Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Mark Merdjadi

Aplikacija za optimizacijo servisnih storitev v gostinskih lokalih

diplomsko delo

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Mentor: doc. dr. Franc Solina

Ljubljana, 2018

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Razvijte in opišite informacijski sistem za optimizacijo servisnih storitev v gostinstvu s pomočjo mobilnih tehnologij. Serviserjem naj sistem nudi pregled naročenih servisov, pregled opravljenih servisov in možnost pisanja in shranjevanja zapisnikov o opravljenem delu.

Rad bi se zahvalil svoji družini, ki me je vseskozi podpirala pri izzobraževalnem procesu. Posebna zahvala gre tudi podjetju OptiBar d.o.o za zaupanje in priložnost izdelave diplome v njihovem podjetju.

Posvečeno družini

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
API	Application programming interface	aplikacijski programski vmesnik
HTTP	HyperText transfer protocol	jezik za označevanje nadbesedila
IDE	Integrated development environment	Integrirano razvojno okolje
REST	Representational state transfer	Arhitektura za izmenjavo podatkov med spletnimi storitvam
SaaS	Software as a service	Programska oprema kot storitev
SQL	Structured Query Langauge	Strukturirani povpraševalni jezik za delo s podatkovnimi bazami
URL	Uniform Resource Locator	Enolični krajevnik vira
XML	Extensible Markup Language	Razširljivi označevalni jezik

Povzetek

Naslov: Aplikacija za optimizacijo servisnih storitev v gostinskih lokalih

Avtor: Mark Merdjadi

Odpreti lasten gostinski obrat je eden izmed najbolj pogostih podjetniških podvigov na svetu. Skoraj ni ulice, ki ne bi imela vsaj enega lokala ali nočnega kluba. Čeprav so ti obrati zaželeni med ljudmi, pa je njihov propad pogost. Do tega večinoma privede slaba organizacija, komunikacija in optimizacija delovnih procesov znotraj lokala. Gostinska oprema, kot so na primer kavni avtomati, je sestavni del kvalitetnega obratovanja lokala. Dobro ohranjena in pravilno delujoča oprema bo povečala kvaliteto izdelkov in storitev. Ko se avtomat pokvari, je v interesu obrata, da se ta popravi čim hitreje in bolje. Cilj diplomske naloge je razvoj informacijskega sistema za optimizacijo servisnih storitev v gostinstvu s pomočjo mobilnih tehnologij, ki so danes vseprisotne. Serviserjem bo sistem nudil pregled naročenih servisov, pregled opravljenih servisov in možnost pisanja in shranjevanja zapisnikov o opravljanem delu. Na drugi strani bo gostincem nudil podajo odzivov na opravljene storitve.

Ključne besede: Android, mobilna aplikacija, HoReCa, popravilo, servis, lokal, gostinstvo.

Abstract

Title: Application for optimization of repair services in catering establishments

Author: Mark Merdjadi

Starting an own catering establishment is one of the most popular business ventures in the world. Rarely are there streets, where at least one bar or a night club is not present. Although such establishment are popular among the people, they seem to suffer from a fairly high failure rate. This is most likely due to poor organization, communication and optimization of processes within the business. Catering equipment, such as coffee machines, are a fundamental part of a quality bar operation. A well kept and properly working equipment will increase the quality of the offered products and services. It is in the best interest of a business, to have their broken machines properly fixed as soon as possible. The goal of this project is to develop an information system for optimization of repair services within catering establishments, with the help of ever more ubiquitous mobile technologies. The system will allow repairmen to keep track of their orders, work and allow them to create and save records of their completed repairs. On the other hand, the catering employees will be given an option to send back feedback on completed services.

Keywords: Android, mobile application , HoReCa, repair, service, coffee shop, catering.

Poglavje 1

Uvod

1.1 Motivacija in cilji

Ustanovitev gostinskega obrata je eden izmed najbolj pogostih podjetniških podvigov. Sodeč po študijah, naj bi v prvem letu propadla slaba tretjina vseh obratov, še ena tretjina pa svoja vrata zapre v naslednjih treh do petih letih [15, 20]. Med najpogostejšimi razlogi za to je slabo vodenje in organizacija. Eden izmed najpomembnejših delov vsakega lokala je seveda njegova oprema, ki pa se ob konstantni uporabi kvari.

V diplomski nalogi bom predstavil razvoj mobilne aplikacije za optimizacijo procesa naročanja in opravljanja servisnih storitev tako, da na eni strani ponudim serviserjem elektronsko alternativo vsej papirologiji, na drugi strani pa zaposlenim v lokalu omogočim nek preprost način podajanja odziva o izvedenih storitvah. Zbrani podatki prav tako koristijo ponudnikom servisov, saj lahko tako natančno ocenijo, kje se delo opravlja dobro, in kje slabo. Aplikacija bo serviserjem omogočala spremljanje in pregled zgodovine servisov ter olajšala izdelavo njihovih zapisnikov, naročnikom oz. lokalom, pa bo omogočila ocenjevanje opravljenih storitev. Aplikacijo bodo na svoje mobilne naprave imeli nameščeno zaposleni v servisnem podjetju in zaposleni v gostinskem obratu. Vsak od njih bo imel prilagojen pogled, preko katerega bo potekala interakcija s sistemom.

Cilj je tako povečati transparentnost delovanja in okrepiti vez med entitetami znotraj domene gostinstva, s čimer bi lahko inovirali njihovo delovanje in jih naredili bolj konkurenčne [19].

1.1.1 Poslovni model in distribucija

Aplikacija je bila narejena v sklopu sistema OptiBar, ki ga razvija podjetje OptiBar d.o.o. Poslovni model trženja aplikacije temelji na modelu SaaS (Software as a Service). To pomeni, da bo podjetje za uporabo aplikacije zaračunavalo mesečni znesek, odvisen od števila gostinskih obratov. Aplikacijo lahko uporablja katerokoli podjetje, ki se ukvarja s ponudbo proizvodov in storitev gostinskim obratom, in le tem nudi tudi servisiranje opreme. Ta podjetja so tudi plačnik aplikacije, saj je uporaba za same gostinske obrate brezplačna. Za podjetja, ki se odločijo za uporabo aplikacije, se izvede tudi interno izobraževanje, na katerem se njihovim zaposlenim, predvsem terenskim delavcem, predstavi način uporabe aplikacije. Omenjeni terenski delavci namreč predstavljajo tudi mrežo, preko katere se širi uporaba v gostinske obrate.

1.2 Zgradba diplomskega dela

V drugem poglavju bom predstavil izbiro tehnologij za razvoj sistema in razlogi za njihovo izbiro. V tretjem poglavju bodo natančneje opisane funkcionalne zahteve projekta. Četrto poglavje se bo nanašalo na načrtovanje in implementacijo podatkovne baze. Nadaljeval bom z opisom razvoja spletne storitve v petem poglavju. Šesto poglavje bo opisalo načrtovanje in razvoj mobilne aplikacija, ki predstavlja osrednji del te diplomske naloge. V sedmem poglavju bom predstavil potek testiranja zaključenega projekta, osmo poglavje se bo nanašalo na mogoče izboljšave, v zadnjem devetem pa bom predstavil sklepne ugotovitve.

Poglavje 2

Uporabljene tehnologije

2.1 MySql

Za hranjenje podatkov smo uporabili sistem MySql. To je odprtokodni sistem za upravljanje z relacijskimi podatkovnimi bazami [18, 17]. Uporablja relacijski model, za katerega je značilno, da so podatki porazdeljeni v tabele (relacije) s stolpci in vrsticami. Vrstice predstavljajo nek zapis, stolpci pa atribute tega zapisa. Vsak zapis je enolično določen s primarnim ključem [25]. Po zapisih lahko poizvedujemo, jih dodajamo, brišemo ali urejamo z poizvedovalnim jezikom SQL. Prednost sistema MySql je v tem, da je zelo varen, skalabilen in kompatibilen z veliko večino operacijskih sistemov [16].

2.2 PHP in CodeIgniter

PHP je odprtokodni skriptni programski jezik, ki se uporablja za razvoj strežniških in dinamičnih spletnih storitev [21]. Gre za enega najbolj priljubljenih in uporabljenih programskih jezikov v spletnem programiranju. Njegova prednost je v tem, da se odlično ujema z MySqlom, ki ga uporabljamo. Prav tako se storitve, napisane v tem jeziku lahko izvajajo na veliki večini strežniških platform, delujejo tako na sistemih, ki bazirajo na UNIXu, kot tudi na sistemih Windows [22]. Gre za poceni, varno, hitro in zanesljivo

rešitev za razvijanje spletnih aplikacij [23].

CodeIgniter je programsko razvojno ogrodje za izgradnjo spletnih aplikacij v jeziku PHP. Njegov cilj je pohitritev razvoja projektov tako, da razvijalcem ponudi bogato množico knjižnic za pogoste primere uporabe. Za uporabo smo se odločili, ker gre za preprosto ogrodje, ki ne potrebuje posebne konfiguracije, ne uporablja velikih monolitičnih knjižnic, je kompatibilen z več verzijami jezika PHP in ima natančno in obsežno dokumentacijo [7].

2.3 Mobilni operacijski sistem Android

Android je operacijski sistem za pametne telefone in prenosne naprave, ki ga razvija Google. Je odprtokodni sistem in bazira na Linuxovem jedru. Nad jedrom tečejo vmesna programska oprema (middleware), knjižnice in programski vmesniki ter aplikacijska oprema, ki teče na aplikacijskem ogrodju, ki je kompatibilno z Java knjižnicami. Program bo tudi napisan v programskem jeziku Java. Razvijanje za Android je smiselno, saj ta sistem zajema nekaj več kot 50% trga na področju mobilnih naprav in je pogosto uporabljen na napravah, ki se masovno uporabljajo za elektronsko poslovanje [1, 3].

2.4 Razvojna okolja

2.4.1 Android studio

Android studio je uradno integrirano razvojno okolje za razvoj Android aplikacije [4], zgrajen na okolju IntelliJ IDEA podjetja JetBrains [6]. Dostopen je na vseh večjih operacijskih sistemih in je nadomestil Eclipse Android Development Tools kot primarno okolje za razvoj. Omogoča nam podporo za projekte, bazirane na sistemu Gradle, grafično okolje za lažji razvoj grafičnih vmesnikov aplikacije, orodja za opazovanje in razhroščevanje aplikacij in vgrajen Anroid emulator, ki nam omogoča testiranje na računalniku brez fizične naprave [5].

2.4.2 PhpStorm

Za razvoj spletnih storitev smo uporabili IDE PhpStorm. Vsebuje urejevalnike besedila za jezike Php, Javascript, HTML in CSS. Omogoča sprotno analizo kode in opozarjanje na napake, ponuja podporo za več verzij jezika Php, vgrajeno ima povezovanje in pregledovanje podatkovnih baz, ki jih uporabljamo v projektu ter je na voljo na večini operacijskih sistemov [24].

