Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Nejc Kambič

Postopek testiranja mobilne aplikacije za izposojo oblačil

diplomsko delo

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Mentor: viš. pred. dr. Igor Rožanc

Ljubljana, 2019

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V diplomskem delu predlagajte postopek za izvedbo testiranja mobilne aplikacije in ga prikažite na konkretnem primeru aplikacije za izposojo oblačil. V ta namen najprej kratko predstavite nekaj za vas pomembnih testnih prvin in okvirne zahteve za mobilno aplikacijo. Nato predstavite ustrezen razvojni cikel mobilne aplikacije in vanj umestite več korakov testiranja. Vsakega izmed njih kratko predstavite in na koncu sestavite v celovit postopek.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
PO		Programska Oprema
PB		Podatkovna zbirka
API	Application Programming Interface	Aplikacijski programski vmesnik
JSON	JavaScript Object Notation	JavaScript objekt za izmenjavo podatkov
XML	Extensible Markup Language	Razširljivi označevalni jezik
HTTP	HyperText Transfer Protocol	Protokol za prenos hiperteksta
FTP	File Transfer Protocol	Protokol za prenos datotek
IMAP	Internet Message Access Protocol	Internetni sporočilni dostopni protokol za prenos informacij
JDBC	Java DataBase Connectivity	Javanski vmesnik za povezovanje s podatkovnimi zbirkami
SOAP	Simple Object Access Protocol	Protokol za preprost dostop do objektov
REST	Representational State Transfer	Predstavitveni prenos stanja

Povzetek

Naslov: Postopek testiranja mobilne aplikacije za izposojo oblačil

Avtor: Nejc Kambič

V diplomski nalogi je predstavljen postopek testiranja pri razvoju Android aplikacije, ki je prikazan na primeru aplikacije za izposojo oblačil. Namen naloge je izpostaviti nekatere značilne posebnosti postopka, s katerim smo izboljšali kakovost aplikacije. Postopek testiranja temelji na uporabi specifičnega razvojnega cikla. Predstavljen je po korakih, v okviru katerega je opisan potek posameznega koraka in orodja, s katerimi smo v tem koraku testirali specifične izdelke.

V zadnjem delu je devet korakov sestavljenih v celovit pristop. Koraki testiranja so: testiranje zahtev aplikacije, testiranje podatkovne zbirke, testiranje programskega vmesnika, testiranje vmesnika API po namestitvi na Heroku, testiranje funkcionalnih zahtev in načrta aplikacije, testiranje Android aplikacije ter alfa in beta testiranje. Pri testiranju smo si pomagali s sledečimi orodji: JMeter, JetBrains Pycharm Community in Espresso Test Recorder.

Ključne besede: mobilna aplikacija, Android, Heroku, potek testiranja, JMeter, JetBrains Pycharm, Espresso Test Recorder.

Abstract

Title: The testing process for a clothes rental mobile application

Author: Nejc Kambič

The thesis presents the integration of testing into the development process of Android application for renting clothes. The purpose of the work is to describe the testing process by which we achieve higher quality of application. Testing process is presented step by step by describing the tools and the workflow of each test step. The main goals are the demonstration of testing, tools we used, the stages of the testing and tested products.

In the last part are nine steps composed into a comprehensive approach. Steps of testing are: the application requirements testing, the database testing, the application programming interface testing, the API testing after installation on Heroku, the functional requirements and the application design testing, the Android testing and the alpha testing and the beta testing.

Testing was done the following tools: JMeter, JetBrains Picharm Community in Espresso Test Recorder.

Keywords: mobile aplication, Android, Heroku, testing process, JMeter, JetBrains Pycharm, Espresso Test Recorder.

Poglavje 1

Uvod

Testiranje je nepogrešljivi del procesa razvoja programske opreme. Dandanes poznamo različna orodja in pristope, ki nam pomagajo pri testiranju. Ker pa je nabor pristopov in orodij velik, je težko izbrati prava orodja za specifične potrebe in jih uporabiti v pravem trenutku.

V diplomski nalogi je opisan proces testiranja, ki je bil uporabljen pri razvoju mobilne aplikacije za izposojo oblačil. Ta ima nekaj posebnih zahtev, ki vključujejo ustrezne pristope. Predstavljen je celoten proces testiranja, in sicer od testiranja specifikacije do beta testiranja.

Motivacija

Motivacijo za diplomsko nalogo smo dobili ob razvoju same aplikacije. Takrat smo se prvič srečali z vprašanjem, kako lahko testiranje vključimo v proces razvoja in pri tem, kar najbolj izkoristimo razpoložljivi čas ter kvalitetno izpeljemo proces testiranja. Pri iskanju različnih orodij in pristopov smo prišli do več različnih možnosti, ki pa niso v celoti zadovoljile naših pričakovanj. Zato smo predstavili svojo rešitev.

Cilj diplomske naloge

Cilj naloge je predstaviti proces testiranja mobilne aplikacije Android od začetka do konca razvoja. Pri posameznih korakih so predstavljeni izvajalci,

namen testiranja, postopek, orodja in rezultati testiranja. Na ta način je predstavljena naša rešitev testiranja, ki je bila uporabljena pri razvoju aplikacije za izposojo oblačil.

Poglavje 2

Vključitev testiranja v razvoj

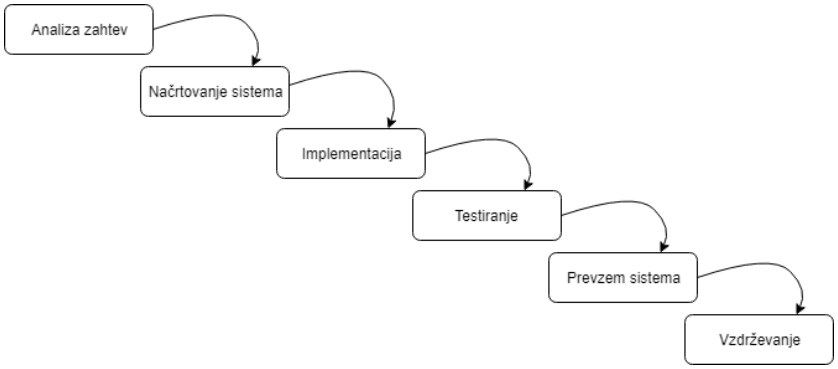
Razvoj programske opreme smiselno vključuje testiranje na različne načine. V našem primeru je razvoj pravzaprav mešanica slapovnega, vzporednega in razvoja v skladu z modelom V. Nadalje so aktivnosti testiranja skrbno načrtovane in izvedene s specifičnimi vrstami testiranja. Vse te tematike so na kratko predstavljene v tem poglavju.

2.1 Življenski cikli razvoja PO

2.1.1 Slapovni model

Slapovni model (ang. waterfall model) je proces razvoja, v katerem se aktivnosti izvajajo zaporedno (slika 2.1). Testiranje je vključeno kot zadnji korak pred namestitvijo aplikacije. Postopek je načrtovan pred začetkom procesa, v njem pa so natančno definirani koraki posamezne aktivnosti [30].

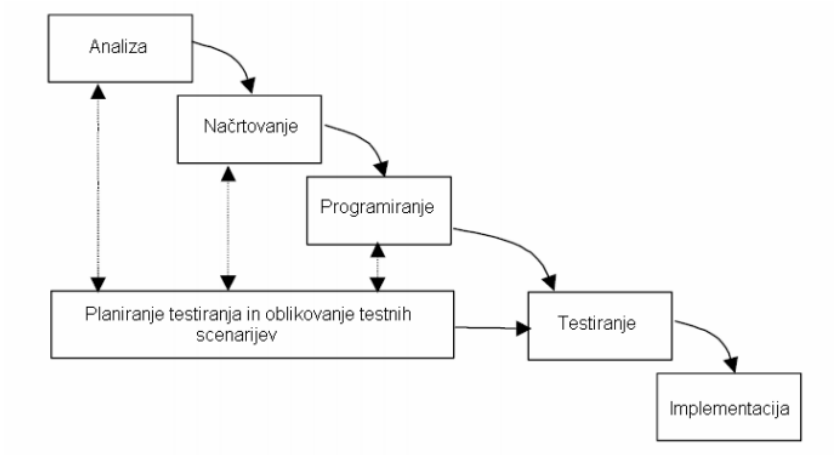
Težava takšnega pristopa je vključitev testiranja po zaključeni izvedbi posameznih faz. V primeru najdenih napak pri testiranju je odprava zahtevna in dolgotrajna, saj zahteva ponavljanje že izvedenih faz razvoja. Uspešnost projekta s slapovnim pristopom je, v primeru modernejših tehnologij običajno nizka.



Slika 2.1: Slapovni model.

2.1.2 Vzporedni model testiranja

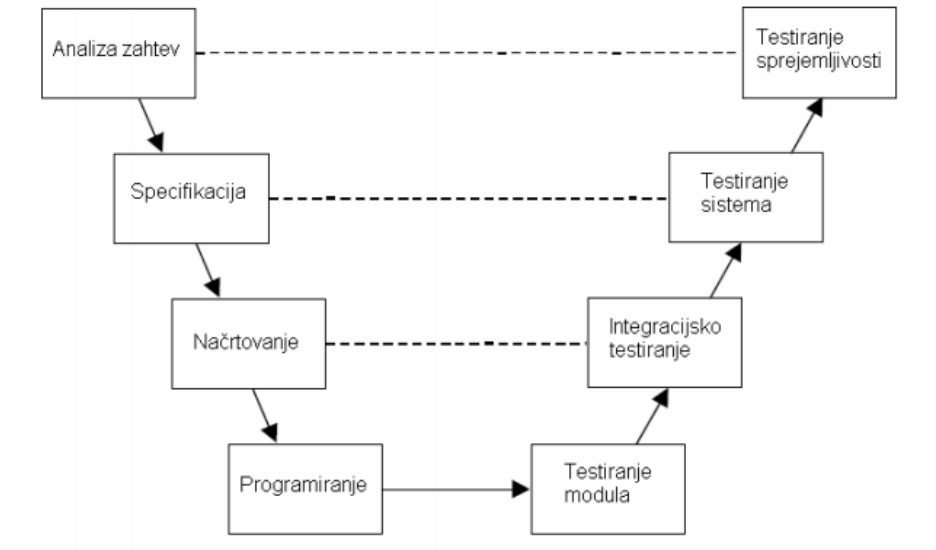
Vzporedni model testiranja (ang. parallel testing cycle) je nastal kot izboljšava slapovnega modela. Proces vsebuje izboljšave pri planiranju in oblikovanje testnih scenarijev ob izvedbi zgodnejših faz razvoja (slika 2.2). Če se med procesom spreminja specifikacija programa, se naredi primerjava z oblikovanimi testnimi plani, ki se po potrebi tudi spreminjajo. Tak način je v primerjavi s slapovnim modelom ponavadi dražji, saj je med razvojem potrebno velikokrat spreminjati testne plane [38].



