Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Vojko Rožič

Sistem za obveščanje raznašalcev pošte v realnem času

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: prof. dr. Viljan Mahnič

Ljubljana, 2018

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:
Sistem za obveščanje raznašalcev pošte v realnem času

Tematika naloge:

Izdelajte sistem, ki bo podjetju, ki se ukvarja z raznašanjem hitre pošte, omogočal sprotno obveščanje raznašalcev na terenu o novih naročilih. Za ta sistem opredelite funkcionalne in nefunkcionalne zahteve ter izberite ustrezna orodja za njegovo realizacijo. Podrobno opišite arhitekturo in sestavne dele izdelane rešitve ter predstavite uporabniške vmesnike, ki so na voljo posameznim vrstam uporabnikov.

Zahvaljujem se prof. dr. Viljanu Mahniču za pomoč in usmerjanje pri izdelavi diplomskega dela.

Kazalo

Povzetek

Abstract

1 Uvod **1**

2 Zahteve sistema **5**

2.1 Tipi uporabnikov 6

2.2 Uporabniške zgodbe in sprejemni testi 7

2.3 Nefunkcionalne zahteve 15

3 Orodje Docker **17**

3.1 Kako si lahko predstavljamo Docker 17

3.2 Kje lahko uporabljamo Docker 18

3.3 Ključni koncepti 19

3.4 Povzetek uporabe 21

4 Arhitektura in postavitev sistema **23**

4.1 Arhitektura sistema in komponente 24

4.2 Primer gradnje zaledne aplikacije 29

4.3 Namestitev orodja Docker 29

4.4 Uporaba orodja Docker pri komponentah sistema 31

5 Uporabniški vmesniki **39**

5.1 Uporabniški vmesniki rešitve ponudnika identitet KeyCloak . . 40

5.2 Uporabniški vmesniki spletne aplikacije 41
5.3 Mobilna aplikacija 47

6 Zaključek **51**

Literatura **53**

Seznam uporabljenih kratic

kratica	angleško
API	Application Programming Interface
APT	Advanced Package Tool
CRUD	create, read, update, and delete
GPG	GNU Privacy Guard
HTTP(S)	Hypertext Transfer Protocol (Secure)
IDP	Identity provider
IMEI	International Mobile Equipment Identity
JAR	Java ARchive
JPA	Java Persistence API
JSON	JavaScript Object Notation
JWT	JSON Web Token
MQTT	Message Queuing Telemetry Transport
POM	Project Object Model
REST	Representational State Transfer
SMS	Short Message Service
TCP	Transmission Control Protocol

Povzetek

Naslov: Sistem za obveščanje raznašalcev pošte v realnem času

Avtor: Vojko Rožič

Diplomsko delo zajema izgradnjo sistema, ki bo optimiziral in olajšal komunikacijo in proces dela v podjetju, ki se ukvarja s prevzemom in dostavo hitrih pošiljk. Predstavili bomo naročnikove zahteve s pomočjo uporabniških zgodb in sprejemnimi testi. Predstavili bomo orodje Docker, ki razvijalcem olajša delo in skrajša čas, ki je potreben pri razvoju in vzpostavitvi naročnikovega sistema. Predstavili bomo njegovo uporabo pri gradnji, distribuciji in vzpostavitvi različnih komponent sistema. Predstavili bomo uporabniške vmesnike, ki jih zaposleni v podjetju uporabljajo pri interakciji s sistemom. Rezultat diplomske naloge bo delujoči sistem, ki omogoča lažjo komunikacijo med telefonisti in raznašalci o novih dostavnih naročilih. Poleg optimizirane komunikacije pa bo sistem omogočal popoln pregled nad naročili, telefonisti, raznašalci in mobilnimi napravami, ki nastopajo v sistemu.

Ključne besede: Uporabniške zgodbe, sprejemni testi, uporabniški vmesnik, spletna aplikacija, mobilna aplikacija, arhitektura sistema, zabojnik, slika.

Abstract

Title: System for real-time notification of couriers

Author: Vojko Rožič

The thesis describes the creation of a system that will optimize and facilitate the communication and workflow for a company engaged in picking up and delivering express shipments. We will present the client's requirements through user stories and acceptance tests. We will introduce a Docker tool, which allows developers to reduce the time needed to build and deploy the system. We will present Docker's use for building, distribution and deployment of various components that are necessary for the system. We will present user interfaces used by employees to interact with our system. The result of this thesis will be a functioning system that facilitates communication about new delivery orders between dispatchers and couriers. In addition to optimized communication, the system will allow for a complete overview of orders, dispatchers, couriers, and mobile devices.

Keywords: User stories, acceptance tests, userinterfaces, web application, mobile application, system arhitecture, container, image.

Poglavje 1

Uvod

Navdih za izdelavo diplomskega dela sem dobil med pogovorom z zaposlenim v podjetju, katerega primarna dejavnost je prevzem in dostava hitrih pošiljk. Znanec mi je potožil, da imajo problem pri komunikaciji med uslužbenci v telefonski centrali, ki sprejemajo naročila, in raznašalci, ki naročila nato prevzamejo in dostavijo. Vsa njihova komunikacija je potekala preko pametnih mobilnih naprav s pomočjo telefonskih klicev in SMS sporočil.

Proces sprejema naročil in posledičnega obveščanja raznašalcev je dolgotrajen. Telefonist sprejme klic stranke, kjer mu stranka sporoči lokacijo prevzema in lokacijo dostave neke pošiljke. Po končanem klicu oz. prejemu vseh potrebnih informacij za izvedbo naročila telefonist pokliče prvega raznašalca, za katerega predvideva, da bi lahko to pošiljko prevzel in dostavil. V najboljšem primeru telefonist naredi samo en klic, lahko pa se zgodi, da se raznašalec v tistem trenutku ne more javiti na telefon ali pa že ima delo in naročila takrat ne more izvesti. Zato je velikokrat potrebno poklicati več raznašalcev, da telefonist prikliče prostega raznašalca za izvedbo naročila. Ko telefonist s klicem potrdi, da lahko raznašalec izvede zahtevano naročilo, mu po končanem klicu pošlje SMS sporočilo s podatki o prevzemni in dostavni lokaciji naročila in morebitne dodatne informacije, kot so kontaktni podatki stranke, kontaktni podatki osebe, ki bo pošiljko prevzela, uro prevzema, ipd. Če je naročil veliko, se pojavi tudi problem zasedenosti telefonista, saj lahko

naenkrat opravlja samo en klic. To pomeni, da morajo nekatere stranke čakati na prostega telefonista ali pa telefonist z zamikom poskuša obvestiti raznašalce o nekem novem naročilu.

Vsi raznašalci za dostavo uporabljajo službeno vozilo, zato nekateri klici in SMS sporočila do raznašalca pridejo med vožnjo. Klicanje med vožnjo pomeni večje tveganje za prometno nesrečo. Da raznašalci zmanjšajo tveganje, za klice uporabljajo sistem za prostoročno telefoniranje. Za pregled SMS sporočil o podatkih o naročilih pa se raznašalec ustavi na najbolj varnem oz. primernem mestu in jih pregleda.

Cilj naše diplomske naloge je optimizacija procesa sprejema novih naročil in posredovanje le-teh raznašalcem na terenu. Za potrebe telefonistov smo razvili spletno aplikacijo preko katere telefonisti vnašajo naročila, ki jih sprejemajo preko telefonske linije. Za raznašalce pa smo razvili mobilno aplikacijo za pametne mobilne naprave, ki bazirajo na operacijskem sistemu *Android* [1], s katero bodo raznašalci lahko rezervirali, preklicali rezervacijo, ali pa samo pregledovali naročila današnjega dne. Dodatni cilj, ki ga želimo doseči, je izbira tehnologij, ki nam omogočajo hitro postavitev sistema na oddaljenem strežniškem sistemu, in izbira tehnologij, ki bodo omogočile hitro osveževanje podatkov v spletni aplikaciji ali mobilni napravi. To pomeni, da se ob vnosu novega naročila preko spletnega vmesnika le-to prikaže raznašalcem na mobilni aplikaciji v čim krajšem možnem času. Isto velja za osvežitev naročila na spletnem vmesniku v primeru, ko raznašalec rezervira naročilo preko mobilne aplikacije.

Strukturo diplomskega dela smo razdelili v šest poglavij. Za lažjo predstavo celotne diplomske naloge je spodaj podan kratek opis vsakega od poglavij.

V drugem poglavju bomo opisali uporabnike, ki nastopajo v našem sistemu, njihove vloge, uporabniške zgodbe ter sprejemne teste. Podali bomo tudi nekaj nefunkcionalnih zahtev, ki jih želimo, da jih naš sistem podpira. V tretjem poglavju bomo na hitro opisali orodje Docker, s katerim smo dosegli, da lahko naš sistem hitro postavimo na oddaljenem strežniku. V četrtem po-

glavju bomo predstavili komponente, ki sestavljajo naš sistem, in praktično uporabo orodja Docker pri razvoju, gradnji ter postavitvi sistema na oddaljenem strežniku. V petem poglavju bomo predstavili uporabniške vmesnike, ki jih uporabljajo uporabniki sistema. Nato pa bomo v zaključnem poglavju podali možne izboljšave in zaključne misli.

Poglavje 2

Zahteve sistema

Zahteve našega sistema smo pridobili v pogovorih z naročnikom in zaposlenimi. Ugotovili smo, da bo sistem zajemal tri različne tipe uporabnikov: *telefonist*, *raznašalec* in *administrator*. Uporabnika tipa telefonist in raznašalec sta v domeni uporabe naročnika, uporabnik tipa administrator pa je v domeni razvijalca sistema. Naš načrt je izdelati sistem, ki bo vseboval spletno aplikacijo, preko katere bodo telefonisti pregledovali in vnašali naročila strank. Naročila se bodo nato prikazovala raznašalcem na mobilni aplikaciji, s katero bodo lahko naročila rezervirali ali pa posledično preklicali rezervacijo. Rezervacija naročila je akcija, s katero raznašalec naročilo prevzame v izvedbo. Izvedba naročila pomeni prevzem pošiljke iz prevzemne lokacije in njeno dostavo na dostavno lokacijo. Preklic rezervacije pa je akcija, s katero raznašalec prekliče izvedbo in s tem omogoči rezervacijo naročila nekemu drugemu raznašalcu. Za lažje razumevanje smo v nadaljevanju podali opise vseh treh tipov uporabnikov in frekvenco uporabe sistema. Nato pa smo podali uporabniške zgodbe in sprejemne teste za naš sistem.

2.1 Tipi uporabnikov

2.1.1 Administrator

Uporabnik tipa administrator bo skrbel za upravljanje z uporabniki tipa telefonist. Kot smo omenili, je administrator v domeni razvijalca sistema, saj hočemo imeti nadzor, kdo lahko upravlja z našim sistemom. Naročnik nam pred uporabo sporoči, koliko telefonistov bo uporabljalo sistem, in za vsakega od njih sporoči ime, priimek, geslo in elektronski naslov. Administrator ima pregled nad vsemi uporabniki tipa telefonist, ki jih po potrebi doda, onemogoči, izbriše, spremeni geslo ali pa spremeni njegove osnovne podatke. Administrator dostopa do sistema zelo redko. Uporablja ga, ko naročnik zaposli novega telefonista ali prekliče delovno razmerje s telefonistom ali pa je potrebno telefonistu spremeniti njegove osebne podatke.