2.4.3 DataGrip

Za lažje upravljanje s podatkovno bazo, sem uporabil okolje DataGrip. Nudi nam grafični pregled, urejanje, kreiranje in brisanje tabel in zapisov. Poleg MySql ima vgrajene gonilnike za podporo sistemom kot so AWS Redshift, DB2, Derby, Exasol, H2, HSQLDB, Microsoft Azure, Oracle, PostgreSQL, SQL Server, Sqlite in Sybase [8].

2.5 BitBucket

Ker je projekt potekal v sklopu večjega sistema, je bila pomembna uporaba verzioniranja kode. To nam je omogočilo lažjo kolaboracijo med različnimi razvijalci, točen pregled nad tem, kdo je odgovoren za določene dele kode in enostavno spremljanje sprememb v projektu. Prav tako nam v primeru hudih napak v sistemu omogoča, da projekt enostavno postavimo nazaj na prejšnjo iteracijo, v kateri je vse še delovalo [28]. Za verzioniranje smo uporabili spletno storitev BitBucket, ki nam omogoča vse funkcionalnosti Gita.

Poglavje 3

Zajem zahtev

Pred začetkom samega razvijanja je bilo potrebno doreči, kakšne funkcionalnosti bi olajšale celoten potek opravljanja servisa in na kakšen način bi te storitve izboljšali na dolgi rok.

3.1 Kako bo potekalo delo serviserja

Za lažjo predstavitev dela smo si zamislili, kakšen je sploh potek dela serviserja z uporabo našega sistema. Ko se v nekem gostinskem obratu pokvari kos opreme, mora eden izmed zaposlenih servis naročiti. Naročanje servisa sicer ni del te diplomske naloge, vendar pa je razumevanje delovanja tega podsistema bistveno tudi za našo aplikacijo. Uporabnik spletnega portala izbere želeni termin servisa, serviserja oz. podjetje, ki nudi servis ter poda kratko obrazložitev problema, s katerim se sooča. Tu sedaj pride na vrsto mobilna aplikacija.

Takoj, ko je naročilo oddano, bo lahko serviser to videl na svojem mobilnem telefonu. Naročilo se mu bo pojavilo na spisku servisov kot novo, nepotrjeno naročilo. Če se bo strinjal z zahtevanim terminom, bo servis lahko potrdil takoj, sicer lahko izbere nov, ustreznejši termin in nato opravi potrditev. Tako naročnik na spletni platformi, kot tudi serviser sedaj vidita, kdaj se bo servis izvedel.

Po izvedenem servisu mora vsak serviser narediti še zapisnik. V zapisniku serviser opiše problem in potek popravila. V aplikaciji bo enostavno izbral naročilo ravnokar opravljenega servisa in izbral možnost za opravljanje zapisnika. V zapisnik bo podal vse potrebne informacije in ga zaključil. Na drugi strani ima nato naročnik na voljo možnost ocenjevanja opravljenih storitev. Preko spletne storitve potrdi, da je bil servis res opravljen in poda svoje mnenje o serviserju in njegovem delu.

3.2 Primeri uporabe

Ko smo definirali celoten potek, smo iz njega natančneje zajeli primere uporabe in dodali še nekatere dodatne funkcionalnosti, ki bi lahko uporabniku olajšale delo.

Pregled prihajajočih servisov - serviserju prikažemo vse prihajajoče servise (kje, kdaj).

Pregled lokalov - serviserju prikažemo vse lokale, za katere je zadolžen.

Pregled zgodovine - serviserju prikažemo zgodovino vseh opravljenih popravil.

Opravi zapisnik izrednega servisa - serviserju omogočimo možnost kreiranja. zapisnika za servise, ki so bili naročeni s strani lokala (izredni servis).

Opravi zapisnik rednega servisa - serviserju omogočimo možnost kreiranja. zapisnika za servise, ki jih opravi preventivno, brez predhodnega naročila (redni servis).

Ocena stanja lokala - serviserju omogočimo, da ob pisanju zapisnika poda oceno generalnega stanja lokala, čistoče lokala in vidnost znamke podjetja, s strani katerega je bil poslan serviser. Prav tako si lahko zabeleži nastavitev gramature kave v kavnih avtomatih.

Pregled naprav - pri vsakem lokalu se izpišejo podrobnosti o njegovih napravah.

Prikaz poti in lokacije lokala - za vsak lokal lahko zaženemo aplikacijo Google Maps, ki nam na podlagi naše lokacije prikaže pot do lokacije lokala.

Poglavje 4

Podatkovna baza

V podatkovno bazo je bilo potrebno dodati 4 dodatne tabele, 2 tabeli pa sta bili dodani že prej. To sta bili tabeli uporabnikov (users) in tabela gostinskih obratov (coffee_shops). Prva hrani podatke o posameznih uporabnikih kot so ime, priimek, e-naslov, zgoščeno geslo ter tip uporabnika, ki nam pove kakšno vlogo ima v samem sistemu (administrator, zaposlen v lokalu, vodja lokala, serviser, ...). Druga že dodana tabela hrani podatke o gostinskih obratih. To so ime, ulica, mesto, poštna številka, regija, država, davčna številka podjetja, prav tako hranimo tudi geografsko dolžino in širino (longitudo in latitudo) lokacije obrata.

Dodati je bilo nato potrebno tabelo servisov (services), ki bo hranila naročene servise. Hranila bo id lokala, v katerem se bo izvajalo popravilo, id serviserja, ki ga bo izvajal, opombo naročnika in polje, ki nam pove ali je servis reden ali izreden. Hranimo tudi 4 različne datume. Ob vnosu novega zapisa se najprej shrani datum njegovega kreiranja v tabeli. Drugi datum nam pove, kdaj naročnik želi imeti servis. Tretji datum nam pove, kdaj bo serviser dejansko prišel, četrti pa je datum opravljenega popravila. Na koncu hranimo še en booleanov atribut, ki nam pove , če je servis opravljen ali ne.

Naslednja tabela je tabela zapisnikov (repair_reports). Tabela bo povezana s tabela services preko atributa service_id, ki predstavlja identifikacijsko številko servisa. Poleg tega bo hranila še samo vsebino zapisnika, atribut, ki

nam bo povedal, ali je naročnik končan servis potrdil, ali ga je zavrnil kot neopravljenega/pomankljivega. Hranila bo tudi 6 različnih ocen. Tri ocene so podane s strani naročnika, ki bo ocenil ustreznost opravljenega servisa. Ostale 3 ocene bo ob zaključku zapisnika podal serviser o stanju lokala. Dva atributa bosta služila beleženju gramature kave v popravljenem aparatu (če gre za kavni aparat), zadnji atribut pa je agregacija ocen, ki služi spletni platformi in ni pomembna za mobilno aplikacijo.

Tabela naprav (shop_devices) hrani podatke o napravah posameznega lokala. Preko atributa shop_id je povezana s tabelo coffee_shop. Hrani še tip naprave, znamko naprave, model naprave in atribut, ki beleži ali je naprave še v uporabi ali ne. Hranimo še datum, ko smo ustvarili zapis.

Zadnja tabela (technician_coffee_shop) je namenjena povezovanju serviserjev z lokali, za katere so zadolženi. Služi kot vmesna tabela, saj je lahko en serviser zadolžen za več lokalov, en lokal pa seveda lahko koristi več serviserjev. Imamo t.i. povezavo mnogo proti mnogo. Tabela hrani id serviserja, ki je povezan s tabelo users, id lokala, ki je povezan s tabelo coffee_shop, atribut active, ki nam pove ali je serviser še vedno zadolžen za ta lokal in datum, ko je bil zapis ustvarjen.

Poglavje 5

Spletna storitev

V tem poglavju sledi opis implementacije spletne storitve. Mobilna aplikacija bo za delovanje potrebovala konstantno povezavo z internetom. Preko vmesnikov storitve bo odjemala in pošiljala potrebne podatke. Gre za spletno storitev REST, torej bo uporabljala zahteve protokola HTTP, podatke pa pošiljala v formatu JSON.

5.1 HTTP

HTTP je komunikacijski protokol, ki se uporablja za prenos podatkov po spletu [9]. Komunikacija poteka med odjemalcem in strežnikom. Odjemalec na strežnik pošlje HTTP zahtevo s poljubnimi podatki, ta pa mu nato odgovori glede na prejete podatke. Odgovor običajno zajema status in neko vsebino, če je bila ta uspešno zahtevana [10, 11].

Protokol uporablja več različnih zahtev. To so zahtevki GET, POST, PUT, HEAD, DELETE, TRACE, OPTIONS in CONNECT [12]. V sklopu projekta sta bila uporabljena le dva:

1. GET - zahtevek za pridobitev neke vsebine

2. POST - zahtevek za pošiljanje podatkov na spletni strežnik

5.2 REST

REST je arhitekturni stil za komunikacijo med napravami v internetu. Baziran je na protoklu HTTP in uporablja vse njegove zahtevke. Storitve, ki so v skladu z REST arhitekturo imenujemo RESTful storitve. Te zagotavljajo povezljivost različnih sistemov brez uporabniškega poseganja. Podatke navadno pošiljajo v formatu JSON, lahko pa tudi v drugih, kot npr. XML ali HTML [26].

5.3 JSON

Je tekstovni format, neodvisen od programskih jezikov, ki se uporablja za izmenjavo podatkov. Gre za preprost format v obliki ključ-vrednost in je lahko berljiv tudi človeku. Podpira ga velika večina modernih programskih jezikov. Prenaša lahko podatkovne tipe kot so števila, nizi, logične vrednosti in ničelo vrednost null. Kot omenjeno prej, podatke lahko hranimo v objektu, sestavljenemu iz ključev in vrednosti. Do posamezne vrednosti dostopamo tako, da se sklicujemo na ključ, pod katerim je shranjena. Format omogoča še strukturo seznama, v katerem do podatka dostopamo tako, da se sklicujemo na kazalec oz. pozicijo, kjer se ta nahaja [13, 14].

5.4 Arhitektura storitve

Spletna storitev bo razdeljena na dva dela in bo podobna arhitekturi MVC, le da bomo v našem primeru potrebovali samo model in krmilnik, ne pa tudi pogleda.

V modelu se bodo nahajale funkcije, ki bodo direktno upravljale s podatki iz podatkovne baze. Funkcije bodo izvajale poizvedbe, pripravljale in obdelovale podatke in jih nato posredovale krmilniku.

