None*		AUTO_INCREMENT
☐	2	**Datum_prijava**	date			No	*None*		
☐	3	**Sat_prijava**	time			No	*None*		
☐	4	**Id_smjestaj**	int(255)			No	*None*		
☐	5	**Id_korisnika**	int(20)			No	*None*		
☐	6	**Id_kvar**	int(100)			No	*None*		
☐	7	**Opis**	text	utf8_general_ci		Yes	*NULL*		
☐	8	**Status**	text	utf8_general_ci		Yes	*NULL*		
☐	9	**Ciscenje**	varchar(2)	utf8_general_ci		Yes	*NULL*		
☐	10	**Id_rjesenik**	int(100)			No	*None*		

Izvor: Autor

3.9.1. Relacijski model baze podataka

Slika 37. Relacijski model baze podataka aplikacije “Održavanje“



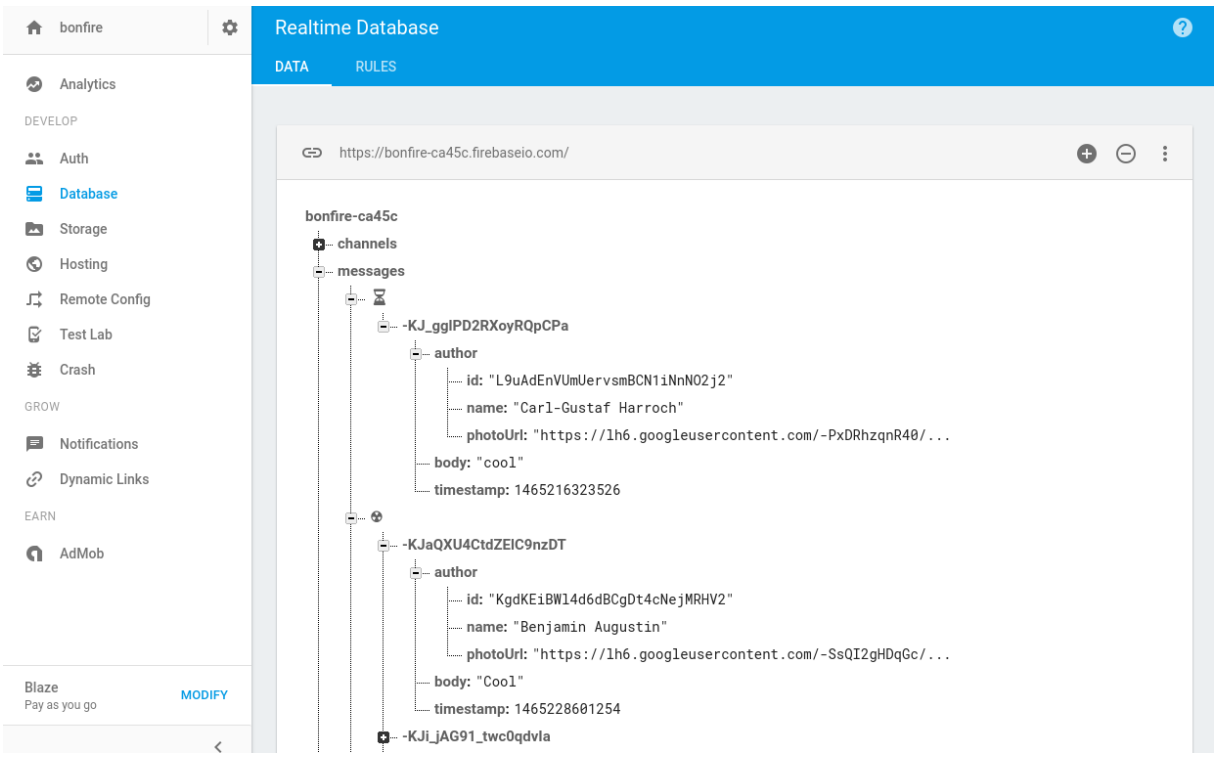
Izvor: Autor

Na prikazanoj slici dat je relacijski dijagram baze podataka aplikacije „Održavanje“ Ono što je važno naglasiti je vrijednost Id_korisnika unutar tablice rjeseni_kvar, prilikom kreacije svakog novog zapisa ima vrijednost null. Ta vrijednost se dodjeljuje automatski iz razloga što Id_korisnika predstavlja korisnika koji je riješio kvar. Pošto kvar nije otklonjen vrijednost ovog stranog ključa postavlja se automatski na vrijednost null. Nakon što je kvar otklonjen u samom procesu potvrđivanja od strane djelatnika održavanja, mijenja se zapis Id_korisnika u vrijednost koja odgovara djelatniku koji je riješio kvar. Ovdje valja napomenuti da Id_korisnika nema jedini vrijednost null, već tu vrijednost imaju i ostali atributi tablice rijeseni_kvar osim

Id_rjesenik koji predstavlja primarni ključ te ujedno strani ključ tablice prijava_kvara.

4. FIREBASE BAZA PODATAKA

Slika 38. Izgled Firebase baze podataka



Izvor: novoda

Firebase Database je oblik baze podataka koja je smještena u oblaku tj. sve podatke sprema u JSON formatu na samom oblaku u realnom vremenu. Firebase je NoSQL tip baze podataka što znači nema tablica, nema stupaca kao kod SQL baza podataka. Podaci su spremljeni unutar JSON objekata te zajedno čine jedno veliko stablo JSON-a. Svakim dodavanjem novog podataka unutar samog stabla stvara se novi čvor sa svojim ključem. Dva ista ključa unutar stabla nepostojne čak i ako teoretski dvoje korisnika pokuša u isto vrijeme unijeti isti podatak svaki će podatak dobiti zaseban ključ, jer se ključ prilikom generiranja jednim dijelom sastoji od nasumično generiranih brojeva.

Što se tiče integriteta podataka kod Firebasea nema očuvanja integriteta

podataka iz razloga što se bilo koji novi podatak zajedno sa ključem(zajedno tvore skup ključ->vrijednost) zapisuju kao čvor unutar baze podataka.

Jedan od prednosti samoga Firebase-a je da se podaci spremaju u memoriju Android uređaja u SQLite bazu podataka ugrađenog u sam uređaj, na taj način podaci su dostupni korisniku i kada nema konekcije sa bazom podataka, te se isti ažuriraju čim se uspostavi konekcija sa bazom.

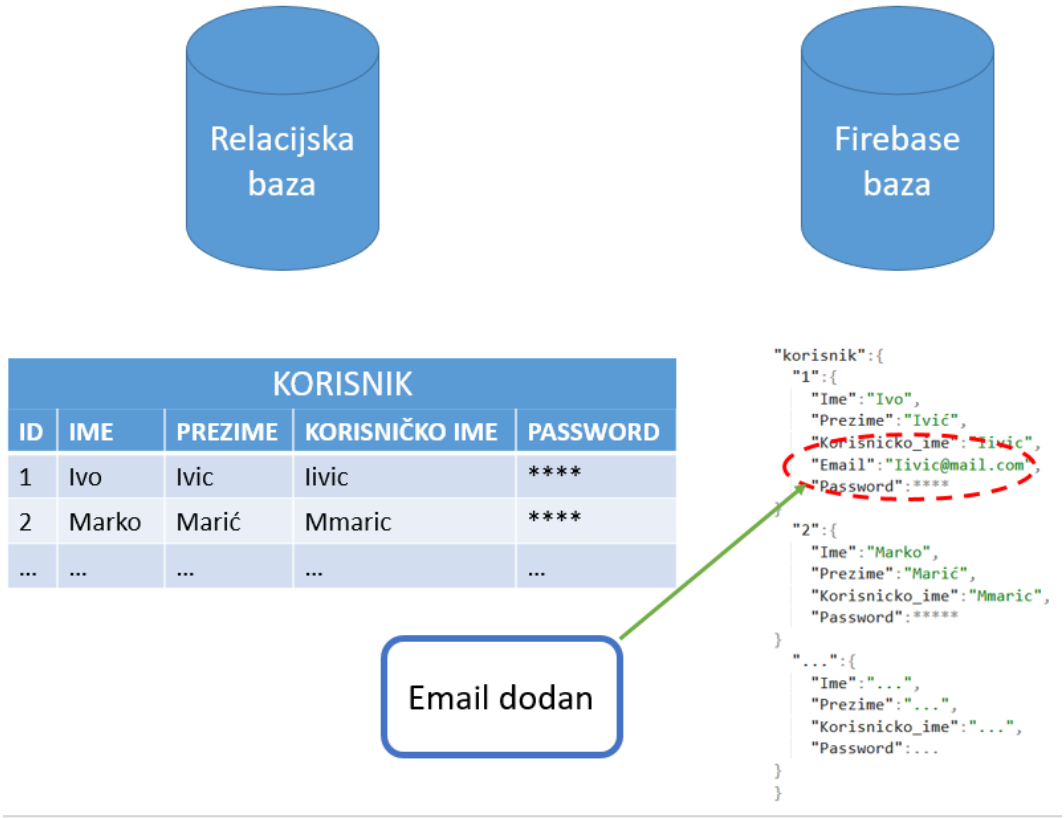
4.1. Usporedba Firebase baze podataka sa MySQL bazom podataka

U ovom poglavlju usporedit ćemo Firebase bazu podataka sa MySQL bazom podataka. Usporedba će se temeljiti na načinu skladištenja podataka, osnovnih operacija nad podacima, autentikaciji i drugih razlika i mogućnosti između obih baza podataka.

4.1.1. Razlike u modelu podataka

Kao što je već rečeno Firebase baza podataka(NoSQL) pohranjuje podatke u obliku čvorova unutar JSON stabla za razliku od MySQL kod kojeg se podaci pohranjuju unutar tablica. Firebase baza podataka ne zahtjeva striktan definirani model podataka. To za programera same baze podataka znači dodavanje različitih tipova podataka unutar JSON stabla za svaki zapis. Primjer je prikazan na sljedećoj slici koja prikazuje umetanje email korisnika za samo jednog korisnika unutar Firebase baze podataka. Za relacijsku bazu to će predstavljati problem jer moramo kreirati novi stupac „email“ za sve korisnike, dok kod Firebase baze podataka to neće biti potrebno, već ćemo dodati novi JSON objekt „email“ samo za tog korisnika. Iz navedenog primjera proizlazi veća fleksibilnost za programere prilikom izrade same baze podataka.

Slika 39. Relacijska i Firebase baza podataka umetanje e-maila



Izvor: Autor

Druga najveća novina jeste nema spajanja tablica(entiteta) prilikom kreiranja upita te se unutar Firebase baze podataka dokumenti ugrađuju jedni u druge tj. JSON čvorovi se formiraju ovisno o podacima koje želimo dohvatit. Firebase prilikom izvršavanja upita dohvaća čitavo stablo što u većini slučajeva osim podataka koje želimo dohvatit i koji su nam potrebni dohvaća nažalost i one podatke koji nisu iz razloga što se dohvaća cijelo JSON stablo što predstavlja oblik „smeća“ tj. nepotrebnih podataka. Osim dohvaćanje nepotrebnih podataka kod Firebase baze podataka pojavljuje nam se problem dohvaćanja čvorova prilikom kreiranja upita. Kao što je poznato kada u SQL bazi podataka želimo dohvatiti podatke SQL upitom između dvije tablice čiji je upit definiran određenim uvjetom koristimo JOIN-ove. Firebase baza podataka nema ugrađenu mogućnost korištenja JOIN-ova te je dati problem riješen na drugačiji način.

