Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matej Jakop

Sistem za avtomatizacijo upravljanja oglasnega prostora

diplomsko delo
na univerzitetnem študiju

Mentor: doc. dr. Peter Peer

Ljubljana, 2012

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01831/2012
Datum: 03.04.2012

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MATEJ JAKOP**

Naslov: **SISTEM ZA AVTOMATIZACIJO UPRAVLJANJA OGLASNEGA PROSTORA**

SYSTEM FOR AUTOMATION OF BILLBOARD MANAGEMENT

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

V uvodu analizirajte potek dela upravljalcev plakatnih mest ter obstoječe sisteme za upravljanje oglasnega prostora. Nato opišite algoritme in koncepte, ki so potrebni v takšnem sistemu. Na podlagi algoritmov za avtomatsko področno dodeljevanje delovnih nalogov, za iskanje optimalne poti obiskovanja mest ter za iskanje idealne razporeditve zasedenosti plakatnih mest, zasnujte in implemenirajte sistem za avtomatizacijo upravljanja oglasnega prostora s pomočjo mobilne aplikacije. Sistem na koncu testirajte in poročajte o učinkovitosti algoritmov.

Mentor:
doc. dr. Peter Peer

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:
prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Matej Jakop, z vpisno številko **63070025**, sem avtor diplomskega dela z naslovom:

Sistem za avtomatizacijo upravljanja oglasnega prostora

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Peter Peer,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 15. septembra 2012 Podpis avtorja:

Zahvaljujem se staršem in vsem, ki so mi tekom študija stali ob strani s svojimi nasveti in predlogi. Hvala doc. dr. Peter Peer za usmerjanje in pomoč pri pripravi diplomskega dela. Nia, hvala za spodbujanje, podporo in strpnost.

Vsem najdražjim.

Kazalo

Seznam uporabljenih kratic in simbolov

AJAX - Asynchronous JavaScript and XML; asinhroni JavaScript in XML
API - Application Programming Interface; programski vmesnik
ARM - Advanced RISC Machine; napredni RISC stroj
CDMA - Code Division Multiple Access; metoda dostopa do kanala pri radijskih komunikacijskih tehnologijah
CRM - Customer Relationship Management; upravljanje odnosov s strankami
DRY - Don't repeat yourself; ne ponavljaj se
EDGE - Enhanced Data rates for Global Evolution; izboljšane vrednosti hitrosti prenosa za globalni napredek
EV-DO - Evolution-Data Optimized; telekomunikacijski stardard za brezžični prenos podatkov preko radijskih signalov, tipično v uporabi za širokopasosvni dostop do interneta
GPS - Global Positioning System; sistem za pozicioniranje
GSM - Global System for Mobile communications; sistem mobilne telefonije
HTTP - HyperText Transfer Protocol; protokol za izmenjavo nadbesedil ter grafičnih, zvočnih in drugih večpredstavnostnih vsebin na spletu
IMAP - Internet Message Access Protocol; standard za sprejemanje e-pošte
JSON - JavaScript Object Notation; oblika zapisa objektov v programskem jeziku JavaScript
LTE - Long Term Evolution; standard za brezžično komunikacijo velikih hi-

trosti za mobilne telefone in terminale

MVC - Model–View–Controller; model-pogled-kontroler

NFC - Near Field Communication; sistem za bližnjo komunikacijo

ORDBMS - Object-Relational DataBase Management System

P2P - Peer-to-peer; vsak z vsakim

POP3 - Post Office Protocol version 3; standard za prejemanje e-pošte

REST - REpresentational State Transfer; stil programske arhitekture za distribuirane sistem

RFID - Radio-frequency identification; radijska identifikacija

RISC - Reduced Instruction Set Computer; računalnik s procesorjem, ki uporablja majhno število preprostih ukazov

SCGI - Simple Common Gateway Interface; vmesnik, ki določa, kako program komunicira z drugim programom

SMTP - Simple Mail Transfer Protocol; standard za pošiljanje e-pošte

SOAP - Simple Object Access Protocol; standard za spletne storitve, ki temelji na XML

SSL - Secure Sockets Layer; sloj varnih vtičnic

TCP/IP - Transmission Control Protocol/Internet Protocol; standardiziran sklad protokolov, na katerem temelji internet

TSP - Travelling Salesman Problem; problem trgovskega potnika

UMTS - Universal Mobile Telecommunications System; univerzalni sistem za mobilno komunikacijo

WiMAX - Worldwide Interoperability for Microwave Access; tehnologija namenjena brezžičnemu širopasovnemu prenosu podatkov

XML - Extensible Markup Language; razširljivi označevalni jezik

Povzetek

Sodobni svet nas na skoraj vsakem koraku bombardira z oglasnimi sporočili. Naj si bo to doma pred televizorjem, med brskanjem po spletu ali med sprehodom po mestu. Oglasna sporočila se nahajajo povsod. Vedno znova prihajajo nova, v različnih oblikah, še bolj drzna in prepričljivejša. Od kod prihajajo? Kdo skrbi za njihovo namestitev? Kako poteka nadomeščanje starih oglasov z novimi? Je možno proces dodatno računalniško podpreti in ga delno tudi avtomatizirati? Na ta in podobna vprašanja smo poskušali odgovoriti v diplomskem delu, v katerem smo zasnovali sistem za podporo poslovanja upravljalcev plakatnih mest, ki za opravljanje vseh ključnih nalog uporablja številne algoritme. Uporabili smo geometrijski algoritem za avtomatsko dodeljevanje delovnih nalogov posameznemu plakaterju, razvili genetski algoritem za avtomatsko planiranje razporeditve zasedenosti plakatnih mest ter izvedli prilagoditev algoritma za reševanje problema trgovskega potnika za delovanje na mobilnem telefonu. S pametno uporabo algoritmov v sistemu, nam je uspelo razviti sistem, ki ni uporaben samo za potrebe oglaševalcev, ampak tudi na drugih področjih dela.

Ključne besede:
Android, Django, splet, poslovna aplikacija, avtomatika, TSP, optimizacija s kolonijami mravelj, genetski algoritem

Abstract

Modern world bombs us with advertisements on every step. At home in front of a TV, while browsing on the internet or during a walk around a city. Advertisements are everywhere. Every day new ones come in different shapes and sizes. They are more and more daring and convincing. Where do they come from? Who puts them there? How do they replace old with new ones? Can this process be supported and automated by a computer? In this diploma thesis we tried to answer to all this questions. We developed software that uses many algorithms to support the business of billboard management. We used geometric algorithm to automatically assign work order to the right person, developed genetic algorithm that can automatically assign advertisement to the best billboard and we made an adjustment to the travel salesman algorithm to run on mobile phone. With smart usage of algorithms we succeeded to develop a system that can be used in solving wide range of problems, not just in advertisement.

Keywords:
Android, Django, web, business application, automatic, TSP, ant colony optimization, genetic algorithm

Poglavje 1

Uvod

Upravljanje s plakatnimi mesti je resen posel. Vedno več denarja se namenja oglaševanju, saj se vedno več podjetij zaveda, da je ključ za uspeh izdelka v dobri prepoznavnosti na trgu. Poslovni svet od upravljalca spletnih mest zahteva hitro prilagajanje, dober pregled nad stanjem na terenu ter odlične organizacijske in pogajalske sposobnosti. V veliko pomoč pri vsakodnevnemu spopadanju s problemi mu je lahko dober informacijski sistem. Za razvoj dobrega informacijskega sistema je potrebno poznavanje značilnosti posla in problemov, ki se pojavljajo.

1.1 Potek dela upravljalcev plakatnih mest

V diplomskem delu smo razvili programsko rešitev za podporo poslovanja upravljalcev plakatnih mest. Opis poteka dela je pridobljen preko pogovora s podjetjem, ki se že vrsto let ukvarja s plakatnimi mesti in predstavlja njihov trenutni potek dela od pridobitve stranke, vse do zaključka oglasne kampanje za določeno stranko in potek same zamenjave plakatov.

1. **Identifikacija želja stranke**: Stranka ima različne želje glede načina oglaševanja in njene ciljne publike. V podjetju, kjer je bila opravljena analiza, od stranke potrebujejo naslednje parametre za izvedbo oglaševalske akcije:

- Obdobje zakupa oglasnega prostora, vse želje glede dolžine zakupa, morajo biti večkratnik števila 14 (14 dni, 28 dni itd.), pri čemer je postavljena zgornja meja zakupa.
- Ciljna publika: moški, ženske, otroci, družine, zabava, hrana, lokacije v bližini šol, lokacije v bližini bank ipd.
- Montaža dodatnih gradiv, promocijskih darilc ipd.

2. **Pregled pogojev za izpolnitev želja stranke**: Na podlagi zbranih želja stranke je potrebna odločitev, ali se lahko stranki zagotovi storitev kot jo le-ta pričakuje. V primeru, da pogoji za popolno zadovoljstvo stranke ne obstajajo, se izvedejo pogajanja, katerih namen je omilitev želja oziroma dosega dogovora, ki je primeren za obe strani.

3. **Popravljanje dodelitve plakatnih mest za že obdelane stranke**: Stranka sama ne more izbrati, kje točno naj se pojavijo njeni oglasi. Poda lahko samo opis želja, dodelitev plakatnega mesta pa opravi izvajalec. Včasih se zgodi, da je potrebno kakšno predhodno dodeljeno plakatno mesto dodeliti drugi stranki. Bolj kot je stranka pomembna za podjetje za upravljanje z oglasnim prostorom, večja je verjetnost, da se bo zaradi te stranke spremenila obstoječa razporeditev.

4. **Dodelitev plakatnih mest stranki**: V kombinaciji s prejšnjim korakov se izvede dodelitev oglasnih mest za novo stranko.

Po zajemu zahtev vseh strank, ki so aktualne za določeno časovno obdobje, se začne postopek zamenjave plakatov na plakatnih mestih. Celoten postopek od priprave razporeda pa vse do zamenjave plakatnih mest je običajno sestavljen iz naslednjih korakov:

1. **Izdelava delovnih nalogov**: Priprava opravil oziroma nalog za posameznega plakaterja,

2. **Prevzem delovnega naloga s strani plakaterja v podjetju**: Poleg samega delovnega naloga prevzame tudi plakate in druga morebitna

gradiva ali promocijske izdelke, ki jih mora namestiti na plakatno mesto.

3. **Potek zamenjeve plakatov**: Plakater se odpravi na posamezne lokacije, kjer opravi potrebne zamenjave. Na delovnem nalogu, ki ga je prevzel v podjetju, je navedeno kateri oglasi morajo na neki lokaciji ostati in katere je potrebno odstraniti. Delovni nalog ne predpisuje točne pozicije plakata znotraj lokacije. O razporedu se plakater odloča sam. Poskrbeti mora le, da je končno stanje kot je bilo predpisano.

4. **Vrnitev delovnega naloga**: Plakater mora v podjetje vrniti izpolnjen delovni nalog. Vrnjen delovnih nalog je osnova za obračun plačila za njegovo delo.

5. **Prepisovanje podatkov iz vrnjenih delovnih nalogov**: Za obračun dela in vodenje stanja na terenu, je potrebno podatke prepisati v elektronsko obliko.

1.1.1 Problemi identificirani pri pogovoru in analizi dela

Preko pogovora in analize poteka dela smo prišli do spoznanja, da ima zgoraj opisan način dela kar nekaj težav oziroma šibkih točk. Z odpravo le-teh bi dosegli, da bi podjetje delovalo bolj ekonomično in z manj vsakodnevnimi težavami. Tako bi se zaposleni lahko več časa posvetili primarni nalogi - iskanju oglaševalcev, ki prinašajo prihodke.

Problem prevzemanja delovnih nalogov

Prevzem delovnih nalogov se vrši v podjetju. Plakater pride v podjetje in prevzame svoj delovni nalog, vključno z vsem kar potrebuje za izvedbo delovnega naloga. Težava se pojavi v primeru, da se naknadno ugotovijo napake na samem delovnem nalogu, oziroma se na delovni nalog dodajo še dodatne

zamenjave. Takrat mora plakater ponovno v podjetje po nov delovni nalog, oziroma je potrebno delavca obvestiti po telefonu in mu sporočiti spremembe.

Pomanjkanje pregleda nad stanjem na terenu

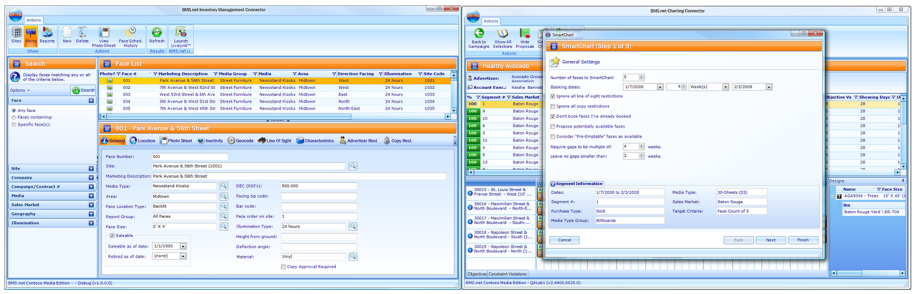
Na podlagi plana razporeditve podjetje ve, kakšno bo stanje na terenu po izvedbi delovnih nalogov s strani vseh plakaterjev. Dejansko stanje med samim izvajanjem zamenjav pa se težko spremlja. Tako odgovorni v podjetju ne vedo kolikšen delež plakatov je že bil ustrezno zamenjan, kakšen je bil vrstni red zamenjav, kateri plakati so (bili) poškodovani (zaželjena tudi slika stanja na terenu) ipd. Da bi bilo spremljanje takšnih in drugačnih parametrov s terena še težje, pripomore tudi dejstvo, da plakatna mesta niso enolično označena – oznake so podane kar opisno, npr. moški WC, 3. ogledalo. Še dodatno zmedo pa povzroča dejstvo, da se plakater samostojno odloča, kje bodo nameščeni posamezni plakati znotraj lokacije in te informacije ne sporoča v podjetje.

Težave pri dodeljevanju plakatnih mest

V vsakem obdobju, ki ga je moč zakupiti, se pojavi kar nekaj strank. Pri vsaki stranki je potrebno upoštevati njene želje in ji omogočiti čim višjo kvaliteto storitve. To prilagajanje in iskanje kompromisov (popravljanje že določenih plakatnih mest ipd.) vzame precej časa, še posebno, če upoštevamo dejstvo, da je potrebno poskrbeti, da velike stranke ostanejo zadovoljne po izvedbi akcije, saj le-te prinašajo največ prihodkov v podjetje.

Goljufanje pri poročanju kilometrine

Podjetje plačuje delavce na podlagi števila zamenjanih plakatov in količino prevoženih kilometrov. Ker je število potrebnih zamenjav plakatov za nekega plakaterja vnaprej znano, se velikokrat zgodi, da delavci poročajo večjo kilometrino oziroma zaradi neorganiziranosti naredijo večje število kilometrov, kot bi bilo potrebno za izvedbo zamenjave. Problem, ki se pojavi v okviru



Slika 1.1: Pogled na uporabniški vmesnik programa Ayuda BMS.

tega je, da odgovorni v podjetju ne vedo koliko kilometrov bi moral plakater opraviti v idealnem primeru, da bi opravil zamenjave vseh dodeljenih plakatov – dodelitev je namreč vsakič drugačna. Natančno ročno računanje bi vzelo kar nekaj časa in zato ponavadi odgovorni v podjetju pristanejo na neko smiselno vrednost.

1.2 Obstoječe rešitve

Pri pregledu trga in iskanju rešitve smo prišli do spoznanja, da je na trgu malo splošnih rešitev za upravljanje s plakatnimi mesti. Predvidevamo, da se poslovni procesi kar precej razlikujejo med sabo in zato podjetja predvsem uporabljajo rešitve po meri. Kljub temu se najde nekaj produktov, predvsem takšnih, ki podpirajo upravljanje elektronskih oglasnih mest (LED zasloni).

1.2.1 Ayuda BMS

Kanadsko podjetje Ayuda Systems [1] razvija sistem BMS [2], ki se uporablja za upravljanje z več kot 300.000 plakatnimi mesti po Kanadi in Združenih državah Amerike. Podpira upravljanje z lokacijami (vključno z GPS), upravljanje s pogodbami, kampanjami, izdelavo raznih poročil, pregled nad zasedenostjo plakatnih mest preko Ganttovega diagrama, pripravo računov, številne opcije izvozov ipd. Sistem v sebi skriva SmartChart tehnologijo (del uporabniškega vmesnika prikazuje slika 1.1), ki z uporabo algoritmov omogoča dobro zadovoljevanje želja strank, pri čemer poskuša maksimizirati

Slika 1.2: Slika prikazuje RF ID čitalec in značko, ki ju uporabljajo pri podjetju europlakat za označevanje plakatnih mest.

prihodek podjetja. To počne preko minimizacije časa nezasedenosti plakatnega mesta in optimizacije stroškov. Poskuša doseči čim večjo zasedenost plakatnih mest ob čim manjših operativnih stroških. Aplikacija ne podpira spletnega dostopa in je na voljo samo v obliki namizne aplikacije za Windows operacijski sistem.

1.2.2 bManage

bManage [3] je sistem za upravljanje z običajnimi in elektronskimi oglasnimi mesti, razvit v podjetju BillboardPlanet Inc. Sistem podpira številne funkcionalnosti, med pomembnejšimi so modul za izdelavo predlogov oglaševanja, upravljanje s pogodbami, virtualna vožnja mimo plakatnega mesta, načrtovanje razporeditve in spletni dostop.

1.2.3 Rešitev podjetja europlakat

Preko raziskovanja spletne strani podjetja Europlakat [4] smo prišli do sklepa, da podjetje uporablja lastno razvito rešitev za upravljanje z oglasnimi mesti. O samem sistemu in njegovih funkcionalnostih preko podatkov na spletni strani ni moč veliko sklepati. Pritegnilo pa nas je dejstvo, da njihov sistem

omogoča evidentiranje opravljenih zamenjav plakatov preko RFID. Z RFID značkami imajo opremljena plakatna mesta velikosti 504×238 cm in 400×300 cm. Plakater ob zamenjavi plakata uporabi poseben čitalec s pomočjo katerega potrdi svoje delo. Ta funkcionalnost sistema jim služi kot osnova za obračunavanje plačil zunanjim sodelavcem ter pošiljanje poročila o zamenjavi naročniku.

1.3 Cilji diplomskega dela

V okviru diplomskega dela smo si zadali cilj, da omogočimo podjetjem, ki se ukvarjajo s plakatnimi mesti boljše, lažje, cenovno bolj učinkovito in napredno delo. V želji po uspešni identifikaciji problemov, smo se odpravili v podjetje za upravljanje z oglasnimi mesti. Skozi sestanke smo identificirali glavne probleme, ki jih mora diplomsko delo reševati. Ti problemi so:

- **težavno sledenje stanju na terenu in posledično pojavljanje napak**: Pri obstoječem načinu dela v izbranem podjetju uporabljajo programsko podporo v programu Microsoft Excel. Samo delovanje programske rešitve je povsem zadovoljivo, problem se pojavi zaradi ročnega vnosa podatkov iz vrnjenih delovnih nalogov. Pri tem prepisovanju se namreč velikokrat zgodi, da stanje na terenu in stanje v programu nista povsem usklajena ter je potem potrebno ukrepati na licu mesta. Včasih pride do situcije, ko plakater ob prihodu na lokacijo ugotovi, da na lokaciji ne more odstraniti zahtevanih plakatov in s tem narediti prostor za nameščanje novega.

- **zajem dodatnih podatkov s terena**: Občasno se na terenu pojavi želja po dodatnem zajemu podatkov. Želja se pojavi predvsem takrat, ko se ugotovi, da je plakatno mesto poškodovano in potrebno popravila. V takšnih primerih bi koristil zajem slike, ki bi naredil avtomatsko povezavo slike z delovnim nalogom.

- **zamudna priprava razporeda plakatnih mest**: Ročna priprava

razporeda zasedenosti plakatnih mest je precej zamudna, še posebej ker je potrebno upoštevati želje strank in njihovo pomembnost za samo podjetje. Razpored se pogosto spreminja in stranka ne more določati, kje točno želi oglaševati.

- **kilometrina**: Plakater je plačan na podlagi števila zamenjanih plakatov in kilometrov, ki jih je moral pri tem opravit. Podjetju je znano potrebno število zamenjav, nimajo pa podatka o tem, koliko znašajo potrebni kilometri da se opravijo vse zamenjave. Za rešitev problema bi bila potrebna informacija o okvirnem številu potrebnih kilometrov. Na žalost lahko pride do situacije, v kateri se plakater lahko izgovarja, da ni vedel, kakšen vrstni red obiskovanja lokacij bi bilo potrebno izbrati, da bilo število kilometrov čim bližje izračunani vrednosti. Za preprečitev takšnega izgovarjanja se potrebuje rešitev, ki plakaterja usmerja od lokacije do lokacije.

1.4 Pregled dela

V uvodu smo predstavili potek dela upravljalcev plakatnih mest in težave s katerimi se soočajo. Identificirali smo točke, kjer je možno izboljšati računalniško podporo poslovanju in točke, kjer lahko poslovanje še dodatno računalniško podpremo. Ogledali smo si tudi nekaj obstoječih rešitev iz izbranega področja.

V drugem poglavju opravimo pregled mobilnih in spletnih tehnologij, ki smo jih uporabili pri izdelavi diplomskega dela. Podamo tudi razloge za izbiro posamezne tehnologije.

V tretjem poglavju predstavimo algoritme in koncepte, s pomočjo katerih smo dosegli ključne funkcionalnosti diplomskega dela. Za posamezen problem smo poiskali več možnih rešitev in identificirali primernejšo rešitev problema.

Četrto poglavje je namenjeno predstavitvi sistema za avtomatizacijo upravljanja z oglasnim prostorom. Ogledamo si arhitekturo rešitve ter posamezne komponente celotnega sistema in njihov način delovanja.

Peto poglavje vsebuje primerjalno analizo algoritmov in konceptov, ki smo jih uporabili. Med sabo primerjamo algoritme za izbiro vrstnega reda obiskovanja lokacij ter si ogledamo kvaliteto delovanja posameznih variacij algoritma za razporejanje zasedenosti plakatnih mest.