2.1.2 Raznašalec

Uporabnik tipa raznašalec uporablja mobilno aplikacijo. Preko aplikacije pregleduje, rezervira, ali pa prekliče rezervacijo naročila. Raznašalec ne more spreminjati podatkov o naročilu, ampak lahko samo spreminja status naročila, če mu je ta akcija glede na status naročila dovoljena. Frekvenca interakcije raznašalca z mobilno aplikacijo je zelo pogosta. V grobem lahko rečemo, da se uporablja večkrat na uro. Uporaba je odvisna od količine sprejetih naročil v tistem dnevu.

2.1.3 Telefonist

Uporabnik tipa telefonist primarno uporablja sistem za dodajanje, urejanje in brisanje naročil, ki jih sprejema v klicnem centru. Druga naloga telefonistov, ki se uporablja redko, je upravljanje z uporabniki tipa raznašalec in mobilnimi napravami, ki jih uporabljajo raznašalci. Telefonist ureja podatke mobilnih naprav v primerih, ko raznašalec pridobi novo mobilno napravo, odstrani poškodovano, ali pa mu je bila mobilna naprava ukradena. Upravljanje z

raznašalci pa je potrebno v primerih, ko naročnik zaposli novega raznašalca ali pa le-temu prekliče delovno razmerje.

2.2 Uporabniške zgodbe in sprejemni testi

Uporabniške zgodbe [4] opisujejo funkcionalnosti, ki bodo na voljo uporabnikom ali naročniku programske opreme. Uporabniško zgodbo sestavljajo tri komponente:

- pisni opis zgodbe, ki nam pomaga pri planiranju
- pogovori o zgodbi, ki nam pomagajo pri razumevanju podrobnosti zgodbe
- testi, ki določajo, kdaj bo zgodba zaključena

Pri obravnavi uporabniških zgodb moramo paziti, da te niso preobširne. V primeru, da je zgodba preobširna, jo moramo razdeliti na več manjših zgodb. A tudi pri deljenju zgodb na manjše enote moramo biti pozorni, da zgodbe ne delimo do najmanjših podrobnosti. Pri izdelavi je pomembno, da čimbolj točno določimo pričakovanja uporabnikov. Ta pričakovanja določimo v obliki sprejemnih testov [4]. Sprejemno testiranje je proces potrjevanja, da je funkcionalnost zgodbe razvita v skladu s pričakovanji naročnika programske opreme. V nadaljevanju so podane uporabniške zgodbe in sprejemni testi za vse tipe uporabnikov našega sistema.

2.2.1 Administrator

Administrator se v sistem prijavi z uporabniškim imenom in geslom - **A1**

- Opis: Naš sistem bo med začetno postavitvijo na oddaljenem strežniku dodal uporabnika tipa administrator z uporabniškim imenom in geslom, ki smo ga določili med postavitvijo.
- Testi:

  - preveri prijavo v administrativni del sistema s pravilnim in napačnim geslom
  - preveri prijavo v administrativni del sistema s pravilnim in napačnim uporabniškim imenom

Administrator dodaja, pregleduje in spreminja podatke o telefonistih - **A2**

- Opis: Po uspešni prijavi administrator preko administracijskega vmesnika za urejanje uporabnikov tipa telefonist doda novega telefonista, ali pa ureja že obstoječe telefoniste.
- Testi:
  - preveri dodajanje telefonista brez elektronskega naslova
  - preveri s spremembo imena, priimka, gesla in elektronskega naslova telefonista
  - preveri z onemogočitvijo obstoječega telefonista
  - preveri podvajanje

2.2.2 Telefonist

Telefonist se v sistem prijavi z uporabniškim imenom in geslom - **T1**

- Opis: Telefonist obišče spletno stran, kjer se nahaja naša spletna aplikacija. Spletna aplikacija zahteva prijavo z elektronskim naslovom in geslom. Po uspešni prijavi se uporabniku prikaže spletna aplikacija.
- Testi:
  - preveri prijavo s pravilnim in napačnim geslom
  - preveri prijavo s pravilnim in napačnim uporabniškim imenom

Telefonist lahko v sistem vnese novo mobilno napravo - **T2**

- Opis: Da lahko raznašalci uporabljajo mobilno aplikacijo, in da naš sistem lahko določi, iz katere mobilne naprave je prišel zahtevek, potrebujemo podatke o mobilnih napravah. Identifikacijska številka IMEI je najbolj pomemben podatek, ki ga moramo hraniti, saj s to številko ugotovimo, s katere mobilne naprave je prišel zahtevek v naš sistem. Poleg številke IMEI želimo za lažjo preglednost nad napravami zabeležiti tudi poljubno ime naprave, ki jo dodamo v sistem.

- Testi:

  - preveri dodajanje naprave z manjkajočim vnosom imena mobilne naprave
  - preveri dodajanje naprave z manjkajočim vnosom identifikacijske številke IMEI
  - preveri dodajanje naprave z napačno identifikacijsko številko IMEI
  - preveri dodajanje naprave z že obstoječim imenom mobilne naprave
  - preveri dodajanje naprave z že obstoječo identifikacijsko številko IMEI
  - preveri dodajanje naprave s pravilno identifikacijsko številko IMEI in imenom mobilne naprave

Telefonist lahko v sistem vnese novega raznašalca - **T3**

- Opis: Sistem mora hraniti podatke o raznašalcih pošiljk. Potrebno je, da je mogoče v sistem vnesti nove ali urejati obstoječe raznašalce. Za vnos raznašalca potrebujemo njegovo ime, priimek, elektronski naslov in ime mobilne naprave, ki jo raznašalec uporablja. Napravo želimo izbrati preko izbirnega polja, ki nam ponudi vse aktivne mobilne naprave, ki so vnesene v sistem.

- Testi:

- preveri dodajanje raznašalca z manjkajočim imenom
- preveri dodajanje raznašalca z manjkajočim priimkom
- preveri dodajanje raznašalca z manjkajočim elektronskim naslovom
- preveri dodajanje raznašalca z manjkajočo izbiro mobilne naprave
- preveri dodajanje raznašalca z mobilno napravo, ki je že dodeljena drugemu raznašalcu
- preveri dodajanje raznašalca s pravilnimi podatki v vnosnih poljih
- preveri z onemogočitvijo obstoječega raznašalca

Telefonist lahko v sistem vnese novo naročilo - **T4**

- Opis: Glavni funkcionalnosti sistema sta vnos in urejanje naročil, ki jih telefonist sprejema preko telefonske linije. Telefonist mora imeti možnost vnosa poljubnega besedila za prevzemno in dostavno lokacijo in morebitno izbiro raznašalca. Omogočiti je potrebno, da se naročilo raznašalcem lahko prikaže ob določenem času. Funkcionalnost je pomembna, ker lahko stranka preko klicnega centra odda naročilo že zjutraj, a zahteva, da se ga prevzame in dostavi šele v popoldanskem času. Telefonist ne sme izbrati časa prikaza, ki je manjši od trenutnega časa.
- Testi:
  - preveri dodajanje novega naročila brez lokacije prevzema
  - preveri dodajanje novega naročila brez lokacije dostave in dodatnega opisa
  - preveri dodajanje novega naročila brez izbranega raznašalca
  - preveri dodajanje novega naročila z izbranim raznašalcem
  - preveri dodajanje novega naročila brez izbire ure prikaza
  - preveri dodajanje novega naročila z izbiro ure prikaza

  - preveri dodajanje novega naročila z izbiro ure prikaza, ki je manjša od trenutnega časa
  - preveri dodajanje novega naročila s pravilnimi podatki v vnosnih poljih

Telefonist lahko spreminja podatke mobilnih naprav - **T5**

- Opis: V primeru, ko je potrebno spremeniti podatke obstoječih naprav, moramo telefonistu to omogočiti. Lahko se zgodi, da se je telefonist pri vnosu identifikacijske številke IMEI zmotil, zato mu moramo omogočiti, da le-to popravi. Za lažjo evidenco lahko telefonist spremeni tudi ime naprave. V primeru, da je bila mobilna naprava ukradena, moramo imeti možnost, da napravo onemogočimo in s tem preprečimo morebitno zlorabo mobilne aplikacije.
- Testi:
  - preveri s spremembo vseh vnosnih polj mobilne naprave s pravilnimi podatki
  - preveri s spremembo identifikacijske številke IMEI z neveljavno številko IMEI
  - preveri s spremembo imena naprave z neveljavnim vnosom imena
  - preveri omogočanje in onemogočanje mobilne naprave
  - preveri podvajanje

Telefonist lahko spreminja podatke obstoječih raznašalcev - **T6**

- Opis: Lahko se zgodi, da je raznašalec spremenil elektronski naslov, ime ali priimek, zato je potrebno omogočiti urejanje obstoječih podatkov raznašalca. Telefonist mora imeti možnost, da raznašalca omogoči oz. onemogoči.

- Testi:
  - preveri s pravilnimi spremembami vseh vnosnih polj raznašalca
  - preveri s praznim vnosom podatkov v poljih imena, priimka in elektronskega naslova
  - preveri z onemogočitvijo raznašalca in preveri, da ni prikazan v seznamu za izbiro pri oddaji novega naročila
  - preveri podvajanje

Telefonist lahko spreminja podatke obstoječih naročil - **T7**

- Opis: Lahko se zgodi, da je po oddaji novega naročila potrebno spremeniti podatke o nekem naročilu. Omogočiti je potrebno spreminjanje vseh podatkov naročila kot tudi izbris naročila v primeru, da je stranka preklicala naročilo.
- Testi:
  - preveri s spremembo izbire raznašalca
  - preveri s spremembo lokacije prevzema
  - preveri s spremembo lokacije dostave in podrobnosti
  - preveri s spremembo časa prikaza naročila v prihodnost
  - preveri s spremembo vseh vnosnih polj s pravilnimi podatki
  - preveri izbris že oddanega naročila

Telefonist mora imeti pregled nad vsemi naročili - **T8**

- Opis: Telefonistom je potrebno omogočiti prikaz vseh naročil tistega dne na eni strani. Seznam naročil mora vsebovati zaporedno številko naročila, uro oddaje naročila, lokacijo prevzema, lokacijo dostave in ime raznašalca, če je bilo naročilo rezervirano.
- Testi:

  - odpri stran, ki prikazuje seznam naročil, in preveri pravilen izpis zahtevanih podatkov
  - preveri urejenost seznama naročil

Telefonist mora imeti pregled nad vsemi mobilnimi napravami - **T9**

- Opis: Telefonistom moramo omogočiti prikaz vseh mobilnih naprav, ki so v sistemu. Seznam mora vsebovati ime mobilne naprave, status naprave in identifikacijsko številko IMEI.
- Testi:
  - odpri stan, ki prikazuje seznam mobilnih naprav, in preveri pravilen izpis zahtevanih podatkov
  - preveri urejenost seznama mobilnih naprav

Telefonist mora imeti pregled nad vsemi raznašalci - **T10**

- Opis: Telefonistom moramo omogočiti prikaz vseh raznašalcev. Seznam mora vsebovati zaporedno številko raznašalca, ime, priimek, elektronski naslov in ime mobilne naprave, ki jo uporablja.
- Testi:
  - odpri stan, ki prikazuje seznam raznašalcev, in preveri pravilen izpis zahtevanih podatkov
  - preveri urejenost seznama raznašalcev

2.2.3 Raznašalec

Raznašalec lahko pregleduje naročila po različnih kategorijah - **R1**

- Opis: Raznašalec mora imeti možnost pregleda naročil po treh različnih kategorijah. Imamo tri kategorije prikaza naročil: prosta naročila, osebno rezervirana naročila, druga rezervirana naročila. Prosta naročila

so naročila, ki jih ni rezerviral še noben raznašalec. Osebno rezervirana naročila so tista, ki jih vidi raznašalec, ki jih je rezerviral. Druga rezervirana naročila pa so naročila, ki so rezervirana s strani drugih raznašalcev.