Slika 2.2: Vzporedni cikel testiranja.

2.1.3 Model V

V tem primeru ima vsaka aktivnost v procesu že vključeno načrtovanje testiranja. Leva stran procesa predstavlja postopek razvoja, desna stran pa postopek testiranja (slika 2.3). Testiranje se izvaja vzporedno z razvojem, rezutati testiranja trenutne faze, pa so povezani z rezultati predhodnje faze testiranja. Z zaključkom testiranja sprejemljivosti brez napak je aplikacija primerna za izdajo na trg. Glavna prednost takšnega pristopa je vključevanje testiranja v vse faze razvoja [28].



Slika 2.3: Razvojni model V.

2.2 Načrtovanje testiranja

Načrtovanje testiranja je začetni korak procesa testiranja, ki ga vodi testni vodja. Pojavi se na vseh ravneh testiranja od testiranja enot do testa namestitve. Namen načrtovanja je natančno določiti, kako doseči cilje pri izvajanju testiranja s čim večjo uspešnostjo (slika 2.4) [40].

Slika 2.4: Ključna vprašanja pri načrtovanju testiranja

Testiranje mora biti dobro pripravljeno, sicer hitro nastane zmeda. Sistem lahko razpade in vse skupaj pripelje do neuspešnosti testiranja [34]. Načrt testiranja pripravi testni vodja z namenom sistematično predstaviti načine, s katerimi bodo cilji testiranja uspešno doseženi. Načrt mora biti pripravljen tako, da upošteva tudi potrebo po usmerjanju samega testiranja. Pripravljen mora biti skladno z razvojem izdelka ali vzdrževanjem, zato je v planiranje vključen tudi projektni vodja. Načrt testiranja mora biti natančen, razumljiv in sestavljen po korakih, po katerih se kasneje izvaja testiranje [39]. Ker vedno obstaja možnost, da se aplikacija v prihodnosti spreminja, mora načrt vključevati načine zbiranja in sledenja ustreznosti meritev aplikacije. Ker zbiranje in sledenje meritev zahteva uporabo različnih orodij, s katerimi testerji kasneje vnašajo podatke o meritvah, je pomembno, da so vključeni tudi podatki o orodjih.

Jasno opredeljeni cilji, začetek in zaključek testiranja ter tehnike na vseh ravneh testov bodo zagotovili optimalno izvedbo testiranja. Rezultati testiranja na posameznih ravneh bodo tako tudi preglednejši in bolj razumljivi [37].

Testiranje je lahko precej zapleten, drag in dolgotrajen proces. Pomembno je, da testni vodja skrbno načrtuje potek testiranja in naveže sodelovanje z drugimi udeleženci projektne skupine. S tem izvemo več informacij, ki

olajšajo postavitev testnega okolja (recimo: arhitekturi sistema, poslovnih procesih in poteku razvoja PO). Z dobrim sodelovanjem vodja tudi lažje organizira sam potek ter si zagotovi proste vire, ko bo to potrebno. Testno ekipo ne sestavljajo le testerji, temveč tudi drugi člani razvojne ekipe. Ti zagotavljanjo pravilno delovanje testnega okolja in so prisotni ob morebitnih težavah.

2.3 Pristopi testiranja

V nadaljevanju so kratko predstavljeni različni prstopi testiranja, ki jih bomo uporabili v našem testnem procesu.

2.3.1 Testiranje enot

Program je običajno sestavljen iz več delov, ki so med seboj povezani. Najmanjši del programa imenujemo enota (recimo metoda ali funkcija). Posamezno enoto lahko testiramo neodvisno od ostalih enot tako, da podamo vhodne podatke in na podlagi rezultata ugotavljamo pravilnost. Testiranje enot (ang. unit testing) izvajajo programerji po pristopu bele škatle (ang. white box) z namenom odkrivanja napak na samostojnih enotah, ki še niso povezane med seboj [34]. Razvijalec sestavi nabor testnih primerov, s katerim pokrije čim več možnosti in s tem poveča verjetnost pravilnega delovanja enote.

Testiranje se običjno opravlja po naslednjih korakih [34]:

- *Pregledovanje kode (ang. code review):* z branjem kode poskušamo ugotoviti, ali se kje skriva napaka. Kodo pri tem primerjamo s specifikacijo.

- *Prevajanje kode (ang. code compiling):* s prevajanjem kode iščemo sintaktične napake v programu.

- *Dinamično testiranje (ang. dynamic testing):* pripravimo in izvajamo testne primere. S tem ugotavljamo ali se vhodni podatki pravilno preslikajo v izhod.

2.3.2 Testiranje integracije

Namen testiranja je tudi preveriti delovanje med seboj povezanih enot. Testiranje integracije (ang. integration testing) je podobno testiranju enot, vendar z eno veliko razliko [37]; pri testiranju enot je testna enota izolirana od ostalih, pri testiranju integracije pa je enota odvisna od drugih enot. S testiranjem hočemo dokazati, da komponente delujejo tako, kot je zapisano v specifikaciji programa. Izvaja ga testna skupina s pomočjo programerjev po pristopu črne škatle (ang. black box) in bele škatle [27].

Testiranje je možno izvajati na tri načine, in sicer z integracijo od spodaj navzgor, integracijo od zgoraj navzdol in kombinirano integracijo.

2.3.3 Sistemsko testiranje

Sistemsko testiranje (ang. system testing) vključuje vse sestavljene komponente, ki sestavljajo različico oz. verzijo sistema. Testiranje preverja skladnost komponent, pravilnost interakcij in ustreznost prenašanja podatkov v pravilnem sosledju preko vmesnikov. Sistemsko testiranje je pomemben del, s katerim zaključimo cikel testiranja z vidika razvijalca. Če je sistem sestavljen iz več podsistemov, ki so jih razvijale različne ekipe, potem v tem koraku preverjamo medsebojno delovanje posameznih elementov [41]. Po nekaterih delitvah delimo sistemsko testiranje po namenu.

Funkcionalni testi

Že samo ime nam pove, da se pri funkcionalnem testiranju (ang. functional testing) posvetimo funkcionalnostim sistema. Testiranje vključuje vrednotenje in primerjavo vsake programske funkcije s poslovnimi zahtevami [41], kar je običajno osrednji način ustreznosti delovanja.

Pomembno je, da testna ekipa razume, kaj je namen sistema in kako ga bo uporabnik uporabljal. Razumeti morajo proces razvoja testirane aplikacije. Prepoznati in razumeti morajo poslovna pravila, zato je pomembno, da so člani vključeni v proces razvoja programske opreme od samega začetka.

Funkcionalno testiranje se običajno pojavi v več delih. Tako kot gre razvoj aplikacije po modulih, tako gre tudi funkcionalno testiranje. Veliki sistemi se vedno razdelijo na več manjših podsistemov. Ti se kasneje razvijajo vzporedno ali še bolj pogosto zaporedno. Ko se določen del zaključi se tudi testirajo njegove funkcionalnosti [8].

Preformančni testi

Preformančno testiranje (ang. performance testing) je pomemben del testiranja sistema, ki pa se velikokrat zanemari. S tem testiranjem simuliramo časovne omejitve, sočasnost uporabnikov in delovanje sistema pod določeno obremenitvijo. Na ta način poiščemo ozka grla v sistemu. Performančne teste običajno izvedemo tako, da najprej posnamemo skripto, ki simulira dejanskega uporabnika [36]. Skripto nato izvajamo z različnimi konfiguracijami (število uporabnikov, število poslanih zahtevkov ipd.).

Z izvajanjem preformančnih testov lahko ugotovimo, ali smo zagotovili delovanje aplikacije za določeno število uporabnikov pod določenimi pogoji. Tako pridobimo podatke o preformačni zmogljivosti aplikacije, ne da bi stranka nameščala in testirala aplikacijo. Pri tem lahko časovno in finančno prihranimo, saj ni potrebno čakati na odziv stranke in nameščati PO pri stranki [36]. Za izvajanje preformančnega testa so pomembne tudi informacije o hitrosti omrežja, zmoglivosti strežnika na katerem teče aplikacija, število že delujočih aplikacij na sistemu itd.

Sprejemni testi

Sprejemni testi (ang. user acceptance testing) so vrsta testiranja, ki jo izvaja naročnik. S testom preverjamo ustreznost aplikacije glede na zahteve, ki so bile zapisane pred začetkom projekta. Testiranje se izvaja v zadnji fazi,

preden se aplikacija prestavi na produkcijsko okolje. Takšno testiranje izvaja stranka v ločenem okolju, ki je enako produkcijskem okolju, in potrdi delovanje sistema glede na specifikacijo [35]. Sprejemni testi se izvajajo po metodi črne škatle. Testiranje je uspešno zaključeno, ko stranka potrdi pravilnost implementiranih funkcionalnih zahtev. Odprava napak v tej fazi testiranja je najdražja, saj je odkrivanje in popravljanje težavno, sistem pa je potrebno ponovno testirati od začetka [34].