V zadnjem, šestem poglavju kritično analizirano razvito rešitev. Izpostavimo prednosti in slabosti ter podamo nekaj smernic za nadaljni razvoj. Razmislimo tudi o uporabi sistema za podporo poslovanja drugim poslovnim modelom.

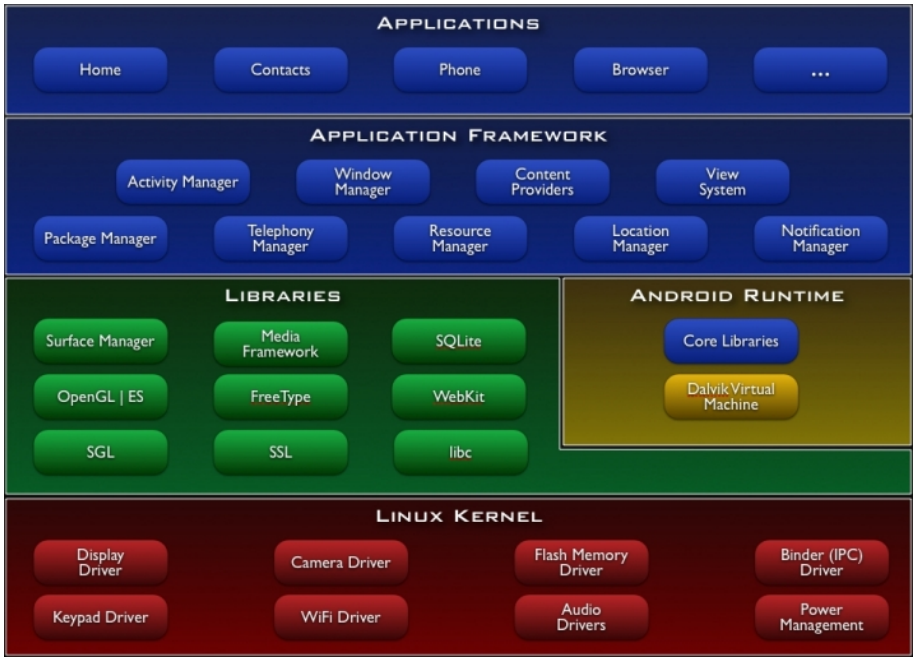
Poglavje 2

Uporabljene tehnologije

Pri izdelavi diplomskega dela smo uporabili številne obstoječe in trenutno aktualne tehnologije.

2.1 Android

Android [5, 6] je odprtokodni operacijski sistem. Razvit je bil kot neposredna konkurenca že uveljavljenim operacijskim sistemom za mobilne naprave, ki so jih razvila podjetja Nokia, Microsoft in Apple. Razvoj operacijskega sistema se je začel leta 2003 v podjetju Android Inc, večjo veljavo na trgu pa je dobil po odločitvi Google za nakup podjetja leta 2005. Svoj javni krst je doživel leta 2007 ob ustanovitvi Open Handset Alliance s strani podjetja Google in v sebi združuje številna podjetja, ki se ukvarjajo z razvojem programske in strojne opreme. Android je v osnovi zgrajen na Linux jedru (arhitekturo sistema prikazuje slika 2.1) okoli katerega so bile razvite številne komponente in aplikacije, ki naredijo operacijski sistem praktično uporaben. Operacijski sistem je na voljo v kar nekaj različicah in je na voljo na številnih napravah. Primarno podprta strojna platforma je ARM, pojavljajo se tudi različice, ki delujejo na x86 platformi. Skupaj z različnimi verzijami sistema [7] pridejo tudi različne verzije razvojnega okolja. Z vidika diplomskega dela so uporabne različice, ki imajo podporo za NFC (2.3.x, 4.0.x). Aplikacije za



Slika 2.1: Arhitektura sistema Android.

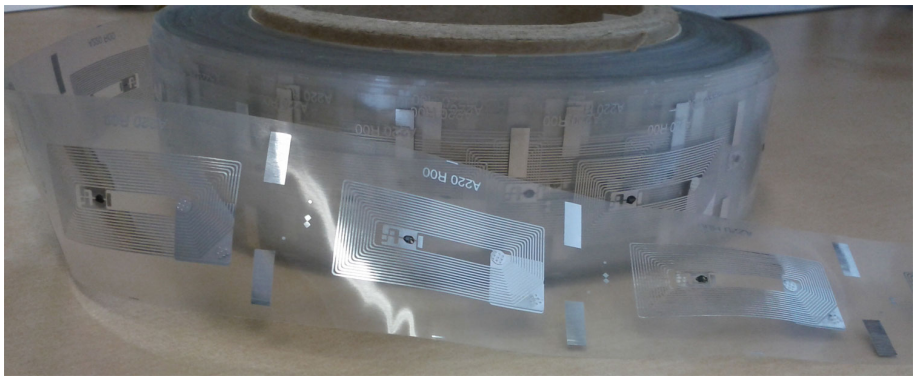
operacijski sistem Android razvijamo z uporabo programskega jezika Java in uporabo XML datotek z uporabo ADT dodatka za Eclipse [8]. Razvojno okolje izvorno kodo prevede v binarno kodo Java virtualnega stroja, le-ta pa se preoblikuje v posebno obliko (dex), ki je primerna za izvajanje v Dalvik virtualnem stroju. Dalvik je procesni virtualni stroj posebej prilagojen za naprave z manj pomnilnika in procesne moči. Google trdi, da je virtualni stroj prilogojen tudi v tej meri, da lahko instočasno in učinkovito teče več virtualnih strojev na eni napravi. Izvajanje aplikacij na mobilnem telefonu je realizirano tako, da vsaka java aplikacija teče pod svojim virtualnim strojem in Linux uporabnikom. Na takšen način so aplikacije popolnoma izolirane med sabo, tako njihovo delovanje kot pravice nad aplikacijskimi datotekami (npr. podatkovna baza). Android v sebi združuje številne funkcionalnosti. Med pomembnejšimi za razvoj aplikacij z namenom podpore poslovnemu procesu so: podpora za podatkovno bazo (SQLite), povezljivost (GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, NFC in WiMAX), podpora različni dodatni strojni opremi (kamera, merilnik pospeška, GPS,

Slika 2.2: NFC značke najdemo v vseh možnih oblikah. Na sliki so primeri različnih nalepk, primer kartice in primer obeska.

termometer, barometer, giroskop in drugi), shranjevanje podatkov na spominsko kartico, sporočanje itd.

2.2 NFC

NFC (ang. Near Field Communication) [9] je skupek standardov, katerih namen je omogočanje bližnje radijske komunikacije med elektronskimi napravami. Elektronska naprava je lahko mobilni telefon, računalnik, kavni avtomat, vhodna vrata ipd. NFC je zasnovan tako, da omogoča komunikacijo zgolj na razdalji do nekaj centimetrov, kar je povsem dovolj za predvidene namene uporabe. V okviru obstoječih aplikacij in aplikacij, ki se jih pričakuje v prihodnosti, se NFC uporablja za mobilno plačevanje, izmenjavo majhne količine podatkov (poslovne vizitke), kontrolo pristopa, kot elektronska vstopnica na prireditve in druge dogodke, potrjevanje prisotnosti



Slika 2.3: Značke na kolutu brez zaščitnega oziroma papirnega sloja pripravljene za nadaljno obdelavo.

Slika 2.4: Primer NFC čitalca ACS ACR122U.

na določeni lokaciji itd. V okviru NFC poznamo več tipov naprav oziroma sestavnih delov. Poznamo aktivne naprave in značke (primeri na sliki 2.2). Pod aktivno napravo spada vsaka elektronska naprava opremljena z NFC čipom in lastnim napajanjem. Aktivne naprave so tako mobilni telefoni, računalniki opremljeni z NFC čitalci (slika 2.4), gospodinjski aparati z NFC čipom itd. Značka je čip, opremljen z anteno in brez lastnega napajanja (primer gole značke je na sliki 2.3). Moč jih je zaslediti v številnih oblikah. Pojavljajo se v obliki potiskanih nalepk, zapestnic, obeskov za ključe, kartic, figuric itd. Zanje velja, da se med sabo razlikujejo tudi po vrsti čipa. V času pisanja so v standardu zajeti naslednji čipi:

- ISO/IEC 14443 Type A,
- ISO/IEC 14443 Type B in
- FeliCa.

Med napravami oziroma akterji NFC je možnih več načinov komunikacije. Tehnologija trenutno omogoča naslednje:

- aktivna naprava z aktivno napravo: npr. komunikacija med dvema mobilnima telefonoma, komunikacija med telefonom in terminalom za plačilo ipd. Sama komunikacija se lahko izvede na več načinov:
  - ena od naprav emulira značko,
  - P2P komunikacija: naprava, ki sporoča, ima aktivno radijsko polje, naprava, ki sprejema ga izklopi. Izmenično, enkrat ena, drugič druga naprava.
- aktivna naprava z značko: Značka potrebuje napajanje in ne deluje dokler ni v polju aktivne naprave. Takrat oživi in lahko se začne dvosmerna komunikacija.

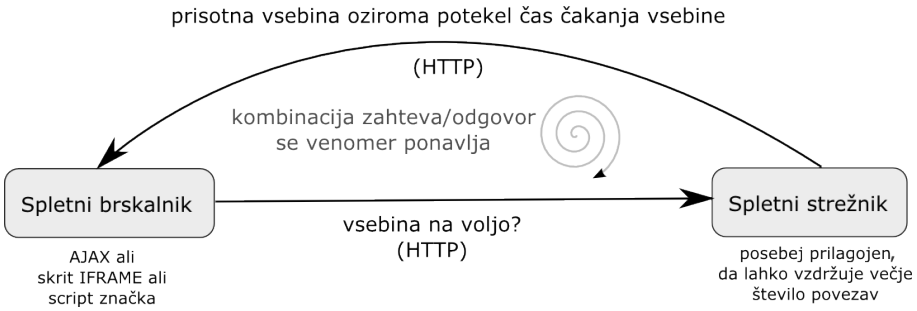
2.3 Django

Django [10, 11] je ogrodje za razvoj spletnih aplikacij v programskem jeziku Python. Spodbuja ponovno uporabo že razvitih komponent in načela DRY (angl. don't repeat yourself). Zasnovan je po vzoru model - pogled - kontroler (ang. MVC, model - view - controller) ahitekturnega vzorca. Osnova Django spletnega ogrodja so objektno-relacijski preslikovalnik (ang. object-relational mapper, predstavlja model v MVC), sistem za predloge spletnih strani (ang. web templating engine, predstavlja pogled v MVC) in url razpečevalnik (ang. url dispatcher, predstavlja kontroler v MVC). Poleg treh osnovnih komponent se v jedrnem delu ogrodja skrivajo enostaven spletni strežnik za testiranje in razvoj, sistem za serializacijo in preverjanje veljavnosti spletnih obrazcev, sistem za prevajanje in številni drugi. V okviru spodbujanja ponovne uporabe Django aplikacij, so razvijalci Django spletnega ogrodja že sami priskrbeli nekaj najpogosteje uporabljenih aplikacij oziroma zaokroženih celot. V paketu tako dobimo sistem za avtentikacijo (vključno s sistemom za pravice), dinamično administrativno stran, sistem za komentiranje, razne aplikacije za zaščito pred napadi ipd.

Django zaradi uporabe interpretativnega jezika deluje na številnih platformah. Naj si bo to Linux, FreeBSD, Windows ali kaj tretjega. Da je platforma podprta je potrebno le, da za platformo obstaja Python tolmač ter da so za izbrano platformo dostopne knjižnjice za povezavo na izbrano podatkovno bazo. Podatkovnih baz, ki jih je moč uporabiti v Djangu je kar nekaj. Podprte so tako PostgreSQL, MySQL, SQLite in Oracle. Pojavljajo se tudi številni dodatki za podporo dodatnih podatkovnim baz, moč je najti tudi podporo za ne-relacijske baze (npr. MongoDB).

2.4 Comet

Comet [12] je spletni aplikacijski model, ki se uporablja za realno-časovno komunikacijo med spletnim strežnikom in uporabnikovim brskalnikom preko HTTP. Sam termin v sebi skriva več tehnologij, ki so že v osnovi prisotne v



Slika 2.5: Shema Comet modela.

aktualnih spletnih brskalnikih. Implementacije Comet aplikacijskega modela razdelimo v dve večji kategoriji:

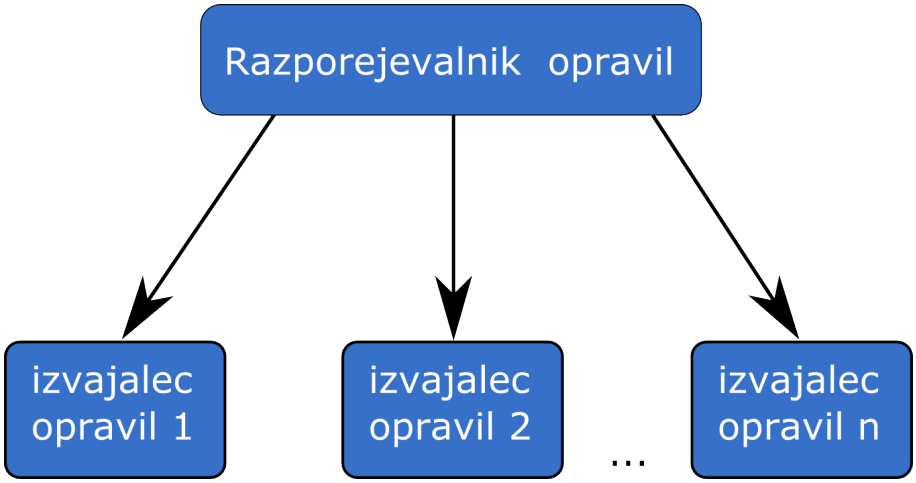
- Pretakanje (angl. Streaming): Aplikacija na Comet strežnik odpre stalno povezavo preko katere dobiva podatke od strežnika, ko se ti pojavijo in izvaja ustrezne operacije. Pretakanje lahko izvedemo na dva načina, s pomočjo skrite iframe značke (angl. Hidden iframe) ali z uporabo XMLHttpRequest-a.

- AJAX dolgo izpraševanje (angl. Ajax with long polling): Pretakanje zaradi stalne povezave s strežnikom ni zanesljivo (izguba povezave s strežnikom) in zaradi tega se uporabljajo še drugi načini implementacije. Tako poznamo še dolgo izpraševanje preko XMLHttpRequest in dolgo izpraševanje z uporabo html script značke (angl. Script tag long polling).

Dobra implementacija Comet protokola skriva v sebi več načinov izvedbe in preklaplja med njimi glede na potrebe.

2.5 Celery

Celery [13] je odprto-kodni sistem, spisan v programskem jeziku Python, za asinhrono distribuirano izvajanje opravil (angl. tasks). Uporablja se za izvajanje opravil, katerih izvajanje lahko preložimo na kasnejši čas oziroma za



Slika 2.6: Shema sistema Celery.

prelaganje izvajanja dalj časa trajajočih opravil z namenom obdržanja dobre uporabniške izkušnje. Sistem sestavljajo razporejevalnik opravil (angl. message broker), eden ali več izvajalcev opravil (angl. working nodes) ter prostor za shranjevanje rezultatov in kasnejši prevzem (slika 2.6 prikazuje arhitekturo sistema). Razporejevalnik opravil prejema zahteve in jih razporeja med izvajalce opravil. Ti izvedejo zahtevano opravilo ter vrnejo rezultat oziroma ga shranijo za morebiten kasnejši prevzem. Sistem premore številne funkcionalnosti. Uporabnejše z vidika uporabe v sistemu razvitega v okviru diplomske naloge so:

- odpornost na napake (angl. fault-tolerant),
- porazdeljenost (angl. distributed),
- istočasnost izvajanja (angl. concurrency),
- shranjevanje rezultatov: v različne podatkovne baze ali pošiljanje obvestilnih sporočil,
- omejevalnik števila zahtev,
- pošiljanje elektronske pošte v primeru problema izvrševanja,
- razporejanje.

2.6 PostgreSQL

PostgreSQL [14] je objektno-relacijski sistem za upravljanje s podatkovno bazo (ang. ORDBMS), zaradi licence primeren za brezplačno uporabo v komercialnih kot tudi v čisto hobi projektih. Na voljo je za številne operacijske sisteme, med drugim za Linux, FreeBSD, Solaris, Microsoft Windows in Mac OS X. Razvoj sistema se je začel že v poznih 80-tih in se intezivno odvija še danes. Zaradi dolgoletnega vlaganja znanja in izkušenj številnih razvijalcev je sistem prišel v fazo, ko se lahko kosa tudi s plačljivimi/dragimi rešitvami. V sebi skriva številne dobre funkcionalnosti. Med njimi so podpora proceduralnim jezikom (v osnovi PL/SQL, Tcl, Perl in Python, možni še drugi jeziki preko dodatkov), indeksi (B+ drevesa, zgoščena tabela, posplošena iskalna drevesa, posplošeni obrnjeni indeksi), prožilci (na izvedbo UPDATE ali INSERT zahtev), dedovanje, replikacija, asinhrona obvestila (za pošiljanje dogodkov poslušalcem na bazi), transakcije, regularni izrazi, iskanje po celotnem besedilu ipd.

2.7 Nginx

Nginx [15] je odprto kodni HTTP, povratno posredovalni (angl. reverse proxy server) in tudi poštni posredovalni strežnik (angl. mail proxy server, SMTP, POP3 in IMAP). Focus pri razvijanju strežnika Nginx je usmerjen v zagotavljanje visoke hitrosti delovanja, sočasnosti izvajanja opravil in nizke porabe pomnilnika (10.000 istočasnih povezav naj bi porabilo okoli 2,5 MB pomnilnika). Strežnik je mogoče poganjati na številnih platformah. Podpira Unix, Linux, različice sistemov BSD, Mac OS X, Solaris, AIX in Microsoft Windows. Nginx se lahko uporablja za serviranje statičnih datotek, prikaz indeksnih datotek map (seznam brskanje po mapah) kot tudi za serviranje dinamičnih HTTP vsebin z uporabo FastCGI, SCGI in uWSG. Dobro se obnese tudi kadar ga uporabimo za uravnavanje obremenjenosti strežnikov. S primernimi vtičniki je možna uporaba strežnika za namene oddajanja video vsebin (FLV in MP4).

Visoko hitrost delovanja in predvidljivo obnašanje pod visoko obremenitvijo mu omogoča dogodkovno voden pristop k izpolnjevanju zahtev. Dogodkovno voden pristop pri izvrševanju zahtev pomeni, da določeno zahtevo Nginx izvršuje ob nastopu določenega dogodka. Npr. Ninx dobi zahtevo po pridobitvi podatka iz X strežnika. Nginx pripravi novo zahtevo in jo pošlje na strežnik X. Med čakanjem na odgovor od strežnika X, Nginx počne druge stvari. Šele ob nastopu dogodka, ki sporoča, da so podatki prispeli, nadaljuje delo na prvotni zahtevi. Opisani pristop je boljši z vidika porabe sistemskih sredstev, kot pristop, kjer se za vsako zahtevo uporablja lastna nit ali proces kot to počno drugi strežniki (npr. Apache). Poleg vsega omenjenega Nginx omogoča še TLS/SSL povezave, imenske oziroma IP virtualne strežnike, avtentikacijo za spletne strani, gzip stiskanje, URL prepisovanje in še številne druge functionalnosti.

2.8 REST

REST [16] (angl. Representational State Transfer) je pristop za izvajanje operacij preko HTTP protokola. Operacija je lahko dodajanje podatkov, branje, brisanje, izvajanje izračunov ipd. REST arhitekturni tip je bil razvit pararelno s HTTP 1.1. Sestavljen je iz odjemalcev in strežnikov. Odjemalec poda zahtevo na določen URL skupaj z vsemi potrebnimi parametri. Strežnik sprocesira zahtevo in vrne rezultat v primernem format. Format je lahko kakršenkoli, največkrat se uporablja XML, JSON ali HTML. REST je za razliko od SOAP (angl. Simple Object Access Protocol) [17] manj tipsko strikten. V primerjavi s SOAP ima še številne dodatne prednosti. Ni omejen na XML format, ne zahteva od ponudnika storitve posebne glave pri komunikaciji, uporablja HTTP kode napak in posledično porabi manj pasovne širine. Zaradi vseh teh prednosti v praksi počasi nadomešča SOAP. REST arhitektura ima kar nekaj ključnih ciljev. Poskuša biti nadgradljiva (skalabilna), zagotavljati splošnost vmesnikov, neodvisnost komponent in vključevati vmesne komponente za zagotavljanje varnosti, zmanjšane zakasnitve in ovijanje

podedovanih sistemov (ang. legacy systems). Ključne cilje poskuša doseči z definiranjem šestih omejitev:

1. Odjemalec - strežnik (angl. client–server): Ločenost odjemalca od strežnika zagotavlja enostavnejši razvoj vsake od komponent in posledično to privede do boljših rešitev. Odjemalec se tako ne ubada s tem, kako je strežnik prišel do rezultata, kako se podatki celotnega sistema hranijo, kako zagotoviti istočasen dostop poljubne količine odjemalcev ipd. Podobno velja za strežnik. Njemu se ni potrebno ukvarjati s tem, kako se bodo podatki predstavili uporabniku, kako bo potekala interakcija z uporabnikom ipd.

2. Brez stanja (angl. stateless): Na strežniku se ne pomnijo podatki o stanju odjemalca oziroma kontekstu med posameznimi zahtevami. Odjemalec mora ob vsaki zahtevi podati vse podatke, ki se potrebujejo za izračun rezultata.

3. Možnost medpomnjenja rezultatov (angl. cacheable): Na spletu se lahko podatki pomnijo (npr. na proxy strežnikih). Zaradi tega mora biti odgovor strežnika eksplicitno označen ali je primeren za medpomnjenje ali ne. Npr. šifrant držav lahko medpomnimo nekaj časa, medtem ko rezultat poizvedbe o trenutni lokaciji določenega letala ne.