- Testi:
  - preveri spremembo prikaza naročil po vseh treh kategorijah

Raznašalec lahko rezervira prosto naročilo - **R2**

- Opis: Če je naročilo v seznamu prostih naročil, ga lahko rezervira katerikoli raznašalec. Rezervacija naročila v drugih dveh statusih ni možna.
- Testi:
  - preveri rezervacijo prostega naročila
  - preveri rezervacijo na rezerviranem naročilu

Raznašalec lahko prekliče rezervacijo naročila - **R3**

- Opis: Raznašalec lahko prekliče rezervacijo naročila, ki ga je pred tem rezerviral. Naročilo, ki je bilo rezervirano s strani drugega raznašalca, ne more preklicati.
- Testi:
  - preveri preklic rezervacije naročila na rezerviranem naročilu, ki ga je raznašalec pred tem rezerviral
  - preveri preklic rezervacije naročila na rezerviranem naročilu, ki ga je pred tem rezerviral drugi raznašalec

2.3 Nefunkcionalne zahteve

Naš sistem ima tudi nekaj nefunkcionalnih zahtev, ki jih moramo vzeti v zakup pri razvoju, gradnji in namestitvi sistema. Zahteve so sledeče:

- sistem se mora namestiti na strežnik z operacijskim sistemom Linux
- odzivni čas spletne in mobilne aplikacije naj bo čim krajši
- podatkovni prenos podatkov mobilne aplikacije naj bo čim manjši
- mobilna aplikacija mora biti razvita za mobilne telefone z operacijskim sistemom Android
- uporabnik tipa telefonist uporablja spletno aplikacijo

Poglavje 3

Orodje Docker

Orodje Docker [9] nam omogoča gradnjo, distribucijo in izvajanje katerekoli aplikacije kjerkoli. Docker se je v zelo kratkem času uveljavil kot standardni način uvajanja programske opreme.

Pred Dockerjem je bilo potrebnih veliko različnih tehnologij, s katerimi smo dosegli uvajanje programske opreme. Primer nekaterih tehnologij oz. orodij so virtualni stroji, orodja za upravljanje s konfiguracijami, orodja za upravljanje z odvisnimi knjižnicami, ipd. Vsa ta orodja je bilo potrebno uporabljati oz. vzdrževati. Za te namene so podjetja zaposlovala inženirje, ki so dobro poznali ta orodja.

Docker nam je s svojo filozofijo gradnje, distribucije in izvajanja aplikacij omogočil, da različni inženirji govorijo en skupni jezik in s tem olajšal komunikacijo. Z uporabo orodja Docker dosežemo, da celoten življenjski cikel programske opreme potuje skozi enoten proces.

3.1 Kako si lahko predstavljamo Docker

Za lažjo predstavitev Dockerja bomo uporabili metaforo. Slovenski prevod besede Docker je pristaniški delavec. Orodje Docker si zato lahko predstavljamo kot delavca v pristanišču, ki natovarja in raztovarja tovor na oz. z ladje. Če je količina tovora, ki ga je potrebno naložiti na ladjo, velika, mora

lastnik pristanišča zaposliti veliko pristaniških delavcev, ki lahko v razumnem času ves ta tovor naložijo na ladjo. Tovor, ki ga je potrebno naložiti na ladjo, je tudi različnih velikosti, oblik in tež, kar pomeni, da je optimalno zlaganje tovora zahtevno, zamudno in drago. Problem optimalnega zlaganja tovora po kosih se reši z uvedbo standardiziranih zabojnikov, v katere delavci naložijo različen tovor in nato zabojnike s pomočjo žerjavov naložijo na ladjo, ki podpira standardizirane zabojnike.

Podobno se dogaja v svetu razvoja programske opreme. Pred Dockerjem je bilo potrebno veliko truda, da se je programsko opremo, ki je bila razvita, uvedlo v različna okolja (razvojno, testno, produkcijsko). Vsa okolja je bilo potrebno postaviti ločeno s pomočjo skript ali namenskih orodij. To pomeni, da je bila potrebna večkratna postavitev okolja in večkratna namestitev programske opreme na vsa okolja, ki so v uporabi. Z uporabo zabojnikov Docker pa našo programsko opremo ali rešitev zapakiramo v enega ali več zabojnikov, ki jih nato preprosto distribuiramo skozi različna okolja. Vse, kar potrebujemo, je nameščeno orodje Docker na okoljih, kjer bomo naše zabojnike izvajali. Ni nam potrebno skrbeti, ali zabojnik Docker izvajamo na računalniku, ki ima nameščen operacijski sistem *RedHat* ali *Ubuntu*, ali pa na virtualnem stroju z operacijskim sistemom *CentOS*. Vse, kar je potrebno storiti, je zagon ukaza `docker run` in slika zabojnika Docker je naložena na ciljni računalnik in pripravljena na zagon. Kaj je slika Docker in kaj zabojnik Docker, bomo pojasnili v nadaljevanju.

3.2 Kje lahko uporabljamo Docker

Porajata se nam vprašanji, zakaj bi nekdo uporabljal Docker in za katere namene bi ga uporabljali? Kratek odgovor na vprašanje „zakaj“ je, da nam Docker lahko zelo hitro prihrani denar. Za odgovor na vprašanje „za katere namene“, pa smo spodaj našteli nekaj možnih uporab v realnem svetu.

- zamenjava navideznih strojev
- prototipi

- pakiranje programske opreme
- uporaba v arhitekturi mikrostoritev
- uporaba v neprekinjeni dostavi

3.3 Ključni koncepti

Ključna funkcija orodja Docker je gradnja, distribucija in izvajanje aplikacije na kateremkoli sistemu z nameščenim Dockerjem. Končnemu uporabniku je Docker terminalni program, s katerim pošiljamo ukaze orodju Docker, ki se izvaja v ozadju in izvršuje podane ukaze. Poglavitni Dockerjevi ukazi, ki jih pogosto uporabljamo, so podani spodaj.

3.3.1 Ključni ukazi Dockerja

- docker build - gradnja slike Docker
- docker run - zagon slike Docker kot zabojnik Docker
- docker commit - shranjevanje spremembe zabojnika v sliko Docker
- docker push - nalaganje slike Docker v register slik Docker

3.3.2 Slike Docker in zabojniki Docker

Obstaja možnost, da bralec še ne pozna koncepta Dockerjeve slike in zabojnika (v nadaljevanju samo slika ali zabojnik). Način, s katerim si lahko predstavljamo slike in zabojnike, je lahko primer nekega programa in procesa (proces je program v izvajanju). Kot je proces program v izvajanju, je zabojnik slika v izvajanju. Drug način predstavitve lahko izpeljemo iz objektno usmerjenega programiranja. Slike si lahko predstavljamo kot razrede in zabojnike kot objekte teh razredov. Zanimiva lastnost je tudi, da lahko iz ene slike ustvarimo več zabojnikov, ki so med seboj izolirani.

3.3.3 Register Dockerjevih slik

Slike so shranjene v registrih Dockerjevih slik (v nadaljevanju samo register slik). Registre slik si lahko predstavljamo kot skladišča, iz katerih lahko uporabniki prenašajo ali pa vanje nalagajo slike, ki so jih pripravili drugi uporabniki ali podjetja. Poznamo zasebne in javne registre slik. Do zasebnih registrov lahko dostopajo samo uporabniki, kateri imajo dovoljenje za dostop, javne pa lahko uporabljajo vsi. Docker nam ponuja javni register slik *DockerHub* [5], na katerem imamo na razpolago veliko osnovnih slik, katere lahko nato uporabimo pri gradnji naše poljubne aplikacije Docker. Veliko podjetij, ki razvijajo namensko programsko opremo, nam le-to ponujajo zapakirano v slikah. Primer take programske opreme je podatkovna baza *MySQL* [12] ali pa spletni strežnik *NGINX*.

3.3.4 Dokument Dockerfile

Sliko lahko ustvarimo na več načinov. Za naše potrebe in razumevanje bomo opisali gradnjo slike s pomočjo dokumenta *Dockerfile*. Dokument *Dockerfile* je tekstovna datoteka, ki vsebuje ukaze, s katerimi povemo orodju Docker, kako naj sliko zgradi. Za boljše razumevanje je spodaj primer preprostega dokumenta *Dockerfile* s kratkimi razlagami ukazov.

```
(1) FROM node
(2) RUN git clone -q https://gitlab.com/some-application.git
(3) WORKDIR some-application
(4) RUN npm install > /dev/nul
(5) EXPOSE 8000
(6) CMD ["npm","start"]
```

Dokument Dockerfile se začne z ukazom `FROM` (1). Ukaz `FROM` pove, katero osnovno sliko bo Docker uporabil pri gradnji. V našem primeru smo uporabili uradno sliko *Node.js* po imenu *node*, ki vsebuje binarne datoteke *Node.js*, potrebne za izvajanje *Node.js* aplikacij. Slika se nahaja na javno

dostopnem registru slik *DockerHub*. Ukaz v vrstici (2) prenese programsko kodo naše aplikacije. Ukaz v vrstici (3) nas premakne v mapo, kjer se nahaja programska koda naše aplikacije. Vrstica (4) izvede ukaz `npm`, ki poskrbi za namestitev in gradnjo *Node.js* aplikacije. Aplikacija posluša zahtevke na vratih 8000, zato z ukazom `EXPOSE` v vrstici (5) določimo, naj zabojnik v izvajanju posluša na vratih 8000. Na koncu v vrstici (6) določimo, naj zabojnik ob zagonu izvede ukaz `npm start`, ki začne z izvajanjem aplikacije v zabojniku.

3.4 Povzetek uporabe

Za boljše razumevanje imamo spodaj nabor ukazov za gradnjo slike, shranjevanje slike v register slik, prenos slike na oddaljeni strežnik in zagon zabojnika iz zgrajene slike.

```
(1) docker build -t moja-slika .
(2) docker push my-registry:5000/moja-slika
(3) docker pull my-registry:5000/moja-slika
(4) docker run moja-slika
```

Ukaz (1) zgradi sliko z imenom *moja-slika* iz dokumenta *Dockerfile*, ki se nahaja v imeniku, kjer smo izvršili ukaz. Ukaz `push` (2) naloži sliko v register slik. Ukaz (3) nato izvedemo na oddaljenem strežniku, kjer imamo nameščeno orodje Docker, ki prenese sliko iz registra slik na oddaljeni računalnik. Zagon zabojnika, ki vsebuje aplikacijo, pa nato izvršimo z ukazom (4).