V določenih primerih, ko je izdelek namenjen širokemu naboru uporabnikov, lahko sprejemno testiranje nadomestimo z alfa in beta testiranjem.

Alfa testiranje

Alfa testiranje (ang. alpha testing) je vrsta testiranja, ki se začne izvajati, ko je izdelek že skoraj v uporabnem stanju in traja do konca razvojnega procesa. Testiranje izvaja testna skupina projekta. Namen testiranja ni zgolj testiranje funkcionalnih zahtev, ampak tudi razumevanje uporabiške izkušnje [31].

Beta testiranje

Beta testiranje (ang. beta testing) aplikacije je podobno alfa testiranju, le da izvajajo testiranje končni uporabniki aplikacije.

Beta različica aplikacije se pripravi za omejeno število končnih uporabnikov, da bi pridobili povratne informacije o kakovosti izdelka. Testiranje zmanjšuje tveganja za napake v aplikaciji in zagotavlja večjo kakovost izdelka s potrditvijo končnih uporabnikov. To je tudi končni preizkus, preden se izdelek pošlje na trg.

Poglavje 3

Opis aplikacije

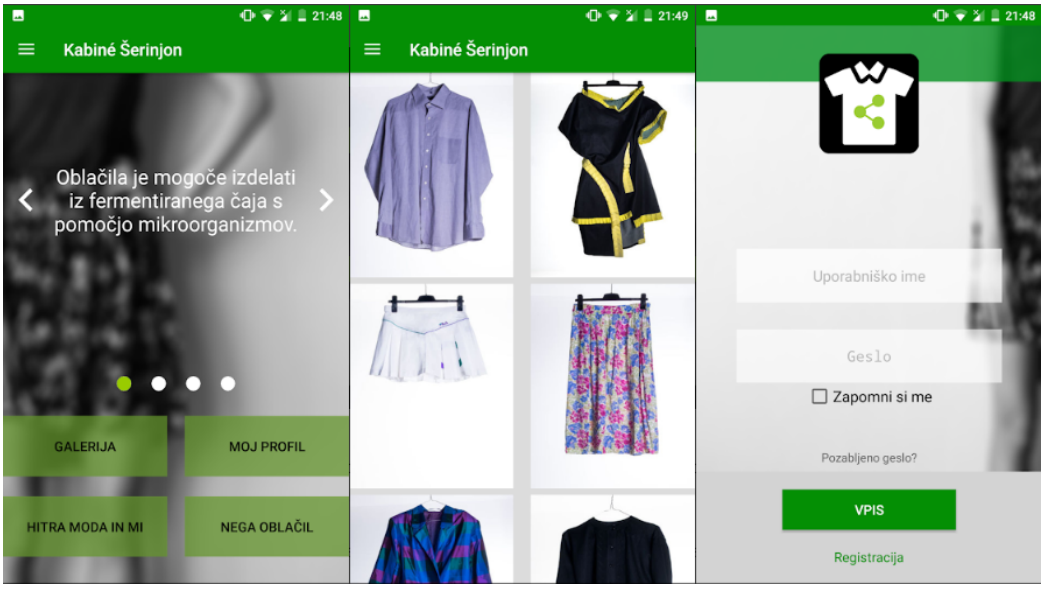
Naš pristop za testiranje mobilne aplikacije bo predstavljen na preprostem zgledu aplikacije za brezplačno izposojo oblačil.

Ideja aplikacije je brezplačna izposoja oblačil za določen čas (slika 3.1), spodbujanje uporabnikov o okoljski problematiki pri proizvodnji in predaja informacij o negovanju oblačil. V aplikaciji so shranjene informacije o različnih kosih oblačil, ki so jih podarili slovenski modni oblikovalci. Registrirani uporabniki te podatke uporabljajo za pregled in izposojo oblačil. Vsak registriran uporabnik si lahko katerikoli kos oblačila brezplačno izposodi, poskrbi zanj po objavljenih navodilih in oblačilo po uporabi vrne nazaj.

Aplikacija je bila razvita za operacijski sistem Android [14], ki je trenutno najbolj razširjen med uporabniki pametnih telefonov. Na trgu je trenutno več kot 80 % naprav [25] s tem operacijskim sistemom. Pri razvoju je bilo uporabljeno orodje Android Studio[1] [18], ki omogoča programiranje v programskem jeziku java ter oblikovanje grafične podobe aplikacije v jeziku XML.

V nadaljevanju poglavja so opisane funkcionalne in nefunkcionalne zahteve ter sestavni deli aplikacije.

[1]Android Studio je integrirano razvojno okolje (ang. integrated development environment) za operacijski sistem Android. Orodje temelji na programski opremi IntelliJ IDEA JetBrains [19] in je posebej zasnovano za razvoj Android aplikacij.



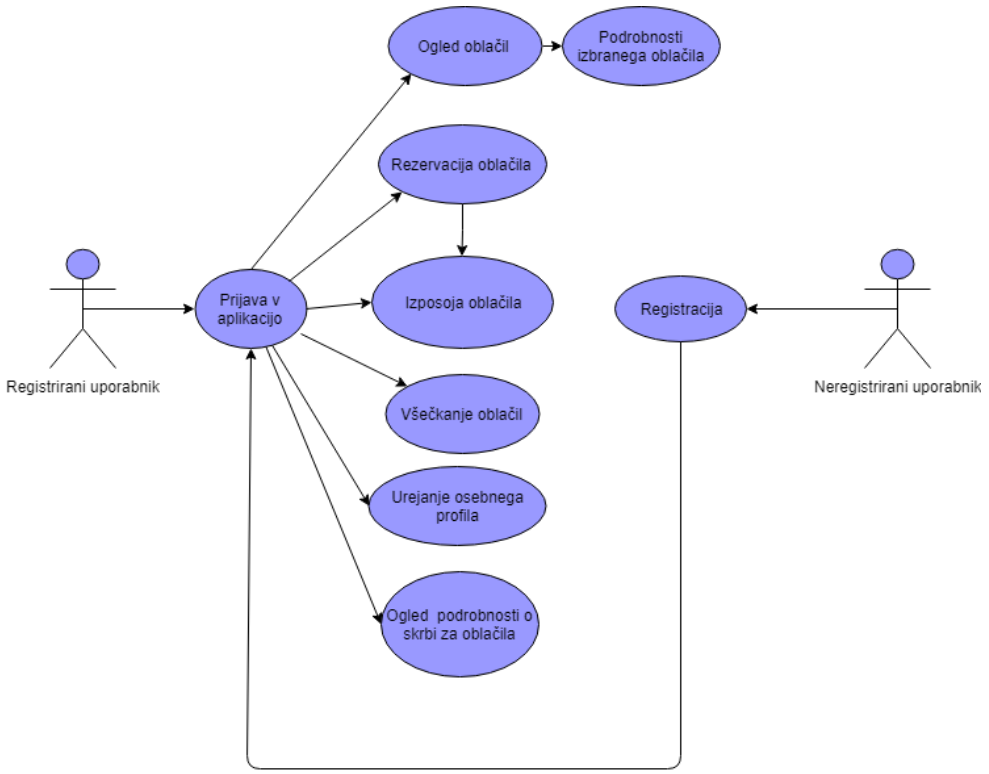
Slika 3.1: Aplikacija za izposojo oblačil.

Primer uporabe aplikacije

Mobilno aplikacijo lahko uporabljajo samo registrirani uporabniki, kar prikazuje diagram uporabe aplikacije (slika 3.2). Neregistrirani uporabnik ima možnost registracije v sistem preko zaslona za registracijo. Ob uspešni registraciji ga aplikacija preusmeri na glavni meni apikacije. Registrirani uporabnik lahko vidi vsa oblačila v sistemu in po želji določeni kos oblačila všečka.

Ob izbiri določenega oblačila na seznamu pridobi podrobne podatke o oblačilu (velikost, kateremu spolu je oblačilo namenjeno in podatki o negi izbranega oblačila). Poleg osnovnih podatkov lahko za vsako oblačilo preveri tudi, ali je že izposojeno. Uporabnik si lahko izposodi kos oblačila za časovno obdobje do štirinajstih dni. V tem času mora pazljivo ravnati z oblačilom in ga negovati v skladu s podanimi informacijami. Oblačilo nato vrne in izposodi si ga naslednji uporabnik. Ko se približuje rok za vrnitev oblačila, uporabnik dobi opozorilo, da mora v roku treh dni vrniti oblačilo. Že izposojena oblačila si uporabnik lahko rezervira in počaka, da pride na vrsto za izposojo.

Na svoji osebni strani ima uporabnik pregled izposojenih oblačil in oblačil, ki so mu všeč. V aplikaciji so tudi navodila o ravnanju z oblačili, njihovem čiščenju in negovanju.



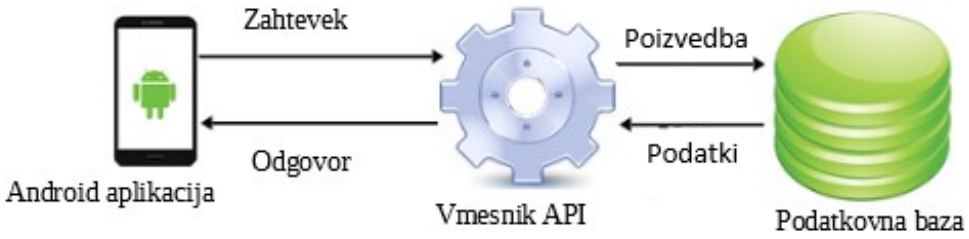
Slika 3.2: Diagram uporabe aplikacije.