4. Večplastni sistem (angl. layered system): Odjemalec ne more vedeti ali je povezan direktno s končnim strežnikom ali vmes nastopajo še različni strežniki. Ti strežniki so lahko strežniki za pomnjenje predhodnih rezultatov, za uravnavanje obremenjenosti strežnikov, varnostni strežniki ipd.

5. Koda na zahtevo (angl. code on demand): Opcijsko lahko strežnik poveča funkcionalnosti odjemalca preko prenosa programskih dodatkov. Dodatki so lahko v obliki izvorne kode (npr. JavaScript) ali prevedeni v ustrezno binarno obliko.

6. Enoten vmesnik (angl. uniform interface): Enoten vmesnik med odjemalcem in strežnikom poenostavi in loči posamezne dele arhitekture med sabo.

Pri implementaciji REST se moramo omejitev držati, ni pa predpisano, na kakšen način jih je potrebno implemetirati.

Poglavje 3

Uporabljeni algoritmi in koncepti

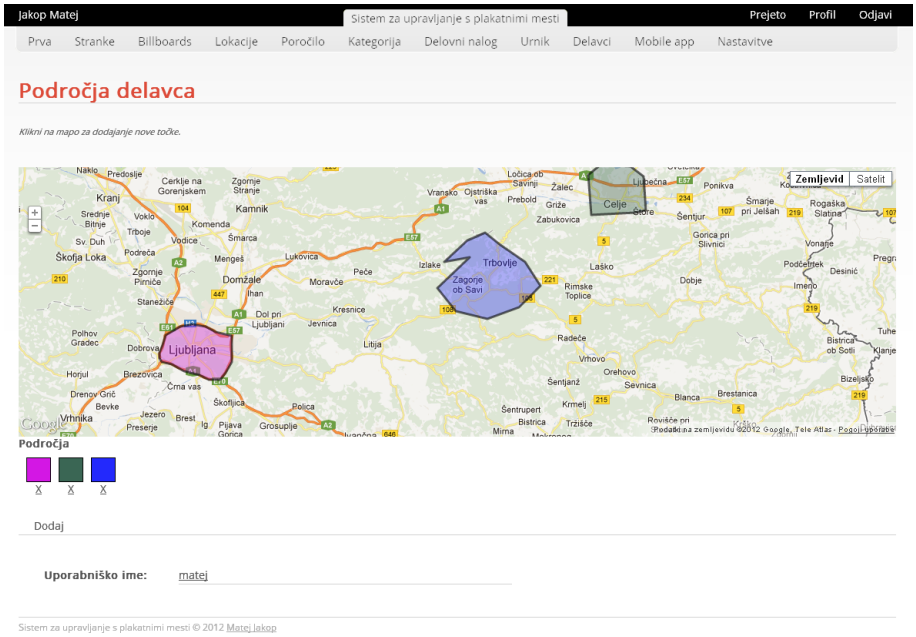
Za uspešno delovanje vseh od sistema zahtevanih nalog je bilo potrebno uporabiti posebne algoritme iz različnih področij. S pomočjo posebnih algoritmov smo sposobni reševati naloge kot so:

- avtomatsko dodeljevanje delovnega naloga delavcu na podlagi področja, ki ga pokriva,
- poiskati (sub)optimalno pot obiskovanja mest in
- poiskati kriterijsko optimalno razporeditev zasedenosti plakatnih mest.

Za vsako izmed zahtev smo preizkusili več algoritmov in se na podlagi izbranih kriterijev odločili, kateri je najprimernejši za vpeljavo v sistem.

3.1 Avtomatsko področno dodeljevanje delovnih nalogov

Problem avtomatskega dodeljevanja delovnega naloga delavcu v sebi skriva dva podproblema. Prvi podproblem je, kako določiti, kaj je vse delovno področje nekega plakaterja. V kakšni obliki to informacijo predstaviti, da bo

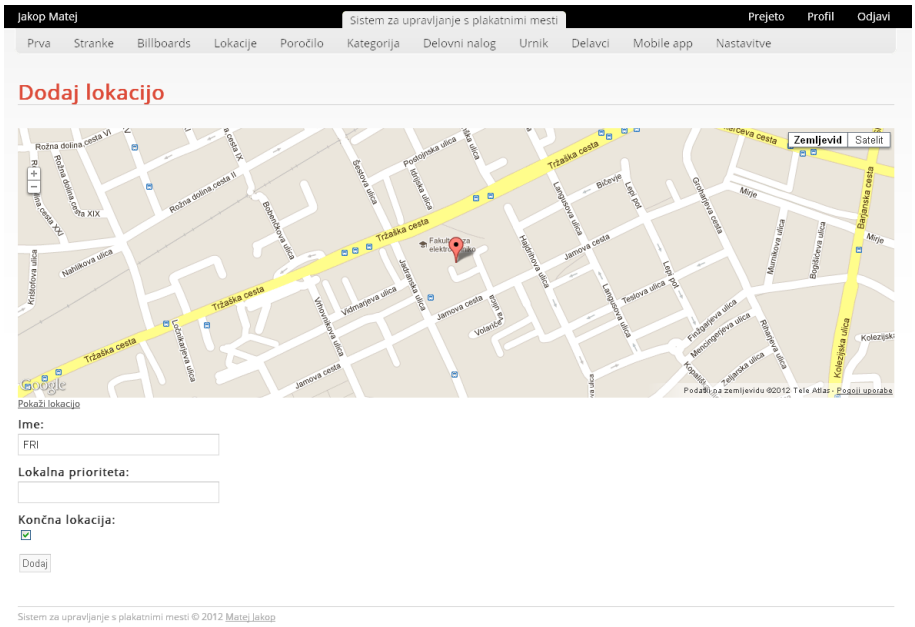


Slika 3.1: Vnos delovnega področja za delavca preko spletnega vmesnika.

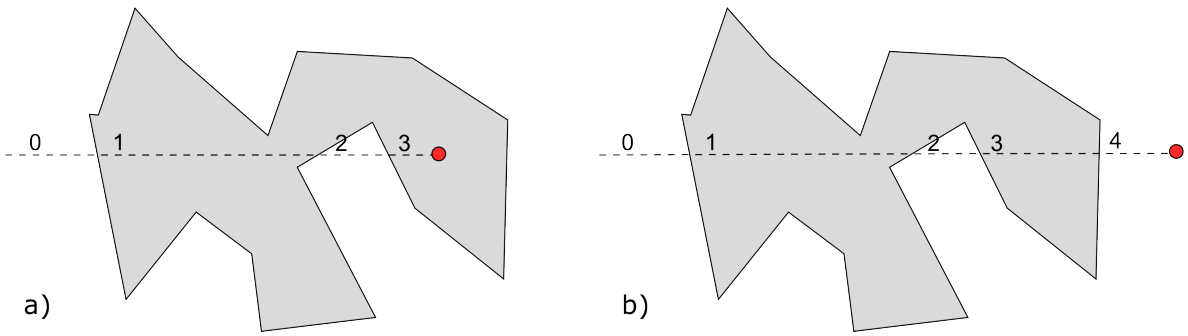
razumljiva uporabniku in jo bo moč uporabiti tudi v izbranem algoritmu? Odločili smo se, da bomo delovno področje predstavili v obliki poligona sestavljenega iz GPS koordinat. Na ta način je vnos delovnega področja za delavca sila enostaven (slika 3.1 prikazuje našo implementacijo vmesnika preko spletnega sistema). Naslednji podproblem, ki se pojavi v okviru reševanja zadane naloge je, kako ugotoviti katera lokacija delovnega naloga je znotraj delovnega področja delavca. To smo rešili tako, da vsaki lokaciji določimo GPS koordinati (slika 3.2 prikazuje dodajanje lokacije in izbiro GPS koordinat). Za izvedbo zahtevane naloge je potrebno samo še ugotoviti ali se lokacija nahaja znotraj delovnega območja, predstavljenega s poligonom (angl. problem Point in polygon). Rešitev smo iskali z metodo usmerjanja žarka in algoritmov za štetje števila obkrožitev.

3.1.1 Metoda usmerjanja žarka

Metoda usmerjanja žarka [18] (angl. Ray casting algorithm), znana tudi kot algoritem števila prehodov (angl. crossing number algorithm) ali algoritem



Slika 3.2: Dodajanje lokacije in določanje GPS koordinate lokacije.



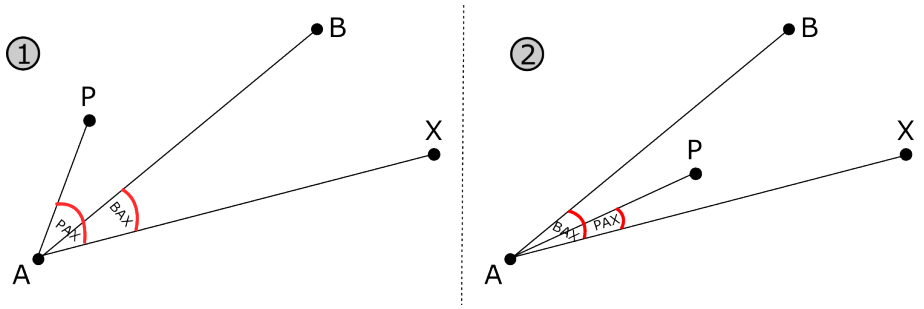
Slika 3.3: Točka znotraj poligona.

sodo-lihega pravila (angl. even-odd rule algorithm), je algoritem, ki nam pri podani točki in preprostem poligonu pove, ali se točka nahaja znotraj ali zunaj poligona. Da pride do te ugotovitve, algoritem šteje kolikokrat žarek seka rob poligona, če žarek potuje od točke, za katero želimo izvedeti ali je znotraj ali zunaj poligona, v poljubni fiksni smeri. Če je število sekanj liho in točka ne leži točno na robu, potem velja, da je točka znotraj poligona. To sledi iz opazovanja situacije, kjer gre žarek od neskončnosti proti opazovani točki tako, da seka robove poligona, pri čemer se izmenjujeta stanji zunaj, znotraj. Tako ob trku žarka z opazovano točko ugotovimo, kje se opazovana točka nahaja. Ob pogledu na sliko 3.3 a) vidimo, da žarek potuje iz neskončnosti proti točki, pri čemer 3-krat seka rob poligona. Število sekanj (3) je liho, kar pomeni, da je točka znotraj poligona. To je razvidno tudi iz slike. Na sliki 3.3 b) je moč zaslediti eno sekanje več, kar privede do sodega števila sekanj (4), kar pomeni, da je točka zunaj poligona. Slika potrjuje teorijo. Formalno je zgoraj podano predstavljeno z Jordanovim krivuljnim teoremom (angl. Jordan curve theorem) [19].

```
1 stevilo = 0
2 za vsako stran poligona:
3     ce zarek seka stran poligona potem
4         povecaj stevilo za 1
5 ce je stevilo liho potem
6     vrni tocka je znotraj poligona
7 drugace
8     vrni tocka je zunaj poligona
```

Koda 3.1: Psevdokoda metode usmerjanja žarka.

Najbolj pomembna in najtežja stvar pri podani psevdokodi 3.1 je, kako ugotoviti ali žarek seka stranico poligona ali ne. Na sliki 3.4 imamo podano daljico $\overline{AB}$. Zanjo mora veljati, da je A vedno pred B in B višje kot A. Kadar to ne velja, točki A in B zamenjamo. Ugotoviti želimo, ali žarek iz izbrane točke P seka daljico ali ne. Da pridemo do odgovora, je potrebno izbrati dodatno točko X. Zanjo velja, da je njena x koordinata večja kot je maksimalna x koordinata na daljici $\overline{AB}$. Med sabo povežemo točki P in A,



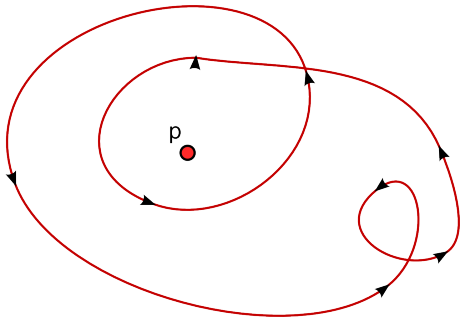
Slika 3.4: Prikaz razmerij med koti v primerih, ko žarek seka segment in kadar ga ne.

ter A in B. Na sliki se nam pojavita dva kota, $\angle PAX$ in $\angle BAX$. V primeru, da velja

$$\angle PAX > \angle BAX \tag{3.1}$$

sledi, da žarek iz točke P, ki potuje neskočno daleč v desno smer, seka daljico $\overline{AB}$. Velja tudi obratno. Če ne drži enačba 3.1 sledi, da žarek iz točke P, ki potuje neskončno daleč v desno smer, ne seka daljice $\overline{AB}$. Oba primera sta prikazana na sliki 3.4 in ju uporablja psevdokoda 3.2.

```
1  Parametri:
2  P : tocka iz katere zarek starta
3  A : koncna tocka segmenta za katero velja, da ima najmanjso y
       koordinato (A mora biti pod B)
4  B : koncna tocka segmenta za katero velja, da ima najvecjo y
       koordinato (B mora biti nad A)
5
6  ce Py = Ay ali Py = By potem
7       Py = Py + eps
8  ce Py < Ay ali Py > By potem
9       vrni ne
10 drugace ce Px > max(Ax, Bx) potem
11      vrni ne
12 drugace
13      ce Px < min(Ax, Bx) potem
14          vrni da
15      drugace
16          ce Ax != Bx potem
```



Slika 3.5: Krivulja okoli točke p za katero velja, da je število obkrožitev 2.

```
17              m_red = (By - Ay)/(Bx - Ax)
18          drugace
19              m_red = neskoncno
20          ce Ax != Px potem
21              m_blue = (Py - Ay)/(Px - Ax)
22          drugace
23              m_blue = neskoncno
24          ce m_blue >= m_red potem
25              vrni da
26          drugace
27              vrni ne
```

Koda 3.2: Psevdokoda iz metode usmerjanja žarka za ugotavljanje ali prihaja do sekanja ali ne.

Algoritem ima lahko nekaj težav s samo natančnostjo izračuna v primerih, kadar točka leži blizu roba, če je algoritem implementiran na računalniku s končno natančnostjo števil. V našem primeru uporabe, natančnost ne predstavlja problema. Pomembnejša nam je hitrost delovanja. Po drugi strani pa velja, da že poligon, ki definira področje dela delavca, ni najbolj točno podan, niti ni potrebe po tem.

3.1.2 Algoritem števila obkrožitev

Število obkrožitev (angl. winding number) [18, 20, 21] zaprte krivulje na ravnini okoli dane točke je celo število, ki pove kolikokrat krivulja potuje v

obratni smeri urinega kazalca okoli točke (na sliki 3.5 je podan primer pri katerem je število obkrožitev 2). Odvisno je od orientacije krivulje in je negativna v primeru, da krivulja potuje okoli točke v smeri urinega kazalca. Matematično je definirana na več načinov, eden izmed njih je:

$$w = \frac{1}{2\pi i} \int_c \frac{1}{z} dz, \tag{3.2}$$

kjer je $z = x + iy$ in c zaprta krivulja v kompleksni ravnini.

V primeru, da se točka nahaja na krivulji, je vrednost nedefinirana. Na podlagi številke lahko ugotovimo lokacijo opazovane točke pri podanem poligonu. Velja naslednje pravilo:

$$w = \begin{cases} 0 & \text{če točka ni znotraj poligona} \\ n > 0 & \text{če poligon obkroži točko n-krat v obratni smeri urinega kazalca} \\ n < 0 & \text{če poligon obkroži točko n-krat v smeri urinega kazalca} \end{cases} \tag{3.3}$$

Eden izmed načinov za izračun števila obkrožitev je, da seštejemo nasprotne kote vsake od stranic poligona [18, 22]. Postopek vključuje časovno zelo potratne inverzne trigonometrične funkcije. Za izračun števila obkrožitev obstaja boljši način. Dovolj je, da štejemo preko katerih kvadrantov potuje krivulja [18]. To nas hitrostno privede na isto raven kot metoda usmerjanja žarka.

3.2 Iskanje optimalne poti obiskovanja mest

Plakater na terenu dobi seznam delovnih nalogov, ki jih mora izvršiti. Delovni nalogi imajo določene lokacije (slika 3.6) in opravila, ki jih je potrebno na lokaciji izvest. Vprašanje, ki se pri tem pojavi je, kako po vrsti izvrševati delovne naloge, da se opravi čim krajša pot med posameznimi lokacijami. Postavljeno vprašanje je ekvivalentno vprašanju, ki se pojavi pri problemu trgovskega potnika (TSP) [23]. Problem trgovskega potnika je definiran na sledeči način. Imamo dan graf s podanimi cenami povezav med posameznimi mesti. Trgovski potnik mora obiskati vsa mesta in sicer tako, da bo obiskal

Tabela 3.1: Naraščanje števila različnih poti ob naraščanju števila mest.

Število mest n	Število različnih poti
1	1
2	2
3	6
4	24
5	120
10	3.628.800
15	1.307.674.368.000
20	2.432.902.008.176.640.000
30	2,65252859812191058636308 48e+32

vsako mesto natanko enkrat in pri tem opravil pot s čim nižjo ceno. Algoritmov za reševanje problema trgovskega potnika je kar precej, predvsem zato, ker gre za reševanje precej pomembnega problema s številnimi aplikacijami v planiranju, logistiki, izdelovanju mikročipov ipd. Algoritme za reševanje TSP lahko razdelimo v dve večji skupini:

- natančni algoritmi,
- hevristični in aproksimacijski algoritmi.

3.2.1 Natančni algoritmi

Namen natančnih algoritmov je izračun optimalne rešitve problema [23]. Trivialna rešitev je pregled vseh permutacij obiskovanja in obdržanje permutacije, ki ima najkrajšo razdaljo obhoda. Trivialna rešitev pri že malce večjem številu mest postane nepraktična ($n >= 20$). Število poti med n mesti, ki jih moramo pregledati je enako $n!$ (tabela 3.1 prikazuje naraščanje števila različnih poti, ki jih moramo pogledati pri n številu mest)

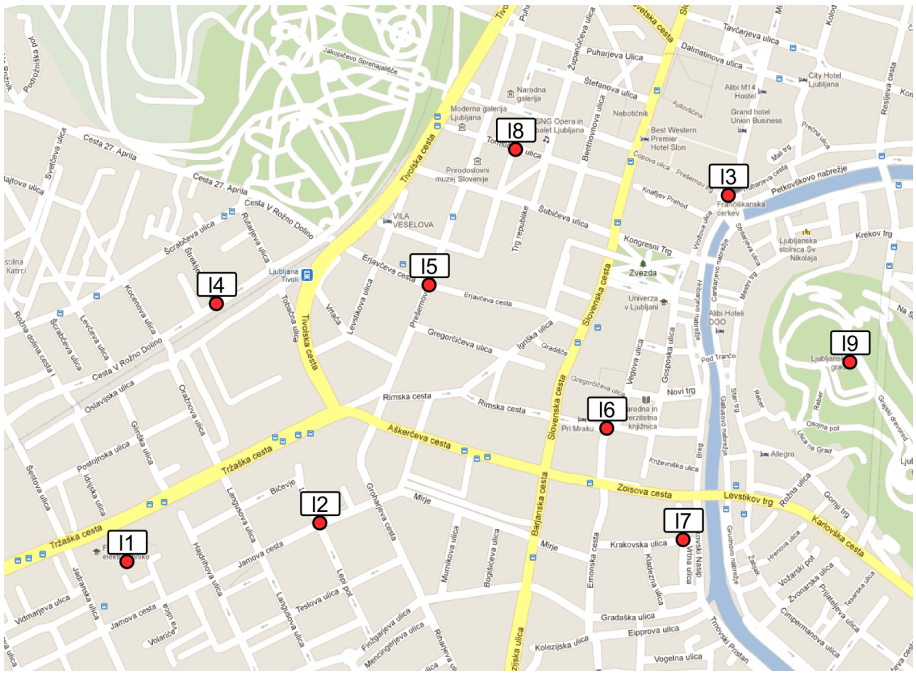
Ugotovljeno in dokazano je bilo, da je problem trgovskega potnika NP-poln. To nam da vedeti, da ne obstaja algoritem, ki bi nam rešitev problema trgovskega potnika izračunal v polinomskem času. Kljub temu so raziskovalci

našli algoritme, ki delujejo boljše kot opisana trivialna rešitev. Pristopi k izboljšavi trivialnega algoritma so bili naslednji:

- uporaba dinamičnega programiranja [23, 24],
- uporaba različnih razveji in omeji algoritmov (za do 60 mest) [23],
- progresivno izboljšavanje rešitve v kombinaciji z linearnim programiranjem (za do 200 mest) [23],
- uporaba različnih razveji in omeji algoritmov z domenskim znanjem (za do 85.900 mest) [25].

3.2.2 Hevristični in aproksimacijski algoritmi

Za hevristične in aproksimacijske algoritme velja, da ne garantirajo optimalne rešitve. Kljub temu se pri iskanju rešitve NP-polnih problemov velikokrat uporabljajo, ker se izkaže, da je za praktično uporabo njihovo delovanje še vedno dovolj dobro. Pri reševanju problema trgovskega potnika imamo na voljo več hevristik in aproksimacijskih algoritmov [26]. Dva izmed mnogih hevrističnih algoritmov sta najbližji sosed (angl. Nearest neighbour algorithm) in optimizacija s kolonijami mravelj (angl. ant colony optimization). Ker aproksimacijskih algoritmi ne dejajo optimalnih rezultatov, so raziskovalci razvili številne hevristike, ki izboljšajo končni rezultat pridobljen s pomočjo enega izmed številnih aproksimacijskih algoritmov. V tej skupini tako poznamo 2-opt, k-opt in njeno posplošitev v-opt metodo [27]. S pomočjo teh hevristik pridemo dokaj blizu spodnje Held-Karp meje. Spodnja Held-Karp meja je ocena, ki jo je moč izračunati s pomočjo simpleks (angl. simplex) metode, za katero velja, da je v povprečju okoli $0,8\%$ od optimalne dolžine poti [28]. Izračun ocene poteka preko izračuna sekvence minimalnih 1-dreves (angl. 1-tree), ki jih ocenimo ter izberemo najvišjo oceno. Najvišja ocena nam predstavlja spodnjo Held-Karp mejo. 1-drevo je na problemu z n mesti definirano na naslednji način. 1-drevo je povezan graf z vozlišči $1, 2, ..., n$ sestavljen iz drevesa na vozliščih $2, 3, ..., n$, ki je z dvema povezavama povezan s



Slika 3.6: Lokacije delovnih nalogov po mestu.

prvim vozliščem. Velja, da je 1-drevo minimalno, kadar je drevo na vozliščih $1, 2, ..., n$ minimalno vpeto drevo (angl. Minimum Spanning Tree) ter je le povezano na prvo vozlišče z dvema najcenejšima povezavama. Pot je 1-drevo v katerem ima vsako vozlišče stopnjo 2. Če je minimalno 1-drevo pot, potem zanj velja, da je to pot z minimalno ceno [28].