Krmilnik bo vseboval API funkcije. To so funkcije preko katerih mobilne naprave komunicirajo s strežnikom. Kličejo potrebne metode iz modela, od

njega prejmejo podatke in nato v primerni JSON obliki pošljejo napravi odziv, na podlagi katerega bo uporabniku podala zahtevan rezultat.

Spodnja funkcija prikazuje izvedbo SQL poizvedbe v ogrodju Codeigniter. Nahaja se v modelu, vrne pa nam zgodovino opravljenih servisov.

```
function getServiceHistory(){
  return $this->db->query(
    "SELECT s.id,
    s.id_coffee_shop,
    s.id_technician, s.done,
    DATE_FORMAT(s.reserved_date, '%d.%m.%Y, %H:%i') AS reserved_date,
    DATE_FORMAT(s.completed_date, '%d.%m.%Y, %H:%i') AS completed_date,
    s.created,
    cs.name AS coffee_shop_name
    FROM services AS s
    JOIN coffee_shops AS cs ON s.id_coffee_shop = cs.id
    WHERE s.done = 1 AND s.id_technician = ?
    ORDER BY s.completed_date DESC",[$this->user_id])->result();
}
```

V krmilniku imamo nato funkcijo z enakim imenom. Najprej preveri žeton uporabnika oz. naprave, ki je poslala zahtevo, v primeru, da je avtorizacija uspešna, pa vrne rezultat v formatu JSON.

```
public function getServiceHistory(){
    $this->verifyToken();
    echo json_encode($this->api_model->getServiceHistory());
}
```

Poglavje 6

Mobilna aplikacija

6.1 Arhitektura aplikacije

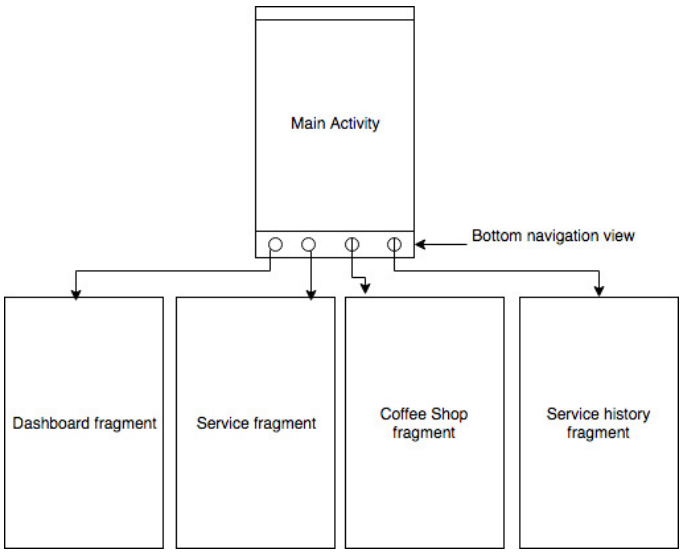
Pred samo izdelavo je bilo potrebno določiti, kako bo aplikacija sploh zasnovana. Po premisleku smo se odločili, da bo aplikacija zasnovana po podobnem principu kot nekateri že izdelani podsistemi, prisotnimi na mobilni platformi. V skladu z zahtevami smo se odločili, da bo aplikacija razdeljena na 4 dele. Prvi del bo začetna plošča, ki bo uporabniku služila kot nek osnovni pregled. Naslednji del bo namenjen pregledu prihajajočih servisov. Tu bo uporabnik imel podrobnejši pregled obveznosti in možnost ustvarjanja zapisnika. Sledi fragment za pregled lokalov, kjer se bo nahajal seznam vseh lokalov, za katere je serviser zadolžen. Tu bo lahko dobil bolj podrobne informacije o njegovih gostinskih obratih. Zadnji del bo namenjen pregledu zgodovine, kjer bo uporabnik lahko pogledal podrobnosti o vseh servisih, ki jih je opravil. Vse te štiri dele bomo lahko elegantno povezali v celovito aplikacijo z uporabo nekaterih priročnih Androidovih tehnologij in elementov, kot so fragmenti in meniji.

6.1.1 Android fragment

Fragment je bil v android predstavljen z namenom, da omogoči načrtovanje bolj dinamične in fleksibilne uporabniške izkušnje. Ena izmed prednosti, ki

jo bomo uporabili tudi v našem projektu je ta, da lahko več fragmentov teče znotraj ene same aktivnosti. Delujejo kot moduli znotraj aktivnosti, kar nam omogoča bolj organizirano in modularno implementacijo naših funkcionalnosti. Fragment vedno gostuje znotraj aktivnosti, njegov življenjski cikel pa je direktno povezan na cikel starševske aktivnosti. To pomeni, da če začasno ustavimo aktivnost, bo ustavljeno delovanje tudi vseh fragmentov znotraj nje. Če aktivnost uničimo, se uničijo tudi fragmenti. Med samim tekom aplikacije, pa je mogoče z vsakim fragmentom upravljati neodvisno od vseh ostalih delov aplikacije. Če recimo želimo fragment ustaviti, zamenjati z drugim ali pa ga uničiti, to lahko storimo brez, da bi vplivali na delovanje aktivnosti [2].

Naša aplikacija bo zato zasnovana tako, da bomo imeli eno osnovno aktivnost. Znotraj te aktivnosti bomo imeli postavljen Androidov element ViewPager. To je element, ki nam poenostavi izvajanje in preklapljanje večih fragmentov znotraj ene aktivnosti. Na dnu naše glavne aktivnosti bo postavljen meni, preko katerega bo lahko uporabnik med fragmenti preklapljal. Privzeto nam ViewPager omogoča tudi preklop med fragmenti tako, da uporabnik lahko po ekranu s prstom podrsa levo ali desno.



Slika 6.1: Grafični prikaz arhitekture aplikacije z uporabo fragmentov

6.2 API modul in Retrofit 2

Poseben modul znotraj aplikacije bo še modul za komunikacijo z internetom. Da lahko naša aplikacije komunicira s spletno storitvijo, ji moramo sprogramirati REST odjemalca. To sem naredil s pomočjo knjižnice Retrofit2. To je REST odjemalec za Android in Javo. Poenostavi nam komunikacijo in izmenjavo strukturiranih podatkov z REST spletnimi storitvami [27]. Za pravilno delovanje knjižnice je potrebna implementacija treh različnih razredov.

Prvi razred model, podatkovna struktura, ki je namenjena preslikavi iz formata JSON v javanski objekt s potrebnimi lastnostmi. Model, ki hrani podatke o napravi v lokalu izgleda takole:

```
public class Device {
    private int id;
    private String manufacturer, device, model;
    public int getId() { return id;}
    public String getManufacturer() {return manufacturer;}
    public String getDevice() {return device;}
    public String getModel() {return model;}
}
```

Drugi razred bo imenovan APIs. Naloga tega razreda je, da definira vse možne HTTP klice na našo spletno storitev. Kot primer klica lahko vzamemo HTTP zahtevo za naprave določenega lokala.

```
@GET("getDevices")
Call<ArrayList<Device>> getDevices(@Query("user_id") int user_id,
    @Query("shop_id") int shop_id);
```

Zadnji razred je razred RestClient. Uprablja naš APIs razred in pa Androidov Builder API, ki mu omogča definiranje končne točke (end point) do naše spletne storitve. Tukaj tudi definiramo način preslikave prejetih JSON podatkov v naš model. Za pretvorbo Retrofit uporablja Googlovo knjižnico Gson. Za lažje razhroščevanje je dodan še HttpLoggingInterceptor, ki nam omogoča beleženje in izpisovanje prejetih podatkov. V glavo vsake zahteve

moramo prav tako dodati žeton (token), s katerim nas spletna storitev avtorizira. Implementacija razreda izgleda takole:

```
public class RestClient {
    private static APIs REST_CLIENT;

    public RestClient(Context context, final String token) {
        String url = context.getString(R.string.api_url);
        HttpLoggingInterceptor interceptor = new HttpLoggingInterceptor();
        interceptor.setLevel(HttpLoggingInterceptor.Level.BODY);
        OkHttpClient.Builder builder = new
            OkHttpClient.Builder().addInterceptor(interceptor);
        if(!Utils.isEmpty(token)) {
            builder.addInterceptor(new Interceptor() {
                @Override public Response intercept(Chain chain) throws
                    IOException {
                    Request request =
                        chain.request().newBuilder().addHeader("Authorization",
                        token).build();
                    return chain.proceed(request);
                }
            });
        }
        Retrofit retrofit = new Retrofit.Builder().baseUrl(url)
                .addConverterFactory(GsonConverterFactory.create())
                .client(builder.build()).build();
        REST_CLIENT = retrofit.create(APIs.class);
    }

    public APIs get() {
        return REST_CLIENT;
    }

}
```

6.3 Vgrajevanje v obstoječ sistem

Aplikacijski del, namenjen serviserjem, je le eden izmed podsistemov znotraj aplikacije. Tako smo pri implementaciji morali paziti, da se je potek aplikacije izvedel pravilno, glede na to, kakšno vlogo ima uporabnik v sistemu.

Na vpisni aktivnosti v aplikaciji se na strežnik izvede klic, ki nam sporoči pravilnost kredenc. Če je bil vpis uspešen, dobimo ostalih podatkov o uporabniku še status o njegovi vlogi. Glede na ta status mora aplikacija zagnati pravilen podsistem. Če ta status pove, da je uporabnik serviser, se nato zažene glavna aktivnost namenjena njihovi vlogi.

6.4 Glavna aktivnost

Kot opisano v razdelku arhitekture bo aplikacije sestavljena iz ene aktivnosti in večih fragmentov. Glavna aktivnost bo tista, ki bo vsebovala element FragmentManager. To je androidov element, ki nam omogoča upravljanje s fragmenti. Vsebovala bo tudi spodnji meni, preko katerega bo uporabnik menjaval fragmente, in zgornjo orodno vrstico, ki bo prikazovala ime trenutnega fragmenta in omogočala izpis uporabnika. Opisani grafični vmesnik je prikazan na sliki.