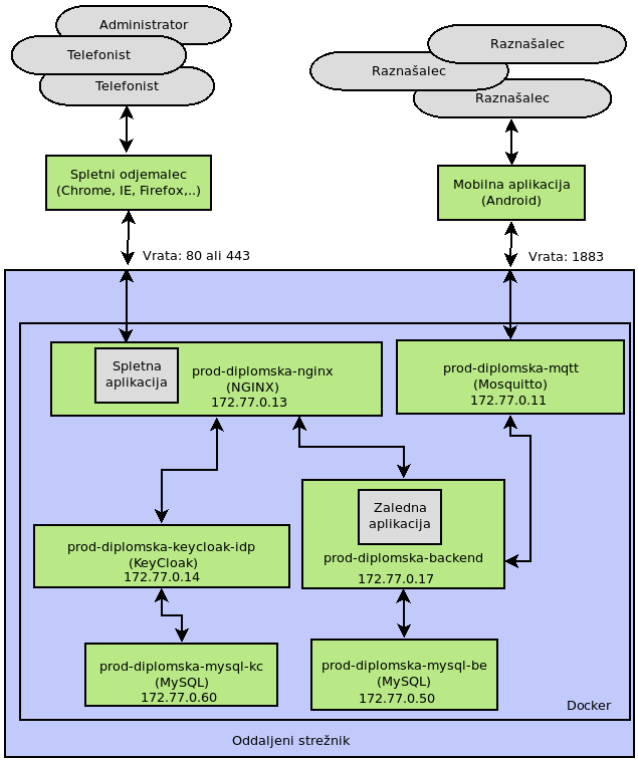
Poglavje 4

Arhitektura in postavitev sistema

V tem poglavju bomo na začetku opisali arhitekturo sistema in naloge komponent, ki sestavljajo sistem. Opisali bomo gradnjo in zagon zaledne aplikacije brez uporabe orodja Docker. Nato pa bomo opisali, kako namestimo orodje Docker na strežniškem ali razvojnem sistemu. Po opisu namestitve bomo podali ukaze orodja Docker, ki jih moramo izvesti, da zgradimo slike komponent in ukaze, ki jih moramo izvesti na oddaljenem strežniku, za izvajanja zabojnikov.

Orodje Docker smo izbrali zaradi preproste uporabe, in ker nam omogoča hitro postavitev sistema na oddaljenem strežniku. Vse, kar moramo storiti na strežniškem računalniku, je namestitev orodja Docker. Ni potrebno nameščati različne programske opreme direktno na operacijski sistem, saj se vsa programska oprema, kot so podatkovne baze, posredniki sporočil, spletni strežniki, ipd. izvaja v zabojnikih. V primeru, da bo naročnik želel sistem prestaviti na drug strežnik, bomo lahko to naredili v zelo kratkem času in bomo prepričani, da bo le-ta deloval tako, kot na predhodnem strežniku.

4.1 Arhitektura sistema in komponente



Slika 4.1: Arhitekturna slika sistema

Na sliki 4.1 je prikazana arhitektura našega sistema. Kot je razvidno, naš sistem vsebuje mobilno aplikacijo ter šest komponent oz. zabojnikov:

- zaledna aplikacija (zabojnik prod-diplomska-backend)
- podatkovna baza zaledne aplikacije (zabojnik prod-diplomska-mysql-be)
- ponudnik identitet KeyCloak (zabojnik prod-diplomska-keycloak-idp)
- podatkovna baza ponudnika identitet KeyCloak (zabojnik prod-diplomska-mysql-kc)
- spletna aplikacija (zabojnik prod-diplomska-nginx)

- posrednik sporočil Mosquitto (zabojnik prod-diplomska-mqtt)

Zaledna aplikacija

Zaledna aplikacija je osrednji del sistema. Skrbi za sprejemanje zahtevkov, ki prihajajo iz mobilne ali spletne aplikacije. Te zahtevke nato preveri, izvede spremembe in podatke shrani v podatkovno bazo. Po shranjevanju poskrbi, da se vse spremembe sporočijo uporabnikom mobilne in spletne aplikacije.

Zaledna aplikacija izpostavlja dostopne točke *REST API*, na katere spletna aplikacija pošilja zahtevke za spremembo stanj hranjenih podatkov. Poleg točk *REST API* zaledna aplikacija izpostavlja protokol *WebSocket* [16], preko katerega sporoči spletni aplikaciji spremembe, ki so se zgodile. Protokol *WebSocket* omogoča polno dvosmerno komunikacijo preko povezave TCP. V našem primeru protokol *WebSocket* uporabljamo samo v smeri zaledne aplikacije proti spletni aplikaciji. S tem dosežemo, da spletna aplikacija ne zahteva svežih podatkov s periodičnimi klici na točke *REST API*, ampak sprejema sporočila, ki jih iz zaledne aplikacije pošiljamo preko protokola *WebSocket*.

Za komunikacijo s podatkovno bazo uporabljamo modul *spring-boot-jpa*. *JPA* nam preslika Javanske razrede v tabele naše podatkovne baze ter nam poleg preslikave ponudi razrede, s katerimi si olajšamo delo z upravljanjem povezav, transakcij in operacijami CRUD nad podatkovno bazo.

Za pošiljanje in sprejemanje zahtevkov mobilne aplikacije uporabljamo posrednika sporočil *Mosquitto* [10]. Posrednik sporočil nam služi kot vmesna točka med mobilno in zaledno aplikacijo. Ko zaledna aplikacija prejme zahtevek mobilne aplikacije o spremembi preko posrednika sporočil, ga preveri, obdela, shrani spremembe in preko posrednika sporočil sporoči spremembe mobilni aplikaciji.

Za komunikacijo s posrednikom sporočil zaledna aplikacija uporablja knjižnico *Eclipse Paho* [7]. Ta knjižnica ponuja razrede za delo s protokolom MQTT in poskrbi za vzpostavljanje povezave s posrednikom sporočil. Protokol MQTT deluje po principu *sporoči-naroči* (ang. publish-subscribe).

Namen protokola MQTT [11] je zmanjšanje porabe energije na mobilnih napravah, optimalno delovanje na komunikacijskih kanalih z majhno pasovno širino in malo odvečnimi podatki (ang. overhead), ki so potrebni za uspešno komunikacijo.

V grobem zaledna aplikacija sprejema akcije, obdeluje in shranjuje podatke ter obvešča spletno in mobilno aplikacijo o spremembah.

Podatkovna baza zaledne aplikacije

Za shranjevanje podatkov o naročilih, raznašalcih in mobilnih naprav, zaledna aplikacija uporablja podatkovno bazo *MySQL*. *MySQL* je odprtokodni sistem za upravljanje s podatkovnimi bazami *MySQL*. *MySQL* je relacijska baza, katere lastnosti so hitrost, zanesljivost, skalabilnost in preprosta uporaba.

Ponudnik identitet KeyCloak

Uporaba spletne aplikacije je dovoljena samo telefonistom. Zato potrebujemo način, s katerim preprečimo uporabo spletne aplikacije uporabnikom, ki za to nimajo dovoljenja. Rešitev za avtorizacijo bi lahko implementirali sami, a obstajajo rešitve, ki nam to delo prihranijo. Ena izmed možnosti, ki je na voljo, je uporaba odprtokodne rešitve *KeyCloak*. *KeyCloak* nam ponuja „rešitev iz škatle“ za upravljanje z uporabniki, izdajanjem žetonov *JWT*, prijavno in registracijsko formo ter mnogo več. V delovanje rešitve *KeyCloak* se zaradi obširnosti ne bomo poglobili. Uporabnik si lahko več prebere na njihovi uradni spletni strani [8].

Podatkovna baza ponudnika identitet KeyCloak

KeyCloak podatke, ki jih potrebuje za delovanje in podatke uporabnikov tipa telefonist, shranjuje v podatkovno bazo. Tudi tukaj uporabljamo podatkovno bazo *MySQL*.

Spletna aplikacija

Spletno aplikacijo uporabljajo uporabniki tipa telefonist. Telefonisti preko spletne aplikacije pregledujejo, dodajajo in urejajo podatke o naročilih, raznašalcih in mobilnih napravah.

Uporabniški vmesnik spletne aplikacije smo razvili s pomočjo ogrodja *ReactJS* [13]. To je knjižnica za izdelavo modularnih komponent uporabniškega vmesnika, ki jih lahko večkrat ponovno uporabimo. Spletno aplikacijo zgradimo s pomočjo orodja *webpack*, ki nam združi in minimizira vse uporabljene datoteke *JavaScript*.

Spletna aplikacija neprijavljenim telefonistom prikaže prijavno stran ponudnika identitet *KeyCloak*. Po uspešni prijavi spletna aplikacija pridobi dostopni (ang. access token) in osveževalni žeton (ang. refresh token). Dostopni žeton spletna aplikacija nato vključi v glavo vsakega zahtevka.

Spletna aplikacija pošilja zahtevke o spremembah na točke *REST API* zaledne aplikacije preko protokola HTTP s pomočjo knjižnice *axios*. Podatke, ki so bili spremenjeni, pa spletna aplikacija prejme preko protokola *WebSocket*. Spletna aplikacija po pridobitvi žetonov pošlje zahtevek za pridobitev podatkov, ki jih bomo prikazali na spletni aplikaciji. Nato se pošlje nov zahtevek za pridobitev dostopne kode, s katero ustvarimo povezavo z zaledno aplikacijo preko protokola *WebSocket*. Vse spremembe podatkov v sistemu se za tem pošiljajo samo preko vzpostavljene povezave. Zahtevke za spremembo stanja, pa vedno pošiljamo preko zahtevkov HTTP na *REST* vmesnik.

Posrednik sporočil Mosquitto

Posrednik sporočil Mosquitto nam služi kot vmesnik za komunikacijo med zaledno in mobilno aplikacijo. Zaledna aplikacija preko posrednika sporočil sprejema zahtevke o spremembah, ki jih pošlje mobilna aplikacija, ter istočasno mobilni aplikaciji pošlje podatke o spremembah.

Mobilna aplikacija

Mobilno aplikacijo uporabljajo raznašalci. Aplikacija je napisana v programskem jeziku *Java* za naprave, ki jih poganja operacijski sistem *Android*. Razvijalno okolje, ki smo ga uporabili pri razvoju mobilne aplikacije, se imenuje *Android Studio* [2].

Raznašalci s pomočjo mobilne aplikacije pregledujejo naročila v različnih statusih ter izvajajo rezervacije ali preklic rezervacij naročil, če so le-ta v dovoljenem statusu oz. stanju.

Mobilna aplikacija ne uporablja točk *REST API*, ampak vzpostavi povezavo s posrednikom sporočil *Mosquitto* in čaka na oddajo ali sprejem sporočila iz posrednika sporočil. Sporočilo vsebuje vse potrebne podatke, ki so potrebni za prikaz naročil, ali pa podatke, ki so potrebni, da raznašalec rezervira ali prekliče rezervacijo naročila. Vedno, ko se spremeni status ali pa podatek nekega naročila, naša zaledna aplikacija pošlje sporočilo s spremembami posredniku sporočil. Mobilna aplikacija nato sprejme to sporočilo in osveži prikazane podatke o naročilu, ali pa ga premakne v drug status oz. stanje. Ko raznašalec izvede akcijo rezervacije ali preklic rezervacije, mobilna aplikacija pošlje sporočilo s potrebnimi podatki posredniku sporočil, od katerega zaledna aplikacije sprejme in obdela sporočilo. V primeru spremembe (uspešne rezervacije ali preklica naročila) zaledna aplikacija pošlje novo sporočilo posredniku sporočil, ki ga mobilna aplikacije sprejme in osveži potrebne podatke.