Ocena minimalne poti

Pri primerjavi algoritmov med sabo je koristno, če vemo, kakšna je spodnja meja, ki jo algoritem lahko doseže na reševanem problemu. Tako lahko nek algoritem ocenimo, kako dober je dejansko za nek reševani problem in ne samo kako dober je v primerjavi z ostalimi. Pri ocenjevanju, kateri algoritem želimo izbrati za uporabo v sistemu pri reševanju problema vrstnega reda obiskovanja plakatnih mest, smo potrebovali algoritem, ki ob podanem seznamu lokacij izračuna spodnjo mejo razdalje, ki jo moramo premagati, da obiščemo vsa mesta. Na voljo je več algoritmov s pomočjo katerih lahko pri-

dobimo oceno. Odločili smo se za algoritem, ki se sprehodi čez vsa vozlišča grafa (graf predstavlja mrežo plakatnih mest) in v vsakem vozlišču izbere najkrajšo razdaljo do naslednjega vozlišča in jo prišteje k skupni vsoti, pri čemer se ne ozira na vrstni red vozlišč.

```
1 dolzina_poti = 0
2 za vsako vozlisce v grafu:
3     min_izhod = MAX_VALUE
4     za vse povezave iz vozlisca:
5         ce je cena izhodne povezave < min_izhod:
6             min_izhod = cena opazovane izhodne povezave
7     vsota = vsota + min_izhod
```

Koda 3.3: Algoritem za izračun absolutne spodnje meje razdalje za obisk vseh vozlišč grafa.

Algoritem (koda 3.3) izračuna absolutno minimalno dolžino pod katero noben algoritem za reševanje problema vrstnega reda obiskovanja plakatnih mest nikoli ne more iti. To sledi iz preprostega dejstva, da algoritem samo sešteva minimalne cene (ki so nespremenljive na danem grafu).

Algoritem najbližji sosed

Algoritem najbližji sosed (angl. Nearest neighbour algorithm) [29] je eden izmed prvih algoritmov za reševanje problema trgovskega potnika. Deluje na sledeči način. Trgovski potnik začne svojo pot v poljubnem mestu X. Za nadaljevanje poti iz mesta X izbere najkrajšo pot do naslednjega mesta Y. V mestu Y se postopek ponovi. Postopek se ponavlja vse dokler niso bila obiskana vsa mesta. Psevdokoda algoritma je prikazana pod koda 3.4.

```
1 zacetno_mesto = random(mesta)
2 dokler niso obiskana vsa mesta:
3     mesto = izberi mesto, ki se bilo ni obiskano in do njega
          vodi najkrajsa pot iz trenutnega mesta v spremenljivki
          zacetno_mesto
4     mesto.obiskano = 1
5     zacetno_mesto = mesto
```

Koda 3.4: Psevdokoda algoritma najbližji sosed.

Algoritem hitro izračuna kratko pot, a na žalost večinoma ne optimalno (rezultat se od optimalne razlikuje v povprečju za 22% [25]).

Optimizacija s kolonijami mravelj

Optimizacija s kolonijami mravelj (angl. Ant colony optimization)[30, 31] je ena izmed paradigem (prvi algoritem v okviru te paradigme je iz leta 1991) za načrtovanje metahevrističnih algoritmov, ki se reševanja specifičnih problemov loti s pomočjo prenosa znanja iz narave. Avtorji paradigme so idejo dobili iz obnašanja mravelj. Paradigma pravi, da se za reševanje uporablja skupina računsko istočasnih in asinhronih agentov (mravelj), ki se premikajo skozi stanja problema v skladu z delnimi rešitvami. Na smer premika vpliva stohastična lokalna odločitvena funkcija, odvisna od dveh parametrov, sledi in privlačnosti. Vsak izmed agentov (mravelj) sestavlja delno rešitev problema in si pri tem pomni pot. Ko agent najde rešitev (v nekaterih primerih pa že med samo potjo), se agent sprehodi nazaj po pomnjeni poti in pušča feromone (pri mravljah v naravi so to kemični signali, ki jih izločajo, medtem ko se premikajo). Tako spuščeni feromoni bodo prihodnjim agentom (mravljam), ki bodo iskali (boljšo) rešitev problema, sporočali, da je nek del poti bolj privlačen kot kateri drugi. Optimizacija s kolonijami mravelj vsebuje tudi zelo pomemben mehanizem za izboljšanje delovanja. Da bi preprečili, da bi se nek del poti prenasičil s feromoni in zaradi tega vse agente preusmeril nase (zaradi svoje neizmerne privlačnosti), so avtorji iz narave prenesli tudi mehanizem izhlapevanja feromonov skozi čas. S pravkar opisano paradigmo in njeno značilnostjo pararelnega reševanja problema, se lahko dokaj dobro rešuje kombinatorične optimizacijske probleme. Optimizacijo s kolonijami mravelj smo v diplomskem delu uporabili na primeru reševanja problema trgovskega potnika. Slika 3.7 prikazuje raporeditev delovnih nalogov po lokacijah v mestu. Za rdeče obarvane delovne naloge velja, da so že bili opravljeni. Zeleno obarvani delovni nalogi na izvršitev še čakajo. Zanje je

potrebno izračunati pot obhoda. Privzeli smo, da lahko iz poljubne lokacije pridemo na poljubno lokacijo. Predpostavka je razvidna tudi iz slike 3.7, kjer so vse lokacije povezane ena z drugo. To precej poenostavi sam algoritem, na uporabo v praksi pa ima zanemarljiv vpliv.

V vsakem vozlišču mora agent izbrati naslednje vozlišče, v katerega želi priti. Izbira naslednjega vozlišča poteka na naslednji način. Agent preveri v katerih vozliščih iz trenutnega vozlišča še ni bil. Tako si ustvari seznam neobiskanih vozlišč. Po vrsti za vsak element i v seznamu neobiskanih vozlišč začne izvajati izračun, ki bo pokazal ali naj izbere pot v vozlišče ali ne. Najprej preko enačbe

$$d_i = pl_i^{\alpha} * (\frac{1}{r_{ij}})^{\beta} \tag{3.4}$$

izračuna lokalno verjetno d_i. Nato s pomočjo enačbe

$$p_i = \frac{d_i}{\sum_{n=1}^{V} d_n} \tag{3.5}$$

izračuna verjetnost za izbiro nekega mesta (p_i). Spremenljivka pl_i predstavlja trenutno vrednost feromonov v vozlišču i. Spremenljivka r_{ij} je enaka razdalji med trenutnim in opazovanim vozliščem. V je število vseh vozlišč. S parametrom α določamo raziskovalno naravnanost agentov. Večja kot je vrednost α, manjši je vpliv feromonov, ki so jih pustili agenti pri predhodnih iteracijah. Parameter β določa vpliv razdalje do mesta. Večja kot je vrednost β, večja je verjetnost, da bo med potmi, ki so še na voljo, agent izbral tisto, ki je krajša. Po izračunu verjetnosti, se agent s pomočjo generatorja naključnih števil odloči ali bo to mesto obiskal. V primeru, da velja

$$pn <= p_i; pn = \text{vrednost generatorja naključnih števil na intervalu [0..1]}, \tag{3.6}$$

potem pot v opazovano mesto izbere, drugače nadaljuje z izračunom za naslednje mesto na seznamu neobiskanih. To počne tako dolgo, dokler se za enega ne odloči oziroma pride v situacijo praznega seznam neobiskanih mest.

Ko algoritem ugotovi, da so vsi agenti obiskali vsa mesta, izvede najprej postopek izhlapevanje feromov po enačbi

$$trenutna\_stopnja * (1 - RHO) \tag{3.7}$$

Parameter RHO predstavlja procent feromonov, ki se naj ohrani. Izhlapevanju sledi postopek dodajanja novih feromonov. Algoritem za vsakega agenta posebej izračuna prispevek feromonov na poti. V poštev pridejo samo deli poti, ki jih je agent obiskal. Prispevek agenta je enak

$$prispevek = \frac{QVAL * RHO}{dolzina\_poti\_agenta} \tag{3.8}$$

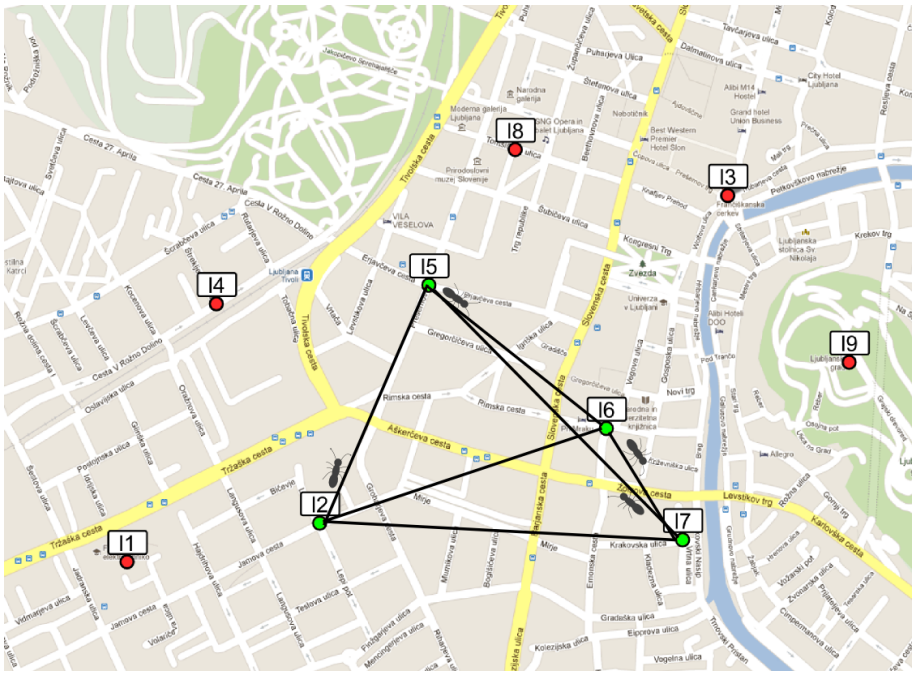
Parameter $QVAL$ predstavlja količini feromov, ki se sprosti določeni poti, ki je agenta pripeljala do končne rešitev. Po dodajanju feromonov na dele poti, algoritem resetira stanje in postopek se ponovi. Delovanje algoritma opisuje psevdokoda 3.5.

```
1 dokler ni dosezeno maksimalno stevilo iteracij ali ni bilo
     sprememb zadnjih 5 iteracij:
2     za vsako vozlisce v grafu:
3         izvedi premik mravelj iz vozlisca
4         ce so vse mravlje ze obiskale vsa mesta in premik ni vec
              mogoc:
5             naj izhlapi nekaj feromonov
6             poisci najboljso pot izmed vseh poti mravelj
7             zapomni si najkrajso pot in resetiraj lokacije
                  mravelj ter zacni nov izracun
8             povecaj stevilo iteracij
```

Koda 3.5: Optimizacija s kolonijami mravelj na problemu trgovskega potnika.

Hevristika Lin–Kernighan

Z uporabo enega izmed hevrističnih algoritmov dobimo običajno neoptimalno rešitev, ki pa se jo z nekaj dodatnega dela lahko izboljša. Ena izmed hevristik za izboljšavo končnega rezultata je hevristika Lin–Kernighan, znana tudi kot 2-opt. Ideja skrita v tehniki je precej enostavna. Zaporedno se sprehodimo čez celotno pot, odstranimo par vozlišč ter ga nadomestimo s krajšo povezavo in tako nazaj povežemo med sabo ločena fragmeta. Vse počnemo v upanju, da bomo na koncu uspeli pridobiti krajšo pot.



Slika 3.7: Izvajanje optimizacije s kolonijami mravelj na lokacijah delovnih nalogov.

3.3 Iskanje idealne razporeditve zasedenosti plakatnih mest

Pri problemu iskanja idealne rezporeditve zasedenosti plakatnih mest imamo na voljo n plakatnih mest in m plakatov, ki jih moramo z upoštevanjem želja strank med sabo povezati. Potrebno je ugotoviti, kateri plakat je potrebno dati na neko plakatno mesto, da bo stranka zadovoljna, mi pa bomo ob tem zmožni sprejeti čim več želja strank in hkrati imeli čim večjo zasedenost plakatnih mest. Pri razporejanju je potrebno upoštevati kar nekaj omejitev, ki problem reševanja še malce otežijo. Pristopov k reševanju je kar nekaj, eden izmed njih je uporaba genetskih algoritmov.

Genetski algoritmi so še eni izmed konceptov, ki poskušajo posnemati obnašanje narave [32]. Prevzemajo idejo evolucije in naravne selekcije. Genetski algoritmi delujejo nad določeno populacijo, pri čemer vsak osebek v

populaciji predstavlja eno hipotezo. Vsak osebek ima v sebi zakodirano hipotezo v obliki genov. Genetski algoritmi delovanje pričnejo z naključno generirano množico osebkov, imenovano tudi populacija oziroma generacija. Prehod iz generacije v generacijo poteka preko genetskih operaciji povzetih po dogajanju v naravi. Te operacije so:

- razmnoževanje: boljši kot je genetsko osebek (kvaliteta hipoteze), večja je verjetnost, da bo imel potomce,
- križanje: vsak potomec je zgeneriran iz dveh drugih osebkov (na izbiro vpliva kvaliteta genoma, boljša kot je, večja je verjetnost, da bo nek osebek izbran za razmnoževanje) na podlagi naključne izbire delov genoma staršev,
- mutacije: geni potomcev se naključno (z neko malo verjetnostjo) spreminjajo.

3.3.1 Genetski algoritem za razporejanje zasedenosti plakatnih mest

Pri izdelavi algoritma za avtomatsko razporejanje zasedenosti plakatnih mest smo se odločili, da bomo uporabili princip genetskih algoritmov. Za uporabo tega principa smo se odločili:

- ker je število vseh možnih kombinacij preveliko, da bi lahko problem rešili z uporabo surove sile (angl. brute force): Vzemimo za primer popolnoma realne številke. Na voljo imamo 5000 plakatnih mest. V izračun za naslednje časovno obdobje smo prejeli seznam želja, ki skupaj zasedejo 600 plakatnih mest pri čemer stranke niso podale nobenih želja, kje naj bo kakšen izmed njihovih plakatov nameščen. S pomočjo binomskega koeficienta izračunamo število vseh množic lokacij na katere lahko razporedimo posamezne plakat (zanemarimo različne razporeditve plakatov znotraj posameznih množic).

$$stevilo\_vseh\_razporeditev = \binom{5000}{600} = 1.0163111950... \times 10^{795} \quad (3.9)$$

Izračun z enačbo 3.9 nam pokaže, da je kljub zanemarjenju različnih razporeditev znotraj izbrane množice plakatnih mest, število kombinacij ogromno.

- smo ob prebiranju člankov prišli do spoznanja, da daje princip genetskih algoritmov dobre rezultate pri generiranju urnikov,
- zaradi naravne predstavitve problema z osebki v populaciji,
- zaradi dokaj enostavne implementacije ob uporabi prosto dostopnih knjižnjic.

Implementacija

Implementacija genetskega algoritma za razporejanje zasedenosti plakatnih mest je bila izvedena v programskem paketu Watchmaker [33]. Programski paket omogoča hitro in enostavno implementacijo željenega obnašanja v programskem jeziku Java z definiranjem le nekaj osnovnih razredov. Potrebno je definirati naslednje razrede:

- razred, ki predstavlja osebek populacije,
- razred, ki predstavlja tovarno novih osebkov populacije,
- razred za izvajanje križanj,
- razred za izvajanje mutacij in
- opcijsko razred za ugotavljanje kvalitete osebka populacije

Pri implementaciji smo za osnovni osebek populacije definirali razred urnik (angl. timetable) z nekaj osnovni metodami. Ena izmed metod je tudi metoda za preverjanje ali lahko osebek živi. Metoda pri iskanju odgovora na vprašanje preveri ali urnik ustreza vsem željam (se posamezni vpisi gibajo v ustreznih časovnih okvirih) ali prihaja do prekrivanja posameznih vpisov ali je neka želja upoštevana samo enkrat ipd. Na podlagi vseh teh podvprašanj poda končno odločitev: osebek lahko živi oziroma mora biti mrtev.

Ocenjevanje kvalitete zgeneriranega razporeda

Pri uporabi genetskega algoritma moramo skozi generacije ocenjevati posamezne osebke, v našem primeru urnike. Pri tem se pojavi vprašanje, kakšen je dober urnik? Po tehtnem premisleku smo prišli do odločitve, da je dober urnik takšen, ki čim bolje zadosti željam. Pogoji za zadostitev so:

- rezervacija traja željeno število dni in znotraj predpisanega obdobja,
- rezervacija se začne čimprej znotraj predpisanega obdobja,
- posamezne rezervacije na neki lokaciji se ne prekrivajo.

Na podlagi pogojev smo zasnovali funkcijo izračuna kvalite. Z njeno pomočjo lahko ločimo med sabo veljavne in neveljavne urnike ter dobre od slabih. Za vrednost kvalitete velja, nižja kot je vrednost rezultata, kvalitetnejši je urnik. V primeru, da je urnik neveljaven, velja

$$kvaliteta = \infty \tag{3.10}$$

Urnik je neveljaven kadar ne zadosti pogojem podanim zgoraj. Kadar ocenjujemo kvaliteto veljavnega urnika pri dani množici

$$X = \{x; x\text{je rezervacija termina v urniku}\} \tag{3.11}$$

velja

$$kvaliteta = \sum_{x \in X} (z(x) * pri(x)) \tag{3.12}$$

in

$$z(x) = zacetek(x) - zelja\_zacetka(x) \tag{3.13}$$

ter

$$pri(x) \text{ predstavlja prioritetni indeks } \in [0..1]\,. \tag{3.14}$$

Za $z(x)$ velja, manjše kot je boljše je. Nižji kot je prioritetni indeks, pomembnejša je lokacija. Vzemimo za primer izračun kvalitete urnika, ki ga dobimo z upoštevanjem želja prikazanih v tabeli 3.2. Recimo, da želimo oceniti urnik, ki ustreza željam, prikazan na sliki 3.8. Za poenostavitev izračuna

upoštevajmo, da imajo vse lokacije enako prioriteto (prioritetni indeks enak 1). Tako dobimo naslednje $z(x)$:

$$z(\text{Rezervacija 1}) = 1 - 1 = 0 \tag{3.15}$$

$$z(\text{Rezervacija 2}) = 2 - 2 = 0 \tag{3.16}$$

$$z(\text{Rezervacija 3}) = 1 - 1 = 0 \tag{3.17}$$

$$z(\text{Rezervacija 4}) = 4 - 2 = 2 \tag{3.18}$$

$$z(\text{Rezervacija 5}) = 11 - 11 = 0 \tag{3.19}$$

Iz tega dobimo, da je:

$$kvaliteta = 0 * 1 + 0 * 1 + 0 * 1 + 2 * 1 + 0 * 1 = 2 \tag{3.20}$$

Vidimo, da je urnik razporeda dokaj v redu. Primer na videz malce slabšega urnika je prikazan na sliki 3.9. Če zanj na podoben način kot prej izračunamo kvaliteto, ugotovimo da je:

$$kvaliteta = 2 + 1 + 0 + 3 + 2 = 8 \tag{3.21}$$

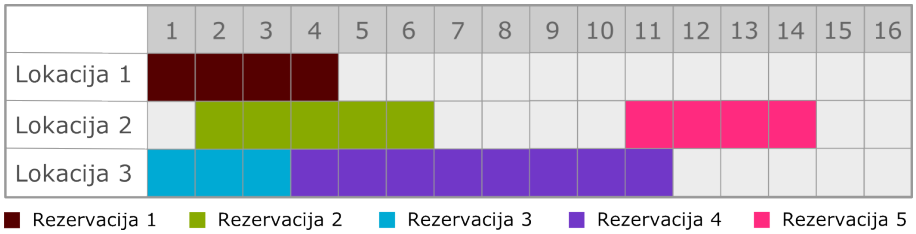
Funkcija kvalitete je pokazala, da gre za slabši urnik. Razporeditev na sliki 3.8 je boljša od tiste na sliki 3.9, kljub upoštevanju istih želja. Pri nadaljnem raziskovanju ocenjevalne funkcije se pojavi vprašanje, kakšen je idealni urnik. Idealni urnik bi bil takšen, za katerega bi veljajo, da je $kvaliteta = 0$. Da bi bilo to res, mora veljati, da se vse rezervacije pričnejo točno takrat kot definira želja. Za želje podane v tabeli 3.2 velja, da idealnega urnika na 3 lokacijah ne moremo doseči. Če zahteve malce spremenimo (tabela 3.3), pridemo do enega izmed idealnih urnikov prikazanega na sliki 3.10.

Tabela 3.2: Seznam rezervacijskih želja.