Za prikazovanje fragmentov je bilo potrebno spisati poseben vmesnik za FragmentManager. Vmesnik je poimenovan RepairPagerAdapter in v osnovi vsebuje seznam vseh fragmentov, ki jih želimo prikazati. Preko metode getItem vračamo želeni fragment.

```
public class RepairPagerAdapter extends FragmentPagerAdapter {
    private ArrayList<Fragment> fragments = new ArrayList<>();
    public RepairPagerAdapter(FragmentManager fragmentManager) {
        super(fragmentManager);
        fragments.add(RepairmanNewDashboardFragment.newInstance());
        fragments.add(RepairmanServiceFragment.newInstance());
        fragments.add(RepairmanCoffeeShopsFragment.newInstance());
        fragments.add(RepairmanRecordFragment.newInstance());
    }
```

```
    @Override
    public Fragment getItem(int position) {return fragments.get(position);}
    @Override
    public int getCount() {return fragments.size();}
}
```

V razredu glavne aktivnosti nato dodamo še metodo prepareViewPager(). Ta metoda se bo klicala ob zagonu glavne aktivnosti in inicializirala vse fragmente in omogočala njihovo menjavo ob interakciji z menijem.

```
void prepareViewPager() {
        adapter = new RepairPagerAdapter(getSupportFragmentManager());
        viewPager.setAdapter(adapter);
        viewPager.addOnPageChangeListener(new
            ViewPager.OnPageChangeListener() {
            @Override
            public void onPageSelected(int position) {
                bottomNavigationView.getMenu().getItem(position).setChecked(true);
                if(actionBar != null) {
                    actionBar.setTitle(page_titles[position]);
                }
            }
        });
        viewPager.setOffscreenPageLimit(3);
    }
```

Ob klicu te metode najprej inicializiramo naš objekt tipa RepairPagerAdapter in ga nastavimo grafičnemu gradniku viewPager kot njegov vir podatkov. Nato mu nastavimo poslušalca addOnPageChangeListener in znotraj njega prepišemo metodo onPageSelected(). V tem metodi se nastavi pravilen izbor na spodnji orodni vrstici in zamenja naslov na zgornji naslovni vrstici. Na koncu s klicem metode setOffscreenPageLimit() povemo gradniku, da lahko naenkrat tečejo le trije fragmenti.

6.5 Začetna plošča

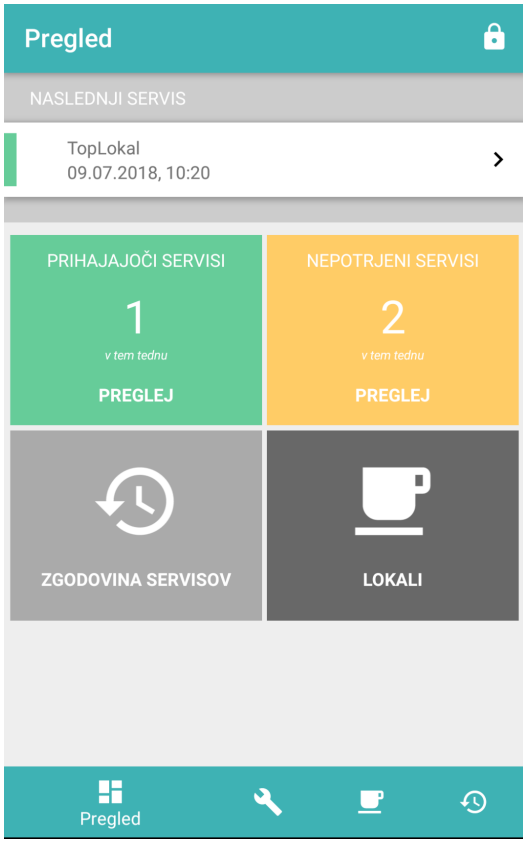
Prvi fragment, ki se bo pokazal bo začetna plošča oz. dashboard. Serviserju bo omogočala nek hiter pregled in osnoven pregled nad njegovim delom in funkcijam, ki mu jih nudi aplikacija. Najprej sem se lotil izdelave uporabniškega vmesnika. Na vrhu zaslona bodo uporabniku prikazani osnovni podatki najbližjega prihajajočega servisa. Videl bo ime lokala in datum servisa, s klikom na prikaz pa se bo odprla aktivnost za podrobnejši pregled servisa. Na sredini ekrana bodo nato štirje različni pogledi. Prvi pogled bo bo prikazal število prihajajočih popravil v tekočem tednu. Drugi pogled bo prikazal število vseh naročil za popravilo, jih serviser še ni potrdil. S klikom na katerega koli od teh dveh pogledov, nas aplikacije pelje na fragment pregleda servisov. Naslednja dva pogleda služita kot gumba, ki uporabnika popeljeta na aktivnosti za pregled zgodovine servisov ali pa na pregled vseh njegovih lokalov. Ta uporabniški vmesnik je bilo nato potrebno vključiti v fragment in mu razviti potrebno funkcionalnost. Za to bo poskrbel RepairmanDashboardFragment, ki bo primarno zadolžen za dve nalogi. Prva naloga bo, da bo s klicem na spletno storitev dobil potrebne podatke, in jih ustrezno prikazal na začetni plošči. Iz fragmenta bomo klicali API getRepairDashboard, ki nam bo vrnil objekt JSON. Ta objekt se bo preslikal v model RepairDashboard, ki hrani podatke o številu prihajajočih in nepotrjenih servisih. Prav tako hrani podatke o najbližjem prihajajočem servisu, shranjenemu kot objekt ServiceEntry. Podatke iz tega razreda nato prikažemo v ustreznih pogledih na grafičnem vmesniku.

```
public class RepairDashboard {
    private ServiceEntry upcoming_service;
    private int upcoming_service_num;
    private int unconfirmed_service_num;
    public ServiceEntry getUpcoming_service() {return upcoming_service;}
    public int getUpcoming_service_num() {return upcoming_service_num;}
    public int getUnconfirmed_service_num() {return unconfirmed_service_num;}
}
```

Druga naloga tega fragmenta bo, da uporabnika ob kliku na poglede pelje na ustrezne lokacije znotraj aplikacije.

```
@OnClick(R.id.upcoming_view)
public void repairButton1(){
    activity.goToPage(1);
}
```

Zgornji zgled prikazuje primer kode, ki se sproži ob kliku na pogled uporabniškega vmesnika.



Slika 6.2: Začetna plošča uporabnika

6.6 Pregled servisov

Fragment za pregled servisov bo moral prikazati prihajajoče servise uporabnika in velja kot glavna funkcionalnost aplikacije. Iz tega fragmenta bo uporabnik dostopal do podrobnosti posameznega servisa in možnosti za opravljanje zapisnika. Uporabniški vmesnik bo sestavljen iz dveh delov. Zgornji del bo zasedel horizontalni koledar, na katerem bodo izpisani dnevi trenutnega tedna. Spodnji del bo seznam, razdeljen na dneve, prihajajoči servisi pa bodo nato prikazani pod njegovim dnevom. Uporabnik bo z menjavo tedna na koledarju prav tako zamenjal pregled v seznamih, če pa bo na posamezni dan pritisnil, pa se bo seznam samodejno zadrsal na primeren datum.

Za upravljanje s fragmentom bo skrbel razred RepairmanServiceFragment. Njegovi primarni nalogi bosta komuniciranje z spletno storitvijo in povezovanje ter zaznavanje interakcij s seznamom in koledarjem. Najprej sem se lotil implementacije seznama.

Ko razvijamo aplikacije za sistem Android, se pogosto srečujemo s seznami. V osnovi je seznam razdeljen na tri dele. Najprej imamo pogled, ki služi kot posamezna celica oz. element seznama, ki bo prikazoval podatke zapisov. Nato imamo vir podatkov. To je običajno neka množica objektov, ki hrani podatke, ki jih želimo prikazati. Na koncu imamo še razred vmesnik oz. adapter. Ta služi kot most med podatki in pogledom. Zadolžen je za to, da iz vsakega objekta iz množice izlušči želene podatke, jih prenese na pogled in jih prikaže uporabniku na zaslonu preko Androidovih gradnikov kot sta ListView ali RecycleView. V tem vmesniku prav tako lahko zaznavamo interakcije s posameznimi zapisi, zapise brišemo, urejamo ali dodajamo.

Naš pogled bo preprosta celica, ki bo prikazala ime lokala, ulico lokala, datum popravila, tip popravila in status popravila(potrjen ali nepotrjen). Ob strani bo vsak zapis imel še barvni statusni trak. Če je datum servisa potrjen, bo trak obarvan z zeleno, nepotrjen bo obarvan z rumeno, če pa je serviser naročilo spregledal in ga ni pravočasno potrdil, pa se bo trak obarval z rdečo. Še en pogled bomo uporabili za prikaz imena dneva v glavi sekcije, po katero bodo našteti zapisi.

Množico podatkov za prikaz, bomo dobili iz spletne storitve. Podatki se bodo preslikali v javanski ArrayList objektov ServiceEntry. To je model, ki smo ga ustvarili za preslikavo iz objekta JSON. Ta seznam bomo podali vmesniku, ki jih bo preslikal na naš pogled.

Naš vmesnik se bo nekoliko razlikoval od standardnega Android vmesnika. Naš vmesnik mora omogočati, da znotraj seznama posamezne zapise razdelimo še na sekcije. V našem primeru so to dnevi v tednu. V osnovi nam Androidov vmesnik tega ne omogoča, zato sem si pomagal z odprto kodno knjižnico SectionedRecyclerViewAdapter. Ta knjižnica nam omogoča uporabo razširitve za standardni RecycleViewAdapter, s katerimi lahko razdelimo zapise v sekcije, jim dodamo glavo, nogo in animacije. Iz nje lahko uporabimo adapter tipa SectionedRecyclerViewAdapter, vanj pa dodamo našo razširitev DaysOfWeekServiceSection, ki deduje iz abstraktnega razreda StatelessSection.