Poleg standardnih tehnologij, ki se uporabljajo pri razvoju aplikacij Android, smo uporabili tudi zunanjo knjižnico *Eclipse Paho Android Service* [3], ki nam olajša delo s protokolom MQTT in olajša delo z vzpostavljanjem povezave ter pošiljanjem in prejemanjem sporočil iz ali na posrednika sporočil. V podrobnosti razvoja mobilne aplikacije se ne bomo spuščali, ampak bomo v naslednjem poglavju opisali uporabniški vmesnik aplikacije in kako raznašalci uporabljajo mobilno aplikacijo.

4.2 Primer gradnje zaledne aplikacije

Zaledna aplikacija je razvita v programskem jeziku *Java* s pomočjo aplikacijskega ogrodja *Spring* [14]. *Spring* vzdržuje in dopolnjuje več projektov, ki nam olajšajo in pohitrijo delo pri razvoju aplikacij. *Spring Boot* [15] je en izmed projektov, s katerim si skrajšamo in olajšamo razvoj aplikacij. Ponuja nam mnogo knjižnic oz. modulov, ki nam omogočajo izdelavo naprednih aplikacij. Med drugimi nam ponuja modul z vgrajenim spletnim strežnikom *Tomcat*, ki skrbi za sprejemanje spletnih zahtevkov. *Spring Boot* nam olajša konfiguracijo uporabljenih knjižnic oz. modulov s pomočjo nastavitvenih datotek in primernih anotacij v programski kodi. Ponuja nam zelo dobro dokumentacijo, kjer je opisano celotno ogrodje s praktičnimi primeri uporabe.

Za gradnjo naše zaledne aplikacije smo uporabili programsko orodje *Apache Maven*, ki je namenjeno gradnji in upravljanju z Javanskimi projekti. Uporabljamo ga za hranjenje odvisnosti med knjižnicami in moduli, navodil o načinu gradnje, vrste pakiranja projekta, itd. Gradnja projekta *Maven* temelji na projektno objektnem modelu (ang. Project Object Model - POM) in nekaterimi vtičniki, ki so vgrajeni v to orodje. Spodaj imamo podan primer ukazov za gradnjo naše zaledne aplikacije in njen zagon.

```
(1) mvn clean package -DskipTests=true
(2) java -jar application.jar
```

Z ukazom (1) zaženemo gradnjo in pakiranje naše zaledne aplikacije v samozadostni paket tipa *JAR* (Java Archive), ki že vsebuje potrebni spletni strežnik *Tomcat* za izvajanje naše aplikacije. Z ukazom (2) nato zaženemo aplikacijo na sistemu, ki ima podporo za izvajanje aplikacij *Java 8*.

4.3 Namestitev orodja Docker

Če hočemo uporabljati funkcionalnosti, ki nam jih ponuja orodje Docker, ga je potrebno namestiti na izbran gostiteljski sistem. Spodaj so podani ukazi,

ki poskrbijo za namestitev orodja Docker na distribuciji *Debian* operacijskega sistema *Linux* [6].

```
(1) wget https://download.docker.com/linux/debian/gpg
(2) sudo apt-key add gpg
(3) echo "deb [arch=amd64]
      https://download.docker.com/linux/debian
      $(lsb_release -cs) stable" | sudo tee -a
/etc/apt/sources.list.d/docker.list
(4) sudo apt-get update
(5) sudo apt-cache policy docker-ce
(6) sudo apt-get -y install docker-ce
(7) sudo systemctl start docker
```

Z ukazoma (1) in (2) v operacijski sistem dodamo ključ GPG skladišča, kjer se nahajajo namestitvene datoteke orodja Docker. S ključem GPG operacijski sistem preveri, ali smo prenesli pravilne datoteke za namestitev. Z ukazom (3) v sistem dodamo uradno skladišče, iz katerega bomo prenesli namestitvene datoteke. Ukaz (4) posodobi bazo *APT*. Z ukazom (5) preverimo, ali smo uspešno dodali željeno skladišče v sistem. Željeni izpis izgleda nekako tako:

```
Docker-ce:
Installed: (none)
Candidate: 17.06.0~ce-0~debian
Version table:
17.06.0~ce-0~debian 500
500 https://download.docker.com/linux/debian stretch/stable
amd64 Packages
```

V primeru pravilnega izpisa lahko nadaljujemo z ukazom (6), ki nam izvede namestitev orodja Docker. Po končani namestitvi zaženemo orodje Docker z ukazom (7).

4.4 Uporaba orodja Docker pri komponentah sistema

V tem poglavju bomo za vsako komponento, ki sestavlja sistem, opisali, kako smo uporabili orodje Docker pri gradnji in distribuciji komponent ter postavitvi našega sistema na oddaljenem strežniku. Poleg opisa uporabe orodja, bomo podali tudi dokumente Dockerfile, ki vsebujejo ukaze za gradnjo slik.

4.4.1 Zabojnik prod-diplomska-backend

Zabojnik *prod-diplomska-backend* vsebuje zaledno aplikacijo. Za gradnjo slike zabojnika potrebujemo dokument Dockerfile. Spodaj je podan dokument Dockerfile, ki poskrbi, da je gradnja slike zabojnika *prod-diplomska-backend* uspešna.

```
(1) FROM java:8
(2) EXPOSE 8055
(3) ADD /target/application.jar application.jar
(4) COPY /cfg/application.properties application.properties
(5) ENTRYPOINT ["java", "-jar", "application.jar"]
```

Ukaz (1) pove orodju Docker, naj za gradnjo naše slike uporabi osnovno sliko iz javnega registra slik *DockerHub*, ki ima nameščeno *Javo 8*. Uradno ime osnovne slike je *java:8*. Naša zaledna aplikacija za spletne zahtevke posluša na omrežnih vratih 8055, zato jih z ukazom (2) izpostavimo. Ukaz (3) vključi našo zaledno aplikacijo v sliko. Podobno naredi tudi ukaz (4), ki prekopira nastavitveno datoteko naše zaledne aplikacije v sliko. Zadnji ukaz (5) pove orodju Docker, naj ob zagonu zabojnika izvede ukaz `java -jar application.jar`, ki zažene aplikacijo v zabojniku.

Ko imamo pripravljen dokument *Dockerfile* za gradnjo slike, jo zgradimo z naslednjim ukazom:

```
sudo docker build -t registry.gitlab.com/*/backend .
```

V tem trenutku slika obstaja samo na razvojnem računalniku. Sliko potrebujemo tudi na oddaljenem strežniku, kjer imamo postavljeno produkcijsko okolje. Za distribucijo je najlažje, da sliko prenesemo v javni ali zasebni register slik. V našem primeru za register slik uporabljamo storitev, ki nam jo ponuja *GitLab*. Poleg registra slik nam *GitLab* ponuja tudi storitev nadzora različic naše programske kode. Spodaj sta podana ukaza, ki izvršita prijavo v zasebni *GitLabov* register slik (1) in nalaganje slike v ta register (2).

```
(1) docker login registry.gitlab.com
(2) sudo docker push registry.gitlab.com/*/backend
```

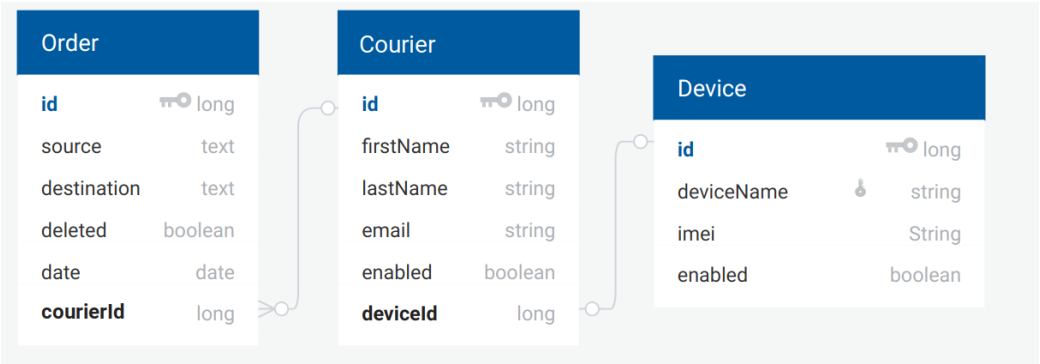
Ko je slika uspešno naložena v register slik, jo lahko prenesemo in začnemo z izvajanjem na oddaljenem strežniku. Ukazi, ki jih moramo izvesti na oddaljenem strežniku, so sledeči:

```
(1) docker login registry.gitlab.com
(2) sudo docker pull registry.gitlab.com/*/backend
(3) sudo docker run
      --name prod-diplomska-backend /
      --expose 8055
      --ip=172.77.0.17
      --net diplomska
      -d registry.gitlab.com/*/backend
```

Z ukazom (1) se na oddaljenem računalniku prijavimo v *GitLabov* register slik. Ukaz (2) prenese sliko na oddaljeni strežnik. Ukaz (3) začne z izvajanjem našega zabojnika iz slike, ki smo jo pred tem naložili iz zasebnega registra slik.

4.4.2 Zabojnik prod-diplomska-mysql-be

Zabojnik *prod-diplomska-mysql-be* vsebuje podatkovno bazo *MySQL*, ki jo uporablja zaledna aplikacija. Pred uporabo podatkovne baze moramo ustvariti podatkovno shemo tabel, ki hranijo naše podatke. Za ustvarjanje sheme smo naredili skripto *notifer.sql* ki vsebuje sql stavke, ki ustvarijo željene tabele. Slika 4.2 prikazuje shemo tabel podatkovne baze.