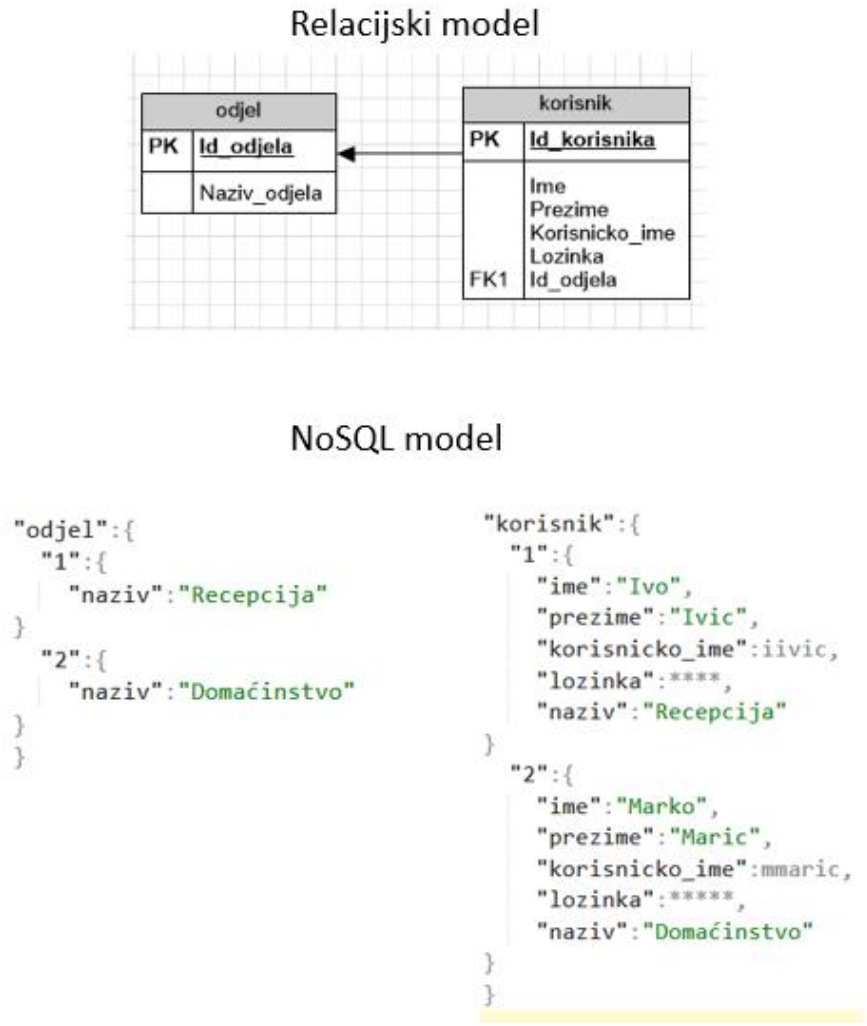
Kao rješenje navedenoga problema dohvaćanja nepotrebnih podataka i nedostatak implementacije JOIN-ova kao što imamo slučaj u SQL bazama podataka

uvodi se pojam denormalizacija. Denormalizacija je proces uvođenja redundancije podataka(ponavljanja) unutar same baze podataka sa ciljem efikasnijeg upravljanja podacima unutar same baze podataka. Razvijatelju aplikacija(programeru) sam pojam denormalizacije izaziva neshvaćanje i protivljenje iz razloga što se većina aplikacija temelji na korišteniju Relacijskih baza podataka za pohranu podataka samih aplikacija te je prelazak ili konverzija već razvijenih aplikacija na NoSQL sustav pohrane podataka kao što je Firebase veoma težak i predstavlja nemoguću misiju samom programeru.

Denormalizacija je dakako veoma važna iz razloga što nam omogućava bolje dohvaćanje samih podataka. Normalno, kada govorimo o denormalizaciji moramo se osvrnuti na sustav provjere referencijalnog integriteta(strani ključ). Kada i kako koristiti denormalizaciju nije jednoznačno određeno već ovisi prvenstveno o samoj aplikaciji tj dohvaćanju podataka(„upitima“) u samu bazu.

Na sljedećoj slici dat je primjer relacijskog modela i NoSQL modela u obliku JSON stabla relacije 1:M između korisnika i odjela. Svaki korisnik pripada jednom odjelu dok jedan odjel može imati više korisnika. Istu usporedbu možemo predočiti unutar samih zapisa koji se nalaze u Firebase bazi podataka(NoSQL) i MySQL(relacijska baza) podataka.

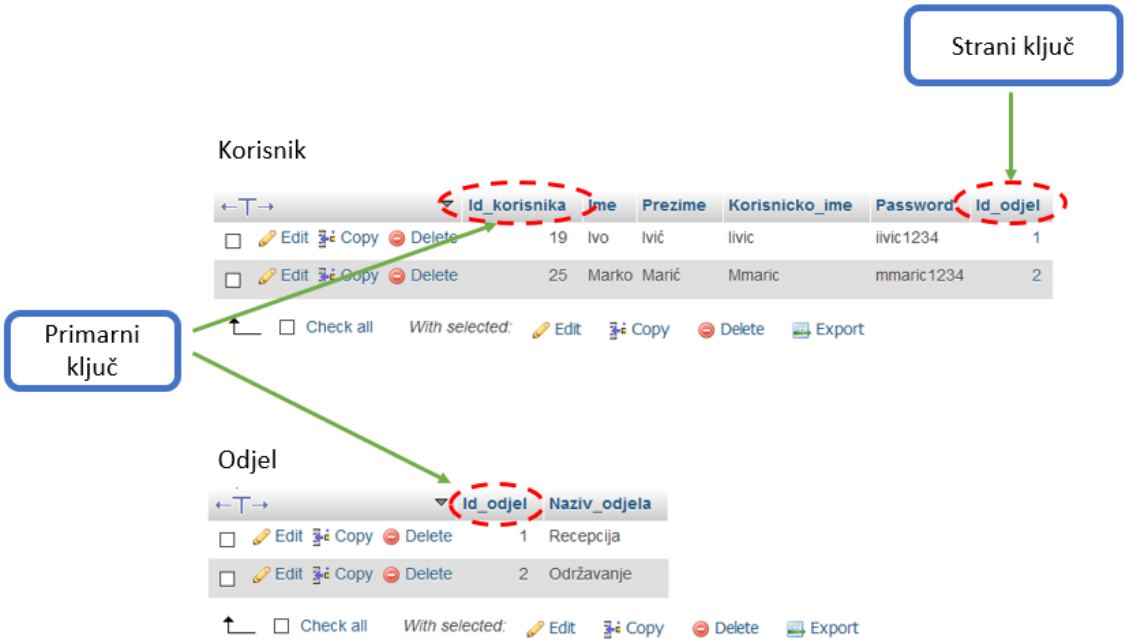
Slika 40. Primjer Relacijskog i Firebase modela



Izvor: Autor

Sljedeća slika prikazuje strukturu entiteta korisnik odjel unutar MySQL baze podataka. Kao što možemo primijetiti tablice se sastoje od primarnoga ključa te se tablica korisnik sastoji od stranog ključa Id_odjel kao referenca na tablicu odjel i očuvanje integriteta podataka.

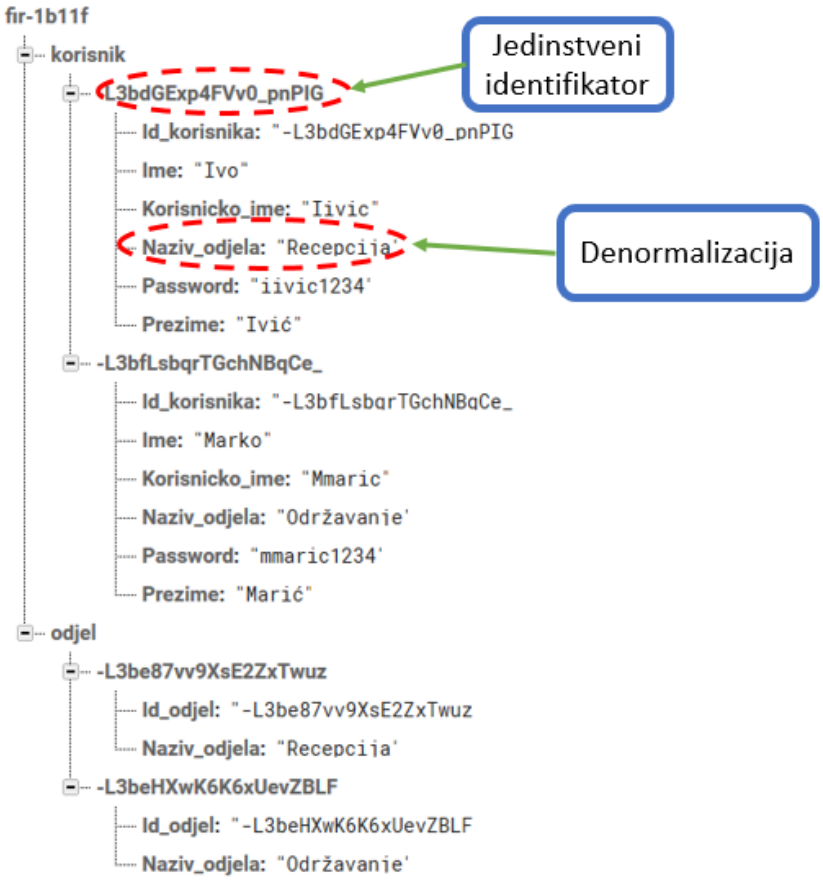
Slika 41. Odjel-korisnik zapis unutar MySQL baze podataka



Izvor: Autor

Sljedeća slika prikazuje izgled tablica korisnik i odjel unutar Firebase baze podataka. Umjesto primarnoga ključa Firebase baza podataka koristi jedinstveni identifikator koji se generira automatski unutar same baze podataka za svakog korisnika. Osim jedinstvenog identifikatora korisnika primjećujemo da nam se Naziv_odjela pojavljuje unutar JSON stabla korisnik i unutar JSON stabla odjel što predstavlja tipičan primjer denormalizacije korištene zbog nepostojanja referencijalnog integriteta i stranoga ključa te kasnijeg prikazivanja svih korisnika zajedno sa odjelima kojima pripadaju.

Slika 42. Odjel-korisnik zapis unutar Firebase baze podataka



Izvor: Autor

U sljedećem poglavlju objasniti ćemo najznačajnije operacije koje se mogu izvršiti unutar obiju baza podataka te pojasniti razlike između pojedinih operacija.

4.1.2.Operacije unutar baza podataka

U ovom poglavlju opisat ćemo glavne operacije nad podacima unutar Firebase i MySQL baze podataka te prokomentirati razliku među njima. Osnovne operacije nad podacima unutar baze podataka su:

- Unos podataka.
- Izmjena podataka.
- Brisanje podataka.
- Pregled podataka.

4.1.2.1. Unos podataka

Sljedeći primjer prikazuje programski isječak napisan u JAVA programskom jeziku za unos novog korisnika unutar Firebase baze podataka. Zbog potrebe kasnijeg prikazivanja svih korisnika zajedno sa odjelima kojima pripadaju i nepostojanje mehanizma stranog ključa i referencijalnog integriteta bili smo primorani osim podataka o korisniku zapisati i naziv odjela unutar JSON objekta korisnik. S druge strane SQL baza podataka osim što ne podržava redundanciju podataka i ima implementirane mehanizme referencijalnog integriteta i stranog ključa, dupliranje podataka nije dozvoljeno što možemo vidjeti na istom primjeru.