6.6.1 Delovanje vmesnika SectionedRecyclerViewAdapter in sekcij DaysOfWeekServiceSection

Abstraktni razred StatelessSection nam omogoča implementacijo večih metod, s katerimi bomo oblikovali našo sekcijo. Vse potrebne podatke bomo razredu podali preko obveznega konstruktorja.

```
public DaysOfWeekServiceSection(String title, ArrayList<ServiceEntry> list,
    Context context) {
    super(new
        SectionParameters.Builder(R.layout.item_repair_service_emergency)
            .headerResourceId(R.layout.item_day_of_week_header)
            .build());
    this.title = title;
    this.entries = list;
    this.context = context;
}
```

V konstruktorju najprej določimo grafične elemente, ki jih bomo uporabili

za prikaz elementov in glavo sekcije. Nato iz parametrov dobimo še naslov sekcije, t.j. dan v tednu. Seznam vseh podatkov, jih želimo prikazati in aplikacijski kontekst, ki nam omogoča dostop raznih aplikacijskih resursov. Sledil je gnezden razred, ki vsebuje reference na grafične elemente našega pogleda celice.

```
class EmergencyServiceViewHolder extends RecyclerView.ViewHolder{
       @BindView(R.id.emergency_shop_name)
       TextView name;
       @BindView(R.id.emergency_location)
       TextView location;
       @BindView(R.id.emergency_date)
       TextView date;
       @BindView(R.id.emergency_status)
       TextView status;
       @BindView(R.id.status_image_view)
       ImageView status_image_view;
       @BindView(R.id.emergency_repair_item)
       LinearLayout item;
       EmergencyServiceViewHolder(View v) {
           super(v);
           ButterKnife.bind(this,v);
       }
   }
```

Preko tega razreda bomo dostopali do grafičnih elementov posamezne celice. Uporabili ga bomo v metodi onBindItemViewHolder(). V tej metodi se nahaja vsa logika, zadolžena za prikaz podatkov vsakega posameznega zapisa. Metoda za vsak posamezen zapis pregleda njegove podatke, nato pa na njihovi podlagi določi vsebino vseh tekstovnih grafičnih elementov in barve statusnih trakov.

```
@Override
public void onBindItemViewHolder(RecyclerView.ViewHolder holder, final int
    position) {
    EmergencyServiceViewHolder eHolder = (EmergencyServiceViewHolder) holder;
    eHolder.name.setText(getEntry(position).getCoffee_shop_name());
    eHolder.location.setText(
        getEntry(position).getStreet()+",
            "+getEntry(position).getPost_code()+"
            "+getEntry(position).getCity());
    if (getEntry(position).getConfirmed_date() != null){
        eHolder.date.setText(getEntry(position).getConfirmed_date());
        eHolder.status.setText(context.getString(
            R.string.repair_approved));
        eHolder.status.setTextColor(ResourcesCompat.getColor(
                context.getResources(), R.color.holo_green, null));
        eHolder.status_image_view.setImageResource(
        R.color.holo_green);
    } else {
        eHolder.date.setText(getEntry(position).getReserved_date());
        eHolder.status.setText(context.getString(
            R.string.repair_unapproved));
        eHolder.status.setTextColor(ResourcesCompat.getColor(
                context.getResources(), R.color.orange_yellow, null));
        eHolder.status_image_view.setImageResource
            (R.color.orange_yellow);
    }
    if (getEntry(position).getNum_of_days() > 0){
        eHolder.date.setTextColor(
                ResourcesCompat.getColor(context.getResources(),
                    R.color.red_6, null));
        eHolder.status_image_view.setImageResource(R.color.red_6);
    } else {
        eHolder.date.setTextColor(
                ResourcesCompat.getColor(context.getResources(),
                    R.color.defaultText, null));
    }
}
```

Zaključen vmesnik je bilo nato potrebno še implementirati in napolniti znotraj razreda fragmenta. Ob ustvarjenju fragmenta bomo iz storitve najprej dobili podatke in jih shranili, nato pa z metodo prepareSectionedRecycler() inicializirali objekt SectionedRecyclerViewAdapter in ga pripeli v naš RecyclerView.

```
void prepareSectionedRecycler(){
    recycler.setHasFixedSize(true);
    recycler.setLayoutManager(new
        LinearLayoutManagerWithSmoothScroller(activity));
    sectionAdapter = new SectionedRecyclerViewAdapter();
    recycler.setAdapter(sectionAdapter);
}
```

V pogled lahko novo sekcijo dodamo s klicem na njegovo metodo addSection(), kjer ServiceArray predstavlja ArrayList vseh servisov, ki se bodo izvajali na ponedeljek v izbranem tednu.

```
sectionAdapter.addSection("1",new DaysOfWeekServiceSection("Ponedeljek",
    ServiceArray, getContext()));
```

6.6.2 Implementacija horizontalnega koledarja

Za implementacijo horizontalnega koledarja si bom pomagal s knjižnico WeekCalender. Ima že narejen grafični vmesnik s vsemi potrebnimi vmesniki, ki jih potrebujemo. Koledar bomo najprej dodali v XML datoteko grafičnega vmesnika fragmenta in mu določili vse potrebne lastnosti.

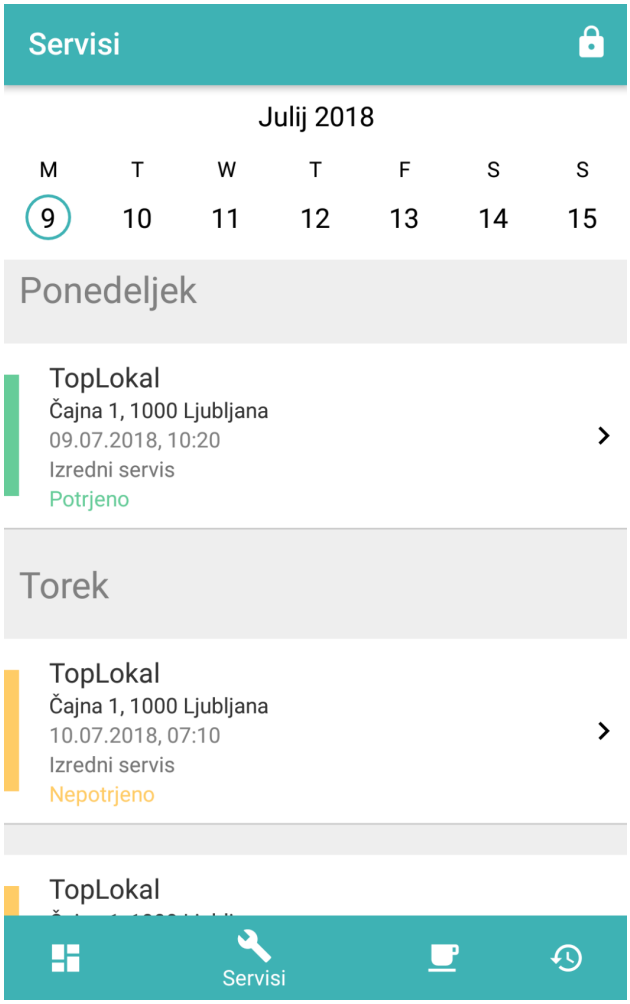
```
<noman.weekcalendar.WeekCalendar
            android:id="@+id/weekCalendar"
            android:layout_width="match_parent"
            android:layout_height="75dp"
            app:daysTextSize="18sp"
            app:weekTextSize="14sp"
            app:numOfPages="20"
            android:background="@color/white"
```

```
        app:daysBackgroundColor="@color/white"
        app:weekBackgroundColor="@color/white"
        app:weekTextColor="#000000"
        app:daysTextColor="#000000"
        app:selectedBgColor="@color/colorPrimary"
        app:todaysDateTextColor="@color/white"
        app:todaysDateBgColor="@color/colorPrimary"/>
```

Nato bomo v razredu fragmenta dodali še možnosti interakcije s koledarjem. Metoda setupWeeklyCalendar() se bo klicala ob kreiranju fragmenta

```
private void setupWeeklyCalendar(){
  weekCalendar.setOnWeekChangeListener(new OnWeekChangeListener() {
    @Override
    public void onWeekChange(DateTime firstDayOfTheWeek, boolean forward) {
    current_selected_date = firstDayOfTheWeek;
    month_name.setText(Utils.getMonthName(getContext(),firstDayOfTheWeek));
    getServices(firstDayOfTheWeek.getWeekOfWeekyear());
   }
});
```

V metodi nastavimo poslušalca OnWeekChangeListener, ki se sproži, kadar uporabnik zamenja teden na koledarju. Ob menjavi tedna si bomo najprej shranili prvi dan v tednu, nato bomo nad koledarjem nastavili ime meseca, na koncu pa bomo še opravili klic na našo spletno storitev, in uporabniku osvežili pogled z novimi servisi.



Slika 6.3: Fragment za pregled servisov

6.6.3 Potrditev in zapisnik servisa

Ob kliku na servis, se nam mora odpreti nova aktivnost, ki bo prikazala podrobnosti naročila, uporabniku podala možnost potrditve termina in dovolila uporabniku ustvariti zapisnik.

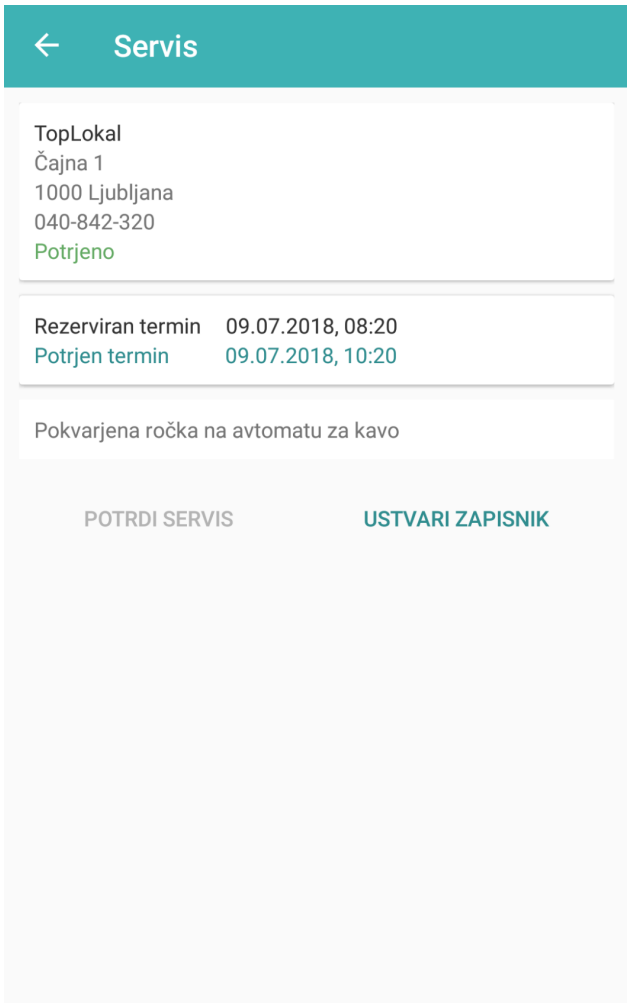
Uporabniški vmesnik bo razdeljen na tri dele. Najprej imamo osnovni pogled aktivnosti, na katerem bodo izpisane podrobnosti servise. Te podrobnosti bodo zajemale ime lokala, lokacijo lokala, kontaktno telefonsko številko odgovorne osebe v lokalu, termin, ki ga želi naročnik, in pa končni potrjeni

termin, ki ga določi serviser. Spodaj bomo imeli dva gumba, ki bosta vodila na ostala dva dela uporabniškega vmesnika te aktivnosti. Prvi gumb bo namenjen potrditvi termina. Odprl bo modalni dialog, na katerem se bo nahajal vmesnik za izbor datuma in ure. Če se serviser strinja z izbranim terminom lokala, bo tega samo potrdil, drugače si bo izbral svoj termin. Drugi gumb bo omogočen šele, ko je termin servisa potrjen. S pritiskom nanj bomo odprli drugi modalni dialog, na katerem bo postavljeno tekstovno polje, kamor bo serviser vnesel svoj zapisnik. Na koncu dialoga ima na voljo še možnost podaje ocene o stanju lokala. To bodo zvezdne ocene, ki bodo ocenjevale generalno stanje lokala, čistočo in skrb za opremo in vidnost znamke podjetja, s katerega prihaja serviser. V primeru, da je serviser popravljal kavni aparat, lahko zabeleži tudi, če je popravil gramaturo kave, ki jo naprava uporablja. Zabeleži koliko gramov kave je naprava uporabljala pred servisom in na koliko je to količino popravil.