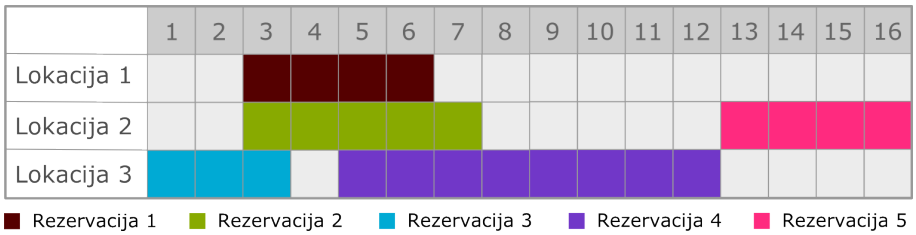
	Želja 1	Želja 2	Želja 3	Želja 4	Želja 5
Obdobje	1. do 6.	2. do 10	1. do 16.	2. do 16.	11. do 20.
Trajanje	4	5	3	8	4
Lokacija	1, 3	2	2, 3	3	2, 3

Tabela 3.3: Spremememenjen seznam rezervacijskih želja.

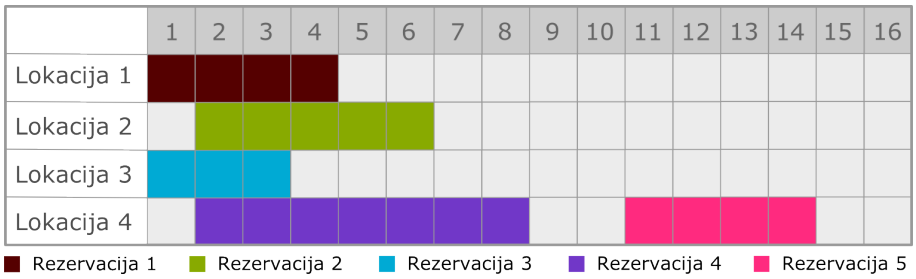
	Želja 1	Želja 2	Želja 3	Želja 4	Želja 5
Obdobje	1. do 6.	2. do 10	1. do 16.	2. do 16.	11. do 20.
Trajanje	4	5	3	8	4
Lokacija	1, 3	2	2, 3	3, **4**	2, 3, **4**



Slika 3.8: Primer urnika za želje podane v tabeli 3.2.



Slika 3.9: Primer slabšega urnika za želje podane v tabeli 3.2.



Slika 3.10: Eden izmed idealnih urnikov za želje podane v tabeli 3.3.

Križanje genov staršev pri ustvarjanju potomca

V genetskem algoritmu se populacija spreminja skozi generacije. Vsaka naslednja generacija bi v teoriji morala biti boljša. V veliki meri vpliva na izboljšavo populacije skozi čas, način križanja genov staršev pri ustvarjanju potomca. Zamislili smo si tri načine križanja genov:

1. Iz seznama rezervacij obeh staršem naključno za vsako željo izberemo rezervacijo in jo dam v potomca.

2. Iz seznama rezervacij obeh staršem za vsako željo izberemo najboljšo rezervacijo (najmanjši zamik začetka izvajanja) in jo dam v potomca.

3. Vedno ustvarimo klon boljšega od staršev in izvedemo postopek za izboljšavo urnika. Postopek poteka na naslednji način. Vsaki izmed rezervacij v urniku prestavimo datum začetka na čim zgodnejši termin, pri čemer to počnem tako, da ne pokvarimo urnika.

Mutacije

Genetski algoritem bi dajal precej slabše rezultate brez mutacij osebkov. V želji po izboljšanju delovanja algoritma smo dodali več načinov mutacij:

1. Na naključno izbrani lokaciji se naključno izbereta dve rezervaciji in se terminsko zamenjata med sabo.

2. Na naključno izbrani lokaciji naključno izberemo rezervacijo, ki jo lahko prestavimo za eno mesto na zgodnejši termin ne da bi s tem ustvarili neveljaven urnik.

3. Vse rezervacije na urniku premaknem za eno mesto na zgodnejši termin.

Ker so nekatere mutacije s precejšnjo verjetnostjo usodne za osebek, smo v implementacijo dodali, da se mutacija iz seznama mutacij izbere naključno. Določili smo, da se mutacija pod številko 1 izbere z verjetnostjo 12%, mutacija pod številko 2 z verjetnostjo 76% in mutacija pod številko 3 z verjetnostjo

12%. Tako smo dosegli boljše delovanje ter preprečili nenadne propade populacije, ki so se nam prej pojavljale. Kljub temu smo ohranili dinamičnost sistema.

Poglavje 4

Sistem za avtomatizacijo upravljanja oglasnega prostora

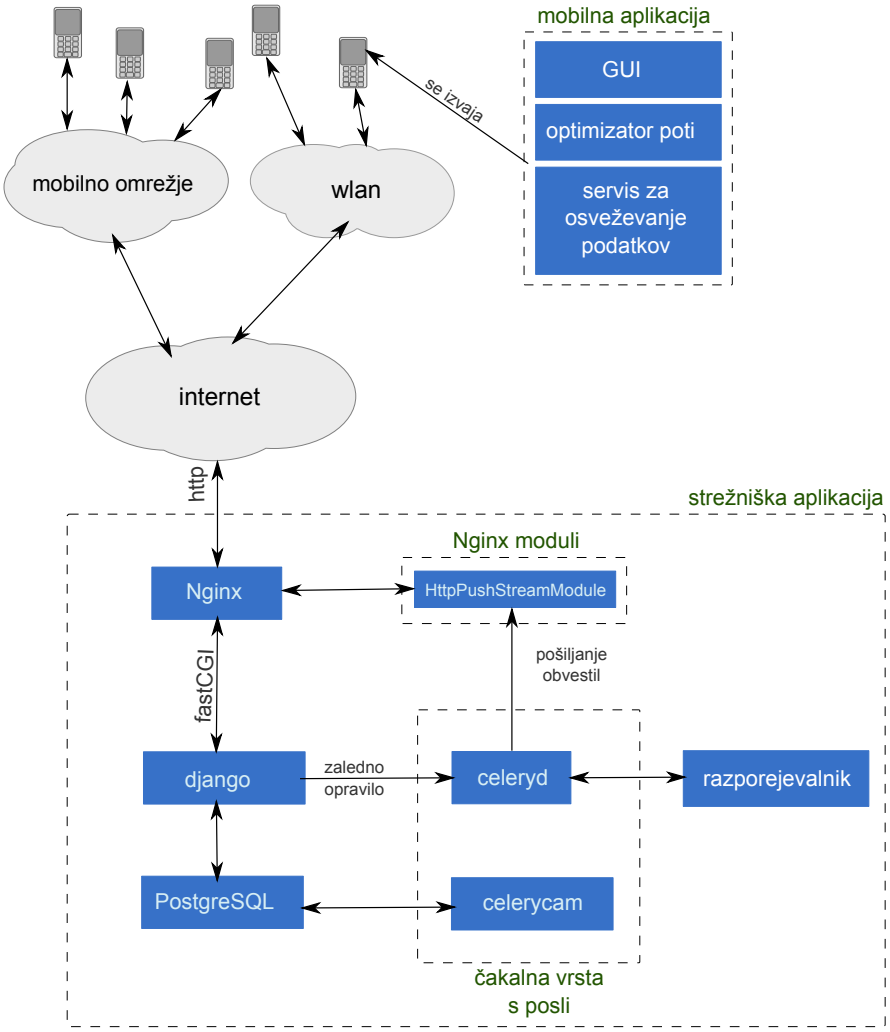
V okviru diplomske naloge smo razvili sistem za podporo pri upravljanju s plakatnimi mesti. Sistem za upravljanje s plakatnimi mesti podpira vse faze dela, od zajema naročnikovih zahtev in vse do končnega poročanja o stanju na terenu.

4.1 Arhitektura rešitve

Slika 4.1 prikazuje arhitekturo sistema. Iz slike je razvidno, da smo zasnovali sistem, ki je v grobem zgrajen iz:

- strežniške aplikacije in
- mobilne aplikacije.

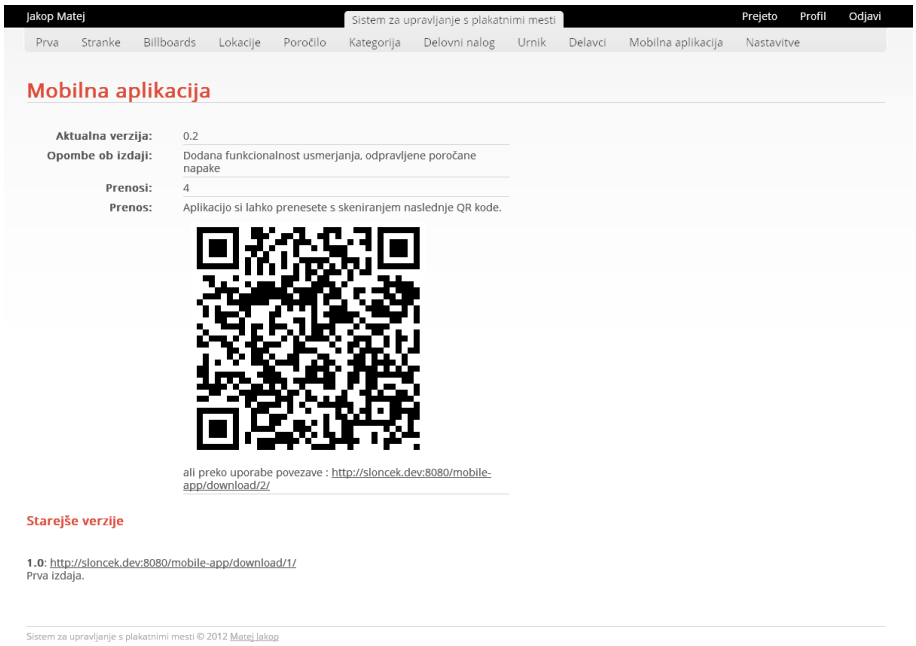
Za razdelitev sistema na dva dela smo se odločili zaradi potrebe po dveh aplikacijah. Prva aplikacija je namenjena delavcem v podjetju, druga pa za delavce na terenu. Aplikaciji tesno sodelujeta med sabo in šele ob sodelovanju obeh komponent lahko sistem zaživi.



Slika 4.1: Arhitektura sistema.

4.2 Strežniška aplikacija

Strežniška aplikacija predstavlja vstopno točko do sistema za vse uporabnike. Do strežniške aplikacije dostopajo delavci v podjetju in plakaterji. Z uporabniškega vidika lahko aplikacijo razdelimo na spletni vmesnik za upravljanje s sistemom, zaledni sistem in vmesnik API. Do spletnega vmesnika dostopajo delavci v podjetju. Preko njega upravljajo s sistemom. Plakaterji pri svojem delu uporabljajo posebno mobilno aplikacijo preko katere se izvajajo REST API zahteve na strežniško aplikacijo. Strežniška aplikacija sestoji iz spletnega strežnika Nginx, Django aplikacije, podatkovne baze PostgreSQL, sistema za izvajanje zalednih opravil Calery in razporejevalnika opravil. Spletni strežnik Nginx je vstopna točka v samo strežniško aplikacijo. Preko njega poteka vsa komunikacija z zunanjim svetom. Nginx-u je pripet HttpPushStreamModule, ki Ngix preoblikuje do te mere, da je mogoča uporaba principa Comet kar na obstoječem strežniku. V rešitvi smo Comet uporabili za obveščanje mobilne aplikacije, da je prišlo do sprememb na podatkih in da je potrebna sinhronizacija podatkov. Nginx je v osnovi nastavljen, da se obnaša kot posredovalni strežnik (angl. proxy) in vse ne Comet zahteve (od Comet zahtev jih loči na podlagi naslova, ki se poda ob klicu na spletni strežnik) pošlje naprej preko FastCGI na Django aplikacijo ter počaka na odgovor. Prejeti odgovor pošlje nazaj k stranki, ki je podala zahtevo. Django aplikacija je del sistema, ki v sebi skriva dve pomembni komponenti. V aplikaciji sta implementirana vmesnik API in logika za upravljanje celotnega sistema. Vmesnik API predstavlja le delček vseh funkcionalnosti Django aplikacije. Večji del aplikacije je namenjen spletnemu vmesniku za upravljanje s sistemom. Določena opravila, ki se izvedejo na zahtevo uporabnika, trajajo preveč časa, da njihovo izvajanje ne bi negativno vplivalo na uporabniško izkušnjo, saj po kliku na neko akcijo, uporabnik pričakuje takojšen odziv. Primer takšnega opravila je izračun razporeda zasedenosti plakatnih mest. Izračun razporeda včasih traja tudi po nekaj minut. Splošno uveljavljena praksa v teh primerih je, da se takšna opravila prestavijo na kasnejše izvajanje. Za uporabo kasnejšega in neodvisnega izvajanje nekega

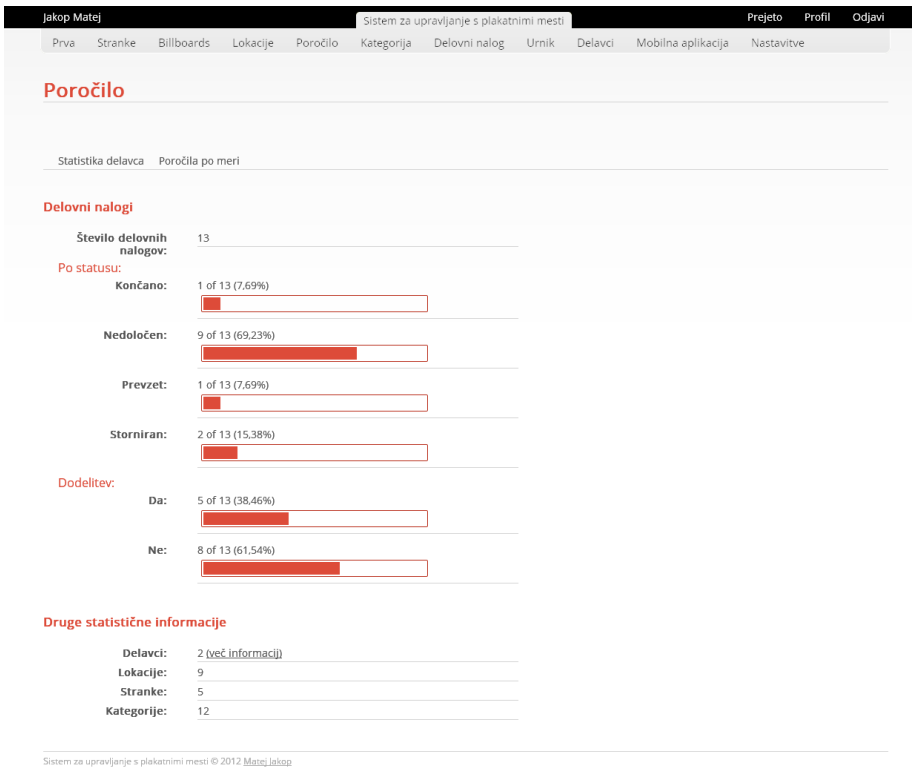


Slika 4.2: Prenos mobilne aplikacije preko QR kode.

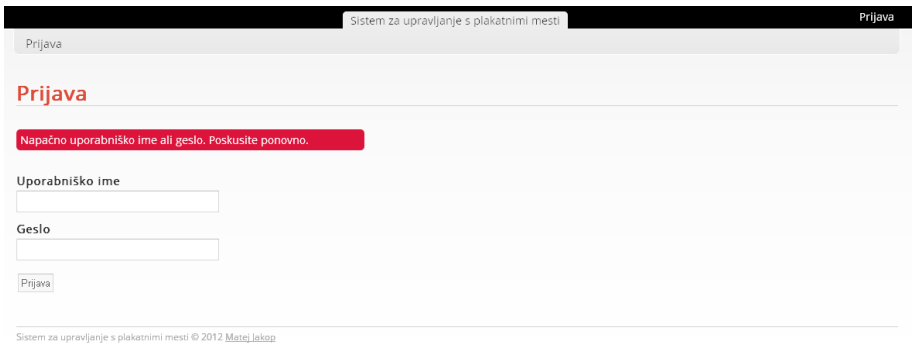
opravila potrebujemo dodaten program, ki bo to opravilo neodvisno izvedel v ozadju in po končanem izračunu obvestil uporabnika o opravljenem delu. V okviru naše rešitve smo uporabili Celery. Zanj smo se odločili, ker je spisan v programskem jeziku Python, močno povezan z Django in v uporabi v produkcijskih sistemih, ki izvajajo na milijone opravil na dan. Razporejevalnik je programska komponenta, spisana v programskem jeziku Java, ki je namenjena izračunu razporeda zasedenosti plakatnih mest. Več o njem v nadaljevanju.

4.2.1 Vmesnik API

Vmesnik API je namenjen komunikaciji med mobilno in strežniško aplikacijo. Aplikaciji med sabo komunicirata preko protokola HTTP, s katerim smo implementirali REST. Protokol HTTP smo izbrali zaradi več razlogov. Prvi razlog je, da je uporaba protokola enostavna. Drugi razlog za izbiro HTTP protokola je, da mora mobilna aplikacija komunicirati s strežniško



Slika 4.3: Poročila o dogajanju v sistemu.



Slika 4.4: Prijava v spletno aplikacijo za namene upravljanja sistema.

aplikacijo tudi kadar ni dostopne WiFi točke in za te razmere je potrebno upoštevati določene posebnosti mobilnega omrežja. Veliko mobilnih operaterjev namreč omogoča povezavo samo preko TCP/IP vrat 80 in blokira poskuse preko drugih. Če bi uporabili ne splošno sprejeti protokol ali katera druga vrata, bi lahko imeli probleme. Naslednji razlog za odločitev za uporabo HTTP protokola je ta, da obstaja ogromno število dobrih in manj dobrih HTTP strežnikov. Številčnost izbire in dejstvo, da so številni izmed njih testirani v produkciji pod hudimi obremenitvami, nam pove, da lahko z izbiro HTTP protokola prihranimo kar nekaj časa, ki bi ga sicer porabili za razvoj in testiranje lastnega strežnika z implementiranim lastnim protokolom. Za zagotavljanje ustrezne komunikacije je potrebna definicija vsebine, ki se prenaša med mobilno in strežniško aplikacijo. Pri načrtovanju aplikacije smo se odločili, da bodo odgovori, ki jih bo mobilna aplikacija prejemala od spletne aplikacije, zapisani v obliki JSON (angl. JavaScript Object Notation) [35]. Za to obliko zapisa smo se odločili, ker je oblika zapisa lahkotnejša [36] v primerjavi z drugimi pogostimi oblikami zapisa informacij (npr. XML). Pomembno težo pri odločitvi je imelo tudi dejstvo, da ni težav z razčlenjevanjem formata, ker je na voljo dovolj preizkušenih in dobrih programskih knjižnjic. Velikost zapisa ima pri komunikaciji preko mobilnega omrežja velik pomen. Večji kot je zapis, dlje časa čaka uporabnik na odgovor, kar prispeva k negativni uporabniški izkušnji. Da bi velikost zapisa še dodatno zmanjšali, smo se odločili, da bomo odgovor stisnili z GZIP algoritmom. Na tak način nam je uspelo zmanjšati velikost odgovora tudi do 70 %. Do sedaj je bilo že precej povedanega glede komunikacijskega protokola in vsebine. Manj pa je znanega o izvajanju zahteve mobilnega telefona do strežniške aplikacije. Kadar mobilna aplikacija potrebuje podatke od strežniške aplikacije ali ji mora nekaj sporočiti, najprej pri sebi pripravi vsebino zahteve zapisane v JSON formatu. Nato oblikuje ustrezen URL naslov in nanj poda zahtevo. Osnovna shema URI naslova za vse API klice je sestavljena na sledeči način:

```
http://naslov_streznika/api/?action=ime_api_klica
```

```
{
    "fields":{
        "revision":8536,
        "model":"Nexus S",
        "app_version_code":2,
        "app_version_name":"0.2",
        "phone_uid":"7b63413d3ad426dbf60cfbe8d759d904171d4005",
        "auth_key":"1
            adb64ef2adcd2353aebd3eae2ebad299341f7d3382a19391276239b
            ",
        "os_version":"Android 4.0.4"
    }
}
```

Koda 4.1: Primer parametrov podanih v JSON obliki pri izvajanju ukaza get_task_type.

Iz osnovne sheme je mogoče opaziti, da niso nikjer navadeni dodatni parametri, ki so potrebni pri določenih API klicih. Dodatni parametri se prenašajo kot vsebina v HTTP POST zahtevi (primer pod koda 4.1). Za ta način prenosa smo se odločili, ker bodo v prihodnosti določeni API klici zahtevali večjo količino parametrov, ki pa je ne bo več mogoče prenesti preko GET metode (pošiljanje preko metode GET je velikostno omejeno tako na strani brskalnikov kot na strani spletnih strežnikov, pri čemer velikost ni standardizirana). Naslednji razlog za to odločitev je, da se preko POST metode lahko pošlje poljubna vsebina. Nismo omejeni na pošiljanje podatkov zgolj v obliki ključ-vrednost. To postane pomembno dejstvo pri pošiljanju slik ali drugih binarnih (v tekstovni obliki nesmiselnih) podatkov. Zahteva preko dostopne točke na mobilnem telefonu (Wlan, mobilno omrežje) prispe do strežniške aplikacije. Strežniška aplikacija sprocesira zahtevo in vrne odgovor mobilni aplikaciji (primer odgovora pod koda 4.2).

```

{
    "status":0,
    "data":{
```

```
5         "types":[
6
7         ],
8         "revision":8536
9     }
10 }
```

Koda 4.2: Primer odgovora na izvedbo ukaza get_task_type.

Le-ta v skladu z odgovorom ustrezno ukrepa. Na opisan način lahko mobilna aplikacija dostopa do naslednjih funkcionalnosti:

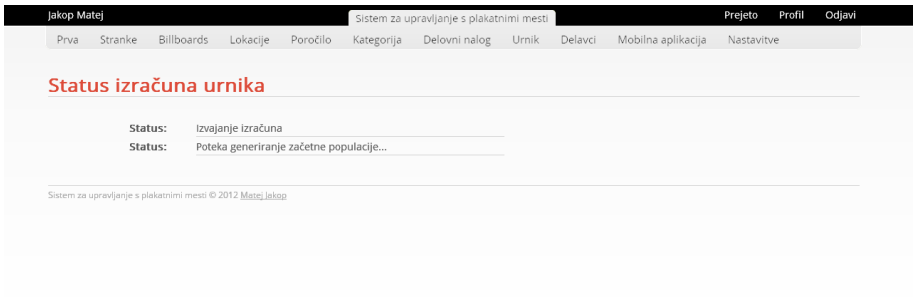
- prijava v sistem,
- prenos šifranta lokacij, statusov, tipov opravil, tipov polj, strank,
- prenos delovnih nalogov,
- pošiljanje izpolnjenih vrednosti delovnega naloga,
- spreminjanje statusa delovnega naloga,
- prevzemanje delovnega naloga in
- povezava lokacije z značko.