Odmah primjećujemo da je SQL naredba za unos podataka puno jednostavnija i na koncu smislenija, te se o samom referencijalnom integritetu brine sama SQL baza podataka dok kod FIrebase baze podataka sve ovisi o razvijatelju aplikacije tj samim podacima koji će se kasnije prikazivati te se i njihovo samo modeliranje a i postojanje unutar Firebase baze podataka ovisiti o samom prikazu podataka unutar aplikacije. To je veliki nedostatak same Firebase baze podataka jer je sve unutar iste stavljeno na pleća rezvijatelju aplikacije te ne postoje nikakvi mehanizmi kontrole unutar same baze podataka. S druge strane kao što je već rečeno NoSQL baza podataka kao što je Firebase baza podataka daje mogućnost unosa novog JSON objekta za određeni čvor ili samo jedan čvor(Primjer: unosa Email-a za jednog korisnika) što u SQL bazi podataka kao što je MySQL nije moguće

Slika 43. Programski isječak unos novog korisnika u Firebase i MySQL bazu podataka

Firebase

```
/**
 * Metoda za spremanje podataka Korisnik
 * u Firebase bazu podataka
**/
private void unosKorisnik() {


    //Dohvaćanje reference ili kreiranje korisnik-a unutar Firebase baze podataka
    databaseKorisnik = FirebaseDatabase.getInstance().getReference("korisnik");

    String ime = editTextIme.getText().toString().trim();
    String prezime = editTextPrezime.getText().toString().trim();
    String korisnicko_ime = editTextKorisnickoime.getText().toString().trim();
    String password = editTextLozinka.getText().toString().trim();

    //provjera popunjenosti polja za unos
    if (!TextUtils.isEmpty(ime) || !TextUtils.isEmpty(prezime) ||
            !TextUtils.isEmpty(korisnicko_ime) || !TextUtils.isEmpty(password) ||
            !TextUtils.isEmpty(spinnervrijednost)) {

        //kreiranje jedinstvenog identifikatora metodom push().getKey()
        // koji ujedno predstavlja Primarni ključ Korisnika
        String id = databaseKorisnik.push().getKey();

        //kreiranje objekta Korisnik
        Korisnik korisnik  = new Korisnik(id, ime, prezime,korisnicko_ime,
                password,spinnervrijednost); //spinnervrijednost predstavlja naziv_odjela

        //Spremanje vrijednosti korisnika u Firebase bazu podataka
        databaseKorisnik.child(id).setValue(korisnik);

        Toast.makeText(this, "Korisnik dodan", Toast.LENGTH_LONG).show();
    } else {

        Toast.makeText(this, "Molim Vas unesite sva polja za unos korisnika", Toast.LENGTH_LONG).show();
    }
}
```

MySQL

```
1 INSERT INTO `korisnik` (`Ime`, `Prezime`, `Korisnicko_ime`, `Password`, `Id_odjel`)
2 VALUES ( 'Ana', 'Anić', 'Aanic', 'anic1234', '1');
```

Izvor: Autor

4.1.2.2. Brisanje podataka

Sljedeći primjer prikazuje brisanje podataka unutar baze podataka i to korisnika i odjela. Želimo izbrisati pojedini zapis o odjelu moramo osim podataka unutar JSON

objekta odjel izbrisati onaj unutar JSON objekta korisnik ako taj odjel više ne postoji. što je suprotna operacija od unosa novog korisnika u Firebase bazu podataka. Što se tiče brisanja podataka unutar SQL baze podataka tablice korisnik vidljivo je iz iste slike da je operacija izvršena jednom kratkom SQL naredbom. Sa druge strane brisanje podataka iz tablice odjel u veći slučajeva rezultirati će porukom o greški i nemogućnošću brisanja iz razloga što se primarni ključ koristi kao strani ključ u nekoj drugoj tablici(više poglavlje Referencijalni integritet relacijskih baza podataka).

Slika 44. Programski isječak za brisanje korisnika i odjela u Firebase bazi podataka i korisnika u MySQL bazi podataka

Firebase

Brisanje korisnik-a

```
private boolean brisanjeKorisnik(String id) {

    //dohvaćanje reference određenog korisnik-a
    DatabaseReference brKorisnik = FirebaseDatabase.getInstance().getReference("korisnik").child(id);

    //brisanje vrijednosti korisnik unutar baze podataka
    brKorisnik.removeValue();
    Toast.makeText(getApplicationContext(), "Korisnik Izbrisan", Toast.LENGTH_LONG).show();

    return true;
}
```

Brisanje odjel-a

```
private boolean brisanjeOdjel(String id) {

    //dohvaćanje reference određenog odjela-a
    DatabaseReference brOdjel = FirebaseDatabase.getInstance().getReference("odjel").child(id);
    //brisanje vrijednosti odjel unutar baze podataka
    brOdjel.removeValue();

    //dohvaćanje reference određenog korisnik-a
    DatabaseReference brKorisnik = FirebaseDatabase.getInstance().getReference("korisnik").child(id);
    //brisanje vrijednosti korisnik unutar baze podataka
    brKorisnik.removeValue();

    Toast.makeText(getApplicationContext(), "Odjel Izbrisan", Toast.LENGTH_LONG).show();

    return true;
}
```

MySQL

```
1 DELETE FROM korisnik WHERE Id_korisnika=3;
```

Izvor: Autor

4.1.2.3. Izmjena podataka

Operacija izmjene podataka unutar Firebase baze podataka ne razlikuje se više od operacije brisanja tj. ovisi nalazi li se podatak koji se želi izmijeniti unutar još kojega čvora. Zbog nepostojanja provjere referencijalnog integriteta i stranog ključa javljaju nam se poteškoće prilikom izmjene podataka zbog dupliranja istih. Naime izmjenom vrijednosti određenog čvora na jednom mjestu neće se automatski izmijeniti na svim mjestima u JSON dokumentu u kojima se izmijenjena vrijednost nalazi. Kao rješenje ovog problema automatski se nameće izmjena vrijednosti na svakoj poziciji na kojoj se izmijenjena vrijednost nalazi unutar Firebase baze podataka, što zahtjeva potrebna znanja o lokacijama izmijenjene vrijednosti unutar same baze podataka. To potkrepljuje sljedeća slika koja prikazuje izmjenu naziva odjela jednog odjela unutar Firebase baze podataka na svim lokacijama na kojima se taj odjel nalazi zbog korištenja denormalizacije i prikaza podataka. Analogija tomu u SQL bazi podataka izmjena naziva odjela unutar tablice odjel vrlo je jednostavna, i jedino što se prilikom izmjene mora znati je id zapisa tj. odjela čiji se naziv želi promijeniti.

Slika 45. Programski isječak izmjene postojećeg korisnika u Firebase i MySQL bazi podataka

Firebase

```
private boolean izmjeniKorisnik(String id,  String Ime, String Prezime, String  Korisnicko_ime,
                                String Password, String Naziv_odjela) {

    //dohvačanje reference određenog korisnik-a(kojeg se želi izmjeniti)
    DatabaseReference izmjena = FirebaseDatabase.getInstance().getReference("korisnik").child(id);

    //kreiranje novog objekta Korisnik
    Korisnik korisnik  = new Korisnik (id, Ime, Prezime, Korisnicko_ime, Password, Naziv_odjela);

    izmjena.setValue(korisnik); //unos novih podataka za odabranog korisnika u bazu podataka

    Toast.makeText(getApplicationContext(), "Korisnik Izmjenjen", Toast.LENGTH_LONG).show();
    return true;
}
```

MySQL

```
UPDATE korisnik SET Ime="Anamarija" WHERE Id_korisnika=25
```

Izvor: Autor

4.1.2.4. Pregled podataka

Kod pregleda ili prikaza podataka po određenom kriteriju kod Firebase baze podataka koristi se objekt po nazivu DataSnapshot. DataSnapshot je objekt koji sadrži podatke sa određene putanje unutar baze podataka. DataSnapshot možemo promatrati kao SQL upit koji sadrži grupirane podatke iz relacijskih tablica po nekom kriteriju za prikazivanje. Firebase baza podataka koristi tkz. „slušače“(*eng. listener*) pomoću koji osluškuje promjene podataka unutar same baze podataka zajedno sa metodom onDataChange() koja se pokreće prilikom detektirane izmjene. Sljedeća slika prikazuje primjer dohvaćanja svih korisnika unutar Firebase baze podataka zajedno sa odjelima kojima pripadaju te SQL upit unutar relacijske baze podataka koja prikazuje korisnike zajedno sa pripadajućim odjelima. Ono što možemo odmah primijetiti da će prikazani podaci kod Firebase baze podataka biti prikazani u realnom vremenu dok kod MySQL baze podataka za prikazivanje podataka u realnom vremenu mogu se koristiti i okidači(*eng. triggers*) koji će se aktivirati prilikom promjene podataka unutar baze podataka. Ovaj diplomski rad ne koristi okidače već je taj dio riješen primjenom drugih metoda obavijesti o promjeni podataka unutar same baze podataka.

Slika 46. Programski isječak prikaza korisnika u Firebase i MySQL bazi podataka

Firebase

```
databaseKorisnik.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {

        //brisanje prijašnjih podataka
        korisnici.clear();

        //iteracija kroz sve čvorove objekta korisnik
        for (DataSnapshot postSnapshot : dataSnapshot.getChildren()) {

            try {
                Korisnik korisnik = postSnapshot.getValue(Korisnik.class);
                //dodavanje korisnika u listu korisnika
                korisnici.add(korisnik);

            }catch(DatabaseException e){

            }
        }
        //kreiranje i prosljeđivanje podataka adapteru za prikazivanje korisnika
        KorisnikList artistAdapter = new KorisnikList(MainActivity.this, korisnici);
        listaKorisnik.setAdapter(artistAdapter);

    }
    @Override
    public void onCancelled(DatabaseError databaseError) {

    }
});
```

MySQL

```
SELECT Id_korisnika, Ime, Prezime, Korisnicko_ime,
Password, odjel.Naziv_odjela FROM korisnik,odjel
WHERE korisnik.Id_odjel=odjel.Id_odjel
```

Id_korisnika	Ime	Prezime	Korisnicko_ime	Password	Naziv_odjela
19	Ivo	Ivić	Iivic	iivic1234	Recepcija
25	Marko	Marić	Mmaric	mmaric1234	Održavanje

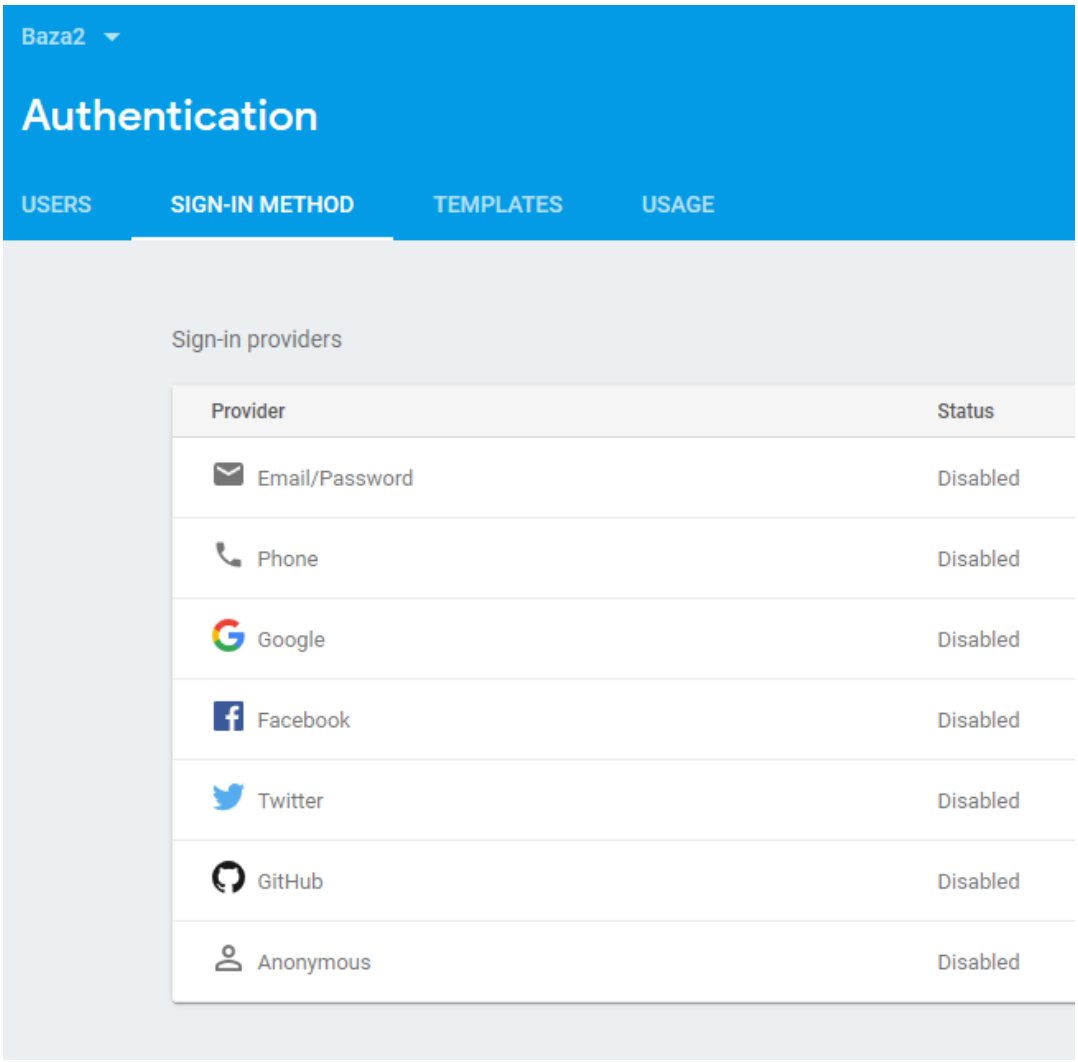
Izvor: Autor

4.1.3. Autentikacija korisnika i pravila pristupa

Firebase baza podataka nudi pregršt mehanizma i načina autentikacije korisnika baze podataka te definira načine manipulacije podataka pohranjenih u samoj bazi podataka. Za svakog korisnika može se dodijeliti što sve može raditi s bazom podataka, dok je sama autentikacija korisnika moguća putem korisničkog imena i lozinke, email računa te autentikacijskog tokena. Firebase baza podataka osim