Sestavni deli aplikacije

Za delovanje aplikacije smo morali ob *Android aplikaciji*, ki skrbi za prikaz podatkov in izposojo oblačil, zasnovati ustrezno *podatkovno zbirko*. Ta skrbi za hrambo uporabniških podatkov, informacij o izposojenih oblekah in rezervacijah. Za komunikacijo med podatkovno zbirko in Android aplikacijo smo izdelali *aplikacijski programski vmesnik (API)* [2]. Vmesnik prejema zahtevke

[2]Vmesnik API (ang. application programming interface) je množica rutin, protokolov in orodij za izdelavo aplikacij. Vmesnik za programsko komponento določa: operacije med podatki ter vhodne in izhodne podatke.

aplikacije, poišče podatke v podatkovni zbirki in jih vrne nazaj (slika 3.3).

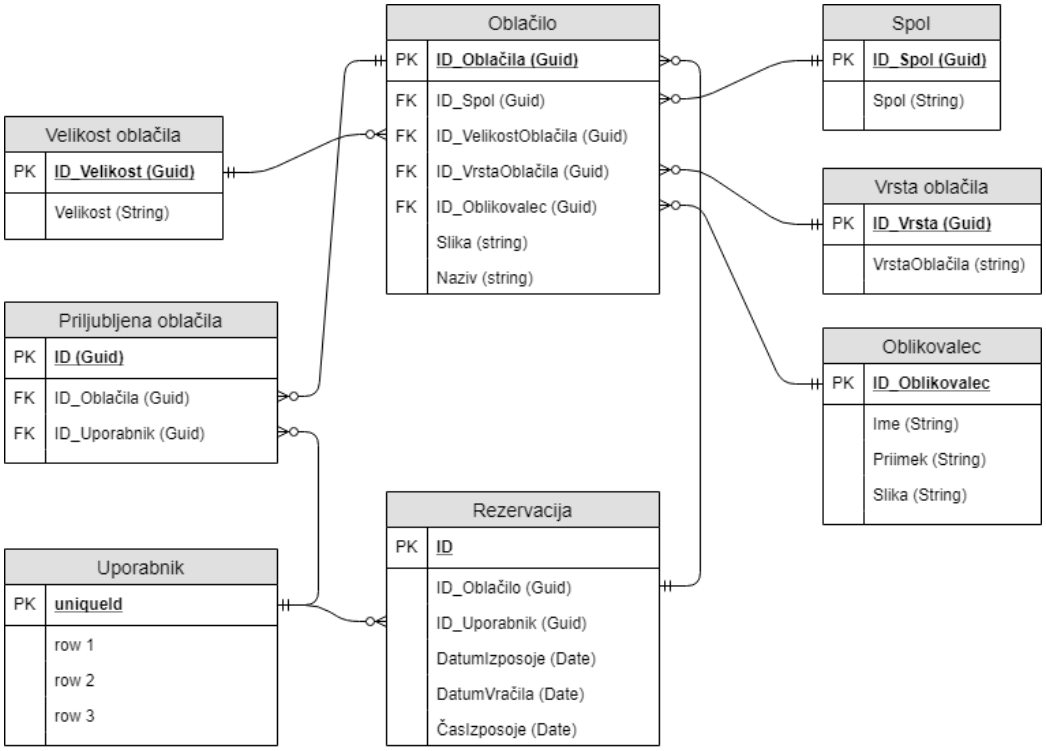


Slika 3.3: Arhitektura prenašanja podatkov iz podatkovne zbirke preko vmesnika API do mobilne aplikacije.

Podatkovna baza

Za hrambo podatkov aplikacije smo izdelali podatkovno bazo z orodjem pgAdmin [33]. Orodje omogoča izdelavo podatkovnih baz PostgreSQL, je odprtokodno ter podprto na operacijskih sistemih Linux, Unix, Mac OS X in Windows.

Za potrebe aplikacije smo izdelali bazo, ki je sestavljena iz osmih tabel (slika 3.4), v katerih so shranjeni posamezni podatki. Tako shranjujemo podatke o oblekah, velikosti posameznega oblačila, o tem, kateremu spolu je namenjeno in podatke o velikosti oblačila. Za posamezno oblačilo imamo shranjen podatek o oblikovalcu in rezervaciji posameznega uporabnika. Prav tako imamo za uporabnike shranjene podatke o priljubljenih oblačilih.



Slika 3.4: Diagram podatkovne baze aplikacije.

Funkcionalne zahteve

Naročnik aplikacije je opisal idejo za izdelavo mobilne aplikacije in podal informacije o načinu delovanja aplikacije. Na podlagi pogovora z narčnikom je bila izdelana specifikacija o delovanju same aplikacije. Pri razvoju mobilne Android aplikacije je bilo potrebno zagotoviti delovanje sledečih funkcionalnih zahtev:

- *Registracija:* uporabnik aplikacije se mora ob prvi uporabi aplikacije registrirati v sistem. Pri tem mora vnesti obvezne podatke: ime, priimek, uporabniško ime in geslo. Ob registraciji ima možnost dodajanja profilne slike iz galerije ali preko fotoaparata. Pri vnosu gesla mora uporabnik vnesti geslo, ki vsebuje vsaj osem znakov, eno veliko črko in eno številko ali poseben znak.

- *Prijava v aplikacijo:* registrirani uporabniki preko prijavnega zaslona vnesejo podatke za prijavo v aplikacijo. Na izbiro morajo imeti polje, kjer lahko izberejo samodejno prijavo ob zagonu aplikacije.

- *Pregled vseh oblačil:* vsi registrirani uporabniki morajo imeti dostop do seznama oblačil. Seznam vsebuje po dve sliki oblačil v vrsti, pri čemer so vse slike enake velikosti. Ob kliku na sliko se odpre okno s podrobnejšimi podatki o izbranem oblačilu.

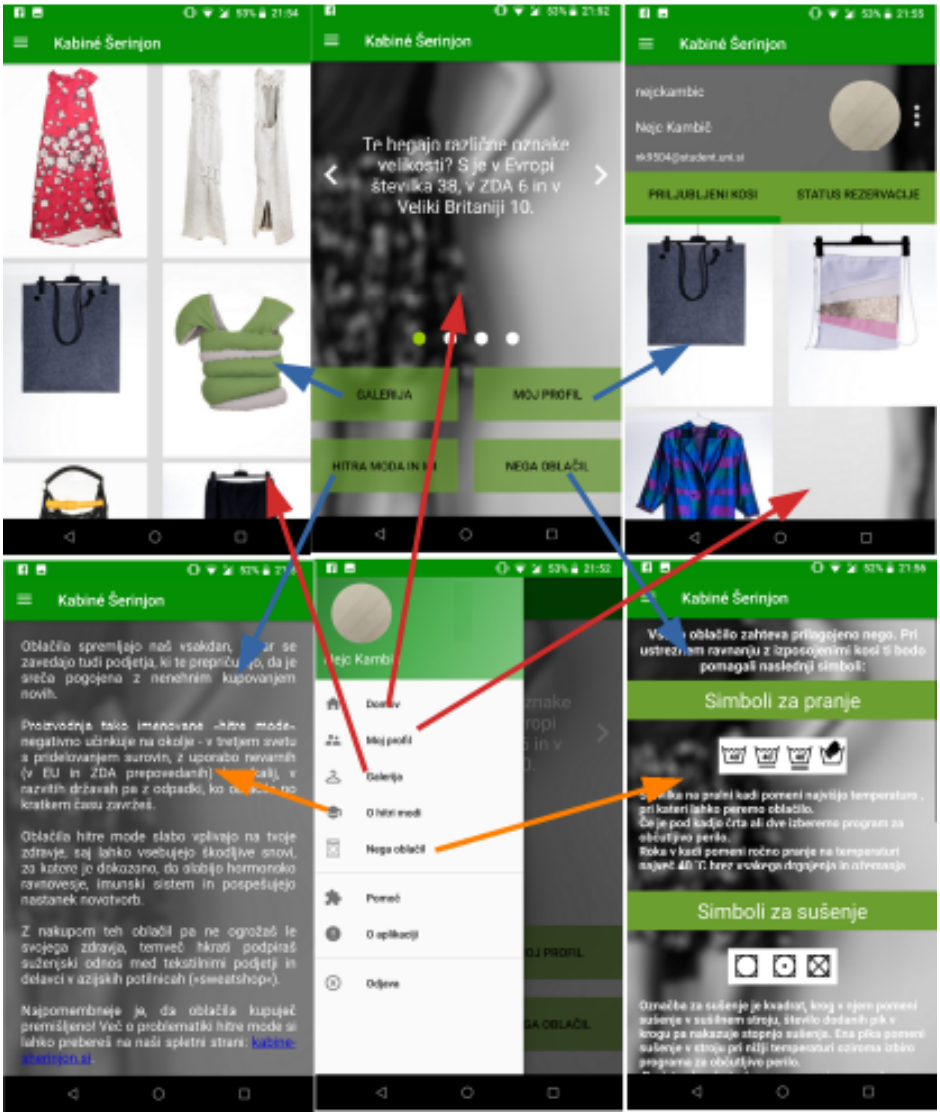
- *Izposoja oblačila:* vsa oblačila v aplikaciji morajo biti na voljo za izposojo. Uporabnik izbere oblačilo in s klikom na gumb potrdi svoj namen. Čas izposoje je omejen na 14 dni. Uporabnik, ki ne vrne oblačila pravočasno, dobi začasno prepoved izposoje. Če se zamujanje pri vračanju oblačil ponavlja, dobi uporabnik dokočno prepoved izposoje. Uporabnik ima hkrati lahko izposojene največ tri kose oblačil.

- *Rezervacija oblačila:* ko je oblačilo izposojeno, ga lahko ostali uporabniki rezervirajo. Ob rezervaciji mora uporabnik dobiti informacijo, kdaj si bo lahko oblačilo izposodil in koliko uporabnikov se nahaja pred njim v čakalni vrsti. Vsa rezervirana oblačila mora uporabnik videti na svoji uporabniški strani.