Da se aktivnost odpre, smo najprej na posamezen element zapisa v fragmentu morali implementirati poslušalca, ki je zaznal na kateri servis je uporabnik pritisnil. Ob pritisku se sproži sproži funkcija, ki odpre novo aktivnost, imenovano RepairServiceActivity. Iz fragmenta se v aktivnost prenesejo vsi potrebni podatki, ki se nato izpišejo v primernih pogledih.

```
emergencyHolder.item.setOnClickListener(new View.OnClickListener() {
  @Override
    public void onClick(View v) {
    Intent intent = new Intent(context, RepairServiceActivity.class);
    Bundle bundle = new Bundle();
    bundle.putInt("id",getEntry(position).getId());
    bundle.putString("name",getEntry(position).getCoffee_shop_name());
    bundle.putString("street",getEntry(position).getStreet());
    bundle.putString("post",getEntry(position).getPost_code()+"
        "+getEntry(position).getCity());
    context.startActivity(intent);
  }
});
```

Podatki se med aktivnostmi prenašajo preko objekta Bundle. Gre za podatkovno strukturo, ki prenaša različne podatkovne tipe v obliki ključ-vrednost.



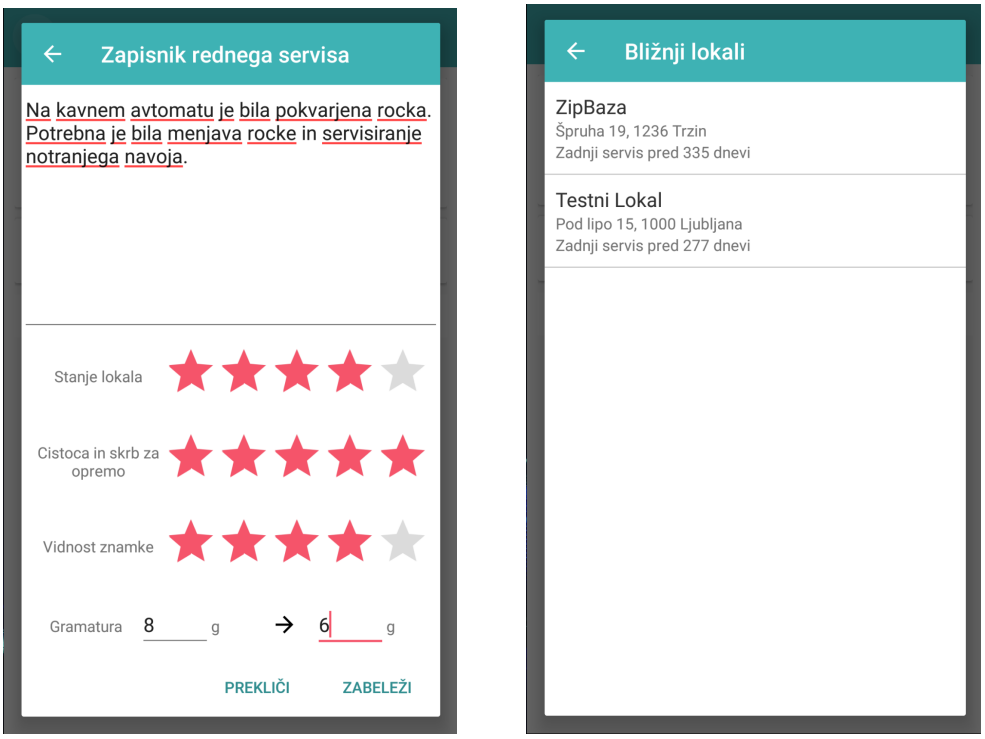
Slika 6.4: Aktivnost za podrobnejši pregled servisa

Sledila je implementacija modalnega dialoga za izbiro in potrditev termina. S klikom na gumb potrdi bomo tudi dialogu preko objekta poslali identifikacijsko številko servisa, in datum termina, ki ga želi naročnik. Koledar, ki bo postavljen na dialogu bomo nastavili na podan termin. Če se bo serviser s terminom strinjal, ga bo lahko potrdil takoj, drugače lahko s koledarjem zamenja termin in ga nato potrdi. Ob potrditvi bo aktivnost

opravila klic na našo spletno storitev, in poslala potrebne podatke za potrditev termina servisa.

Ko je servis uspešno potrjen, omogočimo gumb za ustvarjanje zapisnika. Odpre nam modalni dialog, ki mu moramo prenesti le identifikacijsko številko. V vnosno polje lahko nato uporabnik vpiše vse potrebne podrobnosti o popravilu, ki ga je ravnokar opravil. Pod vnosnim poljem sem moral implementirati še zvezdno ocenjevanje. Za to sem uporabil Androidov element RatingBar. Za vsak ta element je bilo potrebno implementirati poslušalca, ki ob spremembi ocene njegovo vrednost spravi v spremenljivko. Ocene so celoštevilčne od 1 do 5, kjer 1 predstavlja najslabšo oceno, 5 pa najboljšo. Nazadnje je tu še beleženje gramature kave, če je serviser bil zadolžen za popravilo kavnega aparata. Ti aparati imajo predpisano količino kave, ki je najbolj optimalna za delovanje aparata. Gostinci pogostokrat te količine ne upoštevajo, kar lahko privede do okvar. Če serviser ugotovi, da gramatura kave ni bila primerna, si zabeleži kakšna je ta bila, in na koliko jo je popravil. Ko so vsi podatki vnešeni, se s potrditvijo pokliče ustrezna metoda spletne storitve, ki podatke v zapisniku ustrezno razčleni, in jih shrani v podatkovno bazo.

Po potrditvi, da je bil zapisnik ustrezno shranjen, se nato sproži nov dialog za priporočanje lokalov. Namen tega dialoga je, da uporabniku prikaže bližnje lokale, ki že nekaj časa niso imeli opravljene servisa. Takoj, ko je servis potrjen bo aplikacija na strežnik poslala še en klic. Poslala mu bo koordinate lokala, v katerem je serviser pravkar opravil servis. Na podlagi teh koordinat, strežnik preračuna razdaljo do ostalih serviserjevih lokalov. Za vsak lokal v bližini 20km pogleda, kdaj je bil servis opravljen nazadnje. Nato te podatke aplikaciji, ki jih izpiše uporabnik. S klikom na lokal, se nam odpre aktivnost za podroben pregled lokala.



(a) (b)

Slika 6.5: Modalni dialog za ustvarjanje zapisnika (a) in dialog za priporočanje lokalov (b)

6.7 Pregled lokalov

6.7.1 Pregled in iskanje

Na fragmentu za pregled lokalov bo uporabnik imel izpisane vse gostinske obrate za katere je zadolžen. Tako kot pri prejšnjem fragmentu, bomo tudi tu prikazali seznam zapisov z nekaj osnovnimi podatki. Ker ima lahko en serviser potencialno tudi nekaj sto lokalov, bomo tu dodali še polje za iskanje, ki bo ob vnosu sproti filtriralo vse zapise. Tako bomo uporabniku olajšali iskanje želenega lokala.

Fragment bomo implementirali preko razreda imenovanega RepairmanCoffeeShopsFragment. Razred je zadolžen za komuniciranje s spletno sto-

ritvijo. Iz nje bo vrnil vse zapise, nato pa preko RecycleView adapterja napolnil naš prikaz. Tokrat lahko razširimo standardni androidov RecycleView adapter in ne potrebujemo nobenih drugih knjižnic. Razen standardnih metod kot so onBindViewHolder(), je bilo potrebno v adapter dodati še nekaj kode, ki bo služila iskanju lokalov.

```
public void filter(String query){
    filterByName(query);
    notifyDataSetChanged();
}

private void filterByName(String text) {
    coffeeShops.clear();
    if(text.isEmpty()){
        coffeeShops.addAll(coffeeShopsCopy);
    } else{
        text = text.toLowerCase();
        text = text.replace('[U+FFFD]','z');
        text = text.replace('[U+FFFD]','s');
        text = text.replace('[U+FFFD]','c');
        for(CoffeeShop item: coffeeShopsCopy){
        String title = item.getName().toLowerCase()
        .replace('[U+FFFD]','z')
        .replace('[U+FFFD]','s')
        .replace('[U+FFFD]','c');
        if(title.toLowerCase().contains(text)){
            coffeeShops.add(item);
                }
            }
        }
}
```

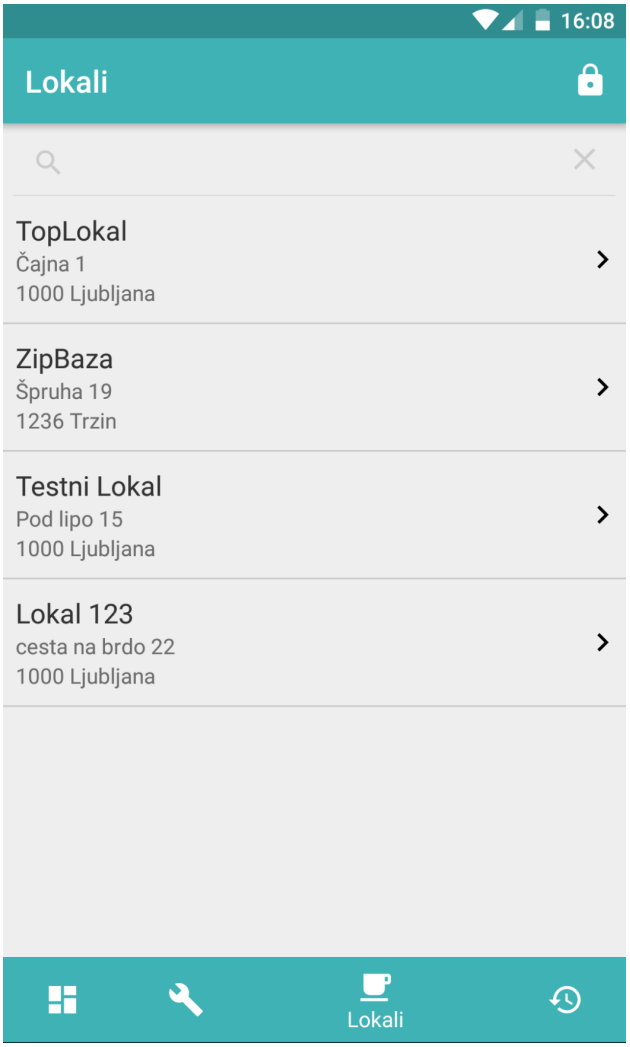
Gornji dve metodi bosta skrbeli za filtriranje. Metoda filterByName prejme kot argument nek tekstovni niz, ki ga uporabnik vnese. V adapterju imamo dva javanska ArrayLista. Prvi je imenovan coffeeShops. Ta hrani zapise, ki jih adapter prikazuje na zaslonu. Drugi je coffeeShopsCopy, ki vedno

hrani vse zapise. Iz njega jemljemo tiste elemente, ki ustrezajo iskanemu nizu, in jih pripenjamo v coffeeShops za prikaz. Metoda bo najprej spraznila seznam za prikaz, iz iskanega niza in imen elementov odstranila šumnike, vse velike črke zamenjala z malimi, in nato primerjala elemente in iskani niz. Če se element ujema z nizom, ga pripne nazaj v glavni seznam. Ta metoda je privatna in je namenjena delovanju znotraj razreda. Do nje dostopamo preko javne metode filter. To metodo bomo klicali v fragmentu, ko bo uporabnik zahteval iskanje. Ta metoda pokliče metodo za filtriranje, nato pa s klicem metode notifyDataSetChenged() na novo naloži celotno vsebino v adapterju, in tako osveži pogled uporabniku.