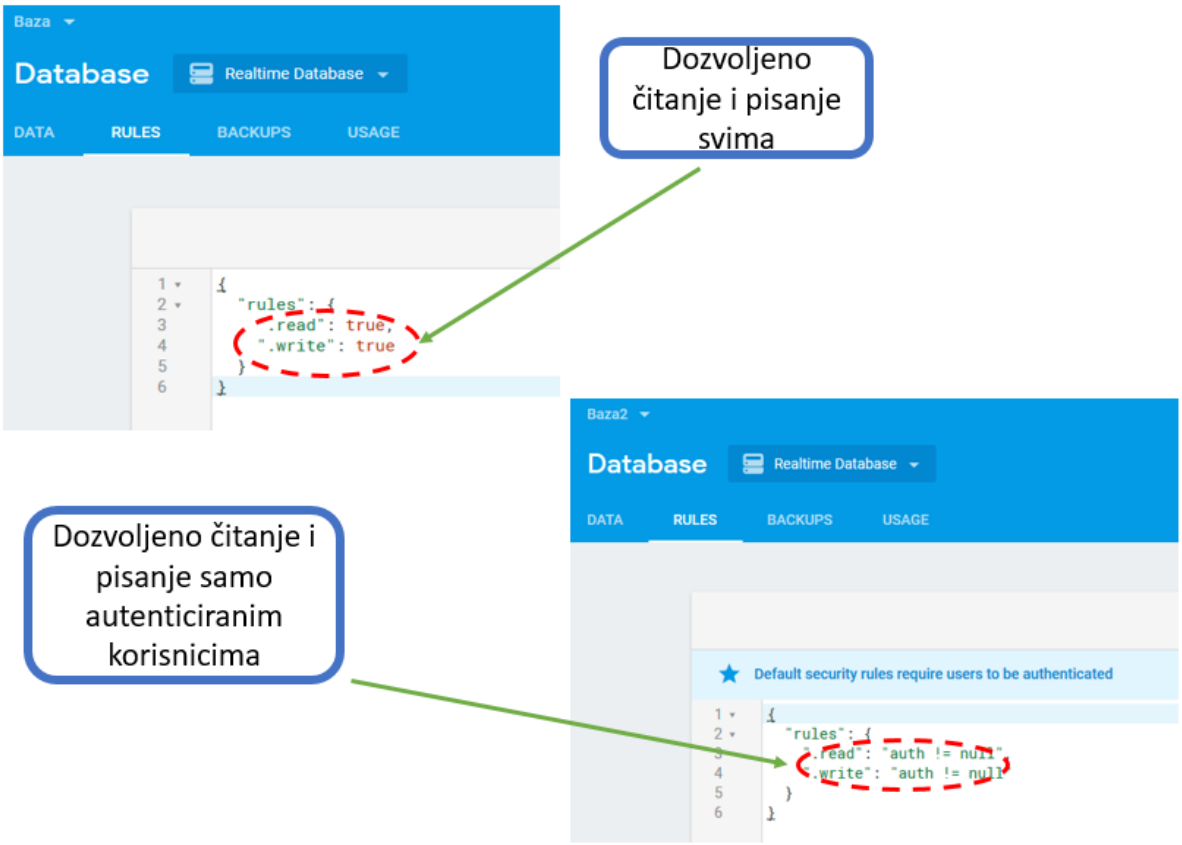
navedene autentikacije korisnika podržava autentikaciju preko socijalnih mreža kao što su Facebook i Twitter. Standardno svim korisnicima koji su uspješno prošli proces autentikacije dozvoljeni je pristup, čitanje i pisanje podatka. Slika(Slika 48.) prikazuje pravila pristupa podacima unutar Firebase baze podataka i to na primjeru dozvole unosa i čitanja samo autenticiranih korisnika i dozvole za čitanje i pisanje svim korisnicima. Pravila kreira sam administrator baze podataka po želji.

Slika 47. Načini autentikacije Firebase baze podataka



Izvor: Autor

Slika 48. Primjer dozvoljenog čitanja i pisanja za sve i samo za autenticirane korisnike



Izvor: Autor

Tablica 2. Pravila pristupa Firebase baze podataka

Pravila pristupa Firebase baze podataka	
.read	Dozvola za čitanje podataka.
.write	Dozvola za unos podataka.
.validate	Definira format koji uneseni podatak treba imati, tip podataka.
.indexOn	Definira indeks na čvoru dijete(korisno kod sortiranja silazno->uzlazno ili obrnuto)

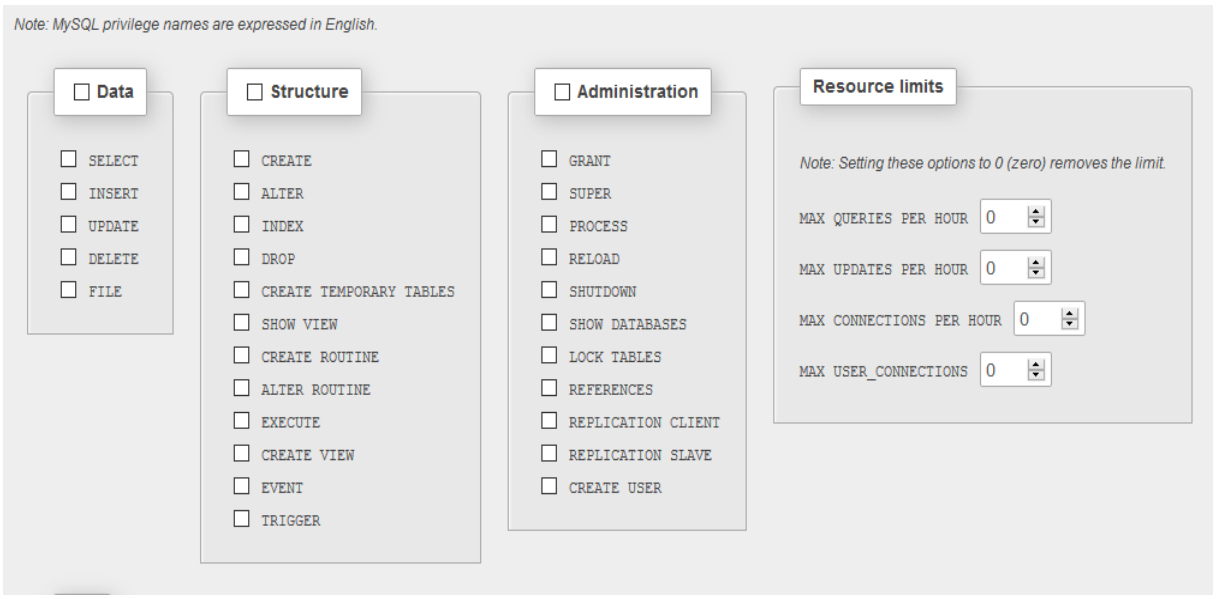
Izvor: Autor

Slika 49. Pravila pristupa za čvor odjel

Izvor: Autor

Na slici su prikazana pravila pristupa za čvor „odjel“. Za čvor „odjel“ dozvoljeno je čitanje i pisanje od svih korisnika baze podataka neovisno jesu li prošli autetikaciju no upisani novi zapis mora biti tipa String. Osim navedenog objekta moguće je definirati pravila i za druge objekte unutar same baze podataka.

Slika 50. Upravljačka ploča programa phpMyAdmin za dodjeljivanje prava korisnicima



Izvor: Autor

Na slici je prikazana upravljačka ploča programa phpMyAdmin za dodjeljivanje prava korisnicima MySQl baze podataka. Kao što možemo primijetiti prava su

razvrstana u četiri kategorije:

- Podaci(*eng. Data*) - osnovne operacije s podacima kao što je unos, brisanje, izmjena podataka
- Struktura(*eng. Structure*) - operacije vezane uz samo strukturu tj. kreiranje i brisanje tablica, izmjena tablica, kreiranje okidača(*eng. Trigger*)
- Administracija(*eng. Administration*) - zaključavanje tablica, kreiranje novih korisnika, dodjeljivanje privilegija(prava) korisnicima baze podataka..
- Ograničenja resursa(*eng. Resource limits*)- maksimalan broj upita i izmjena u bazu podataka po satu, broj konekcija po satu…

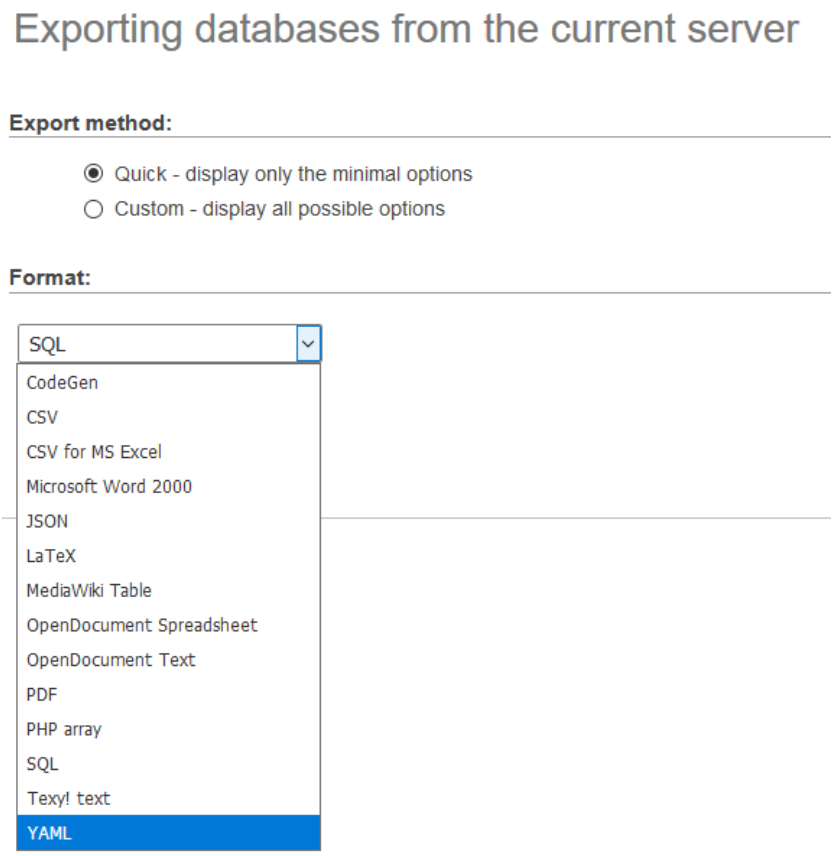
Za autentikaciju se koristi korisničko ime i lozinka korisnika koji prvotno mora biti kreiran od strane administratora baze podataka zajedno sa dodijeljenim pravima unutar same baze podataka

Iz navedenog možemo zaključiti postojanje pregršt mogućnosti implementirane u sami program phpMyAdmin koje omogućuju administratoru fino uštimavanje privilegija i prava za svakoga korisnika za razliku od Firebase baze podataka, dok sama autentikacija nudi mogućnost autenticiranja korisnika jedino preko korisničkog imena i lozinke.

4.1.4. Pohrana podataka i izvoz

Firebase baza podataka pohranjuje podatke u obliku JSON stabla dok su podaci unutar MySQL baze podataka pohranjeni unutar tablica što je već rečeno tokom ovoga rada. Navedeni se podaci mogu izvesti i to kod Firebase baze podataka u obliku JSON datoteke dok kod MySQL baze podataka, podaci se mogu izvesti u nekoliko različitih formata kao što prikazuje sljedeća slika.