Zahteve, ki prihajajo na spletni strežnik je potrebno povezati z uporabnikom. Pošiljanje uporabniškega imena in gesla pri vsaki API zahtevi na strežnik bi bilo nevarno. V primeru, da nekdo uspe pridobiti naše uporabniško ime in geslo na podlagi prisluškovanja prometu ali preko katere izmed drugih metod napada, lahko z njima nemoteno dostopa tudi do spletnega vmesnika sistema za upravljanje z oglasnim prostorom. Zaradi tega se uporabniško ime in geslo pošljeta na strežnik samo pri prijavi (nekriptirano). V primeru, da je bila prijava uspešna, se poleg podatka o uspešnosti prijave v odgovoru na zahtevo mobilne aplikacije pošlje tudi niz znakov, imenovan API ključ, s pomočjo katerega se mora mobilna aplikacija v vseh API klicih identificirati kot določen uporabnik. Na ta način smo zmanjšali količino škode, ki

jo lahko napadalec povzroči v primeru uspešne kraje API ključa. Verjetnost kraje uporabniškega imena in gesla bi lahko še dodatno zmanjšali z uporabo SSL. Vendar se zaradi želje po hitrejšem odzivanju sistema in manjši količini prenesenih podatkov preko mobilnega omrežja, za uporabo SSL nismo odločili. Kljub temu je sistem zasnovan tako, da je mogoča uporaba SSL, v kolikor se pojavi potreba po dodatni varnosti. Razvoj in vzdrževanje sistema poskrbita za vedno nove različice aplikacije. Zaradi številnih različic aplikacije je potrebno poskrbeti za njihovo vodenje ter za način zavračanja uporabnikov, ki uporabljajo nekompatibilno različico. Da lahko strežniška aplikacija spremlja, katera verzija mobilne aplikacije je v uporabi pri uporabniku, se ob izvajanju prijave na strežnik poleg uporabniškega imena in gesla pošlje tudi različica aplikacije. V primeru, da ima uporabnik nameščeno staro nekompatibilno aplikacijo, sistem njegovo prijavo zavrne in uporabniku izpiše obvestilo o potrebni nadgradnji. Kadar pa sistem ugotovi, da uporabnik uporablja aplikacijo, ki je še vedno kompatibilna, a ne najnovejša, se uporabniku na mobilnem telefonu izpiše prošnja o čimprejšnji nadgradnji in dopusti nadaljne delo. Predvidevamo, da se bo aplikacija uporabljala na več različnih telefonih z različnimi različicami Android operacijskega sistema. Zaradi tega si ob prijavi na strežnik zabeležimo različico operacijskega sistema, model telefona in njegov unikatni id. Beleženje teh infomacij nam omogoča, da lahko tako vidimo koliko telefonov je v uporabi znotraj sistema, kakšen operacijski sistema imajo nameščen, kakšni modeli telefonov so prisotni, kdo vse jih uporablja in ali si uporabniki med sabo delijo mobilne telefone. Vse te informacije nam omogoča lažje svetovanje uporabnikom v primeru težav in v primeru kraje ali izgube mobilnega telefona blokado dostopa za mobilni telefon

4.2.2 Razporejevalnik

Razporejevalnik je komponenta strežniške aplikacije za pripravo razporeda zasedenosti plakatnih mest. Genetski algoritem v aplikaciji za izračun in izpis razporeda potrebuje seznam želja, predhodno zajetih preko spletnega



Slika 4.5: Prikaz statusa izračuna.

vmesnika. Seznam želja je podan v posebej formatirani datoteki, ki jo aplikacija ob zagonu prebere, nakar genetski algoritem opravi izračun ter rezultat ponovno zapiše v datoteko. Izračun običajno traja kar nekaj časa in je zaradi zagotavljanja dobre uporabniške izkušnje in omejitev spletnih aplikacij potrebno postopati pri čakanju na rezultat izračuna na malce bolj napreden način. Uporabnik po kliku na gumb izračunaj pričakuje takojšen odziv spletnega vmesnika, čeprav morda račun ni takoj končan. V rešitev smo implemetirali asinhrono izvajanje dalj časa trajajočih opravil. Po kliku na gumb izračunaj, se uporabniku odpre pogled (slika 4.5), preko katerega lahko v realnem času spremlja status izračuna. Ob prikazu strani se uporabnikov brskalnik poveže preko protokola Comet na kanal za obvestila. Prav tako se v ozadju na strežniku ob kliku na gumb izvede priprava zalednega opravila za kasnejšo izvedbo preko namenskega programa Celery. Opravilo ob izvajanju (ni nujno takoj, običajno v roku 1-2 sekund) zažene aplikacijo za izračun razporeda, počaka na njen izračun, shrani delovne naloge v podatkovno bazo ter o končanem izračunu obvesti uporabnika in ga napoti na ogled ustvarjenih delovnih nalogov.

4.2.3 Spletni vmesnik za upravljanje s sistemom

Spletni vmesnik za upravljanje s sistemom omogoča (za uporabo je potrebna prijava, prikaz vmesnika za prijavo je na sliki 4.4) upravljanje s celotnim sistemom. Preko spletnega vmesnika je možno uporabljati naslednje module:

1. Upravljanje s strankami: Uporabnik lahko dodaja, ureja in briše stranke. Vsaki stranki lahko določi prioriteto oziroma vrednost, ki jo ta stranka predstavlja podjetju.

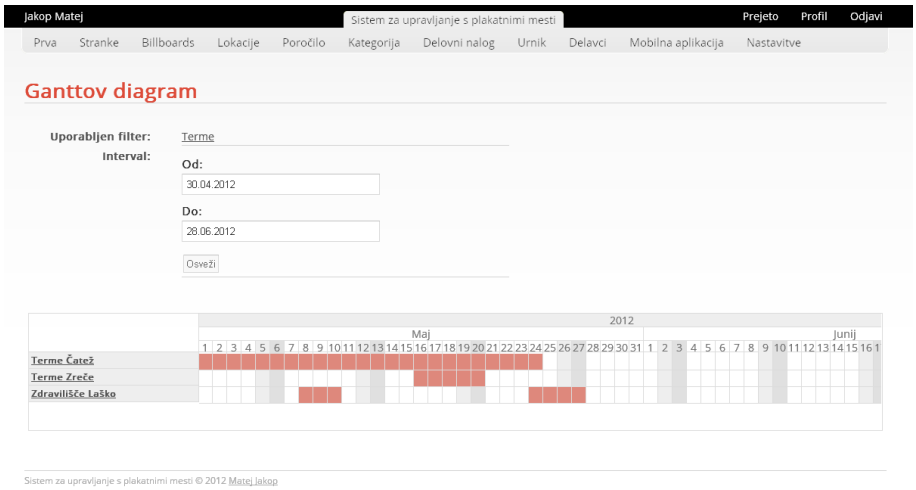
2. Upravljanje z lokacijami: Dodajanje, urejanje in brisanje lokacij. Vsaki lokaciji je možno določiti prioriteto lokacije in določiti kategorije, med katere spada lokacija (npr. družina, moški, ženske, otroci, trgovski centri, bencinske črpalke, terme ipd.). Možen je ogled seznama lokacij na več načinov: hierarhično, po kategorijah in preko seznama vseh lokacij.

3. Upravljanje s kategorijami: Dodajanje, urejanje in brisanje kategorij, ki se uporabljajo v sistemu za dodaten opis lokacij. Kategorija je lahko karkoli in ima pomembno vlogo pri upravljanju lokacij, izračunu razporeditve ipd.

4. Upravljanje z delovnimi nalogi: Dodajanje, urejanje in brisanje delovnih nalogov. Vsakemu delovnemu nalogu je možno določiti opravila in preveriti opravljenost opravil (pregled podatkov, ki jih je vnesel plakater), možno je pregledovati dodeljenost in stanje delovnega naloga.

5. Upravljanje s plakaterji: Vnos in urejanje plakaterjev. Za vsakega plakaterja je možno videti, kateri mobilni aparat uporablja in zgodovino uporabljenih mobilnih aparatov. Enostavno se lahko preveri, katera različica operacijskega sistema in aplikacije se izvajata na mobilnem telefonu in kdaj je uporabnik nazadnje uporabljal aplikacijo. Prav tako je možno za vsakega delavca določiti njegovo območje dela (slika 3.1 prikazuje vmesnik za upravljanje z delovnimi področji).

6. Upravljanje z urnikom oziroma časovno dodelitvijo lokacije stranki: Uporabnik lahko ročno vnese dodelitve oglasnega prostora neki stranki ali uporabi posebnega čarovnika, s katerim zajame zahteve stranke. Čarovniku poda stranko, obdobje in kategorije stranki zanimivih lokacij. S temi podatki čarovnik pripravi vse potrebno za kasnejši izračun



Slika 4.6: Ganttov diagram za pregledovanje zasedenosti lokacij.

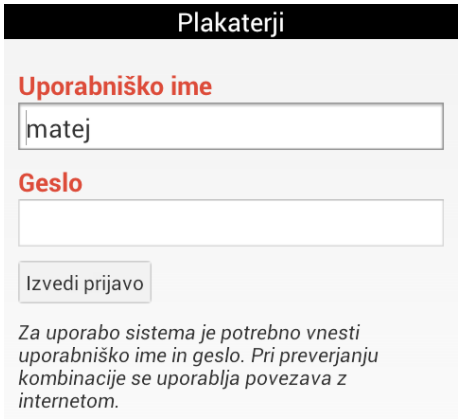
razporeditve. Znotraj sklopa je možno pregledovanje zasedenosti lokacij skozi obdobje. Zasedenost lokacij je mogoče opazovati preko Ganttovega diagrama [34] (slika 4.6 prikazuje primer pregleda) in enostavnega seznama.

7. Upravljanje z ostalimi nastavitvami sistema: Vnos raznih nastavitev sistema, kaj se zgodi ob določeni akciji, kakšen je privzeti status delovnih nalogov po prevzemu ipd.

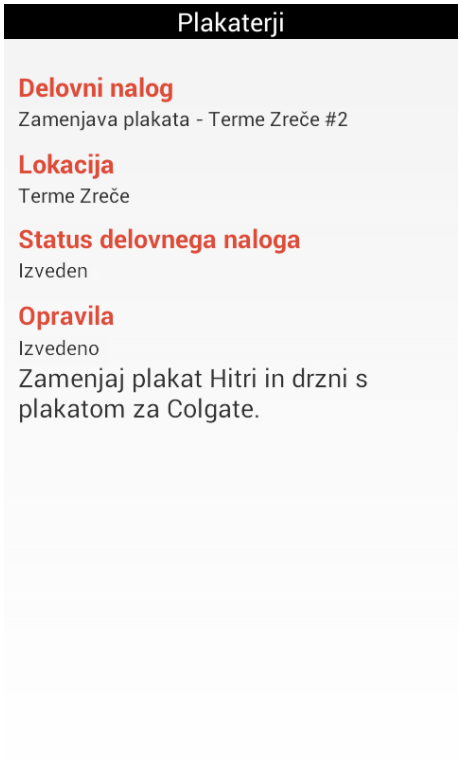
8. Pregledovanje poročila (slika 4.3)

9. Prenos in pregled mobilne aplikacije: Enostaven prenos in namestitev mobilne aplikacije zgolj s skeniranjem QR kode (slika 4.2), pregled opomb ob izdaji in možen prenos starejših različic aplikacije.

10. Pošiljanje in prejemanje privatnih sporočil: Pošiljanje in sprejemanje privatnih sporočil med uporabniki. V predal privatnih sporočil uporabnik prejema tudi vsa obvestila od sistema.



Slika 4.7: Začetni zaslon za prijavo v mobilno aplikacijo.



Slika 4.8: Prikaz podrobnosti delovnega naloga v mobilni aplikaciji.

4.3 Mobilna aplikacija

Mobilna aplikacija predstavlja povezovalni člen med plakaterjem in podjetjem. Plakaterju omogoča, da pregleduje in opravlja svoje delovne obveznosti podane v obliki delovnih nalogov za posamezno lokacijo. Mobilno aplikacijo smo zasnovali tako, da omogoča enostavno in smiselno delo tudi popolnemu začetniku.

4.3.1 Pregledovanje in izvrševanje delovnih nalogov

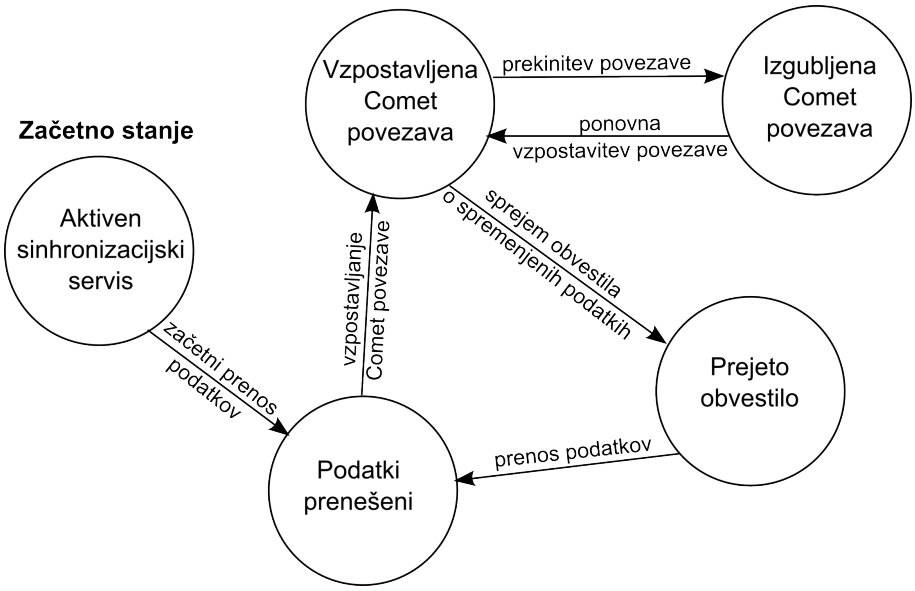
Po uspešni prijavi (slika 4.7) v aplikacijo se nam odpre seznam delovnih nalogov za prijavljenega plakaterja. Delovni nalogi so razvrščeni tako, da bo opravljena pot po izvedbi delovnih nalogov čim krajša.

Po izbiri delovnega naloga iz seznama ali preko dotika značke kot prikazuje slika 4.9, se uporabniku odprejo podrobnosti delovnega naloga (slika 4.8). Podrobnosti delovnega naloga vsebujejo podatke o stranki, lokaciji, roku za izvedbo delovnega naloga in seznam opravil v okviru delovnega naloga. Plakater na podrobnostih delovnega naloga upravlja s stanjem delovnega naloga ali izbere opravilo. V primeru, da izbere opravilo, se mu odpre nov pogled, v katerem so polja za izpolnitev pripravljena s strani odgovornih oseb v centrali. Plakater polja izpolni in zaključi z opravilom. Zaključitev opravila povzroči, da se zbrani podatki v najkrajšem možnem času, prenesejo na strežnik in postanejo vidni v centrali. Za zaključitev celotnega delovnega naloga, mora plakater spremeniti njegovo stanje.

4.3.2 Sinhronizacija podatkov

Mobilna aplikacija za svoje delovanje ne zahteva stalne povezave s centralnim strežnikom preko interneta. Po uspešni prijavi v mobilno aplikacijo je možno večino stvari opraviti v nepovezanem načinu. Za pravilno delovanje v nepovezanem načinu, je potrebno poskrbeti, da se v bazi na mobilnem telefonu nahajajo vsi podatki potrebni pri delu. Za usklajenost podatkov na mobilnem telefonu in centralnem strežniku skrbi sinhronizacija. Za uspešno izvedbo

Slika 4.9: Prikaz podrobnosti delovnega naloga po dotiku značke.



Slika 4.10: Diagram prehajanja stanj v procesu sinhronizacije.

implementacije sinhronizacije se je potrebno odločiti, kdaj bomo sinhronizacijo izvajali, katere podatke bomo sinhronizirali ter način sinhronizacije (se bo sinhronizacija izvajala glede na nek časovni interval, na zahtevo podano s strani uporabnika ali v realnem času?). V implemetaciji smo združili osveževanje na časovni interval in osveževanje v realnem času. Poskrbeli smo, da se ob izvajanju sinhronizacije prenašajo samo spremenjenji podatki. S tem smo dosegli hitrejšo in podatkovno manj potratno sinhronizacijo. Postopek sinhronizacije poteka na sledeči način (slika 4.10):

1. Ob uspešni prijavi uporabnika se izvede prenos vseh spremenjenih podatkov. S tem prenesemo vse podatke, ki so se spremenili v času, ko aplikacija na mobilnem telefonu ni komunicirala s spletnim strežnikom.

2. Po prenosu vseh spremenjenih podatkov servis, ki skrbi za sinhronizacijo podatkov, preklopi v način za realno-časovno sihronizacijo. To poteka tako, da se vzpostavi Comet povezava na Nginx. Uporabili smo način stalno odprte povezave. Komunikacija poteka po naslednjem pravilu. Ko strežnik dobi sporočilo za odjemalca mu ga nemudoma pošlje, drugače strežnik čaka. V primeru, da čaka več kot 30 sekund, potem odjemalcu pošlje prazno sporočilo, da ga obvesti, da je strežnik še aktiven in da je med njima še vedno aktivna povezava. Mobilna aplikacija ob prejemu sporočila dobi informacijo o tem, kateri podatki so se spremenili in jih je potrebno prenesti z uporabo ustreznega API klica. Na opisan način je omogočeno takojšnje osveževanje podatkov. Potrebni sta le 1-2 sekundi, da se spremenjeni podatek v centralni bazi pojavi v bazi na mobilnem telefonu.

3. Zgodi se, da pride do prekinitve internetne povezave. V tem primeru servis, ki skrbi za sinhronizacijo podatkov, preklopi na način intervalnih poskusov ponovne povezave na spletni strežnik. Ko mu uspe ponovno pridobiti povezavo, najprej naredi prenos vseh spremenjenih podatkov, nakar ponovno preklopi v način realno-časovne sinhronizacije opisane v 2. točki.

Problem, kdaj osveževati podatke, smo uspešno rešili na opisan način. Pojavilo pa se je novo vprašanje. Kako vedeti, katere podatke je potrebno prenesti? Kako strežnik ve, kateri podatki se nahajajo na mobilni aplikaciji? Pri iskanju rešitve problema, smo se zgledovali po sistemu za nadzor nad različicami izvorne kode – SVN (Subversion). Za vsak zapis v posamezni tabeli na strežniku smo uvedli številko revizije, bazo pa nastavili tako, da se pri spremembi zapisa v neki tabeli poveča globalna številka revizije in zapiše v stolpec spremenjenega zapisa. Za uspešno prenašanje samo spremenjenih podatkov je tako potrebna samo izmenjava revizijske številke med mobilno aplikacijo in strežnikom. S pomočjo revizijske številke mobilna aplikacija opiše svoje stanje. Postopek določitve, kateri podatki v okviru API klica se morajo prenesti, je tako razdeljen na naslednje korake:

1. Kot parameter v API klicu (primer v koda 4.1) je podana zadnja revizijska številka za izbrani API klic. V primeru, da se klic izvaja prvič, je vrednost parametra enaka 0.

2. Strežniška aplikacija izbere vse tiste zapise v neki tabeli, ki imajo revizijsko številko večjo kot je podana s strani aplikacije v API klicu in jih pošlje mobilni aplikaciji.

3. Mobilna aplikacija prebere vse zapise in poišče največjo revizijsko številko v paketu in si jo shrani v bazo kot zadnjo revizijsko številko za izbrani API klic.

Pravkar opisan postopek se izvede ob vsaki sinhronizacijo podatkov. Velja pa še eno dodatno pravilo. Šifranti se vedno prepišejo z lokalno verzijo na mobilnem telefonu, medtem ko podatki za posamezen izpolnjen delovni nalog ostanejo nespremenjeni. Pravilo velja zato, da se podatki, ki jih je uporabnik vnesel v delovni nalog ne izgubiju ob osveževanju. Teoretično je možno, da bi dva različna plakaterja spreminjala isti delovni nalog in en drugemu prepisovala podatke. Kljub temu da scenarij v praktični uporabi sistema ni mogoč, smo dodali opisano varovalko.

Poglavje 5

Rezultati

5.1 Primerjava algoritmov za določanje vsebovanosti točke znotraj poligona

V podpoglavju 3.1 smo predstavili dva algoritma, ki ju lahko uporabimo za ugotavljanje ali neka točka leži znotraj poligona ali ne. Kot smo omenili že pri sami predstavitvi, nas za praktično uporabo ne zanima v največji meri natančnost dodeljevanja. Bolj nam je pomembna sama hitrost izvajanja. Za določitev v praksi hitrejšega izmed algoritmov, smo izvedli testiranje na množicah velikosti 100, 500, 1.000, 5.000, 20.000, 100.000, 1.000.000, 2.000.000 in 3.000.000. Uporabili smo implemetacijo obeh algoritmov najdenih na [37] v programskem jeziku Python ter implementacijo usmerjanja žarka v knjižnici Shapely [38, 39], ki smo jo uporabili v končnem izdelku. Testiranje smo izvedli na računalniku z Intel Core 2 1.66 GHz procesorjem, z 2 GB glavnega pomnilnika ter operacijskim sistemom Ubuntu. Rezultati testiranja, zbrani v tabeli 5.1, nam povedo, da je implementacija algoritma, ki šteje število obkrožitev hitrejša kot implementacija z usmerjanjem žarka. Nobena od implementacij pa se ne more kosati z implementacijo v knjižnici. Razlog za hitro delovanje knjižnjice je, da je knjižnica Shapely [38] v svojem jedru implementirana v programskem jeziku C in zaradi tega občutno hitrejša od implementacij v interpretativnem jeziku. Odločili smo se, da zaradi hi-

Tabela 5.1: Rezultati testiranja hitrosti delovanja algoritmov za ugotavljanje ali je točka znotraj poligona.