Slika 4.2: Slika podatkovne baze

Za gradnjo slike zabojnika *prod-diplomska-mysql-be* moramo ustvariti nov dokument Dockerfile, ki nam skripto z sql stavki prekopira v sliko. Primer dokumenta Dockerfile je podan spodaj. Z ukazom (1) povemo, da želimo našo sliko zgraditi iz osnovne slike z imenom *mysql*, ki je na voljo na javnem registru slik *DockerHub*. Z ukazom (2) kopiramo našo skripto za ustvarjanje podatkovne baze v sliko. Z ukazom (3) pa izpostavimo vrata 3306, preko katerih se bo naša zaledna aplikacija povezala s podatkovno bazo.

```
(1) FROM mysql
(2) COPY ./scripts/notifer.sql
      /docker-entrypoint-initdb.d/notifer.sql
(3) EXPOSE 3306
```

Sedaj, ko imamo dokument Dockerfile pripravljen, moramo zgraditi (1) in naložiti (2) sliko na naš privatni register slik. Sedeče dosežemo s spodnjimi ukazi:

```
(1) $sudo docker build -t registry.gitlab.com/*/mysql-be .
(2) $sudo docker push registry.gitlab.com/*/mysql-be
```

Za zagon zabojnika na strežniškem sistemu pa moramo zagnati spodnji ukaz, s katerim določimo ime zabojnika, uporabniško ime in geslo, ki ga rabimo za povezavo na podatkovno bazo, IP naslov zabojnika, omrežje ter ime naše slike, ki smo jo pred tem naložili na privatni register slik.

```
$sudo docker run \
   --name prod-diplomska-mysql-be \
   -e MYSQL_ROOT_PASSWORD=********* \
   -e MYSQL_DATABASE=db_name \
   -e MYSQL_USER=db_user \
   -e MYSQL_PASSWORD=********* \
   --ip 172.77.0.50 \
   --net diplomska \
   -d registry.gitlab.com/*/mysql-be
```

4.4.3 Zabojnik prod-diplomska-mysql-kc

Zabojnik *prod-diplomska-mysql-kc* vsebuje podatkovno bazo *MySQL*, ki jo uporablja ponudnik identitet *KeyCloak*. Za zabojnik *prod-diplomska-mysql-kc* ni potrebno ustvariti slike po meri, saj *KeyCloak* ob zagonu in povezavi na podatkovno bazo sam ustvari podatkovno shemo. Na oddaljenem strežniku zato samo poženemo ukaz `docker run` z osnovno sliko *mysql* iz registra slik *DockerHub* in nekaterimi dodatnimi atributi. Primer zagona zabojnika je podan spodaj.

```
$sudo docker run \
   --name prod-diplomska-mysql-kc \
   -e MYSQL_ROOT_PASSWORD=*******  \
   -e MYSQL_DATABASE=keycloak \
   -e MYSQL_USER=keycloak \
```

```
-e MYSQL_PASSWORD=******* \
--ip 172.77.0.60 \
--net diplomska \
-d mysql
```

4.4.4 Zabojnik prod-diplomska-keycloak-idp

Zabojnik *prod-diplomska-keycloak-idp* vsebuje rešitev *KeyCloak*. Ker nam osnovno sliko ponovno ponuja register slik *DockerHub*, je ne potrebujemo zgraditi sami. Zato za zagon zabojnika uporabimo ukaz `docker run` s parametri, ki so podani v spodnjem izpisu.

```
$sudo docker run \
   --name prod-diplomska-keycloak-idp \
   --ip 172.77.0.14 --net diplomska \
   -e KEYCLOAK_USER=usename \
   -e KEYCLOAK_PASSWORD=******* \
   -e MYSQL_DATABASE=keycloak \
   -e MYSQL_USER=keycloak \
   -e MYSQL_PASSWORD=******* \
   -e MYSQL_ROOT_PASSWORD=******* \
   -e MYSQL_PORT_3306_TCP_ADDR=172.77.0.60 \
   -e MYSQL_PORT_3306_TCP_PORT=3306 \
   -e PROXY_ADDRESS_FORWARDING=true \
   -d jboss/keycloak
```

4.4.5 Zabojnik prod-diplomska-mqtt

Zabojnik *prod-diplomska-mqtt* vsebuje posrednika sporočil *Mosquitto*. Za pravilno delovanje sporočilnega strežnika moramo v sliko dodati nastavitvene datoteke. Kot vidimo v spodnjem izpisu dokumenta Dockerfile, v ukazu (1) za osnovno sliko vzamemo uradno sliko, ki vsebuje posrednika sporočil *Mosquitto*. Z ukazom (2) prekopiramo nastavitveno datoteko *mosquitto.conf*,

v kateri omejimo prijavo samo tistim uporabnikom, ki jih podamo v datoteki *passwd*. Z ukazom (3) nato to datoteko prekopiramo v našo sliko. Na koncu izpostavimo vrata 1883, preko katerih se mobilna aplikacija poveže na posrednika sporočil.

```
(1) FROM eclipse-mosquitto
(2) COPY ./cfg/mosquitto.conf
      /mosquitto/config/mosquitto.conf
(3) COPY ./cfg/passwd
      /mosquitto/config/passwd
(4) EXPOSE 1883
```

Ukaz (1) zgradi našo željeno sliko. Ukaz (2) pa sliko naloži na privatni *GitLabov* register slik.

```
(1) $sudo docker build -t registry.gitlab.com/*/mosquitto .
(2) $sudo docker push registry.gitlab.com/*/mosquitto
```

Za zagon posrednika sporočil na oddaljenem strežniku izvedemo sledeči ukaz:

```
$sudo docker run \
   --name=prod-diplomska-mqtt \
   --ip=172.77.0.11 \
   --net diplomska \
   -p 1883:1883 \
   -d registry.gitlab.com/*/mosquitto
```

4.4.6 Zabojnik prod-diplomska-nginx

Zabojnik *prod-diplomska-nginx* vsebuje spletni strežnik *NGINX*, ki streže oz. ponuja spletno aplikacijo. Spodaj je prikazan dokument Dockerfile, ki vsebuje ukaze za gradnjo slike. Ukaz (1) pove, da bomo za osnovno sliko vzeli javno dostopno sliko na registru slik *DockerHub* po imenu *nginx*. Z ukazom (2) ustvarimo imenik, kjer se bodo nahajale datoteke, ki sestavljajo spletno aplikacijo. Da lahko spletni strežnik ponudi spletno aplikacijo, moramo

njene datoteke prekopirati v sliko z ukazom (3). Ukazi (4,5,6,7), prekopirajo nastavitvene datoteke, ki so potrebne za uspešen zagon spletnega strežnika in usmerjanje zahtevkov do rešitve KeyCloak ali zaledne aplikacije. Nastavitvene datoteke tudi določajo, ali bo spletni strežnik uporabljal varni protokol *HTTPS*, kje se nahajajo varnostni certifikati, itd. Zadnji ukaz (9) izpostavi vrata 443, ki so namenjena zahtevkom, ki prihajajo po protokolu *HTTPS*.

```
(1)FROM nginx
(2)RUN mkdir /home/website/
# datoteke spletne aplikacije
(3)COPY -R ./dist/ /home/website/
#konfiguracijske datoteke nginx strežnika
(4)COPY ./default   /etc/nginx/sites-enabled/default
(5)COPY ./ssl-params.conf /etc/nginx/snippets/ssl-params.conf
(6)COPY ./ssl-dd.net.conf /etc/nginx/snippets/ssl-dd.net.conf
(7)COPY ./nginx.conf /etc/nginx/nginx.conf
(8)COPY ./mime.types /etc/nginx/mime.types
(9)EXPOSE 443
```

Kot smo naredili v primeru gradnje slike zaledne aplikacije ali podatkovne baze *MySQL*, moramo tudi tu zgraditi prilagojeno sliko. To ponovno storimo z ukazom `docker build`:

```
$sudo docker build -t registry.gitlab.com/*/nginx .
$sudo docker push registry.gitlab.com/*/nginx
```

Na koncu za zagon zabojnika na strežniškem računalniku uporabimo ukaz `docker run`:

```
$sudo docker run \
   --name prod-diplomska-nginx  \
   -v /etc/letsencrypt/:/etc/letsencrypt/ \
   -v /etc/ssl/:/etc/ssl/  \
   --ip 172.77.0.13 \
```

```
--net diplomska \
-p 443:443 \
-d registry.gitlab.com/*/nginx
```

Poglavje 5

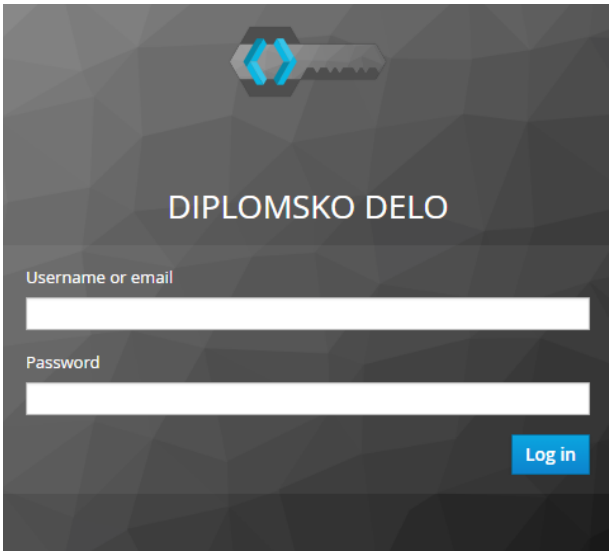
Uporabniški vmesniki

V tem poglavju bomo opisali uporabniške vmesnike in vnosne maske, ki jih uporabljajo uporabniki našega sistema. V našem sistemu bomo uporabniške vmesnike in vnosne maske razdelili v tri sklope. Prvi sklop uporabniških vmesnikov in vnosnih mask nam ponuja rešitev ponudnika identitet *KeyCloak*, drugi sklop pripada spletni aplikaciji, ki smo jo razvili za telefoniste, tretji sklop pa opisuje uporabniški vmesnik mobilne aplikacije, ki smo jo razvili za raznašalce.

5.1 Uporabniški vmesniki rešitve ponudnika identitet KeyCloak

5.1.1 Prijavna stran

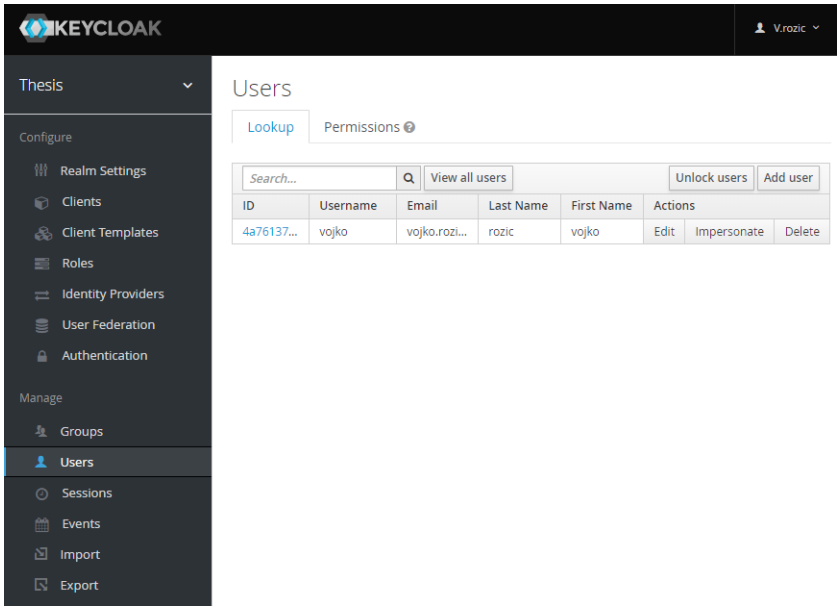
Rešitev *KeyCloak* in spletno aplikacijo lahko uporabljajo samo uporabniki tipa administrator in telefonisti, ki imajo dovoljenje za dostop. Uporabniki morajo zato pred uporabo sistema vnesti uporabniško ime in geslo preko prijavne strani, ki je prikazana na Sliki 5.1. Prijavna stran je del rešitve *KeyCloak* in nam je ni bilo potrebno razviti. Prijavna stran se navezuje na uporabniški zgodbi **A1** in **T1**.



Slika 5.1: Prijavna stran ponudnika identitet KeyCloak

5.1.2 Administracijski vmesnik za upravljanje telefonistov

Da lahko telefonist uporablja spletno aplikacijo, ga je potrebno dodati v sistem. Administrator to stori preko administracijske strani (Slika 5.2), ki nam jo ponuja ponudnik identitet *KeyCloak*. Administracijska stran je namenjena upravljanju z uporabniki tipa telefonist in jo pokriva uporabniška zgodba **A2**.