- *Všečkanje oblačil:* če je uporabniku določen kos oblačila všeč lahko to tudi označi s klikom na gomb v obliki srčka. Oblačilo se prikže v seznamu všečkanih oblačil na uporabnikovi strani.

- *Urejanje in pregled osebnega profila:* uporabnik ima možnost urejanja osebnih podatkov in profilne fotografije.

Na levi stani uporabniškega vmesnika mobilne aplikacije se nahaja navigacijski meni (slika 3.5). S pomočjo menija lahko uporabnik preklopi med različnimi zasloni v aplikaciji. Tako ima uporabnik dostop do vseh zaslonov ne glede na trenutni prikazani zaslon.



Slika 3.5: Prehodi med zasloni aplikacije. Zgoraj: *galerija, glavni zaslon, uporabniški profil.* Spodaj: *o hitri modi, meni, nega oblačil.*

Nefunkcionalne zahteve

Naročnik je zahteval delovanje aplikacije na mobilnih napravah z operacijskim sistemom Android verzije 4.4 [29] [3] ali novejši. Delovanje aplikacije mora biti zagotovljeno ne glede na velikost zaslona in znamko pametnega telefona.

V testni fazi mora biti zagotovljeno stabilno delovanje ob 5000 registriranih uporabnikih in podpora za 300 oblači v sistemu. Aplikacija mora biti enostavna za uporabo, velikost pisave pa ne sme biti premajhna. Ob prenosu oblačil se morajo oblačila prikazati v nekaj sekundah, tudi če aplikacija vsebuje 300 oblačil.

[3] Google v sodelovanju s podjetji združenja Open Handset Alliance (OHA) razvija operacijski sistem Android. Prva verzija v uporabi je bila ver. 1.1, ki je izšla leta 2009. Android 4.4 (z drugim imenom KitKat) je izšel leta 2013.

Poglavje 4

Postopek testiranja aplikacije

V tem delu je predstavljen naš pristop k testiranju tovrstnih specifikacij. Pri tem izhajamo iz naslednjih izhodišč:

1. Pred začetkom razvoja aplikacije je potrebno razmisliti o vključevanju testiranja v proces razvoja.

2. Aplikacija mora biti dokončana v določenem roku, v katerega spada tudi testiranje in odprava napak.

3. Pomembno je, da so izpolnjene tako funkcionalne kot nefunkcionalne zahteve.

4. Vemo, da imamo na voljo le določeno število programerjev, testerjev in ostalih članov skupine. Tu ima glavno nalogo vodja projekta, ki skrbi za pravilno razdeljeno delo in neoviran potek procesa razvoja.

Pri načrtovanju procesa razvoja PO smo si postavili dve ključni vprašanji, ki sta bili pomembni za razvoj in testiranje aplikacije:

Kako in kdaj vključiti testiranje v proces razvoja? S pravilnim vključevanjem testiranja lahko hitreje in bolj učinkovito odkrijemo napake. S tem zmanjšamo čas, ki bi bil namenjen odpravi napak v kasnejših fazah razvoja.

Zaključitev projekta v določenem roku: Če želimo zadovoljne uporabnike, mora biti aplikacija zaključena v vnaprej določenem roku in delovati pravilno.

4.1 Vključitev testiranja v razvoj

Preden smo se odločili, kako bomo testiranje vključili v razvoj, smo pregledali že obstoječe modele. Modele smo analizirali in poiskali njihove dobre in slabe lasnosti. Po pregledu smo se odločili za sledeče:

- Testiranje ne sme biti samo en korak v procesu pred predajo aplikacije naročniku.
- Testiranje se mora izvajati v več korakih v okviru različnih faz procesa razvoja aplikacije.
- Če je le mogoče, se mora testiranje izvajati vzporedno z razvojem PO. S tem bomo najbolje izkoristili čas, ki je na voljo za dokončanje projekta.
- Upoštevati je potrebno možne zamike pri razvoju aplikacije in vzporedno planirati začetek testiranja specifičnega izdelka.

Izdelali smo diagram procesa razvoja programske opreme (slika 4.1), ki je izpeljan iz modela V in slapovnega modela. Diagram temelji na podlagi obeh ključnih vprašanj, predhodni analizi obstoječih modelov in predhodno postavljenih izhodišč. Posamezne korake testiranja in razvoja smo poskusili povezati tako, da bi čim več aktivnosti potekalo vzporedno. Zato smo določili vse aktivnosti v procesu razvoja in za posamezno fazo preučili, ali jo lahko izvajamo neodvisno od ostalih.

Naš proces razvoja PO vsebuje dva vzporedna toka razvoja, ki se na koncu združita.

Razvoj podatkovne zbirke in vmesnika API: sestavljen je iz treh korakov razvoja: *načrtovanja podatkovne zbirke*, *izdelave podatkovne zbirke*

in *načrtovanja in razvoja vmesnika API.* V vsak korak je vključeno tudi testiranje. Koraki testiranja so bili naslednji: testiranje načrta podatkovne zbirke, testiranje podatkovne zbirke in testiranja vmesnika API.

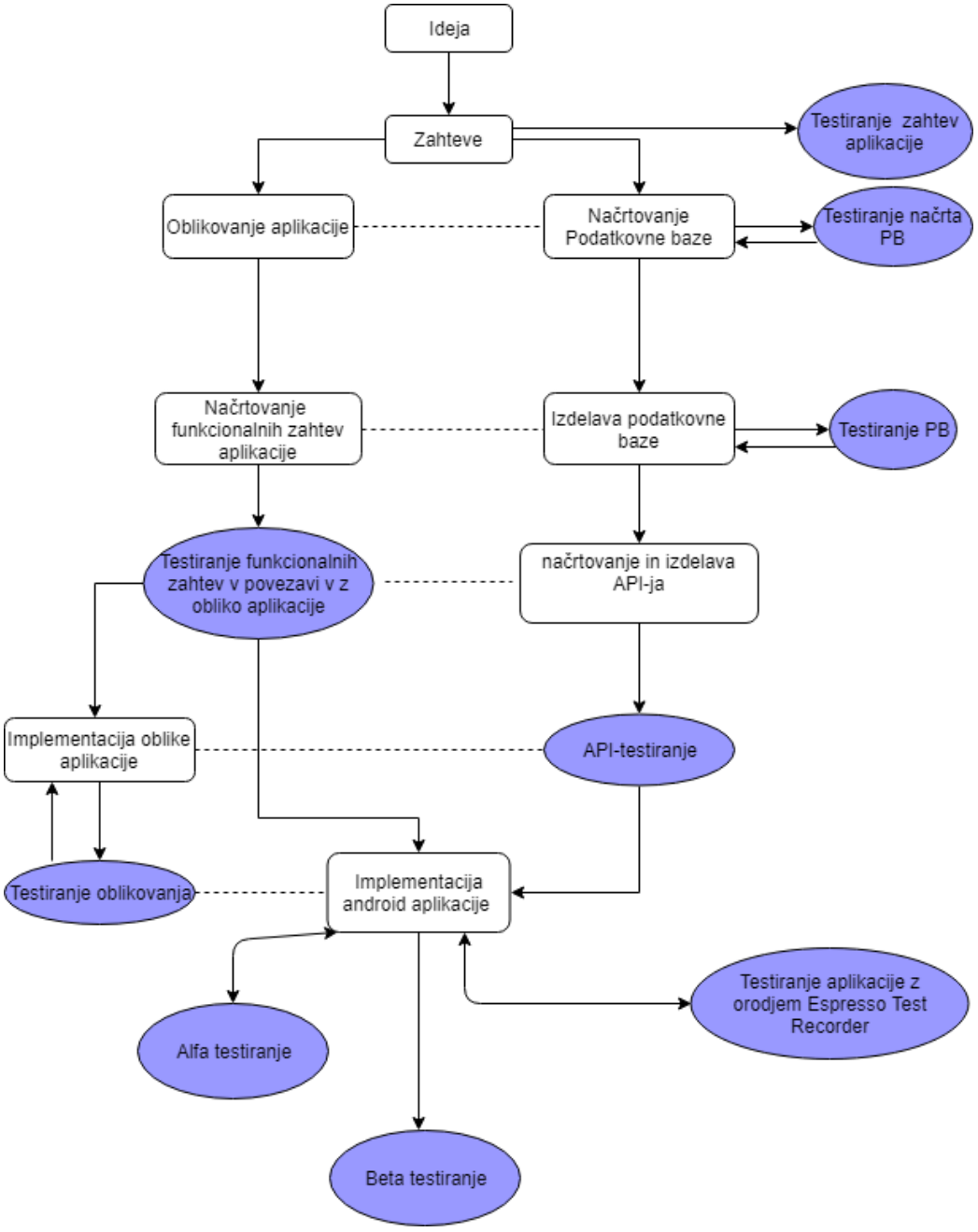
Načrtovanje in implementacija izgleda aplikacije: sestavljen je iz *načrtovanja izgleda in analiziranja funkcionalnih zahtev aplikacije.* Na koncu sledi še *testiranje funkcionalnih zahtev v povezavi z načrtom aplikacije.* Tok razvoja se nato razdeli še na dva dela: implementacija načrta s testiranjem in implamentacija Android aplikacije.

Oba glavna toka procesa razvoja se nato združita v naslednji korak *implementacije Android aplikacije,* ki poteka vzporedno s *testiranjem načrta.* Po zaključeni implementaciji Android aplikacije smo izvedli še alfa in beta testiranje aplikacije.

Določen korak testiranja je vključeval testiranje specifičnega izdelka, ki je nastal v razvojnem koraku. Izdelek je nato prešel v fazo testiranja, kjer so ga testerji preizkusili in podali rezultate. Pri določenih fazah testiranja smo si pomagali s posebnimi orodji za testiranje, s pomočjo katerih je bilo testiranje lažje in bolj učinkovito.