Da bo iskanje delovalo popolnoma, je potrebno klice za filtriranje povezati z iskalno vrstico na fragmentu. uporabniškemu vmesniku na vrhu dodamo iskalno vrstico SearchView. Nato v naš razred fragmenta implementiramo razširitev SearchView.OnQueryTextListener. To nam omogoči, da razred naši iskalni vrstici podamo kot poslušalca, vanj pa implementiramo metodo onQueryTextChange().

```
@Override
public boolean onQueryTextChange(String newText) {
    adapter.filter(newText);
    return true;
}
```

Vsakič, ko uporabnik vnese neko črko v iskalno vrstico, bo poslušalec sprožil to metodo, ki bo klicala naše filtrirne operacije v adapterju. Tako bomo omogočili, da bo uporabnik ob iskanju vedno imel ažuren pregled.



Slika 6.6: Fragment za pregled gostinskih obratov

6.7.2 Podroben pregled

Ob kliku na zapis v fragmentu se bo uporabniku odprla aktivnost za podrobne informacije o lokalu. Aktivnost bo imenovana CoffeeShopActivity, uporabniku pa bo prikazala osnovne podatke o lokalu, seznam vseh prisotnih naprav v lokalu, seznam vseh prihajajočih izrednih servisov, podala možnost, da opravi redni servis in odpre storitev GoogleMaps, ki bo uporabniku začrtala pot do lokala.

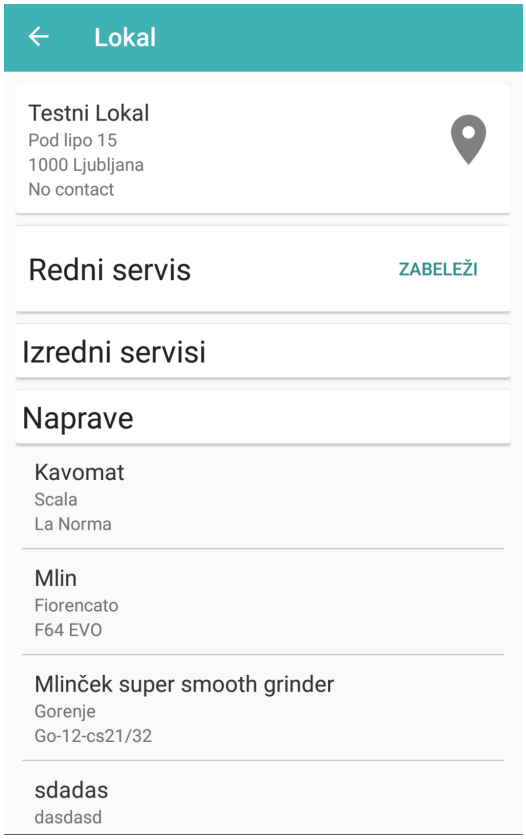
V fragmentu bomo na posamezen element v prikazu seznama najprej dodali poslušalca, ki ob kliku odpre aktivnost in ji preko objekta Bundle prenese osnovne podatke o lokalu. Aktivnost ob kreaciji tudi naredi dva klica na našo spletno storitev. Prvi klic nam bo napolnil prikaz izrednih servisov, drugi klic pa prikaz naprav. Dodamo še razdelek za redne servise. Vanj damo gumb, ki bo odprl modalni dialog, podoben tistemu, ki smo ga uporabili pri izdelavi zapisniki za naročene izredne servise. Funkcionalnost dialoga je identična, le da moramo tu spletni storitvi sporočiti, da ne gre za naročen izredni servis ampak za rednega.

Na koncu je sledila še implementacija storitve GoogleMaps. Na aktivnost smo dodali ikono, katero lahko uporabnik pritisne, če želi, da se mi prikaže pot od njegove trenutne lokacije do lokala, ki ga je izbral.

```
@OnClick(R.id.shop_location_image_view)
public void openMap(){openGoogleMaps(this.latitude,this.longitude);}

private void openGoogleMaps(double latitude, double longitude){
    String uri = "https://www.google.com/maps/dir/?api=1&destination=
        "+Double.toString(latitude)+","+Double.toString(longitude);
    Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(uri));
    startActivity(intent);
}
```

Ob kliku na ikono se sproži metoda openGoogleMaps(). Kot argumenta ji podamo latitudo in longitudo trenutnega lokala. Metoda bo sestavila url naslov, nato pa zagnala aktivnost. Ker android prepozna shemo url naslova, bo takoj vedel, da mora zagnati aplikacijo GoogleMaps, če je ta nameščena na uporabnikovi napravi. Če je uporabnik nima, odpre naslov v spletnem brskalniku.



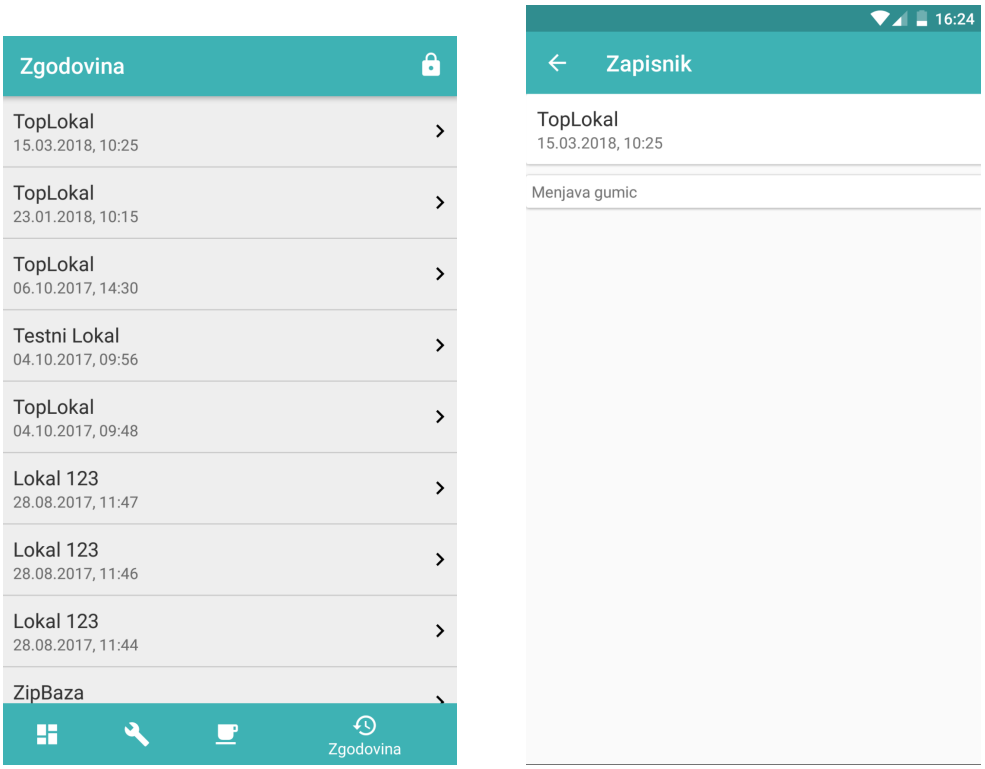
Slika 6.7: Aktivnost za podroben pregled lokala

6.8 Pregled zgodovine

Zadnji fragment bo fragment za pregled zgodovine servisov. Je najpreprostejši fragment v aplikaciji. Fragment bo zajemal en seznam zapisov, na katerih bodo izpisani ime lokala, datum zapisnika in ura zapisnika. Ob kliku na ikono, se bo odprla nova aktivnost, na kateri bodo na vrhu izpisani prej omenjeni podatki, spodaj pa bo prikazano besedilo zapisnika, ki ga je napisal serviser po opravljenem servisu.

Podobno prejšnjim fragmentom smo najprej ustvarili uporabniški vmesnik z gradnikom RecyclerView. Za ta pogled smo ustvarili potrebne celice, nato pa spisali adapter, ki je prejel ArrayList zapisov o zgodovini, in jih ustrezno prikazal na zaslonu. V tem fragmentu nismo potrebovali nobenih

dodatnih funkcionalnosti kot npr. sekcioniranje ali iskanje, zato je bila implementacija osnovna. Vsem elementom smo dodali primerne poslušalce, ki so ob uporabniški interakciji odprle novo aktivnost za podrobnejši pregled.



(a) (b)

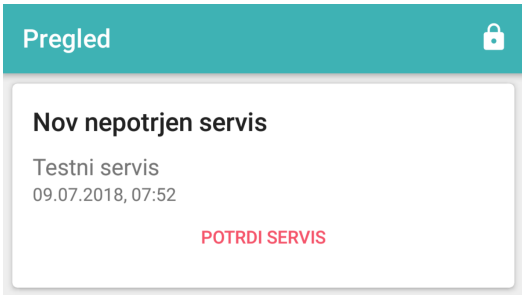
Slika 6.8: Fragment pregleda zgodovine zapisnikov in podroben pregled posameznega zapisa

6.9 Ocenjevanje storitev s strani naročnika

Kot omenjeno v poglavju 6.3, je serviserjev del aplikacije le podsistem v veliko večjem sistemu. Tako kot serviserji, imajo tudi zaposleni v gostinskih obratih v uporabi svoj del aplikacije.