Slika 5.2: Administracijska stran rešitve KeyCloak

5.2 Uporabniški vmesniki spletne aplikacije

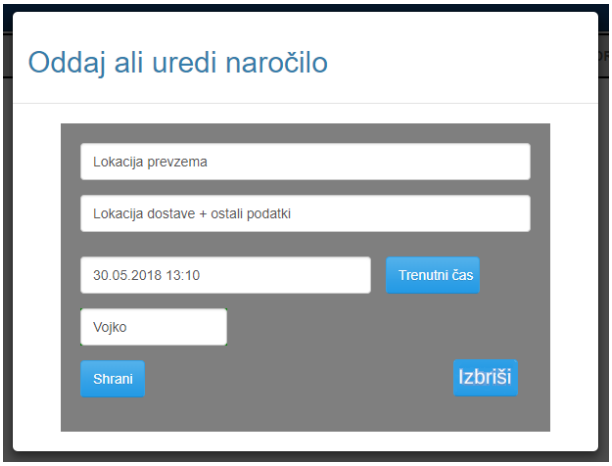
Spletno aplikacijo lahko telefonisti uporabljajo po uspešni prijavi v naš sistem. Kot smo omenili, to storijo na prijavni strani (Slika 5.1) z vnosom uporabniškega imena in gesla. Telefonistu se po prijavi odpre stran spletne aplikacije, kjer so prikazana naročila. Spletna aplikacija ponuja tri strani: *naročila*, *raznašalci* in *naprave*.

5.2.1 Naročila

Najpomembnejša stran spletne aplikacije, je stran, za pregled in urejanje naročil. Stran je prikazana na Sliki 5.3. Telefonist na strani naročil vidi vsa naročila, ki so bila ustvarjena tisti dan. Vsaka vrstica v tabeli predstavlja eno naročilo. Telefonist za vsako naročilo vidi zaporedno številko naročila, datum in uro oddaje naročila, lokacijo prevzema, lokacijo dostave z opcijskimi dodatnimi podatki in podatke o raznašalcu, ki je naročilo rezerviral. Za lažji pregled smo naročila, ki so že rezervirana, obarvali zeleno in naročila, ki še

niso bila rezervirana, z rdečo barvo. Poleg vizualnega pregleda naročil ima telefonist možnost dodajanja naročila s klikom na gumb *Novo* in za urejanje naročila s klikom na gumb *Uredi*, ki ga vidimo poleg vsakega naročila. Pregled naročil zajema uporabniška zgodba **T8**.

Slika 5.3: Pregled naročil



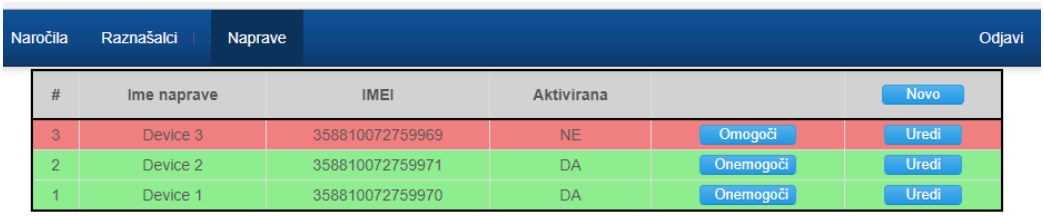
Slika 5.4: Vnosna maska za dodajanje ali urejanje naročil

Zgornja slika (5.4) prikazuje vnosno masko, ki se telefonistu prikaže, ko želi dodati ali urediti naročilo. Kot vidimo, ima telefonist štiri vnosna polja, kjer vpiše podatke, ki so potrebni za vnos ali posodobitev naročila. V prvo polje vpiše lokacijo prevzema, ki je lahko poljubno besedilo. Lokacija prevzema je v veliko primerih štirimestna številka stranke, saj veliko strank uporablja storitev vsakodnevno. V primeru, da naročnik dobi novo stranko,

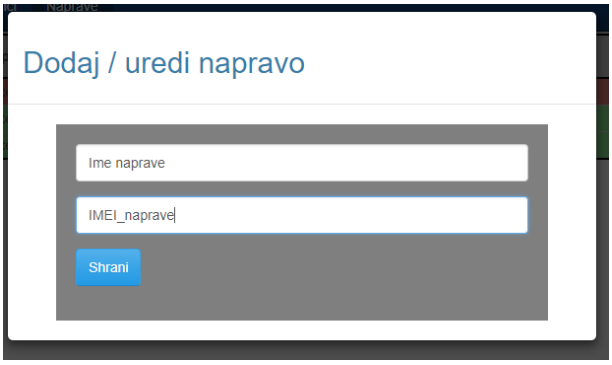
ki je še ne pozna, pa lahko v lokacijo prevzema napiše malo daljše besedilo z več podrobnostmi. Drugo polje, je lokacija dostave, ki vsebuje poljubno besedilo dostave. V večini primerov je to samo naslov, lahko pa telefonist doda več podrobnosti, kot je ime prevzemnika pošiljke ali pa telefonska številka osebe, ki bo pošiljko prevzela. Telefonist ima tudi možnost, da naročila ne odda v trenutku, ko je vnesel podatke. Če želi, da se naročilo raznašalcem prikaže čez dve uri, lahko to stori, s spremembo datuma in ure v tretjem vnosnem polju. Telefonist ima pri vnosu možnost, da za neko naročilo izbere točno določenega raznašalca. To stori z vpisom imena raznašalca v četrto vnosno polje. V primeru, da je polje prazno, se naročilo prikaže vsem. Naročilo se shrani v sistem po kliku na gumb *Shrani*. Če telefonist želi naročilo izbrisati, pa mora klikniti gumb *Izbriši*. Vnos novega naročila in urejanje ali izbris obstoječega naročila pokrivata uporabniški zgodbi **T4** in **T7**.

5.2.2 Naprave

Telefonisti imajo poleg vnosa naročil, tudi nalogo, da urejajo ostale podatke, ki so potrebni, da sistem pravilno deluje. Dodajanje in urejanje mobilnih naprav je ena izmed teh nalog. Spletna stran za pregled in urejanje je prikazana na Sliki 5.5. Lahko omenimo, da je uporaba te spletne strani zelo redka. Največkrat se uporablja po postavitvi sistema, nato pa v primeru, če se mobilna naprava pokvari ali pa je bila ukradena. Kot vidimo na Sliki 5.5, telefonist na enem mestu vidi vse mobilne naprave. Če je bila mobilna naprava ukradena, jo lahko onemogoči s klikom na gumb *Onemogoči* in s tem prepreči morebitne zlorabe. Podobno kot pri naročilih, imamo tudi tu možnost dodajanja mobilne naprave s klikom na gumb *Novo* in urejanje obstoječe naprave s klikom na gumb *Uredi*. V primeru, da je naprava omogočena, imamo v stolpcu *Aktivirana* napisano besedilo *DA*, v primeru onemogočene naprave pa besedilo *NE*. Pregled mobilnih naprav zajema uporabniška zgodba **T9**.



Slika 5.5: Pregled mobilnih naprav

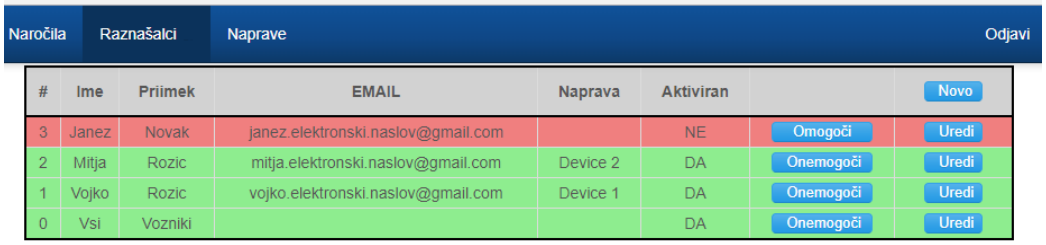


Slika 5.6: Vnosna maska za dodajanje ali urejanje mobilnih naprav

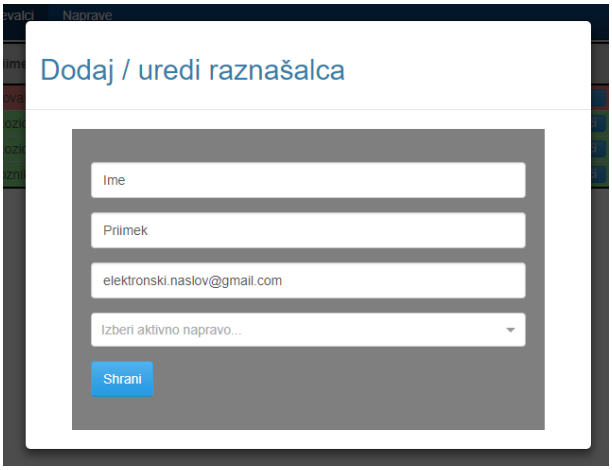
Slika 5.6 prikazuje vnosno masko, ki se telefonistu odpre, v primeru klika na gumb *Novo* ali *Uredi*. Telefonist ima na vnosni maski samo dve vnosni polji. Prvo polje je poljubno ime naprave, ki nam služi za lažjo identifikacijo le-te in drugo polje, ki vsebuje identifikacijsko številko IMEI naprave. Vnosna maska naprav se uporablja samo pri dodajanju nove naprave in v primeru, če se je telefonist pri prvem vnosu naprave zmotil pri številki IMEI, ali pa želi popraviti ime naprave. S klikom na gumb *Shrani* nato v sistem dodamo novo napravo. Po uspešnem vnosu mobilne naprave mora nato telefonist na strani, kjer pregleduje naprave (Slika 5.5), le-to omogočiti. To stori s klikom na gumb *Omogoči*. Vnos naprave in urejanje obstoječe naprave pokrivata uporabniški zgodbi **T2** in **T5**.

5.2.3 Raznašalci

Telefonisti imajo poleg dodajanja mobilnih naprav in naročil še tretjo nalogo, ki je pregled, dodajanje in urejanje raznašalcev. Spletna stran je prikazana na Sliki 5.7. Akcije, ki jih telefonist tu izvaja, so zelo podobne kot pri upravljanju z mobilnimi napravami. Telefonist s klikom na gumb *Novo* ali *Uredi* odpre vnosno masko za dodajanje novega raznašalca (Slika 5.8). V tabeli raznašalcev imamo prikazane njihove podatke, kot so ime, priimek, elektronski naslov in naprava, ki mu jo raznašalec dodeli. Tudi pri raznašalcih imamo možnost, da ga omogočimo v primeru, da smo ga pred tem na novo dodali. Lahko pa se zgodi, da raznašalec prekine delovno razmerje in ga je zato potrebno onemogočiti. To storimo s klikom na gumb *Omogoči* oz. *Onemogoči*. Če smo raznašalca omogočili se v stolpcu *Aktiviran* prikaže besedilo *DA* v nasprotnem primeru pa besedilo *NE*. Tudi tu moramo omeniti, da se spletna stran za pregled in urejanje raznašalcev uporablja zelo redko. Pregled raznašalcev zajema uporabniška zgodba **T10**.