Vsi koraki testiranja so bili skrbno načrtovani že pred začetkom specifične faze razvoja. Pri tem smo jasno določili namen testiranja, potek, testiran izdelek in izvajalce. Določili smo tudi predviden začetek in trajanje testiranja. Ker pri procesu razvoja prihaja do težav in časovnega zamika, smo korake procesa razvoja načrtovali vzporedno z razvojem. Tak način sicer prinaša dodatna usklajevanja nadaljnih faz razvoja, vendar je učinek običajno večji. Pri planiranju razvoja je bilo potrebno upoštevati tudi odpravo odkritih napak pri testiranju. Zato smo pri določitvi začetka vsake faze razvoja predvideli tudi čas za odpravo odkritih napak.

Ko je izdelek uspešno prestal proces testiranja, se je začela naslednja faza v procesu razvoja. Če pa so bile pri testiranju odkrite napake, smo izdelek predali nazaj v specifično fazo razvoja. Koraka razvoja in testiranja sta se izvajala tako dolgo, dokler ni izdelek uspešno prestal testiranja.



Slika 4.1: Proces razvoja in testiranja aplikacije.

V nadaljevanju diplomske naloge bomo vse korake testiranja sistematično opisali.

4.2 Testiranje zahtev aplikacije

Prvi testirani izdelek je bila *specifikacija aplikacije*. Testiranje se je začelo po zaključitvi načtovanja zahtev. V njem so bile testirane funkcionalne in nefunkcionalne zehteve zapisane v specifikaciji. Ko je izdelek uspešno prestal testiranje, sta se vzporedno začela izvajati fazi načrtovanja izgleda aplikacije in načrtovanja podatkovne zbirke.

Pri testiranju so sodelovali *projektni vodja*, *programer (vodja razvoja)* in *naročnik aplikacije*. Njihova naloga je bila preverjanje funkcionalnih in nefunkcionalnih zahtev v specifikaciji

Namen

Pravilno definirane specifikacije aplikacije so ključni izdelek procesa razvoja. Na podlagi specifikacije se izvajajo vse naslednje faze razvoja in testiranja, zato je pomembno preveriti pravilnost specifikacije, saj bi napake povročile verižno reakcijo; pojavile bi se namreč tudi v naslednjih fazah in celoten proces razvoja bi bilo potrebno ponoviti.

Namen testiranja specifikacije je preverjanje funkcionalnih zahtev aplikacije z naročnikom. S tem se odpravjo vse kasnejše težave pri razvoju projekta. Naročnik je seznanjen z delovanjem aplikacije in predlaga spremembe.

Potek testiranja

Testiranje se je izvajalo, še preden se začene razvoj aplikacije. Sodelujoči so pregledali specifikacijo aplikacije za izposojo oblačil in iskali napake. Skupaj so pregledali sledeče:

- postopek za izposojo oblačil,
- postopek prijave in registracije,
- diagram uporabe aplikacije ter
- nefunkcionalne zahteve aplikacije.

Rezultat testiranja

Rezultat te dejavnosti so bile odkrite napake v specifikaciji aplikacije glede na zahteve naročnika. Vse napake so bile zapisane in komentirane v poročilu testiranja. Dokument je vseboval sledeče napake:

1. Napačno definirana registracija: ko uporabnik klikne gumb *registracija*, mora imeti možnost dodajanja slike. Takrat se mora pokazati sporočilo, v katerem lahko izbere način dodajanja slike. Poleg tega mora registracija vsebovati povezavo do splošnih pogojev uporabe.

2. Napaka pri rezervaciji oblačil: če ima uporabnik že izposojena tri oblačila, ne more rezervirati naslednjega kosa. Rezervacija je omogočena, ko vrne vsaj eno izmed izposojenih oblačil.

3. Napaka pri izposoji: če uporabnik ne prevzame rezerviranega oblačila v treh dneh, se njegova rezervacija prekliče. Oblačilo je na voljo naslednjemu uporabniku v vrsti.

4. Napaka pri samodejni prijavi v aplikacijo: če uporabnik spremeni geslo preko aplikacije, mora pri naslednji prijavi ponovno vnesti geslo in ga potrditi.

Specifikacijo je bilo potrebno popraviti glede na poročilo testiranja. Uspešnost te faze testiranja se nam je obrestovala pri nadaljnjem razvoju. Napake v tej fazi lahko kasneje povzročijo dražje popravilo.

4.3 Testiranje načrta podatkovne zbirke

Pri tem koraku testiranja je bil testiran izdelek *načrt podatkovne zbirke*. Testiranje je izvajal *administrator podatkovnih baz*. Ker je načrt podatkovne zbirke zelo pomemben za samo izdelavo, je potrebno v njem poiskati morebitne napake in jih odpraviti pred tvorbo PB.

Namen

Namen testiranja je preverjanje pravilnosti načrta podatkovne zbirke. Tako lahko zagotovimo dostopnost do vseh podatkov, pravilne podatkovne tipe, možnost urejanja in shranjevanja ter pravilnost povezanih entitet.

Neodkrite napake v načrtu podatkovne zbirke povzročajo neustrezno podatkovno zbirko. S tem testiranjem se izognemo kasnejšemu iskanju in odpravljanju napak, kar je običajno veliko težje.

Potek testiranja

Tudi podatkovno zbirko preverjamo glede na specifikacijo aplikacije. Vsi potrebni podatki za delovanje morajo biti pravilno shranjeni v podatkovni bazi. Testiranje je potekalo po sledečih korakih:

1. *Pregled tabel, atributov in podatkovnih tipov:* glede na specifikacijo aplikacije je potrebno preveriti, ali so vsi atributi pravilno shranjeni. Atributi morajo biti v ustrezni tabeli in pravilnega podatkovnega tipa. Vsaka tabela, v kateri so shranjeni podatki, mora imeti pravilno definiran primarni ključ (ang. primary key) in tuje kjuče (ang. foreign keys).

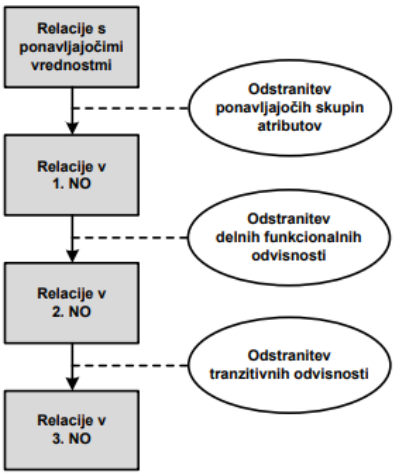
2. *Obveznost in neobveznost podatkov:* v specifikaciji aplikacije je določeno, kateri podatki so obvezni in kateri neobvezni (primer obveznih podatkov: ime, priimek, uporabniško ime; primer neobveznih podatkov: cena oblačila, telefonska številka uporabnika itd.). Tako je potrebno preveriti, ali je v načrtu pravilno določena obveznost specifičnega podatka.

3. *Relacije in številnosti med tabelami:* podatki so v podatkovni bazi shranjeni v različnih tabelah. Da lahko v aplikaciji prikažemo pravilne podatke (na primer, katera oblačila imamo izposojena, kateri smo v čakalni vrsti za specifično oblačilo ipd.), morajo biti tabele pravilno povezane med seboj. Pri tem postopku preverjamo relacije med posameznimi tabelami.

4. *Testiranje vzdržnosti transakcij na podatkovni bazi:* ta korak je nadaljevanje testiranja predhodnega koraka. Preveriti moramo, ali lahko iz določene tabele pridobimo podatke preko relacij med tabelami (relacijo sestavljajo tuji in primarni ključi). Poleg pridobivanja podatkov moramo preveriti še shranjevanje in urejanje posameznega podatka v tabeli.

5. *Normalizacija podatkovne zbirke:* to je zadnji korak v testiranju načrta podatkovne zbirke. Normalizacija procesa, ki vodi relacijsko shemo PB skozi zaporedje testov. Z vsakim testom preverimo, ali so izpolnjeni določeni pogoji (slika 4.2 prikazuje proces normalizacije). S tem procesom se znebimo podvajanja podatkov ter poskrbimo, da v bazi ne prihaja do anomalij med podatki. Proces vsebuje tri korake: *odstranitev ponavljajočih skupin atributov*, *odstranitev delnih funkcionalnih odvisnosti* in *odstranitev tranzitivnih odvisnosti.*

Pri postopku testiranja načrta podatkovne zbirke smo prišli do sledčih rezultatov.



Slika 4.2: Postopek normalizacije podatkovne zbirke.

Rezultat testiranja

Rezultati testiranja podatkovne zbirke za aplikacijo za izposojo oblačil so prikazani po posameznih korakih:

1. *Pregled tabel in podatkovnih tipov* je pokazal, da so napačno izbrani nekateri podatkovni tipi. Atribut za shranjevanje telefonske številke bi moral biti tipa *niz (ang. string)* namesto *celoštevilskega tipa (ang. integer)*. Tako lahko omogočimo shranjevanje telefonske številke v pravilnem formatu *"+386"*

2. *Pregled obveznosti in neobveznosti podatkov* je pokazal, da morajo biti atributi, ki opisujejo uporabnika (telefonska številka, slika, prebivališče in datum rojstva) neobvezni, saj niso pogoj pri registraciji. Prav tako se je odkrila napaka na oblačilih, saj atribut za shranjevanje slik ni bil obvezen (vsa oblačila v aplikaciji morajo imeti sliko).

3. *Pri testiranju relacij in številnosti med tabelami v podatkovni zbirki* ni bilo odkritih napak.

4. *Pri testiranju vzdržnosti transakcij na podatkovni bazi* ni bilo odkritih napak.

5. *Test normalizacije podatkovne zbirke* je pokazal, da je bil načrt že v tretji normalni obliki. To pomeni, da na bazi ne bo prišlo do anomalij in podvajanja podatkov.

Rezultati testiranja so bili predstavljeni v dokumentu, ki je vseboval potek iskanja napak z opisom, v kateri fazi so bile napake odkrite in na katerem nivoju testiranja je prišlo do napak.