Slika 51.Grafičko sučelje izvoza podataka unutar programa phpMyAdmin



Izvor: Autor

Kod Firebase baze podataka svi su podaci spremljeni u oblaku, te bazu podataka kao istu korisnik ne posjeduje već je ona unajmljena i nalazi se na Googleovim poslužiteljima. Kada govorimo o MySQL bazi podataka ista može biti unajmljena, a može se koristiti i na vlastitim poslužiteljima. Stoga MySQL nudi malu prednost iz razloga što istu možemo koristiti na vlastitom poslužitelju ili je imati unajmljenu što je jedan od glavnih ciljeva ovoga diplomskoga rada jer postoji želja korisnika da on sam posjeduje MySQL bazu podataka i ima je instaliranu na vlastitom poslužitelju, a ne iznajmljenu bazu od drugog pružatelja usluga. Stoga misleći na neku budućnost u kojoj će postojati i web verzija aplikacije kojoj će osim Android uređaja moći pristupati i drugi uređaji preko web preglednika, isplativije je napravit SQL bazu podataka i skladištiti podatke u istu od Firebase-a. U slučaju da radimo aplikaciju tipa instant messaging(Viber) koja će postojati u samo mobilnoj verziji i potrebno je automatsko ažuriranje promjena podataka u bazi koristit ćemo Firebase bazu podatak

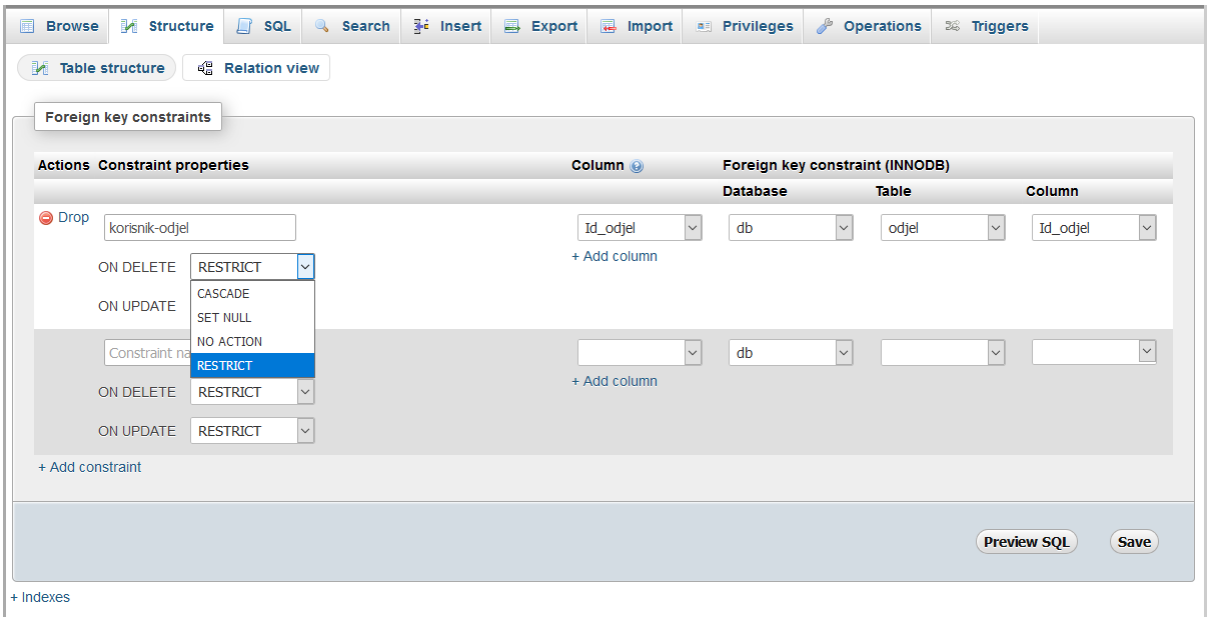
4.1.5. Referencijalni integritet relacijskih baza podataka(MySQL)

Već smo puno puta spomenuli tokom ovog diplomskog rada referencijalni integritet te je konačno došlo i vrijeme da objasnimo što je to. Referencijalni integritet je skup pravila i ograničenja čija je zadaća briga o konzistentnosti podataka pohranjenih u bazu podataka. Pod pojmom konzistentnosti podataka mislimo na dosljednost samih podataka zapisanih unutar tablica u bazi podataka. Konzistentnost se osigurava stranim i primarnim ključem unutar samih tablica. Za primjer ćemo uzeti odnos korisnika i odjela gdje je primarni ključ odjela ujedno i strani ključ korisnika. Strani ključ tablice korisnik zajedno sa primarnim ključem tablice odjela čini vezu u kojoj su vrijednosti obiju ključa isti. Na taj način osigurava se konzistentnost samih zapisa unutar baze podataka. Referencijalni integritet definira ujedno pravila brisanja i izmjene(ažuriranja) podataka. Sljedeća slika prikazuje izgled korisničkoga sučelja programa phpMyAdmin koji služi za kreiranje referencijalnog integriteta među tablicama tj. kreiranju veza između tablica i ponašanje(ograničenja) prilikom izmjene i brisanja podataka iz navedenih tablica.

Neovisno radi li se o brisanju ili izmjeni podataka, dostupna su sljedeća ograničenja:

- **CASCADE** - (odabirom opcije CASCADE automatski se briše ili ažurira vrijednost u obje tablice(odjel i korisnik), ovisno o izabranoj akciji brisanja ili ažuriranja podataka.
- **SET NULL** - (postavlja vrijednost NULL u tablicama u kojima se nalazi strani ključ).
- **RESTRICT** - (akcija brisanja i izmjene podataka biva odbačena u slučaju da postoji strani ključ u nekoj od tablica(referencijalni integritet)).
- **NO ACTION** - (ekvivalent RESTRICT načinu(akcije brisanja i izmjene bivaju odbačene od strane baze podataka u slučaju da strani ključ postoji i kreiran je referencijalni integritet u nekoj od tablica)).

Slika 52. Grafičko sučelje za kreiranje referencijalnog integriteta unutar programa phpMyAdmin



Izvor: Autor

Firebase baza podataka nema implementirane ovakve mehanizme očuvanja integriteta podataka, već je taj zadatak svaljen na pleća samo razvijatelja aplikacije uz korištenje denormalizacije koja je ranije objašnjena u ovome radu.

4.1.6. Zaključak usporedbe između MySQL(SQL) i Firebase(NoSQL) baze podataka

Usporedba između MySQl i Firebase baze podataka što je ujedno usporedba između SQL i NoSQL baza podataka imala za cilj prikazivanje svih onih dobrih strana obiju baza podataka. Iako svaka stvar ima i svoje loše strane, o lošim stranama možemo govorit u smislu jednog konstruktivnog zapažanja trenutačnih nedostataka i mana koje bi se trebale u budućnosti ako je to moguće ispraviti. Kao glavnu lošu stranu Firebase baze podataka je nepostojanje stranog ključa i mehanizma referencijalnog integriteta. U nekim situacijama mogućnost dodavanja novog zapisa za samo određeni čvor predstavlja prednost što je potkrepljeno primjerom dodavanja E-maila za samo jednog korisnika. Funkcija brisanja je veoma loše izvedena i predstavlja prijetnju u slučaju da brisanje određenoga čvora nije izvršeno i u svim ostalim čvorovima gdje se brisana vrijednost nalazi. Isto vrijedi i u slučaju izmjene podataka koji se nalaze u više

čvorova. Što se tiče autentikacije, Firebase baza podataka podržava više načina autenticiranja korisnika(Facebook, Twitter, Email...) dok su sama pravila pristupa puno bolje odrađena u MySQL bazi podataka stavljajući na raspolaganje administratoru pregršt pravila i opcija za definiranje pristupa bazi podataka.

5. ODABIR TEHNOLOGIJA I MEHANIZMI DOHVAĆANJA PODATAKA UNUTAR APLIKACIJE

U ovom poglavlju raspravit ćemo o mehanizmima dohvata samih podataka sa poslužitelja i poteškoćama prilikom odabira tehnologija i izrade same aplikacije.

5.1. Odabir tehnologija

Prije početka pisanja ovoga rada, u doba kada još i sama aplikacija „Održavanje“ nije postojala, u fazi odabira tehnologija, autor ovoga rada našao se u nedoumici između korištenja standardnih alata ugrađenih u JAVA programski jezik koji je ujedno i službeni jezik za pisanje Android aplikacija ili koristiti PHP za dohvaćanje podataka iz MySQL baze podataka i izvršavanje SQL upita. JAVA programski jezik omogućuje nam korištenje JDBC(Java Database Connectivity), sučelje za programiranje aplikacija koji sadrži skup metoda i alata za definiranje spajanja klijenta(uređaja, korisnika) na MySQL bazu podataka. JDBC je dio Java Development Kit(JDK) kojeg možemo definirati kao skup svega što je potrebno za programiranje(izradu) Java aplikacija(biblioteke klasa, prevoditelj...).

Autoru ovoga rada a ujedno i aplikacije korištenje JDBC-a kao posrednika za dohvat podataka između Android aplikacije i MySQL baze podataka, činio se kao pun pogodak, iz razloga što pojednostavljuje programiranje(izradu) same aplikacije. Korištenjem JDBC-a automatski eliminiramo korištenje PHP-a te nam je osigurana puna kompatibilnost između baze podataka i aplikacije. Na sljedećoj slici prikazan je programski isječak same aplikacije.

Slika 53. Programski isječak spajanja na MySQL bazu podataka korištenjem JDBC-a zajedno sa SQL upitom(Android)

```
final String DB_URL = "jdbc:mysql://192.168.1.251/db";
final String USERNAME = "dbkorisnik";
final String PASSWORD = "mypass";
Class.forName("com.mysql.jdbc.Driver");
Connection conn = DriverManager.getConnection(DB_URL, USERNAME, PASSWORD);
if (conn != null) {
    // SQL upit
    String query = "SELECT DATE_FORMAT(pk.Datum_prijava, '%d.%m.%Y.godine') " +
            "AS Datum_prijava, pk.Sat_prijava," +
            " pk.Opis,pk.Status, k1.Korisnicko_ime AS k1_ime," +
            " kvar.Naziv_kvar, smjestaj.Br_sobe, lokacija.Naziv_lokacija" +
            " FROM prijava_kvara pk" +
            " LEFT OUTER JOIN korisnik k1 ON pk.Id_korisnika=k1.Id_korisnika" +
            " LEFT OUTER JOIN kvar ON pk.Id_kvar=kvar.Id_kvar" +
            " LEFT OUTER JOIN smjestaj ON pk.Id_smjestaj=smjestaj.Id_smjestaj" +
            " LEFT OUTER JOIN lokacija ON smjestaj.Id_lokacija=lokacija.Id_lokacija" +
            " WHERE pk.Status='Nedodijeljeno' OR pk.Status='Dodijeljeno'";
    Statement stmt = conn.createStatement();
    ResultSet res = stmt.executeQuery(query);
```

Izvor: Autor

Slika 54. Rezultat dobiven SQL upitom(Android)

```
System.out: ----------------------------------
System.out: Smještaj: 102
System.out: Lokacija: Hotel Diamant
System.out: Datum prijave: 08.12.2017.godine
System.out: Sat prijave: 16:16:00
System.out: Kvar: Svjetla
System.out: Opis: Svjetlo hodnik
System.out: Status kvara: Nedodijeljeno
System.out: Prijavio: Iivic
System.out: ----------------------------------
System.out: ----------------------------------
System.out: Smještaj: A5
System.out: Lokacija: Apartmantsko naselje Diamant
System.out: Datum prijave: 08.12.2017.godine
System.out: Sat prijave: 16:38:00
System.out: Kvar: TV
System.out: Opis: Nema slike
System.out: Status kvara: Nedodijeljeno
System.out: Prijavio: Iivic
System.out: ----------------------------------
System.out: ----------------------------------
System.out: Smještaj: 108
System.out: Lokacija: Hotel Diamant
System.out: Datum prijave: 08.12.2017.godine
System.out: Sat prijave: 16:42:00
System.out: Kvar: Wifi
System.out: Opis:
System.out: Status kvara: Nedodijeljeno
System.out: Prijavio: Iivic
System.out: ----------------------------------
```

Izvor: Autor

Iz navedenih slika odmah uočavamo SQL upit koji je ukomponiran unutar same aplikacije kao dio programskog rješenja za izvršavanje upita u bazu. Druga stavka je definiranje glavnih elementa za pristupanje bazi podataka a to su sam URL baze podataka sa nazivom, korisničkim imenom i lozinkom. Na taj način eliminirali smo korištenje PHP-a te ujedno cijelu aplikaciju učinili puno jednostavnijom i manje složenom. Iako se ovo na prvi pogled čini kao idealno rješenje u praksi i nije baš tako.