Slika 5.7: Pregled raznašalcev

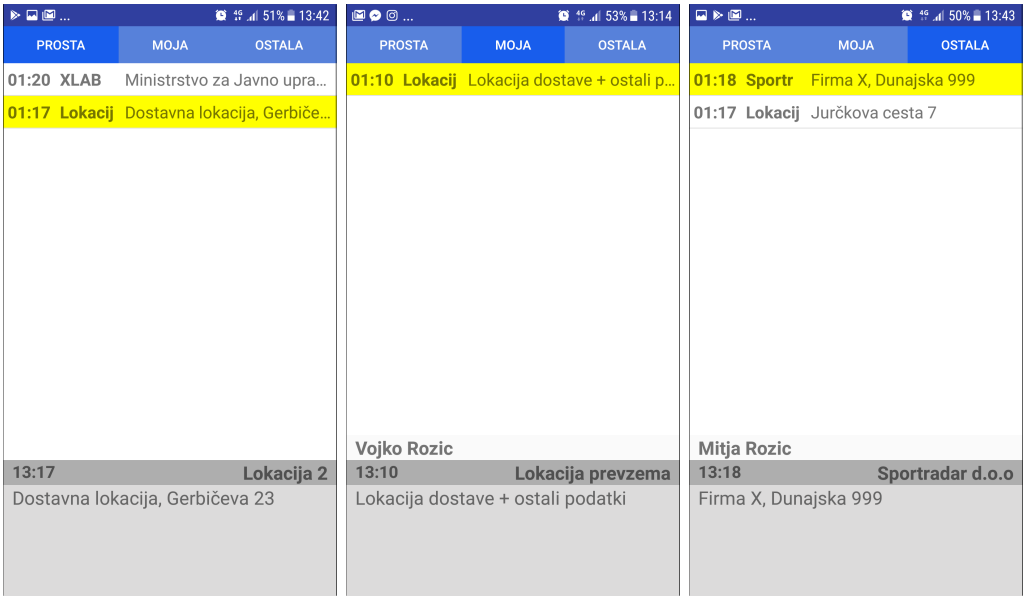


Slika 5.8: Vnosna maska za dodajanja ali urejanje raznašalcev

Slika 5.8 prikazuje vnosno masko, ki se telefonistu odpre v primeru klika na gumb *Novo* ali *Uredi*. Telefonist v prvo vnosno polje vnese ime raznašalca, v drugo priimek, v tretje polje elektronski naslov in v četrtem polju telefonist izbere napravo, ki jo raznašalec uporablja. S klikom na gumb *Shrani* nato v sistem dodamo novega raznašalca. Vnos raznašalca in urejanje obstoječega raznašalca pokrivata uporabniški zgodbi **T3** in **T6**.

5.3 Mobilna aplikacija

Za pregled in rezervacijo ali preklic rezervacije naročil raznašalci uporabljajo mobilno napravo z nameščeno aplikacijo, ki smo jo razvili za operacijski sistem Android. Uporabniški vmesnik mobilne aplikacije sestoji iz treh različnih pogledov, ki prikazujejo naročila v treh različnih stanjih. Vse tri poglede prikazuje Slika 5.9. Vsak pogled je razdeljen na tri komponente. Prva komponenta (obarvano modro) je navigacijska vrstica, preko katere raznašalec izbira med različnimi pogledi. Druga komponenta je seznam naročil, ki so prikazana glede na izbrano stanje. Seznam prikazuje uro, kdaj je bilo naročilo vneseno, del besedila lokacije prevzema in del besedila lokacije dostave z morebitnimi dodatnimi podatki. Tretja komponenta (obarvano sivo) prikazuje podrobnosti izbranega naročila z imenom in priimkom raznašalca, če je stanje naročila rezervirano. Izbiro različnih pogledov pokriva uporabniška zgodba **R1**.



(a) Prosta naročila (b) Moja naročila (c) Druga naročila

Slika 5.9: Grafični vmesnik mobilne aplikacije

5.3.1 Prosta naročila

Če telefonist odda novo naročilo, brez izbranega raznašalca, se na mobilni aplikaciji prikaže v pogledu prostih naročil (Slika 5.9a). Za prikaz podrobnosti naročila raznašalec izbere naročilo iz seznama, ki se nato v seznamu obarva z rumeno barvo. Aplikacija poleg izbranega naročila prikaže podrobnosti v spodnjem delu (obarvano v sivi barvi) uporabniškega vmesnika. Ker v tem primeru naročilo ni rezervirano, se ime raznašalca ne prikazuje.

Raznašalec ima sedaj možnost rezervacije naročila z uporabo geste *Poteg* (ang. swipe) s prstom z leve proti desni po spodnjem sivem delu uporabniškega vmesnika, kjer prikazujemo podrobnosti. Če je proces rezervacije uspel, se naročilo prestavi iz pogleda *prostih* v pogled *mojih* naročil. Ostali raznašalci vidijo rezervirano naročilo v pogledu *ostalih* naročil. Rezervacijo prostega naročila pokriva uporabniška zgodba **R2**.

5.3.2 Rezervirana naročila

Slika 5.9b prikazuje naročila, ki jih je raznašalec rezerviral. Pogled je zelo podoben pogledu prostih naročil. Razlika med njima je v prikazu imena in priimka raznašalca, v delu, kjer prikazujemo podrobnosti naročila. Tam bo vedno ime in priimek raznašalca, ki uporablja mobilno aplikacijo, saj so to naročila, ki jih je rezerviral on.

V tem pogledu lahko raznašalec prekliče rezervacijo naročila in ga prepusti drugim raznašalcem, da ga rezervirajo. To stori podobno, kot je storil pri rezervaciji. V prvem koraku iz seznama izbere naročilo. V drugem koraku pa po sivem delu uporabi gesto *Poteg* s prstom z desne proti levi. Če je bil proces preklica rezervacije uspešen, se naročilo prestavi iz pogleda *mojih* v pogled *prostih* naročil. Preklic rezervacije naročila pokriva uporabniška zgodba **R3**.

5.3.3 Ostala rezervirana naročila

Tretji pogled raznašalci uporabljajo samo za pregled naročil, ki so jih rezervirali drugi raznašalci. Pogled je prikazan na Sliki 5.9c. Raznašalcu se v primeru izbire naročila, v prikazu podrobnosti, prikažejo podatki o naročilu ter ime in priimek raznašalca, ki je to naročilo rezerviral. V tem delu raznašalec nima možnosti rezervacije ali preklica rezervacije, saj naročila niso prosta ali njegova. Lahko rečemo, da je ta pogled s strani raznašalca namenjen samo za branje oz. pregled. Uporabniška zgodba **R3** se nanaša na neuspešen preklic rezervacije naročila.

Poglavje 6

Zaključek

V začetnem poglavju smo predstavili problem, ki je bil povod za izdelavo sistema. Opisali smo proces, s katerim naročnik trenutno rešuje problem in si zadali cilj, da razvijemo sistem, ki bo proces izboljšal. Pogovori z naročnikom in zaposlenimi v njegovem podjetju so nam služili kot osnova pri načrtovanju sistema. Ugotovili smo, da mora naš sistem vsebovati tri tipe uporabnikov. Zahteve sistema in uporabnikov smo zajeli v obliki uporabniških zgodb in s sprejemnimi testi določili, kdaj bo posamezna uporabniška zgodba zaključena.

Opisali smo orodje Docker, čemu je orodje namenjeno in podali preprost primer, kako smo orodje uporabili pri razvoju našega sistema.

Po opisu smo predstavili komponente, ki sestavljajo naš sistem. Podali smo primere uporabe orodja Docker pri gradnji slik in izvajanju zabojnikov na oddaljenem strežniku.

Nadaljevali smo z opisom uporabniških vmesnikov sistema, njihove uporabe in povezave z uporabniškimi zgodbami.

Razviti sistem ima še veliko možnosti za izboljšavo. Trenutna postavitev sistema še vedno zahteva nekaj ročnega dela, a to bi lahko izboljšali z uporabo sprotne dostave in sprotne integracije. Telefonistom bi lahko dodali možnost tiskanja poročil ali pa možnost prikaza lokacije raznašalca na zemljevidu. Ker ima naročnik veliko stalnih strank, bi lahko le-tem omogočili, da naročila

oddajo same, preko spletnega vmesnika, brez klicanja telefonistov.

Tekom razvoja smo spoznali in preizkusili različne tehnologije in omenili samo tiste, ki so nam olajšale delo. Orodje Docker je ena izmed izbranih tehnologij, zato smo orodju namenili eno poglavje.

Rezultat tega diplomskega dela je delujoč sistem, s katerim si naročnik optimizira proces obveščanja. Dosegli smo, da je postavitev sistema na kateremkoli oddaljenem strežniku, ki ga poganja operacijski sistem Linux z nameščenim orodjem Docker, relativno hitra in zagotovili, da se sistem na akcije uporabnikov in spremembe podatkov odziva zelo hitro.

Literatura

[1] Android Operation System. Dosegljivo: https://en.wikipedia.org/wiki/Android_(operating_system), 2018. [Dostopano: 1. 8. 2018].

[2] Android Studio. Dosegljivo: https://developer.android.com/studio/, 2018. [Dostopano: 1. 8. 2018].

[3] Eclipse Paho Android Service. Dosegljivo: https://www.eclipse.org/paho/clients/android/, 2018. [Dostopano: 1. 8. 2018].

[4] Mike Cohn. *User Stories Applied*. New York: Addison-Wesley, 2008.

[5] Explore Official Repositories. Dosegljivo: https://hub.docker.com/explore/, 2018. [Dostopano: 1. 8. 2018].

[6] Get Docker CE for Debian. Dosegljivo: https://docs.docker.com/install/linux/docker-ce/debian/, 2018. [Dostopano: 1. 8. 2018].

[7] Eclipse Paho Java Client. Dosegljivo: https://www.eclipse.org/paho/clients/java/, 2018. [Dostopano: 1. 8. 2018].

[8] KeyCloak. Dosegljivo: https://www.keycloak.org/, 2018. [Dostopano: 1. 8. 2018].

[9] Hobson Sayers A Miell I. *Docker in practice*. Shelter Island: Manning Publicatons Co., 2016.

[10] Eclipse Mosquitto - An open source MQTT broker. Dosegljivo: https://mosquitto.org, 2018. [Dostopano: 1. 8. 2018].

[11] MQTT. Dosegljivo: https://github.com/mqtt/mqtt.github.io/wiki, 2018. [Dostopano: 1. 8. 2018].

[12] MySQL. Dosegljivo: https://en.wikipedia.org/wiki/MySQL, 2018. [Dostopano: 1. 8. 2018].

[13] React - A JavaScript library for building user interfaces. Dosegljivo: https://reactjs.org/, 2018. [Dostopano: 1. 8. 2018].

[14] Spring Framework. Dosegljivo: https://spring.io/, 2018. [Dostopano: 1. 8. 2018].

[15] Spring Boot. Dosegljivo: http://spring.io/projects/spring-boot, 2018. [Dostopano: 1. 8. 2018].

[16] WebSocket. Dosegljivo: https://en.wikipedia.org/wiki/WebSocket, 2018. [Dostopano: 1. 8. 2018].