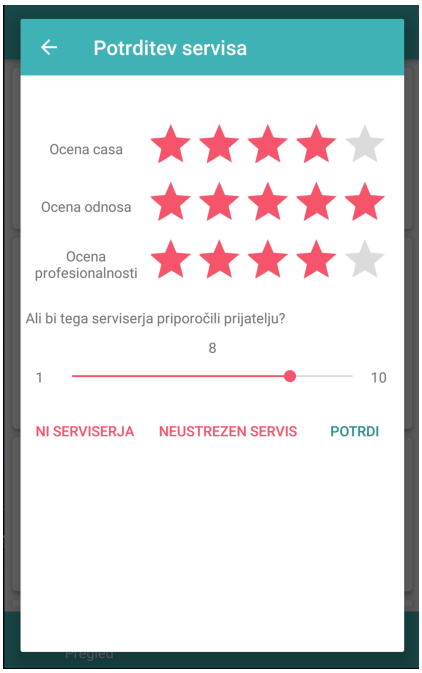
Na podoben način kot pri našem podsistemu, ima tudi tu uporabnik začetno ploščo, na kateri lahko vidi osnovne podatke o svojih obveznostih.

Odločili smo se, da bomo preko te začetne plošče uporabnika obvestili o novem opravljenem servisu. Ko bo serviser zaključil in potrdil svoj zapisnik, se bo uporabniku ob naslednjem obisku začetne plošče na vrhu prikazalo opozorilno okno. V tem oknu bo izpisano ime podjetja, ki je serviserja poslalo ter datum in uro opravljene storitve.



Slika 6.9: Obvestilno okno na začetni plošči natakarja

Ob pritisku na gumb za potrditev bomo uporabniku prikazali modalno okno za podajanje ocene opravljene servisne storitve. Uporabnik bo z zvezdnimi ocenami ocenil hitrost servisa, splošen odnos serviserja in profesionalnost opravljenega dela. Spodaj podamo še drsnik z ocenami od ena do pet, s katerim ocenimo ali bi serviserja priporočili še drugim gostincem. Na koncu dialoga so trije gumbi. S prvim gumbom lahko sporočimo, da serviserja sploh ni bilo in je zapisnik servisa ni bil narejen pravilno. Z drugim gumbom lahko sporočimo, da je bil servis sicer opravljen, vendar neustrezno. Zadnji gumb nam potrdi servis in pošlje vse ocene na strežnik.



Slika 6.10: Modalni dialog za ocenjevanje opravljene servisne storitve

Poglavje 7

Testiranje

Po vsaki implementaciji obsežnejše funkcionalnosti sistema je bilo njeno delovanje potrebno testirati in odpraviti vse morebitne tehnične napake.

7.1 Testiranje spletne storitve

Pri testiranju delovanja spletne storitve smo preverjali delovanje APIjev. S pomočjo orodja Postman, smo na vsako končno točko izvajali klice z različnimi podatki. Opazovali smo pravilnosti odzivov strežnika, pravilnost shranjevanja podatkov v podatkovno bazo in bili pozorni na porabo strežniških resursov.

7.2 Testiranje mobilne aplikacije

Pri mobilni aplikaciji se je testiranje izvedlo v dveh fazah. Tekom razvoja smo sproti vse funkcionalnosti testirali na emulatorjih s pomočjo programa Genymotion. Aplikacijo smo izvajali na emulirani napravi Nexus 5X, z operacijskim sistemom Android Marshmallow (6.0.0). Če so vsi sistemi znotraj aplikacije delovali v skladu z zahtevami, smo nato aplikacijo namestili na več različnih fizičnih naprav. Te naprave so bile različnih znamk, imele različne verzije operacijskega sistema in imela različne velikosti in resolucije ekranov.

Če se je testirana funkcionalnost pravilno izvajala na vseh napravah, je bil testiran del zaključen.

Poglavje 8

Sklepne ugotovitve

V okviru diplomske naloge je bil razvit sistem za optimizacijo servisnih storitev v gostinskih obratih. Sistem omogoča serviserju lažji pregled nad naročenimi in opravljenimi servisi, pregled podatkov o obratih, za katere je zadolžen, in mu omogoča pisanje in shranjevanje zapisnikov o opravljenem delu. Na drugi strani sistem omogoča zaposlenim v obratu, da podajo odziv na kvaliteto opravljenih storitev. S tem zagotovimo ponudnikom storitev določeno transparentnost opravljenega dela.

Za dosego zadanih ciljev smo najprej razvili zaledni strežniški sistem, ki je zajemal podatkovno bazo in spletno storitev.V podatkovni bazi, razviti s pomočjo sistema za upravljanje s podatkovnimi bazami MySql, smo hranili podatke o uporabnikih in njihovem delu, spletna storitev pa je bila namenjena komunikaciji z mobilno aplikacijo. To smo razvili s pomočjo ogrodja CodeIgniter v jeziku PHP.

Mobilna aplikacije je predstavljala osrednji del tega projekta. Interakcija uporabnika s sistemom v celoti poteka preko njegove mobilne naprave. Razvita je bila za operacijski sistem Android, preko Googlovega razvojnega okolja Android studio, v programskem jeziku Java. Razdeljena je bila na dva modula, in sicer na modul za serviserje in modul za gostince. Vsak od njiju zajema potrebne zgoraj opisane funkcionalnosti.

Prednosti aplikacije so poenostavljen dostop do vseh potrebnih informacij

za opravljanje dela, priročen pregled že opravljenega dela, nudenje transparentnosti delovanja ponudnikom servisnih storitev in poenostavljeno podajanje odziva gostincev na opravljene storitve.

Predpostavljene omejitve aplikacije bi se lahko pojavile pri usposabljanju uporabnikov. Lahko pridemo v situacijo, kjer uporabnik ne bo imel dovolj tehničnega znanja oz. tehnološke pismenosti, da bi se mu uporaba aplikacije zdela preprostejša. Prav tako je problem sistema Android v tem, da je raznolik glede na proizvajalca naprav, kar lahko povzroči nekonsistentno delovanje na različnih napravah.

Nekaj prostora imamo tudi za morebitne izboljšave. Odvisne bi bile predvsem od odzivov, ko bo aplikacija enkrat prišla v testno fazo z uporabniki. V prvi vrsti bi lahko prišlo do izboljšav pri uporabniškem vmesniku in izkušnji. Aplikacija bi na strani gostinca lahko nudila tudi možnost naročanja servisov, kar je trenutno izvedljivo le preko spletne platforme. Tako bi lahko posamezni zaposleni v trenutku, ko se neka naprava pokvari, naročil ustrezno popravilo.

Na koncu smo ocenili, da je bil projekt v prvi fazi izvedeni uspešno. Ustrezno smo zadostili vsem zajetim funkcionalnim zahtevam, testiranje pa je do sedaj pokazalo, da je delovanje sistema konsistentno in pravilno.

Literatura

[1] Android (operating system). Dosegljivo: https://en.wikipedia.org/wiki/Android_(operating_system). [Dostopano: Junij 2018].

[2] Android fragments. Dosegljivo: https://developer.android.com/guide/components/fragments. [Dostopano: Junij 2018].

[3] Android (operacijski sistem). Dosegljivo: https://sl.wikipedia.org/wiki/Android_(operacijski_sistem). [Dostopano: Junij 2018].

[4] Android studio. Dosegljivo: https://developer.android.com/studio/. [Dostopano: Junij 2018].

[5] Android studio. Dosegljivo: https://en.wikipedia.org/wiki/Android_Studio. [Dostopano: Junij 2018].

[6] Android studio: An ide built for android. Dosegljivo: https://android-developers.googleblog.com/2013/05/android-studio-ide-built-for-android.html. [Dostopano: Junij 2018].

[7] Codeigniter user guide. Dosegljivo: https://www.codeigniter.com/user_guide/. [Dostopano: Junij 2018].

[8] Datagrip features. Dosegljivo: https://www.jetbrains.com/datagrip/features/. [Dostopano: Junij 2018].

[9] Hypertext transfer protocol – http/1.1 - introduction. Dosegljivo: https://tools.ietf.org/html/rfc2616#page-7. [Dostopano: Junij 2018].

[10] Hypertext transfer protocol – http/1.1 - request. Dosegljivo: https://tools.ietf.org/html/rfc2616#page-35. [Dostopano: Junij 2018].

[11] Hypertext transfer protocol – http/1.1 - response. Dosegljivo: https://tools.ietf.org/html/rfc2616#page-39. [Dostopano: Junij 2018].

[12] Hypertext transfer protocol – http/1.1 - method definitions. Dosegljivo: https://tools.ietf.org/html/rfc2616#page-51. [Dostopano: Junij 2018].

[13] Uvod v json. Dosegljivo: https://www.json.org/json-sl.html. [Dostopano: Junij 2018].

[14] Working with json. Dosegljivo: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON. [Dostopano: Junij 2018].

[15] Chris Muller and Robert H. Woods. The real failure rate of restaurants, hospitality review. *FIU, Hospitality Review Journal*, Vol. 9 : Iss. 2 , Article 7, 1991.

[16] 8 advantages of using mysql. Dosegljivo: https://devops.com/8-advantages-using-mysql/. [Dostopano: Junij 2018].

[17] Mysql. Dosegljivo: https://en.wikipedia.org/wiki/MySQL. [Dostopano: Junij 2018].

[18] Mysql. Dosegljivo: https://sl.wikipedia.org/wiki/MySQL. [Dostopano: Junij 2018].

[19] Michael C Ottenbacher. Innovation management in the hospitality industry: different strategies for achieving success. *Journal of hospitality & tourism research*, 31(4):431–454, 2007.

[20] HG Parsa, John T Self, David Njite, and Tiffany King. Why restaurants fail. *Cornell Hotel and Restaurant Administration Quarterly*, 46(3):304–322, 2005.

[21] What is php? Dosegljivo: http://php.net/manual/en/intro-whatis.php. [Dostopano: Junij 2018].

[22] What can php do? Dosegljivo: http://php.net/manual/en/intro-whatcando.php. [Dostopano: Junij 2018].

[23] 12 'must-know"advantages of php. Dosegljivo: https://www.vandelaydesign.com/advantages-of-php/. [Dostopano: Junij 2018].

[24] Phpsotrm features. Dosegljivo: https://www.jetbrains.com/phpstorm/features/. [Dostopano: Junij 2018].

[25] Traditional relational database solutions. Dosegljivo: https://docs.microsoft.com/en-us/azure/architecture/data-guide/relational-data/. [Dostopano: Junij 2018].

[26] Relationship to the world wide web and rest architectures. Dosegljivo: https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest. [Dostopano: Junij 2018].

[27] Retrofit, a type-safe http client for android and java. Dosegljivo: http://square.github.io/retrofit/. [Dostopano: Junij 2018].

[28] What is version control. Dosegljivo: https://www.atlassian.com/git/tutorials/what-is-version-control. [Dostopano: Junij 2018].