Kao što je poznato Android instalacijska datoteka ima nastavak .apk i kao takva se koristi za instaliranje aplikacija na Android pametne uređaje. Ta ista datoteka se može konvertirati tj. pretvoriti iz instalacijske datoteke u projektne datoteke[8]. Ovo ne bi bio problem da se u projektnim datotekama ne nalazi glavni elementi za spajanje na bazu podataka a to su kao što je već navedeno URL sa nazivom baze podataka, korisničko ime i lozinka, zajedno sa SQL upitima. Iz navedenog, autor ovog diplomskog rada zaključuje da korištenjem JDBC biblioteke za spajanje i dohvat podataka sa MySQL baze predstavlja veliki sigurnosni problem. To je jedan od razloga odabira i korištenja PHP datoteka u kojima se nalaze SQL upiti za bazu podataka, te se kompletan proces komunikacije između baze podataka i same aplikacije ne odvijaju na Android uređaju već preko PHP datoteka pohranjenim na samom poslužitelju.

5.2. Mehanizmi dohvaćanja podataka unutar aplikacije

Nakon odabira PHP-a kao skriptnog jezika za izvršavanje SQL upita u bazu podataka, zbog ranije navedenog sigurnosnog rizika, trebalo je kreirati mehanizme koji bi se približili onima same Firebase baze podataka. Dohvaćeni podaci nisu bili razumljivi Android operacijskom sustavu, te je iste bilo potrebno pretvoriti u JSON notaciju sličnu onoj koju Firebase koristi za skladištenje istih u svoju bazu podataka.

Na sljedećoj slici vidimo jedan PHP(MySQL) upit u bazu podataka te spremanje rezultata upita unutar JSON notacije tj. niza parova(ključ->vrijednost). Takav je niz potrebno parsirati.

[8] Primjer web verzije Android dekompailera

Slika 55. Primjer PHP upita

```
$query="SELECT  DATE_FORMAT(pk.Datum_prijava, '%d.%m.%Y.godine'), pk.id_prijava, pk.Sat_prijava,
              pk.Opis,pk.Status, k1.Korisnicko_ime AS k1_ime, kvar.Naziv_kvar, smjestaj.Br_sobe,
              lokacija.Naziv_lokacija,DATE_FORMAT(rjeseni_kvar.Datum_rjesenja,'%d.%m.%Y.godine'),
              rjeseni_kvar.Sat_rjesenja,k2.Korisnicko_ime AS korisnicko_ime,
              pk.Id_smjestaj,pk.Id_kvar,pk.Ciscenje
              FROM prijava_kvara pk
              LEFT OUTER JOIN korisnik k1 ON pk.Id_korisnika=k1.Id_korisnika
              LEFT OUTER JOIN kvar ON pk.Id_kvar=kvar.Id_kvar
              LEFT OUTER JOIN smjestaj ON pk.Id_smjestaj=smjestaj.Id_smjestaj
              LEFT OUTER JOIN lokacija ON smjestaj.Id_lokacija=lokacija.Id_lokacija
              LEFT OUTER JOIN rjeseni_kvar ON pk.Id_rjesenik=rjeseni_kvar.Id_rjesenik
              LEFT OUTER JOIN korisnik k2 ON k2.Id_korisnika=rjeseni_kvar.Id_korisnik
              WHERE pk.Status='Riješeno' AND pk.Ciscenje!='NULL'";


$res = mysqli_query($connection, $query) or die(mysqli_error($connection));
$result = array();

while($row = mysqli_fetch_array($res)){
array_push($result,
array(

'prijavadatum'=>$row[0],
'prijavaid'=>$row[1],
'prijavasat'=>$row[2],
'kvaropis'=>$row[3],
'status'=>$row[4],
'prijavakorisnik'=>$row[5],
'kvarnaziv'=>$row[6],
'brsobe'=>$row[7],
'nazivlokacija'=>$row[8],
'datumrjesenja'=>$row[9],
'satrjesenja'=>$row[10],
'rjesio'=>$row[11],
'idsmjestaj'=>$row[12],
'idkvar'=>$row[13],
'ciscenje'=>$row[14]

));
}
echo json_encode(array("result"=>$result));
```

Izvor: Autor

Proces parsiranja prikazan je sljedećim programskim isječkom na slici 56. Rezultat parsiranja pohranjuje se unutar Hashmape(rasute tablice) sa vrijednostima ključ->podatak. Takav se skup Hashmapa pohranjuje unutar niza. Nakon formiranja takvog niza veoma je jednostavno pristupanje podacima unutar istih.

Slika 56. Programski isječak parsiranja JSON objekta

```
JSONObject jsonObj = new JSONObject(jsonString);
JSONArray rijeseni = jsonObj.getJSONArray(TAG_RESULTS);

for (int i = 0; i < rijeseni.length(); i++) {
    JSONObject o = rijeseni.getJSONObject(i);

    String brsobe = o.getString(TAG_BRSOBE);
    String prijavadatum = o.getString(TAG_PRIJAVADATUM);
    String prijavasat = o.getString(TAG_PRIJAVASAT);
    String kvarnaziv = o.getString(TAG_KVARNAZIV);
    String kvaropis = o.getString(TAG_KVAROPIS);
    String prijavakorisnik = o.getString(TAG_PRIJAVAKORISNIK);
    String status = o.getString(TAG_STATUS);
    String datumrjesenja = o.getString(TAG_DATUMRJESENJA);
    String satrjesenja = o.getString(TAG_SATRJESENJA);
    String rjesio = o.getString(TAG_RJESIO);
    prijavaid = o.getString(TAG_PRIJAVAID);
    idsmjestaj =o.getString(TAG_IDSMJESTAJ);
    idkvar = o.getString(TAG_IDKVAR);
    lokacija = o.getString(TAG_LOKACIJA);
    String ciscenje = o.getString(TAG_CISCENJE);

    HashMap<String, String> vrijednost = new HashMap<>();

    vrijednost.put(TAG_BRSOBE, brsobe);
    vrijednost.put(TAG_PRIJAVADATUM, prijavadatum);
    vrijednost.put(TAG_PRIJAVASAT, prijavasat);
    vrijednost.put(TAG_KVARNAZIV, kvarnaziv);
    vrijednost.put(TAG_KVAROPIS, kvaropis);
    vrijednost.put(TAG_PRIJAVAKORISNIK, prijavakorisnik);
    vrijednost.put(TAG_STATUS, status);
    vrijednost.put(TAG_PRIJAVAID, prijavaid);
    vrijednost.put(TAG_IDSMJESTAJ, idsmjestaj);
    vrijednost.put(TAG_IDKVAR, idkvar);
    vrijednost.put(TAG_LOKACIJA, lokacija);
    vrijednost.put(TAG_DATUMRJESENJA, datumrjesenja);
    vrijednost.put(TAG_SATRJESENJA, satrjesenja);
    vrijednost.put(TAG_RJESIO, rjesio);
    vrijednost.put(TAG_CISCENJE, ciscenje);

    listarijeseniS.add(vrijednost);
}
```

Izvor: Autor

Za izvođenje PHP upita koristimo Android klasu AsyncTask ili asinkroni zadatak. AsyncTask se koristi vrlo često pogotovo ako su u pitanju mrežni upiti u bazu. Prednost AsyncTask je u tome što se isti uvijek izvodi u pozadinskoj dretvi ne glavnoj(UI dretvi), dok se prikaz samih rezultata dobivenih izvođenjem AsyncTask

klase uvijek prikazuju u glavnoj(UI dretvi). Isti nam pruža mogućnost izvođenja dijela operacija u pozadini nevezano za trenutačno izvođenje programa(aplikacije).

Sam asinkroni zadatak sastoji se od sljedećih metoda:

- onPreExecute().
- doInBackground(Params…).
- onProgressUpdate(Progress…).
- onPostExecute(Result).

Metoda onPreExecute() poziva se iz korisničkog sučelja i u većini slučajeva služi za prikazivanje Spinnera ili neke poruke da je izvršavanje zadatka u tijeku.

doInBackground(Params…) je glavna metoda, ona koja obavlja zadatak, poziva se u zasebnoj dretvi i izvršava zadatak koji joj je namijenjen. U ovom diplomskom radu koristi se za izvršavanje PHP upita i parsiranje dobivenih JSON podataka.

onProgressUpdate(Progress…) izvršava se na UI dretvi. Vrijeme same aktivacije i izvršavanja date metode nije vremenski definiran iz razloga što je njen konkretni zadatak prikaz stanja izvršavanja glavnog zadatka u ovom slučaju izvršavanje metode doInBackground(Params..).

Metoda onPostExecute(Result) je metoda korisničkog sučelja koja služi za prikazivanje rezultat dobivenog od metode doInBackground(Params..). U istoj se metodi prekida izvođenje metode onProgressUpdate(Progress…). U ovom diplomskog radu koristi se za pozivanje i predaju podataka adapteru zaslužnog za prikaz prijavljenih kvarova.

Slika 57. Primjer AsyncTask klase korištene u aplikaciji „Održavanje“

```
private class rijeseniSPHP extends AsyncTask<Void, Void, Void> {

    @Override
    protected Void doInBackground(Void... arg0) {
        HttpHandler httphandler = new HttpHandler();
        String jsonString = httphandler.makeServiceCall(url);
        Log.e(TAG_RESULTS, "Rezultat url: " + jsonString);

        if (jsonString != null) {
            try {

                listarijeseniS.clear();//brisanje prijasnje listarijeseniS
                JSONObject jsonObj = new JSONObject(jsonString);
                JSONArray rijeseni = jsonObj.getJSONArray(TAG_RESULTS);

                for (int i = 0; i < rijeseni.length(); i++) {
                    JSONObject o = rijeseni.getJSONObject(i);

                    String brsobe = o.getString(TAG_BRSOBE);
                    String prijavadatum = o.getString(TAG_PRIJAVADATUM);
                    String prijavasat = o.getString(TAG_PRIJAVASAT);
                    String kvarnaziv = o.getString(TAG_KVARNAZIV);
                    String kvaropis = o.getString(TAG_KVAROPIS);
                    String prijavakorisnik = o.getString(TAG_PRIJAVAKORISNIK);
                    String status = o.getString(TAG_STATUS);
                    String datumrjesenja = o.getString(TAG_DATUMRJESENJA);
                    String satrjesenja = o.getString(TAG_SATRJESENJA);
                    String rjesio = o.getString(TAG_RJESIO);
                    prijavaid = o.getString(TAG_PRIJAVAID);
                    idsmjestaj =o.getString(TAG_IDSMJESTAJ);
                    idkvar = o.getString(TAG_IDKVAR);
                    lokacija = o.getString(TAG_LOKACIJA);
                    String ciscenje = o.getString(TAG_CISCENJE);

                    HashMap<String, String> vrijednost = new HashMap<>();

                    vrijednost.put(TAG_BRSOBE, brsobe);
                    vrijednost.put(TAG_PRIJAVADATUM, prijavadatum);
                    vrijednost.put(TAG_PRIJAVASAT, prijavasat);
                    vrijednost.put(TAG_KVARNAZIV, kvarnaziv);
                    vrijednost.put(TAG_KVAROPIS, kvaropis);
                    vrijednost.put(TAG_PRIJAVAKORISNIK, prijavakorisnik);
                    vrijednost.put(TAG_STATUS, status);
                    vrijednost.put(TAG_PRIJAVAID, prijavaid);
                    vrijednost.put(TAG_IDSMJESTAJ, idsmjestaj);
                    vrijednost.put(TAG_IDKVAR, idkvar);
                    vrijednost.put(TAG_LOKACIJA, lokacija);
                    vrijednost.put(TAG_DATUMRJESENJA, datumrjesenja);
                    vrijednost.put(TAG_SATRJESENJA, satrjesenja);
                    vrijednost.put(TAG_RJESIO, rjesio);
                    vrijednost.put(TAG_CISCENJE, ciscenje);

                    listarijeseniS.add(vrijednost);
                }
            } catch (final JSONException e) {
                Log.e(TAG_RESULTS, "Greška-JSON Parsiranje:  " + e.getMessage());
                getActivity().runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        Toast.makeText(getContext(),
                                "Greška-JSON Parsiranje:  " + e.getMessage(),
                                Toast.LENGTH_LONG)
                                .show();
                    }
```

```
                });

            }
        } else {
            Log.e(TAG_RESULTS, "Nemogućnost dohvata JSON sa servera..");
            getActivity().runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    Toast.makeText(getContext(),
                            "Nemogućnost dohvata JSON sa servera.!",
                            Toast.LENGTH_LONG)
                            .show();
                }
            });

        }

        return null;
    }
```

Izvor: Autor

Prvo se preko `HttpHandler()` pomoćne klase izvršava upit preko http protokola u bazu. Od dobivenog JSON niza koji je ujedno i odgovor iz baze izvršava se JSON parsiranje. Dohvaća se JSON objekt za svaku vrijednost te se ta vrijednost sprema u varijable tipa String(`String brsobe=o.getString(TAG_BRSOBE);`)

Nakon sto se svaki podatak JSON niza kroz jednu iteraciju spremi unutar varijable tipa String iste se dodaju unutar rasute tablice(Hashmape) naziva „vrijednost“ po sistemu ključ->vrijednost(`vrijednost.put(TAG_BRSOBE, brsobe);`. Nakon dodavanja svih vrijednosti Stringova unutar Hashmape vrijednost iste se spremaju unutar niza rasutih tablica `listarijeseniS.add(vrijednost);`

Na kraju same doInBackground() metode nalaze se kontrole za hvatanje iznimaka u slučaju nemogućnosti dohvata podataka sa poslužitelja ili izvođenja JSON parsiranje. Naravno prije svakog JSON parsiranja briše se prethodna lista podataka naredbom listarijeseniA.clear(). Kao rezultat završetka doInBacground() metode poziva se onPostExecute(Result) metoda koja listu kvarova prosljeđuje klasi adapterRijeseniS() za prikaz svakog pojedinog kvara.