4.4 Testiranje programskega vmesnika

Po izdelavi in testiranju podatkovne zbirke je sledila implementacija vmesnika API. Ta ima nalogo prenašanja oblačil v aplikacijo, skrbi za izposojo in

rezervacijo oblačil ter prijavo in registracijo oblačil. To je programer izvedel z uporabo testiranja enot in testiranja integracije, in sicer s pristopom bele škatle.

Namen

Namen testiranja je preverjanje delovanja posameznega dela vmesnika API in nato še delovanje teh delov, ko so povezani med seboj. Tako preverimo pravilno delovanje posamezne funkcionalnosti, ki jo potrebujemo za delovanje aplikacije. Testiranje vmesnika API je eden najpomembnejših korakov testiranja, saj skrbi, da aplikacija pridobi potrebne podatke za njeno delovanje. Napake na vmesniku API bi lahko v našem primeru pomenile odpoved delovanja Android aplikacije. Aplikacja preko vmesnika pošilja podatke o registraciji, prijavi, rezervaciji in izposoji oblačil. Ti podatki se na vmesniku obdelajo po določenem delovnem toku in nato aplikacija pridobi odgovor v obliki rezultata obdelave (primer rezultata: oblačila v PB, informacija o rezervaciji, pravilnost vnešenih podatkov za prijavo ipd.).

Orodja za testiranje

Pri testiranju smo vključili knjižnice, ki omogočjajo pošiljanje zahtevkov za posamezno funkcijo vmesnika API in preverjanje rezultatov.

Za testiranje smo uporabili sledeča orodja in knjižnice:

1. *Programsko orodje Pycharm* [20] za programiranje v programskem jeziku python. Orodje je bilo razvito s strani podjetja JetBrains in omogoča namestitev na operacijske sisteme Windows [17], Linux [15] in macOS [16]. Orodje omogoča vključevanje različnih knjižnic, ki nam pomagajo pri programiranju (recimo: knjižnica `math` [22], ki omogoča uporabo funkcij za seštevanje seznama števil, računanje logaritmov ipd.).

2. *Knjižnica unittest* [23] je ena izmed knjižnic, ki nam omogoča pisanje testnih metod. Napisana je bila po zgledu knjižnice JUnit [13] in

podpira avtomatizacijo testov, zaustavitev izvajanja kode ter pregled stanja v trenutku začasne zaustavitve in združevanje testov v zbirke.

3. Knjižnico *flask* [21] in *JSON* [12].

Potek testiranja

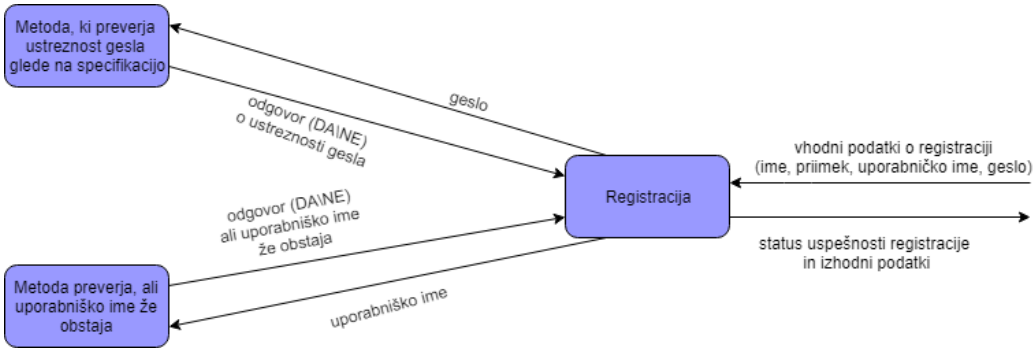
Programer je pri testiranju vmesnika API napisal več testnih primerov z izbranim programskim orodjem in pripadajočimi knjižnicami. Testiranje je zajemalo pravilnost statusa zahtevka in prejetih podatkov. Slika 4.4 prikazuje primere testov, kjer lahko vidimo testiranje enote, ki preverja, ali uporabnik obstaja v PB. Ta metoda vrača odgovor *da (uporabnik ostaja) ali ne (uporabnik ne obstaja),* glede na prejeto uporabniško ime, ki je vhodni parameter. Testna metoda primerja odgovor s pričakovanim rezultatom. Na podlagi prejetega odgovora in pričakovnega rezultata testna metoda vrača uspešnost testiranja enote (slika 4.5).

Po testiranju enot je bilo izvedeno še testiranje integracije. Iz slike 4.4 lahko vidimo primera testiranja registracije in prijave. Metoda *registracija* je povezana z metodo, ki preverja, ali uporabnik že obstaja (uporabniško ime je že zasedeno), in z metodo, ki preverja, ali sta bili pri registraciji vnešeni gesli enaki (pri registraciji uporabnik vnese gesli, ki morata biti enaki). Pri tem testiranju smo preverjali pravilnost delovanja povezanih enot med seboj (slika 4.3) in pri tem preverjali, ali metoda vrača pravilen status in podatke ob napaki (recimo: neveljavno geslo, uporabniško ime že obstaja ipd.). Prav tako smo preverjali pravilnost statusa in podatkov ob uspešni registraciji.

Testiranje se je izvajalo, dokler niso bile odpravljene vse nepravilnosti vmesnika. Odkrite napake so se odpravile glede na poročilo o napakah, ki smo ga dobili preko izbranega orodja (slika 4.5 prikazuje poročilo o napakah pri izvajanju testov).

Na podlagi rezultatov testiranja je bilo potrebno odpraviti napake in ponovno izvesti testiranje. Vsako naslednje testiranje je poleg že napisanih testov zajemalo tudi nove testne primere. Tako je bilo v celoti testirano

delovanje sistema.



Slika 4.3: Povezovanje metod *registracija*, preverjanje gesla in obstoja uporabniškega imena.

Rezultati

Rezultat testiranj so podatki o pognanih testih (slika 4.5). Uporabljena orodja nam kot rezultat vračajo uspešno izvedene teste in neuspešne teste. Razvijalec vmesnika tako dobi vpogled v rezultate in napake.

Na sliki 4.5 lahko vidimo primer štirih testnih metod. Razvidno je, da je pri dveh testih prišlo do napake, zato je potrebno napake poiskati in jih odpraviti. Tu so nam v pomoč rezultati testov. Iz slike 4.3 lahko vidimo, da sta bili testirani dve povezani metodi, pri katerih je prišlo do napake. Metoda za registracijo je povezana z enoto, ki preverja zasedenost uporabniškega imena. Lahko sklepamo, da je velika verjetnost napake pri testiranju metode *registracija* posledica nedelovanje metode za preverjanje zasedenosti uporabniškega imena. Rezultati testiranja so nam bili v pomoč pri odkrivanju napak, saj vemo, da moramo najprej preveriti samostojno metodo, ki je neuspešno prestala testiranje. Šele po odpravi napak v tej metodi, lahko na podlagi rezultatov iščemo napake v metodi za registracijo

```
def test_userExist(self):
    self.assertEquals(True, backend.TSHM.routes.user_exist("serinjonac"))
    self.assertEquals(True, backend.TSHM.routes.user_exist("Ursula"))
    self.assertEquals(False, backend.TSHM.routes.user_exist("Nejc.Novak"))

def test_Login(self):
    response =self.app.post('/login',data=json.dumps(dict(UporabniskoIme='gr.p', Geslo='8c17c981d86e1334dd7d04fe4e45
    data = json.loads(response.data)
    self.assertEquals(200, response.status_code)
    self.assertEquals(False, "error" in data)

    response = self.app.post('/login',data=json.dumps(dict(UporabniskoIme='ga.p', Geslo='8c17c981d86e1334dd1d04fe4e4
    data = json.loads(response.data)
    self.assertEquals(401, response.status_code)
    self.assertEquals(True, "error" in data)

def stest_Registration(self):
    data = json.dumps(dict(UporabniskoIme='User',Ime='ime',Priimek = 'Priimek', Geslol='8c17c981d86e1334dd7d04fe4e45
    response = self.app.post('/register',data = data, content_type='application/json')
    self.assertEquals(200, response.status_code)
    self.assertEquals(False, "error" in data)

    data = json.dumps(
        dict(UporabniskoIme='User', Ime='ime', Priimek='Priimek', Geslol='8c17c981d86e1334dd7d04fe4e4567d',
            Slika='', Geslo2='8c17c981d86e1334dd7d04fe4e4567d'))
    response = self.app.post('/register', data=data, content_type='application/json')
    self.assertEquals(401, response.status_code)
    self.assertEquals(False, "error" in data)
    self.assertEquals('Uporabnisko ime ze obstaja', json.loads(response.data)['error'])
```

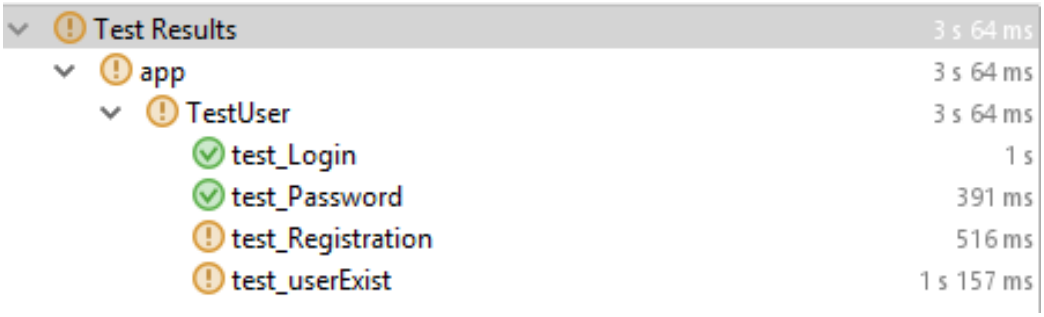
Slika 4.4: Primer testnih funkcij.