N	Usmerjanje žarka [sec]	Alg. števila obkrožitev [sec]	Shapely [sec]
100	0,000184	0,000119	7,7e-05
500	0,000811	0,000494	7,8e-05
1000	0,0013	0,001012	7,2e-05
5000	0,006284	0,004517	0,000104
20000	0,025965	0,021931	9e-05
100000	0,11349	0,090039	0,000106
1000000	1,333774	1,099509	8,1e-05
2000000	2,432915	1,889606	6,5e-05
3000000	4,091906	3,46197	84e-05

trostnih razlogov v končni implementaciji uporabimo knjižnjico Shapely.

5.2 Primerjava algoritmov za izbiro vrstnega reda obiskovanja lokacij

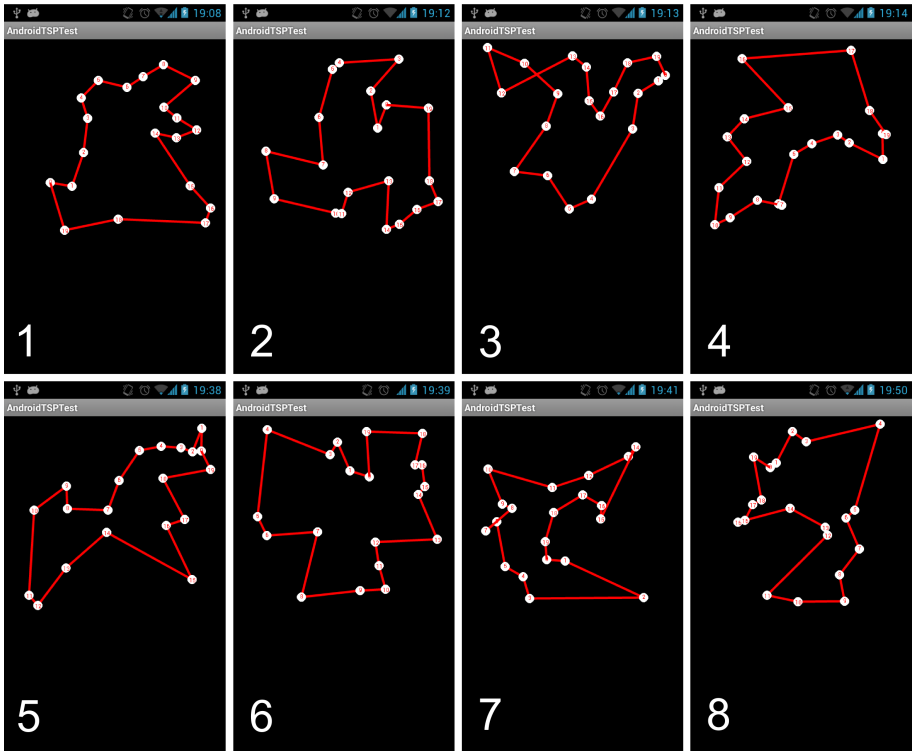
Pri načrtovanju funkcionalnosti sistema, ki določa vrstni red izvrševanja delovnih nalogov v odvisnosti od trenutne lokacije plakaterja, se je bilo potrebno odločiti, kateri algoritem bomo uporabili. V poglavju namenjenemu predstavitvi sistema, smo samo omenili izbrani algoritem, nismo pa nič povedali o razlogih za to odločitev.

V postopku testiranja smo med sabo primerjali dva algoritma, algoritem najbližji sosed in optimizacijo s kolonijami mravelj. Vsak izmed algoritmov je nastopal v več različicah:

1. **Min**: Vrednost predstavlja absolutno spodnjo mejo razdalje.

2. **NN**: Osnovna verzija algoritma najbližji sosed.

3. **NN2**: Osnovna verzija algoritma najblizji sosed z izboljšavo rezultata s pomočjo Lin-Kernighan hevristike.



Slika 5.1: Slika prikazuje osem testnih množic lokacij z izračunanim razultatom obiskovanja posameznih lokacij preko algoritma ACO2.

4. **ACO**: Osnovna verzija optimizacije s kolonijami mravelj. Vrednosti α in β sta enaki 1.

5. **ACO2**: Enako kot ACO vendar z dodatno optimizacijo poti s pomočjo Lin-Kernighan hevristike.

6. **ACOS**: Verzija optimizacije s kolonijami mravelj, kjer sta vrednosti α in β izbrani za bolj raziskovalno naravnanost agentov. Vrednost α je 1,6; vrednost β pa 0,4.

7. **ACOS2**: Enako kot ACOS. Dodatna optimizacija rezultata s pomočjo Lin-Kerninghan hevristike.

8. **ACOL**: Verzija optimizacije s kolonijami mravelj, kjer sta vrednosti alfa in β izbrani tako, da imajo večjo težo krajše povezave. Vrednost α je enaka 0,4, vrednost β pa 1,6.

9. **ACOL2**: Enako kot ACOL. Dodatna optimizacija rezultata s pomočjo Lin-Kerninghan hevristike.

Izbrani algoritem se bo v sistemu izvajal na mobilnem telefonu. Zaradi tega smo že v fazi testiranja poskrbeli za testiranje na mobilnem telefonu. Uporabili smo mobilni telefon Google Nexus S z operacijskim sistemom Android 4.1.1.

Testiranje smo izvedli na večih različnih množicah podatkov. Na sliki 5.1 so izrisane nekatere izmed testnih množic. Testirali smo tako, da smo iz vsake izmed testnih množic (vsaka izmed njih je vsebovala 20 lokacij) postopoma jemali posamezne lokacije in gradili pot obhoda. Tako smo na vsaki izmed množic zgradili poti za 2 mesti, 3 mesta itd. Rezultate za prvo testno množico smo povzeli v tabeli 5.2, rezultate za drugo testno množico pa v tabeli 5.3. Po posameznih lihih vrsticah so zapisane dolžine poti za posamezen algoritem pri N številu mest, v sodih vrsticah pa so zapisani časi izvajanja v sekundah. Odebeljeno smo prikazali zapise za algoritme, ki so se pri določenem številu mest najbolje odrezali. Na dnu tabele smo naredili povzetek za lažjo ocenitev uspešnosti algoritmov. Vključili smo tri količine:

1. **Naj** - predstavlja število najboljših uvrstitev algoritma (prvo mesto).

2. **Odm** - predstavlja povprečen faktor odmika od absolutne spodnje meje dolžine poti.

$$Odm = \frac{\sum_{i=1}^{N} \frac{alg(i)}{min(i)}}{N} \tag{5.1}$$

3. **Pov** - predstavlja povprečen čas izračuna na posamezno iteracijo. Vrednost je zapisana v milisekundah.

$$Pov = \frac{\sum_{i=1}^{N} time(i)}{N} \tag{5.2}$$

Na podlagi dodatno uvedenih količin lahko algoritme razvrstimo glede na uspešnost pri posamezni količini. Pri razvrstitvi smo poleg rezultatov v tabeli 5.2 in tabeli 5.3 upoštevali tudi rezultate testiranj, ki jih zaradi količine zavzetega prostora nismo vključili v diplomsko delo. Posamezne rezultate smo združili na naslednji način. Algoritem za prvo mesto v kategoriji dobi 10 točk, drugi 9, tretji 8 itd. V primeru, da si dva algoritma delita mesto, dobita oba enako število točk, naslednje mesto pa se preskoči (npr. algoritma si delita drugo mesto, naslednje mesto uvrstitve je četrto). Podobno kot na določenih športnih tekmovanjih. Rezultati združeni na pravkar opisan način, so pokazali uspešnost delovanja, kot je prikazano v tabeli 5.4.

Iz tabele 5.4 je razvidno, da je v kategorijah Naj in Odm najbolj uspešen algoritem ACO2. Na podlagi vrstnega reda ostalih algoritmov, je moč sklepati, da ima na dolžino izračunanih poti Lin-Kernighan hevristika precejšnjo vlogo. Vse osnovne različice posameznega osnovnega algoritma so slabše od verzije, ki še dodatno uporablja Lin-Kernighan hevristiko. Rezultati nam sporočajo tudi, da se lene mravlje (algoritma ACOL in ACOL2) odrežejo bolje kot tiste, ki veliko raziskujejo (algoritma ACOS in ACOS2). Pri razporeditvi po hitrosti izvajanja je vrstni red popolnoma drugačen. Najboljši je NN, ki je najbolj preprost algoritem izmed vseh, ki smo jih testirali. Sledi mu njegova izboljšava, nekje daleč zadaj pa so algoritmi zasnovani na optimizaciji s kolonijami mravelj.

Tabela 5.2: Tabela prikazuje rezultate izvajanja algoritmov na prvi množici.

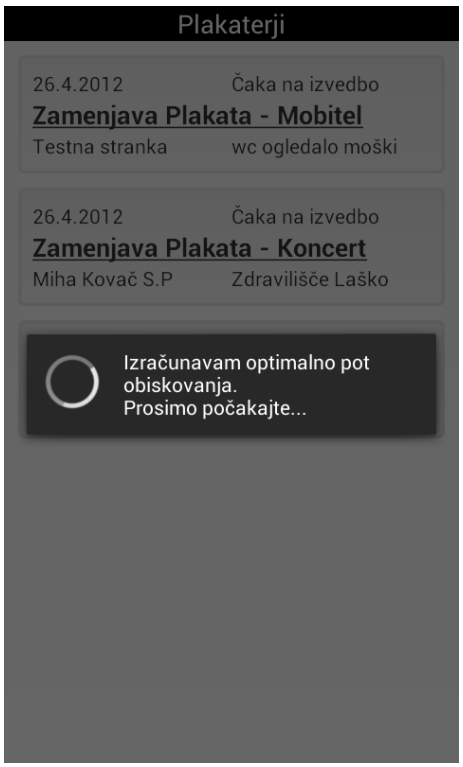
N	Min	NN	$NN2$	ACO	$ACO2$	$ACOS$	$ACOS2$	$ACOL$	$ACOL2$
		$T[s]$	$T[s]$	$T[s]$	$T[s]$	$T[s]$	$T[s]$	$T[s]$	$T[s]$
2	768	**768**	**768**	**768**	**768**	**768**	**768**	**768**	**768**
		0,001	*0,001*	*0,009*	*0,009*	*0,006*	*0,007*	*0,004*	*0,004*
3	586	**903**	**903**	**903**	**903**	**903**	**903**	**903**	**903**
		0,0	*0,0*	*0,014*	*0,014*	*0,019*	*0,02*	*0,013*	*0,013*
4	744	**1061**	**1061**	**1061**	**1061**	1124	1124	1124	1124
		0,0	*0,0*	*0,021*	*0,021*	*0,015*	*0,015*	*0,013*	*0,014*
5	805	**1093**	**1093**	**1093**	**1093**	**1093**	**1093**	**1093**	**1093**
		0,0	*0,0*	*0,04*	*0,041*	*0,026*	*0,027*	*0,022*	*0,023*
6	791	**1095**	**1095**	**1095**	**1095**	**1095**	**1095**	**1095**	**1095**
		0,0	*0,001*	*0,085*	*0,087*	*0,044*	*0,045*	*0,035*	*0,036*
7	798	1244	1244	**1169**	**1169**	1330	1276	**1169**	**1169**
		0,0	*0,002*	*0,036*	*0,037*	*0,067*	*0,067*	*0,046*	*0,046*
8	797	1257	1257	**1198**	**1198**	1257	1257	**1198**	**1198**
		0,001	*0,004*	*0,108*	*0,108*	*0,063*	*0,064*	*0,09*	*0,09*
9	812	1421	1408	**1242**	**1242**	1415	1301	**1242**	**1242**
		0,001	*0,003*	*0,131*	*0,132*	*0,107*	*0,11*	*0,22*	*0,223*
10	930	1366	1366	**1307**	**1307**	1560	1427	**1307**	**1307**
		0,001	*0,002*	*0,191*	*0,192*	*0,214*	*0,216*	*0,183*	*0,183*
11	954	1391	1391	1368	1368	1810	**1332**	**1332**	**1332**
		0,001	*0,002*	*0,195*	*0,195*	*0,206*	*0,208*	*0,292*	*0,293*
12	899	1391	1391	**1332**	**1332**	1368	1368	**1332**	**1332**
		0,001	*0,002*	*0,192*	*0,193*	*0,291*	*0,292*	*0,305*	*0,305*
13	866	1395	1395	**1336**	**1336**	1693	1448	1353	1353
		0,001	*0,002*	*0,319*	*0,32*	*0,301*	*0,303*	*0,395*	*0,396*
14	955	1459	1424	**1337**	**1337**	1884	1399	**1337**	**1337**
		0,001	*0,004*	*0,612*	*0,613*	*0,43*	*0,435*	*0,587*	*0,588*
15	984	1749	1590	**1427**	**1427**	2148	1453	1568	1470
		0,001	*0,004*	*0,8*	*0,801*	*0,733*	*0,739*	*0,479*	*0,482*
16	952	1560	1525	**1429**	**1429**	1868	1763	1464	1464
		0,001	*0,005*	*0,735*	*0,737*	*1,274*	*1,279*	*0,83*	*0,831*
17	888	1575	1438	**1429**	**1429**	1870	1638	1531	**1429**
		0,001	*0,007*	*0,86*	*0,862*	*0,935*	*0,939*	*1,128*	*1,132*
18	842	1577	**1440**	1448	1448	1846	1530	1500	1500
		0,002	*0,007*	*1,181*	*1,183*	*1,24*	*1,245*	*1,274*	*1,276*
19	888	1647	**1508**	1521	1521	2073	1651	1587	1521
		0,002	*0,008*	*1,527*	*1,531*	*1,407*	*1,416*	*2,08*	*2,086*
20	950	1693	**1554**	1577	1575	1841	1746	1603	1601
		0,002	*0,009*	*2,823*	*2,829*	*3,445*	*3,45*	*2,182*	*2,188*
Naj		5	8	15	15	4	5	11	12
Odm		1,574	1,528	1,48	1,48	1,764	1,571	1,508	1,493
Pov		0,902	3,271	519,955	521,305	569,641	572,503	535,669	537,377

Tabela 5.3: Tabela prikazuje rezultate izvajanja algoritmov na drugi množici.

N	Min	NN	$NN2$	ACO	$ACO2$	$ACOS$	$ACOS2$	$ACOL$	$ACOL2$
		$T[s]$	$T[s]$	$T[s]$	$T[s]$	$T[s]$	$T[s]$	$T[s]$	$T[s]$
2	240	**240**	**240**	**240**	**240**	**240**	**240**	**240**	**240**
		0,0	*0,001*	*0,008*	*0,008*	*0,005*	*0,005*	*0,008*	*0,008*
3	418	**585**	**585**	**585**	**585**	**585**	**585**	**585**	**585**
		0,0	*0,001*	*0,025*	*0,025*	*0,031*	*0,033*	*0,029*	*0,03*
4	375	615	615	**585**	**585**	**585**	**585**	**585**	**585**
		0,001	*0,001*	*0,048*	*0,048*	*0,017*	*0,017*	*0,016*	*0,016*
5	443	825	**783**	825	**783**	**783**	**783**	813	801
		0,0	*0,001*	*0,021*	*0,022*	*0,048*	*0,049*	*0,025*	*0,026*
6	620	1099	1046	1069	**1028**	**1028**	**1028**	1058	1046
		0,0	*0,003*	*0,2*	*0,201*	*0,09*	*0,094*	*0,059*	*0,06*
7	561	1158	1123	1128	**1087**	1129	**1087**	1129	**1087**
		0,0	*0,002*	*0,087*	*0,088*	*0,057*	*0,058*	*0,055*	*0,057*
8	636	1179	**1151**	1163	**1151**	1198	**1151**	1163	**1151**
		0,002	*0,009*	*0,122*	*0,122*	*0,075*	*0,079*	*0,104*	*0,105*
9	578	1246	1234	1198	1186	1329	**1175**	**1175**	**1175**
		0,0	*0,002*	*0,192*	*0,193*	*0,151*	*0,152*	*0,18*	*0,18*
10	598	1273	1266	**1164**	**1164**	1302	1251	1179	1179
		0,0	*0,002*	*0,265*	*0,266*	*0,259*	*0,26*	*0,261*	*0,262*
11	659	1391	1321	1241	**1234**	1320	1320	1241	**1234**
		0,001	*0,002*	*0,303*	*0,305*	*0,305*	*0,306*	*0,206*	*0,208*
12	748	1432	1337	**1222**	**1222**	1391	1265	**1222**	**1222**
		0,001	*0,004*	*0,512*	*0,513*	*0,339*	*0,342*	*0,342*	*0,343*
13	863	1438	1383	1368	**1296**	1562	1348	1323	1319
		0,002	*0,003*	*0,579*	*0,582*	*0,353*	*0,357*	*0,488*	*0,491*
14	908	1493	1438	**1351**	**1351**	1459	**1351**	1441	1402
		0,002	*0,004*	*0,696*	*0,697*	*0,541*	*0,545*	*0,482*	*0,487*
15	980	1544	1509	1554	**1478**	1748	1509	1548	1507
		0,001	*0,007*	*0,556*	*0,56*	*0,742*	*0,746*	*0,579*	*0,582*
16	969	1545	1510	**1404**	**1404**	1841	1456	1480	1478
		0,001	*0,009*	*1,635*	*1,636*	*0,9*	*0,909*	*0,905*	*0,909*
17	1026	1600	1565	1488	**1486**	1812	1525	1583	1582
		0,001	*0,01*	*0,826*	*0,83*	*1,086*	*1,094*	*0,758*	*0,767*
18	1067	1719	1636	1628	1604	1993	1726	1628	**1577**
		0,002	*0,016*	*0,944*	*0,95*	*1,358*	*1,368*	*1,5*	*1,506*
19	1155	1764	1681	1622	1622	2157	1903	1696	**1603**
		0,002	*0,015*	*0,871*	*0,873*	*1,535*	*1,546*	*1,248*	*1,253*
20	1190	1827	1701	**1604**	**1604**	2079	1738	1751	1650
		0,002	*0,039*	*2,019*	*2,022*	*2,01*	*2,019*	*1,856*	*1,865*
Naj		2	4	8	16	5	9	5	10
Odm		1,717	1,662	1,619	1,594	1,788	1,645	1,637	1,613
Pov		1,005	6,987	521,445	523,192	521,297	525,201	479,023	481,809

Tabela 5.4: Tabela prikazuje razvrstitev uspešnosti delovanja algoritmov na podlagi treh ocenitvenih funkcij.

	1.	2.	3.	4.	5.	6.	7.	8.
Naj	ACO2	ACOL2	ACO	ACOL	NN2	ACOS2	NN	ACOS
Odm	ACO2	ACOL2	ACO	NN2	ACOL	ACOS2	NN	ACOS
Pov	NN	NN2	ACO	ACO2	ACOS	ACOS2	ACOL	ACOL2



Slika 5.2: Obvestilo prikazano med izračunom optimalne poti na mobilnem telefonu.

Na podlagi analize rezultatov smo se odločili, da v končni implementaciji sistema uporabimo algoritem ACO2. Zanj smo se odločili zaradi njegovih odličnih uvrstitev v kategorijah *Naj* in *Odm* prikazanih v tabeli 5.4. Algoritem je kljub dobri kvaliteti delovanja še vedno dovolj hiter, da je primeren za izvajanje tudi na mobilnem telefonu v realnem času. Izračun za dvajset lokacij na mobilnem telefonu Google Nexus S z operacijskim sistemom Android 4.1.1 porabi okoli 1,5 sekunde. Slika 5.2 prikazuje dogajanje na zaslonu mobilnega telefona tekom izračuna.

5.3 Variacije algoritma za razporejanje zasedenosti plakatnih mest

Avtomatsko razporeditev zasedenosti plakatnih mest smo implementirali s pomočjo genetskega algoritma. Za uspešno uporabo genetskega algoritma je potrebno sprejeti nekaj pomembnih odločitev. Potrebno je uporabiti prave metode za operaciji križanja in mutacije. Katere so primerne za praktično uporabo smo ugotovili s pomočjo testiranja kombinacij metod definiranih v prejšnjih poglavjih.

Za testiranje smo si zamislili realni scenarij v katerem nastopajo štiri stranke in šest lokacij (trafika 1 in 2, gostišče 1 in 2 ter banka 1 in 2). Vsaka izmed strank ima svoje želje in muhe, ki smo jih strnili v naslednjih 7 želja:

- **W11**: stranka 1, oglasno mesto v trafiki ali banki, željeno obdobje od 2. 9. 2012 do 6. 9. 2012 za 3 dni,
- **W12**: stranka 1, oglasno mesto v gostišču 1, željeno obdobje od 7. 9. 2012 do 9. 9. 2012 za 2 dni,
- **W21**: stranka 2, oglasno mesto v banki, željeno obdobje od 1. 9. 2012 do 14. 9. 2012 za 3 dni,
- **W22**: stranka 2, oglasno mesto v gostišču ali trafiki ali banki, željeno obdobje od 9. 9. 2012 do 14. 9. 2012 za 4 dni,

- **W31**: stranka 3, oglasno mesto v banki, željeno obdobje od 3. 9. 2012 do 5. 9. 2012 za 2 dni,
- **W41**: stranka 4, oglasno mesto v gostišču 1, željeno obdobje od 1. 9. 2012 do 14. 9. 2012 za 5 dni,
- **W42**: stranka 4, oglasno mesto trafiki, željeno obdobje od 3. 9. 2012 do 8. 9. 2012 za 2 dni.