Klasu tipa AsyncTask() posjeduje svaki fragment unutar same aplikacije za izvršavanje upita preko PHP datoteke i parsiranje dobivenog rezultata u JSON notaciji.

Na sljedećoj slici dan je programski isječak PHP datoteke zaduženu za dohvaćanje podataka iz baze podataka u kojoj možemo vidjeti izvršavanje samog upita te spremanje podataka unutar niza i kodiranje istog u JSON format.

Slika 58. Primjer PHP upita u bazu

```php
<?php
require("db_connect.php");
$query="SELECT  DATE_FORMAT(pk.Datum_prijava, '%d.%m.%Y.godine'), pk.id_prijava, pk.Sat_prijava,
                pk.Opis,pk.Status, k1.Korisnicko_ime AS k1_ime, kvar.Naziv_kvar, smjestaj.Br_sobe,
                lokacija.Naziv_lokacija,DATE_FORMAT(rjeseni_kvar.Datum_rjesenja,'%d.%m.%Y.godine'),
                rjeseni_kvar.Sat_rjesenja,k2.Korisnicko_ime AS korisnicko_ime,pk.Id_smjestaj,
                pk.Id_kvar,pk.Ciscenje
                FROM prijava_kvara pk
                LEFT OUTER JOIN korisnik k1 ON pk.Id_korisnika=k1.Id_korisnika
                LEFT OUTER JOIN kvar ON pk.Id_kvar=kvar.Id_kvar
                LEFT OUTER JOIN smjestaj ON pk.Id_smjestaj=smjestaj.Id_smjestaj
                LEFT OUTER JOIN lokacija ON 6=lokacija.Id_lokacija
                LEFT OUTER JOIN rjeseni_kvar ON pk.Id_rjesenik=rjeseni_kvar.Id_rjesenik
                LEFT OUTER JOIN korisnik k2 ON    k2.Id_korisnika=rjeseni_kvar.Id_korisnik
                WHERE smjestaj.Id_lokacija=6 AND pk.Status='Riješeno' AND pk.Ciscenje!='NULL'";

$res = mysqli_query($connection, $query) or die(mysqli_error($connection));
$result = array();

while($row = mysqli_fetch_array($res)){
array_push($result,
array(

'prijavadatum'=>$row[0],
'prijavaid'=>$row[1],
'prijavasat'=>$row[2],
'kvaropis'=>$row[3],
'status'=>$row[4],
'prijavakorisnik'=>$row[5],
'kvarnaziv'=>$row[6],
'brsobe'=>$row[7],
'nazivlokacija'=>$row[8],
'datumrjesenja'=>$row[9],
'satrjesenja'=>$row[10],
'rjesio'=>$row[11],
'idsmjestaj'=>$row[12],
'idkvar'=>$row[13],
'ciscenje'=>$row[14]

));
}

echo json_encode(array("result"=>$result));
mysqli_close($connection);
?>
```

Izvor: Autor

Metoda doInBackground() izvršit će se samo jednom i to ili prilikom pristupa fragmentu kojem pripada ili povratku u isti. Pošto Firebase omogućava izvještavanje promjene podataka u realnom vremenu, da bi smo se približili istom uveden je mehanizam ponavljanja izvođenja AsyncTask klase i to uvođenjem klase Timer i klase TimerTask. Klasa TimerTask je vremenska klasa koja se okida nakon prolaska određenog vremena. Pozivanjem same AsyncTask klase unutar ove repetitivne klase uveden je mehanizam ponavljanja iste koja će se izvršiti u određenom vremenskom razdoblju kojeg programer same aplikacije definira. Na sljedećoj slici dat je programski isječak klase TimerTask za pozivanje AsyncTask klase.

Slika 59. Primjer Timer i TimerTask klase korištene u projektu

```
private Timer timerAsync;
private TimerTask timerTaskAsync;

public void repeatrijeseniSPHP() {
    final Handler handler = new Handler();
    timerAsync = new Timer();
    timerTaskAsync = new TimerTask() {
        @Override
        public void run() {
            handler.post(() -> {
                    try {
                        sPHP = new rijeseniSPHP();
                            sPHP.execute();

                    } catch (Exception e) {
                        e.printStackTrace();
                    }
            });
        }
    };
    timerAsync.schedule(timerTaskAsync, 0, 50000);
}
```

Izvor: Autor

Navedeni programski isječak prikazuje metodu repeatrijeseniPHP() koja ponavljanjem AsyncTask klase rijeseniPHP() dohvaća sve prijavljene kvarove koje nisu riješeni iz baze podataka pomoću PHP upita. Samo okidanje izvršava se svaki 50000 milisekunda sto je ekvivalent od 50 sekundi. Ista se može povećati no smanjivanje nije preporučljivo zbog samog opterećenja za Android uređaj učestalim ponavljanjem asinkronog zadatka.

Sada kada smo razjasnili način dohvata podataka iz baze podataka, njihovo interpretiranje, te objasnili proces prikupljanja novih podataka, ostaje nam samo riješiti sustav obavijesti korisnika o promjeni unutar baze podataka. Pošto se podaci prikupljaju svakih 50 sekundi, aplikacija još uvijek nezna jesu li dobiveni novi podaci zaista novi, tj. dali se zaista desila kakva promjena zapisa, dodavanje ili brisanje unutar baze. Za samu provjeru dali su novi podaci zaista novi, kao rješenje nam se samo nameće određeni mehanizam usporedbe.

Sustav usporedbe koji je korišten u ovoj aplikaciji temelji se na spremanju svakog parsiranog rezultata upita iz baze tj. spremanje iz Hashmape(rasute tablice) u memoriju Android uređaja. Kod načina spremanja imamo više mogućnosti unutar Androida. Možemo koristiti SharedPreferences, ugrađenu bazu unutar Android operacijskog sustava(SQLite) i kreiranje obične txt. datoteke unutar samog Android operacijskog sustava. U ovoj aplikaciji odabrana je obična txt. datoteka no prije upisa potrebno je izvršiti konverziju podataka tj. pretvaranje niza Hashmape(rasute tablice) u string. Nakon pretvaranja, potrebno je učitati stare podatke iz txt. datoteke koji su već pretvoreni u string i izvršiti usporedbu između dva stringa. Ako oni nisu isti obavještava se korisnika da je došlo do ažuriranja podataka, prikazuje se na ekranu nova lista kvarova i sprema se novi niz unutar same baze podataka. Na te načine Android uređaj svakog korisnika čuva svoju listu podataka i ista se može usporediti sa novom. Na početku svakog izvođenja klase AsyncTask, niz Hashmape se briše i time oslobađa za primanje novih podataka za usporedbu.

Odabir obične txt. datoteke pao je iz razloga što u isti upisujemo niz podataka pretvoren u string te usporedba se vodi između dva takva stringa. Zbog tog razloga korištenje unutarnje baze podataka samog Android operacijskog sustava činio se nepotrebnim, dok SharedPreferences ima manu postojanja tih podataka nakon izlaska iz aplikacije jer se oni brišu iz memorije uređaja prilikom izlaska iz iste, no u ovom slučaju to ne odgovara zbog trajnog pohranjivanja podataka.

Na sljedećim slikama koji su ujedno i isječak koda prikazane su metode dohvata, konverzije, spremanja i uspoređivanja podataka te metoda za obavijest korisnika o nastalim promjenama.

Slika 60. Metode za spremanje i čitanje niza(liste) iz tekstualne datoteke

```
private void upisListe(String l) {

    getActivity().deleteFile("NerijeseniS.txt");

    try {
        FileOutputStream fileout=getActivity().openFileOutput("NerijeseniS.txt", MODE_PRIVATE);
        OutputStreamWriter outputWriter=new OutputStreamWriter(fileout);
        outputWriter.write(l);
        outputWriter.close();
        System.out.println("Upis u TXT:  "+l);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private void citanjeListe() {

    try {
        FileInputStream fileIn=getActivity().openFileInput("NerijeseniS.txt");
        InputStreamReader InputRead= new InputStreamReader(fileIn);

        char[] inputBuffer= new char[READ_BLOCK_SIZE];
        String str="";
        int charRead;
        while ((charRead=InputRead.read(inputBuffer))>0) {
            String readstring=String.copyValueOf(inputBuffer,0,charRead);
            str+=readstring;
        }
        InputRead.close();
        lista2=str;

    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Izvor: Autor

Slika 61. Primjer metode za uspoređivanje i pretvaranje u string

```
private void usporediNerijeseniS(String lista1){
    String title="Lista ažurirana";
    String text="Molim vas pogledajte listu kvarova";
    citanjeListe();
    try {
        if (lista1.contains(lista2)) {
            System.out.println("UsporediNerijeseniS: Liste su iste ");
            System.out.println("1. lista: " + lista1);
            System.out.println("2. lista: " + lista2);

        } else {
            System.out.println("UsporediNerijeseniS: Liste su različite ");
            System.out.println("1. lista: " + lista1);
            System.out.println("2. lista: " + lista2);
            createNotification(title, text);
        }
    }catch(NullPointerException e){

    }
}


private String pretvoriuString( ArrayList<HashMap<String, String>> nerijeseni){
    String arrayList1=null;
    try{

        Gson gson = new Gson();
        arrayList1 = gson.toJson(nerijeseni);

    }
    catch(ConcurrentModificationException e){

    }
    return arrayList1;
}
```

Izvor: Autor

Slika 62. Metoda za kreiranje obavijesti

```
private void createNotification(String contentTitle, String contentText) {

    Uri soundUri = RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION);