Testiranje smo izvedli za tri metode križanja kot je opisano v poglavju 3. Pri tem smo privzeli uporabo mutacij na način opisan v podpoglavju 3.3.1. Vse metode smo testirali na populacijah velikosti 50, 100, 200, 500, 800, 900, 1000 in 1100. Vsak izračun metode smo na posamezni populaciji ponovili 15×. Pri tem smo opazovali minimalno število generacij ($MinG$), povprečno število generacij ($AvgG$), maksimalno kvaliteto ($MaxQ$), povprečno kvaliteto ($AvgQ$), število neuspehov izračuna (F, npr. populacija je izumrla, nismo dobili rezultata), uspešnost pridobitve rezultata (s) in čas izvajanja (T). Testiranje smo izvedli na računalniku z Intel Core 2 1.66 GHz procesorjem, z 2 GB glavnega pomnilnika ter operacijskim sistemom Ubuntu. Rezultate testiranja smo povzeli v tabeli 5.5. Iz tabele najprej opazimo, da nobena od metod ne dosega 100% zanesljivosti pridobitve rezultata. To je posledica dejstva, da so nekatere mutacije za osebek morda usodne. Če med sabo primerjamo povprečno število generacij za pridobitev rezultata, ugotovimo, da nobena od metod križanja po tej metriki ne izstopa. Drugače pa je, če pogledamo metriko maksimalne kvalitete in povprečne kvalitete. Tukaj metoda 3 močno izstopa. Ima zelo nizke povprečne vrednosti (manjše je vrednost, boljša je - glej podpoglavje 3.3.1) in izmed vseh metod je našla najboljši razpored rezervacij. Metoda 1 in metoda 2 sta si dokaj ekvivalentni. Odločitev pri metodi 2, da izbiramo boljšo rezervacijo, ne prinese veliko k izboljšavi delovanja. Pogled na stolpca F in S razkrije, da imajo vse metode nekaj težav pri iskanju rešitev v primeru manjše začetne populacije. S prehodom preko meje 500 osebkov v začetni populaciji, se uspešnost delovanja vseh verzij algoritma izboljša. Primerjava časa izvajanja metod pri isti veli-

kosti začetne množice nam razkrije, da ni bistvenih razlik med posameznimi metodami. Vse so si dokaj blizu.

V končni implementaciji smo uporabili metodo 3, saj se je na podlagi testiranj obnesla najboljše z vidika kvalitete podane rešitve. To nam pri reševanju problema največ pomeni, saj tako dobimo boljšo razporeditev plakatnih mest.

Tabela 5.5: Rezultati testiranja različnih metod križanja genov na različnih velikostih začetne populacije.

Metoda	M	$MinG$	$AvgG$	$MaxQ$	$AvgQ$	F	S	T[s]
Metoda 1	50	154	154	31	31	14	6,6 %	0,227
	100	51	116,5	32	33,75	11	26 %	0.237
	200	51	83,75	31	37	11	26 %	0.337
	500	51	462	22	33,76	2	86,67 %	0,652
	800	51	72,16	20	28,0	3	80 %	1,278
	900	51	106,8	22	30,3	5	66 %	1,409
	1000	52	96	20	25,35	1	93 %	1,550
	1100	51	104	20	24,07	2	86 %	1,666
Metoda 2	50	51	64,5	29	34,5	13	13,3 %	0,179
	100	51	51	27	30,0	12	20 %	0,236
	200	51	155	27	34,44	6	60 %	0,298
	500	51	63,12	19	26,0	7	53,33 %	0,805
	800	51	154,57	22	32	1	93,3%	1,355
	900	51	84,85	20	26,21	1	93,33 %	1,379
	1000	51	63,9	20	27	5	66,6 %	1,431
	1100	51	61,57	18	27,35	1	93,33 %	1,622
Metoda 3	50	52	131,33	21	21,33	12	20 %	0,202
	100	52	80,8	19	21,0	10	33,33 %	0,242
	200	59	83,3	19	20,66	12	20,0 %	0,381
	500	52	88	20	24,16	9	40 %	0,898
	800	52	113,0	17	19,91	3	80 %	1,470
	900	52	139,45	17	22,27	4	73,33 %	1,555
	1000	52	101,21	17	20,57	1	93,22 %	1,674
	1100	52	293,08	18	21,33	3	80 %	2,075

Poglavje 6

Zaključek

6.1 Prednosti in slabosti

Pri razvoju sistema za upravljanje s plakatnimi mesti je bilo potrebno sklepati kompromise za dosego željenih funkcionalnosti. Kompromisi poleg številnih prednosti prinesejo tudi nekaj slabosti.

6.1.1 Prednosti

Analiza sistema razkrije številne prednosti, ki jih sistem prinaša uporabnikom. Največja prednost sistema je v tem, da nudi računalniško podprto poslovanje, ki prinaša manjše stroške in boljši pregled nad poslovanjem. S pomočjo sodobnih komunikacij in uporabe sistema razvitega v okviru diplomskega dela lahko podjetje prihrani na potnih in materialnih stroških ter navsezadnje na času potrebnem za vsakdanja opravila. Prihranek na materialnih in potnih stroških neposredno sledi iz dejstva, da delovni nalogi prihajajo neposredno na mobilni telefon plakaterja. Ni potrebe po obisku podjetja zgolj za prevzem delovnega naloga na papirju. Plakater se lahko tako v določenih primerih odpravi samo na prevzem potrebnega gradiva k tiskarju. Dodatna prednost sistema je, da lahko na podlagi seznama preostalih neopravljenih delovnih nalogov sistem predlaga plakaterju vrstni red zamenjave, ki naj bi na koncu prinesel nižjo skupno opravljeno dolžino vožnje. S

pomočjo računalniško podprtih delovnih nalogov odpade dolgotrajno prepisovanje ročno vnesenih podatkov iz delovnih nalogov na računalnik. To prinese s sabo ogromen prihranek na porabljenem času. Sistem poskuša s pametnim načrtovanjem razporeditve zasedenosti plakatnih mest poskrbeti za večje zadovoljstvo in uspešnost strank pri oglaševanju brez dodatno vloženega dela s strani uporabnika. Preko dodatnega opisa strank in komentarjev je mogoče spremljati celotno zgodovino dela s stranko, kar s sabo prinese še dodatne pozitivne učinke, čeprav gre v osnovi za zelo oklesten sistem za upravljanje s strankami (angl. CRM). Uporaba sistema za upravljanje ni omejena na določeno strojno ali programsko opremo. Zaradi spletnega vmesnika je možno do sistema dostopati kjerkoli, kadarkoli in s poljubne naprave.

6.1.2 Slabosti

Implementacija sistema za upravljanje s plakatnimi mesti razvita v okviru diplomskega dela v sebi skriva nekaj slabosti, ki pa ne vplivajo na uporabnost rešitve. Na slabosti rešitve lahko pogledamo z različnih vidikov, tako iz ekonomskega, tehničnega, akademskega in še katerega drugega.

Če ocenimo rešitev z ekonomskega vidika pridemo do sklepa, da odločitev za uporabo rešitve prinese s sabo kar nekaj začetnih stroškov. Najprej je potrebno plakaterjem zagotoviti android NFC mobilne telefone. Glede na stanje trga mobilnih aparatov v času izdelave diplomskega dela to predstavlja kar visoko ceno. Drug dokaj velik strošek pri postavitvi sistema so NFC značke. Cena ene značke je še vedno dokaj visoka v primerjavi s tiskom identifikacijske nalepke.

Tehnična ocena rešitve razvite v diplomskem delu razkrije nekaj pomanjkljivosti, ki pa za praktično uporabo niso problematične in se bodo s časom večinoma samodejno odpravile. Pri načrtovanju rešitve smo privzeli, da bo GSM signal z možnostjo povezave na internet povsod prisoten. V realnosti obstaja kar nekaj lokacij, kjer komunikacija ni mogoča tudi v gosto naseljenih mestih. Primeri lokacij, kjer komunikacija ni mogoča, so npr. podzemne garaže brez dodatno nameščenih anten, kleti ipd. Dobra stran oglaševanja

je, da je uporabnost takšnih lokacij minimalna in ne predstavlja ovire za uporabo sistema. V primeru, da se plakater kljub temu znajde v takšni situaciji, lahko svoje delo še vedno opravi. Razlika je samo v tem, da bodo podatki odgovornim delavcem v centrali na voljo kasneje. Večje probleme kot nedostopnost mobilnega omrežja predstavlja nedostopnost GPS signala. Le-ta je nedostopen znotraj stavb. Tako je znotraj stavb onemogočeno natančno lociranje plakatnega mesta z uporabo GPS. Naslednji problem je poraba baterije na mobilnem telefonu. Lahko se zgodi, da v primeru večurne uporabe baterija mobilnega telefona opeša. Opisani problem je enostavno rešljiv. Lahko uporabimo dodatno baterijo ali uporabimo avtopolnilec.

Pogled z akademsko-raziskovalnega vidika nam razkrije, da imajo implementacije algoritmov uporabljenih v sistemu nekaj slabosti. Ena izmed slabosti je hitrost izvajanja algoritmov. Z raziskovanjem bi lahko ugotovili kako posamezne implementirane algoritme pospešiti. Kar nekaj hitrostne rezerve je še pri iskanju rešitve za problem trgovskega potnika. Večji problem kot je sama hitrost izvajanja posameznih algoritmov, predstavlja dejstvo, da trenutno implementirani genetski algoritem, v primeru nemogočih pogojev oziroma želja, ne more podati odgovora na vprašanje, kaj je potrebno spremeniti, da bo realizacija želja strank mogoča in čim boljša.

6.2 Nadaljnje delo

Razvoj programske rešitve se nikoli ne konča, pa naj si bo programska rešitev enostavna ali malce zahtevnejša. Tudi v primeru sistema opisanega v diplomskem delu so možne številne izboljšave.

6.2.1 Mobilna aplikacija

Mobilna aplikacija deluje dobro, bi pa lahko zanjo pripravili še številne izboljšave. Izboljšave so možne tako na uporabniškem vmesniku kot z vidika funkcionalnosti. V obstoječi aplikaciji je pregled delovnih nalogov narejen v obliki seznama. Precej boljše bi bilo, če bi se delovni nalogi prikazovali na

zemljevidu. Vsak delovni nalog bi bil na zemljevidu označen s pikico, pikice pa bi bile med sabo povezane v vrstnem redu obiskovanja. Opisano prikazuje slika 6.1. Zelo uporabna funkcionalnost bi bila, če bi odgovorne osebe v sistemu lahko dodeljevale pravice za delo posameznemu plakaterju. Tako bi se določene servisne funkcionalnosti (npr. povezava lokacije in značke) omejile za uporabo samo določenim plakaterjem. Celotno idejo uporabe pravic bi bilo možno speljati še dalje. V mobilno aplikacijo bi se lahko dodala funkcionalnost ustvarjanja novih delovnih nalogov. To bi bilo koristno v primerih, ko bi delavec na terenu ugotovil, da je plakatno mesto poškodovano, sam pa ga ni zmožen popraviti. Preko izdelave novega delovnega naloga bi tako obvestil druge delavce. Opisana funkcionalnost je bolj napredna, zato bi bilo potrebno nadzorovati, kdo lahko kreira delovni nalog iz posameznih predlog (npr. vsakdo ne more kreirati delovnega naloga za zamenjavo plakata). Preko spletnega vmesnika odgovorni delavci določijo katere podatke mora plakater izpolniti v okviru delovnega naloga. Verzija razvita v diplomskem delu omogoča zajem številčnih, da/ne in opisnih odgovorov. Koristen bi bil zajem drugih podatkov, predvsem iz senzorjev, ki se nahajajo v telefonu. V mnogih primerih bi bil dobrodošel zajem slike za namene popisa stanja. Zelo uporaben bi bil tudi zajem GPS koordinat.

6.2.2 Spletna aplikacija

Pri razvoju spletne aplikacije smo že v osnovi poskušali doseči, da bi le-ta omogočala čimveč. Kljub temu je prostora za številne izboljšave še dovolj. Verjetno najbolj uporabna izboljšava bi bila funkcionalnost poljubnega generiranja poročil. Spletni uporabnik bi si lahko preko uporabniškega vmesnika pripravil njemu ustrezno poročilo, ki bi ga lahko na koncu shranil v PDF ali celo izvozil v kakšni nadaljni obdelavi prijazni obliki (JSON, XML, pogojno CSV). Preko vmesnika za ustvarjanje poljubnih poročil, bi tako lahko uporabnik dobil npr. povprečno, maksimalno in minimalno vrednost neke količine, število vnosov ipd. V najbolj izpopolnjeni obliki bi bilo možno, da bi se za pripravo poročil lahko uporabil tudi skriptni jezik. Na tak način bi

Slika 6.1: Drugačen in bolj pregleden prikaz delovnih nalogov na terenu.

sistem res pokrival poljubna poročila. To bi s sabo prineslo težavno učenje uporabnikov .

Zelo uporabna funkcionalnost bi bilo komentiranje delovnih nalogov. Posamezen delovni nalog bi lahko tako komentirali tako plakaterji kot odgovorni delavci. Komentiranje bi bilo omogočeno tako preko spletne strani kot preko mobilnega telefona. Na takšen način bi bilo bolj enostavno reševati vprašanja v okviru nekega delovnega naloga kot preko privatnih sporočil s stalnim sklicevanjem na določen delovni nalog.

Sistem za upravljanje s plakatnimi mesti ne omogoča izdajanja,vodenja in plačevanja računov. To niti ni namen v tem diplomskem delu opisanega sistema. Ker pa podjetja, ki sistem uporabljajo, te funkcionalnosti potrebujejo, je potrebno poskrbeti tudi za to komponento njihovega poslovnega procesa. S tem namenom se lahko na sistemu naredi izboljšava v smeri izvažanja podatkov za pripravo računov v drugih računovodskih programih (npr. Pantheon).

6.2.3 Izboljšave uporabljenih algoritmov

Pri izdelavi sistema smo uporabili kar nekaj algoritmov, ki v večini primerov dobro služijo svojemu namenu. Izračun optimalnega vrstnega reda obhoda v trenutni verziji sistema upošteva zračne razdalje med lokacijami. Za še boljše delovanje bi bilo potrebno upoštevati dolžino dejanske cestne povezave. Da bi bilo to možno, bi bilo potrebno povezati sistem na zunanjo storitev za navigacijo uporabnikov in preko te storitve pridobivati podatke o realni cestni razdalji.

Algoritem za izračun optimalnega razporeda zasedenosti plakatnih mest bi lahko doživel tudi kakšno spremembo ali dve. V prvi fazi bi bilo potrebno zagotoviti upoštevanje izključevanja konkurence. Zamislimo si plakatna mesta v nekem lokalu. Z vidika nekega oglaševalca ne bi bilo dobro, da se v istem lokalu oglašuje konkurenčno podjetje (npr. na eni strani lokala reklama za Mobitel, na drugi reklama za Si.mobil). Da bi to dosegli bi bilo potrebno v samo predstavitev plakatnih mest uvesti skupine (gruče), znotraj

katerih se ne smeta pojaviti konkurenčni podjetji. Druga dokaj pomembna izboljšava bi bila uvedba preverjanja ali se iz podanih zahtev lahko oblikuje urnik in da se v primeru, ko oblikovanje urnika ni možno, izpišejo potrebni popravki. Ta funkcionalnost sistema bi bila zelo dobrodošla, da se stranki že takoj ob pogovoru in zajemu zahtev sporoči, kaj lahko in česa ne more. Za potrebe izračuna v realnem času bi bilo potrebno v okviru nadaljnjega dela poskrbeti za hitrejše izvajanje. V primeru, da imamo kar nekaj zahtev (od 1000 naprej), se lahko izračun časovno zavleče v nekaj minutno čakanje na rezultat. Veliko na samo hitrost izračuna vplivajo tudi želje strank. V primerih kadar večina strank želi točno določen termin brez dovoljenih dodatnih dni maneverskega prostora znotraj željenega obdobja, se lahko zgodi da zaradi prevelikega povpraševanja in togosti premikanja terminov izračun ne bo uspel oziroma se bo zavlekel za precej časa.

6.2.4 Uporaba sistema za druge namene uporabe

Zaradi dobre zasnove sistema razvitega v okviru diplomskega dela, bi bilo le-tega možno uporabiti za številne druge namene. Med njimi so:

- montiranje stavbnega pohištva,
- sestavljanje sobnega pohištva,
- servis aparatov na domu,
- čiščenje prostorov,
- popis stanja na terenu,
- izvajanje anket ipd.

Iz seznama je razvidno, da je sistem moč uporabiti povsod, kjer se opravljajo naloge po različnih lokacijah na terenu. Le čas bo pokazal, kaj vse razviti sistem pokrije v poslovnem svetu.

Slike

Tabele

Literatura

[1] (2012) Ayuda Media Systems. Dostopno na: http://oohsuppliersmarketplace.com/company.php?id=1154795&company=Ayuda%20Media%20Systems.

[2] (2012) Ayuda Media Systems. Dostopno na: http://www.ayudasystems.com/BMS.

[3] (2012) bManage. Dostopno na: http://www.billboardplanet.com/Promotions/bManage.htm.

[4] (2012) RF ID evidentiranje. Dostopno na: http://www.europlakat.si/orodja-in-raziskave/orodja-in-raziskave/rf-id-evidentiranje/.

[5] (2012) Android (operating system). Dostopno na: http://en.wikipedia.org/wiki/Android_(operating_system).

[6] (2012) Android, the world's most popular mobile platform. Dostopno na: http://developer.android.com/about/index.html.

[7] (2012) Android platform overview. Dostopno na: http://source.android.com/source/overview.html.

[8] (2012) Android Developer tools. Dostopno na: http://developer.android.com/tools/index.html.

[9] (2012) Near field communication. Dostopno na: http://en.wikipedia.org/wiki/Near_field_communication.

[10] (2012) Django. Dostopno na: https://www.djangoproject.com/.

[11] A. Holovaty, J. Kaplan-Moss, The definitive guide to Django : Web development done right, New York, ZDA: Apress, 2009.

[12] (2012) Comet (programming). Dostopno na: http://en.wikipedia.org/wiki/Comet_(programming).

[13] (2012) Celery: Distributed task queue. Dostopno na: http://celeryproject.org/.

[14] (2012) PostgreSQL. Dostopno na: http://www.postgresql.org/.

[15] (2012) Nginx. Dostopno na: http://wiki.nginx.org/Main.

[16] (2012) Representational State Transfer (REST). Dostopno na: http://en.wikipedia.org/wiki/Representational_state_transfer.

[17] (2012) Simple Object Access Protocol (SOAP). Dostopno na: http://en.wikipedia.org/wiki/SOAP.

[18] (2012) Point in polygon. Dostopno na: http://en.wikipedia.org/wiki/Point_in_polygon.

[19] (2012) An efficient point classification algorithm for triangle meshes. Dostopno na: http://granada.academia.edu/JuanTorres/Papers/870943/An_efficient_point_classification_algorithm_for_triangle_meshes.

[20] D. G. Alciatore, R. Miranda, A Winding Number and Point-in-Polygon Algorithm, *Glaxo Virtual Anatomy Project research report*, Colorado State University (CSU), January, 1995. Dostopno na: http://www.engr.colostate.edu/~dga/dga/papers/point_in_polygon.pdf.

[21] (2012) Fast Winding Number Inclusion of a Point in a Polygon. Dostopno na: http://softsurfer.com/Archive/algorithm_0103/algorithm_0103.htm.

[22] (2012) Algorithm Design: Computational Geometry. Dostopno na: http://www.cim.mcgill.ca/~sveta/COMP102/Lecture19.pdf.

[23] (2012) Travelling salesman problem. Dostopno na: http://en.wikipedia.org/wiki/Travelling_salesman_problem.

[24] M. Held, R. M. Karp, A Dynamic Programming Approach to Sequencing Problems, *Journal of the Society for Industrial and Applied Mathematics 10*, str: 196–210, marec 1962. Dostopno na: http://jurnalinternasional.files.wordpress.com/2011/01/dpsequencing.pdf.

[25] T. Stützle, The Traveling Salesman Problem: State of the Art, tehnično poročilo, Darmstadt University of Technology, 2003. Dostopno na: http://www.sls-book.net/Slides/tsp.pdf.

[26] C. Nilsson, Heuristics for the Traveling Salesman Problem, tehnično poročilo, Linköping University (LiU),Švedska,2003. Dostopno na: http://www.ida.liu.se/~TDDB19/reports_2003/htsp.pdf.

[27] I. Bryan, The Traveling Salesman Problem, tehnično poročilo, University of North Carolina Wilmington (UNCW),ZDA,2009. Dostopno na: http://www.uncw.edu/math/documents/MAT495/Papers/Bryan.pdf.

[28] C. L. Valenzuela, A. J. Jones, Estimating the Held-Karp lower bound for the geometric TSP, European Journal Of Operational Research 102 (1), str: 157-175, 1997. Dostopno na http://orca.cf.ac.uk/31865/.

[29] (2012) Nearest neighbour algorithm. Dostopno na: http://en.wikipedia.org/wiki/Nearest_neighbour_algorithm.

[30] (2012) Ant colony optimization algorithms. Dostopno na: http://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms.

[31] L. M.Gambardella, Ant colony optimization, tehnično poročilo, Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA), Switzerland, 2004. Dostopno na: http://www.idsia.ch/~luca/aco2004.pdf.

[32] I. Kononenko, M. Kukar, Machine Learning and Data Mining: Introduction to Principles and Algorithms, UK: Horwood Publishing Limited, 2007.

[33] (2012) The Watchmaker framework. Dostopno na: http://watchmaker.uncommons.org/.

[34] (2012) Gantt chart. Dostopno na: http://en.wikipedia.org/wiki/Gantt_chart.

[35] (2012) Introducing JSON. Dostopno na: http://www.json.org/.

[36] (2012) JSON: The Fat-Free Alternative to XML. Dostopno na: http://www.json.org/xml.html.

[37] (2012) Routine for performing the "point in polygon"inclusion test. Dostopno na: http://www.dgp.toronto.edu/~mac/e-stuff/point_in_polygon.py.

[38] (2012) Shapely. Dostopno na: http://toblerity.github.com/shapely/.

[39] (2012) geos::algorithm::RayCrossingCounter Class Reference. Dostopno na: http://geos.osgeo.org/doxygen/classgeos_1_1algorithm_1_1RayCrossingCounter.html.