    Notification.Builder builder = new Notification.Builder(getContext())
            .setSmallIcon(R.drawable.pocetna)
            .setAutoCancel(true)
            .setContentTitle(contentTitle)
            .setSound(soundUri)
            //.SetDefaults (NotificationDefaults.Sound | NotificationDefaults.Vibrate)
            .setContentText(contentText);



    mNotification = builder.getNotification();
    mNotificationManager.notify(NOTIFICATION_ID, mNotification);
}
```

Izvor: Autor

Korištenjem gore navedenih metoda, klasa i mehanizma autor ovoga rada uspio se približiti mehanizmu rada Firebase baze podataka, te iako nije identičan iz razloga što se upiti stalno ponavljaju s odgodom od 50 sekundi, isti su zanemarivi jer se očuvalo pravilo korištenja MySQL-a kao baze podataka za pohranu podataka te očuvala mogućnost daljnje nadogradnje sustava na web inačicu istog.

ZAKLJUČAK

Android operacijski sustav svakim danom uzima sve više maha nasuprot konkurenata na tržištu. Android operacijski sustav više nije instaliran samo u pametne telefone već i u druge uređaje(npr. satove, televizije). Izrada aplikacija za Android postala je rutina iz razloga što se za gotovo svaku PC verziju nastoji napraviti njezina mobilna inačica i obuhvatiti što više korisnika. S druge strane mobilne aplikacija postale su u nekim okruženjima nužno zlo, jer ta ista okruženja ne dozvoljavaju izradu PC izdanja aplikacije. Na komfor koji pružaju mobilne aplikacije se korisnici svakim danom sve više privikavaju te polako postoji bitnije imati mobilnu aplikaciju od PC verzije.

Ovaj diplomski rad bavi se izradom mobilne aplikacije(Android) za prijavu kvarova u hotelijerstvu. Ista je zamišljena da može funkcionirati samostalno ili se integrirati u isti takav sustav u PC izdanju koji koristi MySQL za bazu podataka uz ostavljenu mogućnost daljnjeg razvoja i unaprjeđivanja istog.

Najveći zadatak prilikom programiranja ove aplikacije bio je osmisliti mehanizam automatske provjere promjene zapisa unutar MySQL baze podatka. Pošto Firebase nismo smjeli koristiti zbog budućih „mutacija“ same aplikacije i posjedovanju podataka na vlastitom poslužitelju, morali smo mehanizme Firebase baze podataka imitirati kroz samu Android aplikaciju korištenjem Androidovih mogućnosti uparenih sa drugim tehnologijama (npr. PHP).

Ova aplikacija je dokazala da je moguće približiti se Firebase mehanizmima automatskog obavještavanja promjena u bazi.Za dobivanje najidealnijeg sustava kao sam odgovor nameće se suživot Firebase baze podataka i MySQL baze podataka zajedno u takvoj zajednici gdje bi se svaka promjena nad podacima i upis novih zapisivao u obje baze podataka, naravno uz privolu samog korisnika.

SAŽETAK

Ovaj rad bavi se izradom mobilne aplikacije za prijavu kvarova u hotelijerstvu. Dosadašnji načini bilježenja kvarova su u dnevnik kvarova ili pomoću PC aplikacija. Aplikacija je razvijena za Android operacijski sustav. Cilj aplikacije bio je smanjenje vremena od same detekcije pa do otklanjanja kvara. Ograničenje je bilo korištenje MySQL za bazu podataka. Iako Google ima svoje rješenje u pogledu baze podataka(Firebase) zadatak ovog rada bio je usporediti te dvije baze podataka i imitirati mehanizme Firebase baze podataka. Isto se postiglo suradnjom same Android aplikacije, PHP-a(isti nam omogućuje daljnje razvijanje i unaprjeđenje same aplikacije) i JSON-a.

Ključne riječi: *mobilna aplikacija, Android operacijski sustav, baza podataka, Firebase, prijava kvara*

SUMMARY

This paper deals with the design of mobile applications for failure registrations in hotel. Forms of malfunction have been reported in a malfunction diary or PC application. The application is developed for the Android operating system. The goal of the application was to reduce the time from the detection itself to eliminating the failure. The limit was to use MySQL for the database. Although Google has its own database solution (Firebase), the task of this paper was to compare these two databases and imitate the Firebase database mechanisms. The same is achieved through collaboration of the Android application itself, PHP (the same allows us to further develop and improve the application itself) and JSON.

Keywords: *mobile applications, Android operating system, database, Firebase, malfunction report.*

LITERATURA

Knjige

- Android Programming: The Big Nerd Ranch Guide (2nd Edition) ,Autori: Bill Phillips, Chris Stewart, Brian Hardy, Kristin Marsicano, Nakladnik: Big Nerd Ranch(2015).
- Asynchronous Android Programming Second Edition, Autor: Helder Vasconcelos, Nakladnik: Packt Publishing(2016).
- Firebase Essentials - Android Edition(First Edition), Autor: Neil Smyth, Nakladnik: Payload Media, Inc.(2017).

Internet izvori

- http://www.androidbegin.com/tutorial/android-search-filter-listview-images-and-texts-tutorial/ Pristupljeno(7.8.2017)
- https://www.androidpit.com/forum/550920/tutorial-how-to-add-search-function-to-custom-listview Pristupljeno(7.8.2017)
- https://stackoverflow.com/questions/26919099/implement-search-in-listview-inside-fragment Pristupljeno(7.8.2017)
- https://stackoverflow.com/questions/7230893/android-search-with-fragments Pristupljeno(10.8.2017)
- http://www.edumobile.org/android/action-bar-search-view/ Pristupljeno(11.8.2017)
- https://www.androidhive.info/2013/11/android-working-with-action-bar/ Pristupljeno(20.8.2017)
- http://semycolon.blogspot.hr/2014/11/first-android-app-step-7-populate.html Pristupljeno(15.6.2017)
- https://stackoverflow.com/questions/34699273/android-turn-on-off-camera-flash-programatically-with-camera2 Pristupljeno(23.8.2017)
- http://hmkcode.com/android-menu-handling-click-events-changing-menu-items-at-runtime/ Pristupljeno(11.4.2017)

- http://www.viralandroid.com/2016/01/turn-on-and-off-camera-led-flashlight-programmatically.html Pristupljeno(23.8.2017)
- https://stackoverflow.com/questions/33560219/in-android-how-to-set-navigation-drawer-header-image-and-name-programmatically-i Pristupljeno(3.9.2017)
- https://www.androidhive.info/2013/11/android-sliding-menu-using-navigation-drawer/ Pristupljeno(4.9.2017)
- http://blog.teamtreehouse.com/add-navigation-drawer-android Pristupljeno(4.9.2017)
- http://codetheory.in/android-navigation-drawer/ Pristupljeno(4.9.2017)
- https://stackoverflow.com/questions/4993765/how-to-stop-the-timer-in-android Pristupljeno(17.9.2017)
- https://stackoverflow.com/questions/43359837/tab-layout-with-fragments Pristupljeno(18.9.2017)
- http://www.cs.dartmouth.edu/~campbell/cs65/lecture08/lecture08.html Pristupljeno(18.9.2017)
- https://stackoverflow.com/questions/8665676/how-to-use-notifydatasetchanged-in-thread Pristupljeno(18.9.2017)
- https://spin.atomicobject.com/2016/08/19/android-listview/ Pristupljeno(24.9.2017)
- https://stackoverflow.com/questions/25743290/run-a-task-on-ui-thread-from-fragment Pristupljeno(24.9.2017)
- http://www.techotopia.com/index.php/Creating_an_Android_Tabbed_Interface_using_the_TabLayout_Component Pristupljeno(1.10.2017)
- https://www.androidhive.info/2015/09/android-material-design-working-with-tabs/ Pristupljeno(1.10.2017)
- https://stackoverflow.com/questions/10774871/best-way-to-compare-dates-in-android Pristupljeno(1.10.2017)
- https://alvinalexander.com/java/simpledateformat-convert-date-to-string-formatted-parse Pristupljeno(1.10.2017)
- https://stackoverflow.com/questions/26032540/how-to-delete-row-item-from-list-view-in-android Pristupljeno(7.10.2017)

- https://stackoverflow.com/questions/5497580/how-to-dynamically-remove-items-from-listview-on-a-button-click Pristupljeno(13.10.2017)
- https://www.androidcode.ninja/android-notification-bar-sound-icon/ Pristupljeno(13.10.2017)
- http://www.mysamplecode.com/2011/09/android-asynctask-httpclient-with.html Pristupljeno(21.4.2017)
- http://simpleandroidtutorials.blogspot.hr/2012/06/periodically-update-data-from-server-in.html Pristupljeno(21.10.2017)
- https://stackoverflow.com/questions/36853696/android-service-mysql-check-for-database-change Pristupljeno(4.11.2017)
- https://stackoverflow.com/questions/24264093/how-to-use-android-alarmmanager-with-small-intervals-like-1-minute Pristupljeno(10.11.2017)
- https://stackoverflow.com/questions/43467229/android-alarm-manager-is-repeating-after-5-seconds-and-ignoring-interval-time Pristupljeno(11.11.2017)
- https://coderwall.com/p/qfoxfg/schedule-a-service-using-alarmmanager Pristupljeno(11.11.2017)
- http://androidhiker.blogspot.hr/2015/10/android-timer-and-timertask-scheduling.html Pristupljeno(13.11.2017)
- https://stackoverflow.com/questions/31025568/run-task-in-background-periodically Pristupljeno(13.11.2017)
- http://blog.jana.com/blog/2015/09/10/apple-cant-touch-androids-market-share-in-emerging-markets Pristupljeno(27.11.2017)
- https://icons8.com/ Pristupljeno(23.10.2017)
- https://www.tutorialrepublic.com/php-tutorial/php-mysql-crud-application.php Pristupljeno(24.5.2017)
- https://www.codeofaninja.com/2011/12/php-and-mysql-crud-tutorial.html
- Pristupljeno(24.5.2017)
- https://www.mysql.com/ Pristupljeno(28.11.2017)
- https://www.concretepage.com/android/android-alarm-clock-tutorial-to-schedule-and-cancel-alarmmanager-pendingintent-and-wakefulbroadcastreceiver-example Pristupljeno(14.11.2017)
- https://www.presentationgo.com/presentation/swot-analysis-powerpoint-template-with-cycle-matrix/ Pristupljeno(25.11.2017)

- https://www.code-sample.com/2017/09/how-to-setup-firebase-environment.html Pristupljeno(27.11.2017)
- https://www.tutorialspoint.com/android/android_json_parser.htm Pristupljeno(28.11.2017)
- https://gonehybrid.com/firebase-database-best-practices/ Pristupljeno(30.11.2017)
- https://howtofirebase.com/collection-queries-with-firebase-b95a0193745d
- https://db-engines.com/en/system/Firebase+Realtime+Database%3BMySQL%3BSQLite Pristupljeno(30.11.2017)
- https://crisp.chat/blog/why-you-should-never-use-firebase-realtime-database/ Pristupljeno(30.11.2017)
- https://www.idc.com/promo/smartphone-market-share/os Pristupljeno(29.11.2017)
- https://www.w3schools.com/js/js_json_intro.asp Pristupljeno(29.11.2017)
- https://www.json.org/ Pristupljeno(26.11.2017)
- https://firebase.google.com/ Pristupljeno(29.11.2017)
- http://www.c4learn.com/android/android-os-architecture/ Pristupljeno(1.12.2017)
- https://datayo.wordpress.com/2015/10/11/android-programming/ Pristupljeno(1.12.2017)
- https://wccftech.com/idc-report-q1-android-window-ios/ Pristupljeno(1.12.2017)
- https://www.tutorialspoint.com/xamarin/xamarin_android_activity_lifecycle.htm Pristupljeno(2.12.2017)
- https://www.novoda.com/blog/bonfire/ Pristupljeno(2.12.2017)
- https://www.sitepoint.com/mysql-foreign-keys-quicker-database-development/ Pristupljeno(18.12.2017)
- http://www.informit.com/articles/article.aspx?p=1216889&seqNum=4 Pristupljeno(5.1.2018)
- https://dev.mysql.com/doc/refman/5.5/en/create-table-foreign-keys.html Pristupljeno(6.1.2018)

- https://stackoverflow.com/questions/21010367/how-to-decompile-an-apk-or-dex-file-on-android-platform Pristupljeno(23.1.2018)
- http://www.vogella.com/tutorials/MySQLJava/article.html Pristupljeno(1.2.2017)

Izvori fusnota i slika

- Android službena stranica https://www.android.com/
- Google službena stranica https://www.google.com/intl/en/about/
- Open Handset Alliance https://www.openhandsetalliance.com/
- Softpedia službena stranica http://news.softpedia.com/news/Google-Details-Upcoming-Update-for-Search-App-for-Android-464847.shtml
- wccftech službena stranica https://wccftech.com/idc-report-q1-android-window-ios/
- JANA službeni blog http://blog.jana.com/blog/2015/09/10/apple-cant-touch-androids-market-share-in-emerging-markets
- datayo službena stranica https://datayo.wordpress.com/2015/10/11/android-programming/
- JAVA službena stranica https://www.java.com/en/
- Tutorialspoint službena stranica https://www.tutorialspoint.com/xamarin/xamarin_android_activity_lifecycle.htm
- MySQL službena stranica https://www.mysql.com/
- PHP službena stranica http://php.net/
- JSON službena stranica https://www.json.org/
- Developer Android službena stranica https://developer.android.com/about/dashboards/index.html
- droidlife službena stranica https://www.droid-life.com/2016/01/05/android-distribution-update-for-january-2016/
- novoda službena stranica https://www.novoda.com/blog/bonfire/
- Android dekompailer https://www.apkdecompilers.com/

POPIS SLIKA

POPIS TABLICA