4.5 Testiranje programskega vmesnika po namestitvi na Heroku

Heroku [6] je storitvena platforma (ang. platform as a service), ki razvijalcem omogoča gostovanje ter upravljanje aplikacij v oblaku in analizo prometa. Podpira namestitev aplikacij, ki so napisane v različnih programskih jezikih (na primer: python, node.js, java, ipd.) ter gostovanje podatkovnih baz.

Namen

Poleg testiranja enot in integracije je pomembno vmesnik testirati tudi po namestitvi na specifično platformo. Izvesti je potrebno preformačne teste in preveriti delovanje, odzivnost ter stabilnost. Rezultati testiranja morajo ustrezati nefunkcionalnim zahtevam iz specifikacije aplikacije. Vmesnik API mora po namestitvi na platformo delovati enako, kot je bilo predvideno pred namestitvijo. Če rezultati niso zadovoljivi, je potrebno preveriti težave s



Slika 4.5: Rezultat testnih metod.

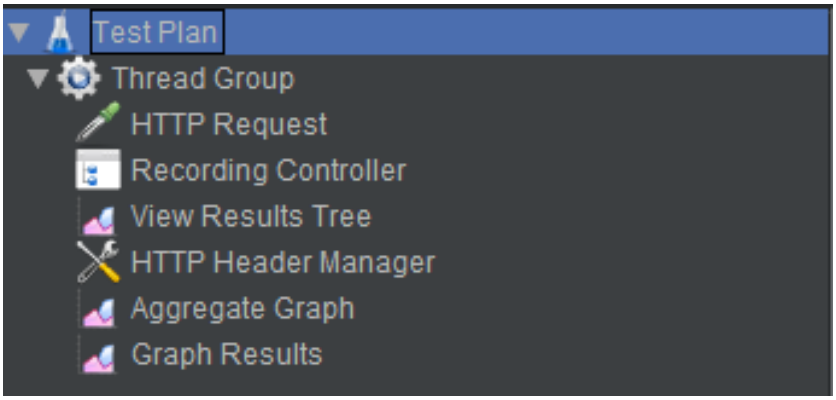
strani ponudnika platforme. Druga možnost je, da se težava nahaja v vmesniku.

Orodja za testiranje

Testiranje je bilo izvedeno s pomočjo orodja *JMeter* [5]. Orodje nam omogoča izvajanje obremenitvenih in preformačnih testov. Orodje je brezplačeno in odprtokodno, napisano v programskem jeziku java ter podprto na različnih operacijskih sistemih. Omogoča testiranje različnih protokolov in aplikacij, na primer: HTTP [9], FTP [7], IMAP [10], SOAP [26] in REST [24] spletnih servisov. Omogoča tudi povezovanje na podatkovno zbirko preko vmesnika JDBC [11] ipd.

Orodje ima na voljo velik nabor različnih komponent, s katerimi izvajamo teste. Vsak testni scenarij mora vsebovati osnovno komponento testni plan (slika 4.6 prikazuje primer testnega plana), znotraj katere nato nastavimo aktivnosti za avtomatizirano testiranje. Orodje deluje kot odjemalec in simulira uporabnike, ki pošiljajo zahtevke na strežnik. Testiranje se izvede po vnaprej določenem testnem scenariju. Kot rezultat nam orodje s pomočjo elementa poslušalec prikaže rezultate. Rezultate nam lahko prikaže v grafični ali tabelarični obliki.

V naprej določene testne plane lahko izvajamo neposredno preko ukazne vrstice (ang. command line).



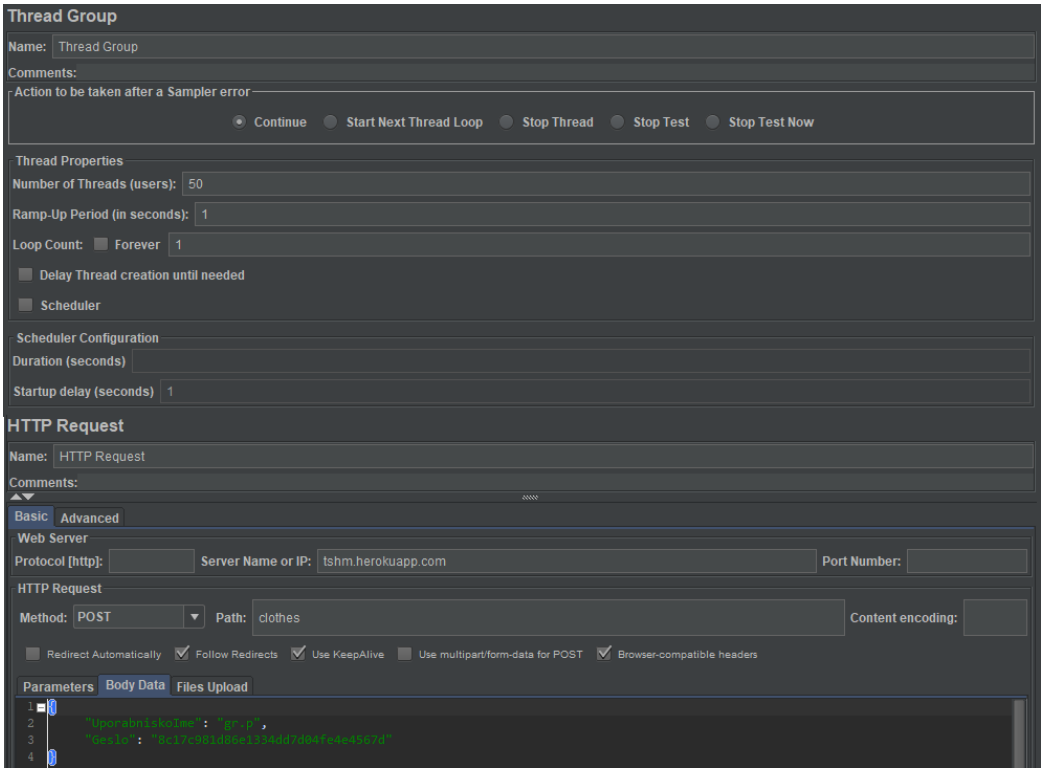
Slika 4.6: Komponente testnega plana.

Potek testiranja

S pomočjo izbranega orodja smo testirali delovanje vmesnika na platformi Heroku. Testiranje smo izvedli s komponento *HTTP request* (slika 4.7). Komponenti smo določili nastavitve HTTP zahtevka: naslov strežnika kjer se nahaja vmesnik, metodo in podatke zahtevka. V komponenti `HTTP Header Manager` smo nastavili pošiljanje podatkov v format JSON.

Sledila je še nastavitev virtualnih uporabnikov (slika 4.7). S to aktivnostjo testnega načrta imamo možnost nastavitve števila virtualnih uporabnikov (ang. Tread Group) za simulacijo. Aktivnost generira določeno število uporabnikov v določenem časovnem obdobju po enakomerni porazdelitvi. Z zanko smo določili, kolikokrat naj se generiranje izvede.

Dodali smo še element *poslušalec* (ang. listener), s katerim prikažemo oziroma analiziramo podatke. Ob pričetku testiranja z orodjem pošljemo zahtevke na izbran naslov, poslušalec pa nato čaka na odgovore. Ob analizi odgovorov nam element izriše graf ali prikaže podatke v tabeli z različnimi informacijami (recimo: število poslanih zahtevkov, količina prenesenega prometa, uspešno prejeti zahtevki ipd.).

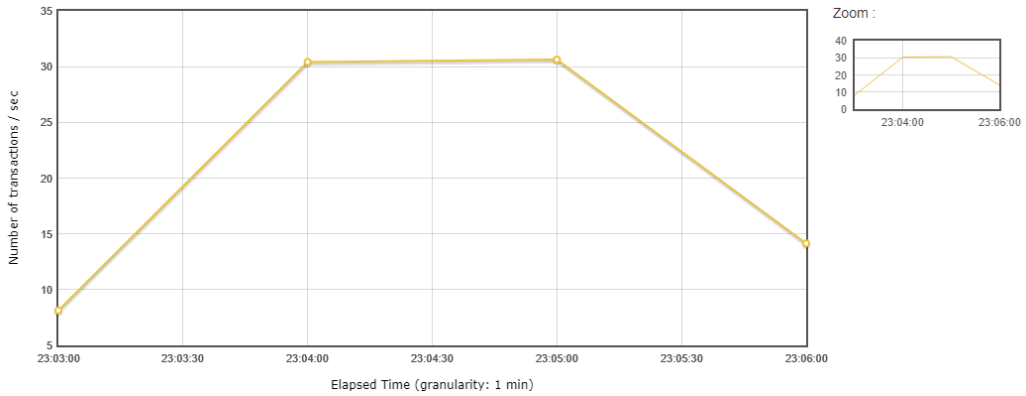


Slika 4.7: Konfiguracija virtualnih uporabnikov in HTTP zahtevka.

Rezultati

Pretočnost je sestavni del nefunkcionalnih zahtev, ki se meri kot skupno število zahtev na sekundo. Na ta način pridobimo podatke o zmogljivosti strežnika, ki je eden od pomembnih kazalnikov pri ocenjevanju uspešnosti uporabe. Ob testiranju uspešnosti aplikacije je potrebno upoštevati tudi nekatere druge kazalnike kot so recimo odzivni čas, zakasnitev itd.

Naš test je vseboval 5000 navideznih uporabnikov. Testirali smo pretočnost pri pridobivanju oblačil iz podatkovne zbirke. Tu je ozko grlo aplikacije, saj se morajo iz podatkovne zbirke v aplikacijo prenašati slike. Pričakovano je, da bo prepustnost tu najmanjša. Izkazalo se je, da lahko vmesnik API obdela 30 zahtevkov (slika 4.8 in slika 4.10) na sekundo, kar zadovolji potrebe aplikacije.



Slika 4.8: Pretočnost vmesnika API na platformi Heroku.

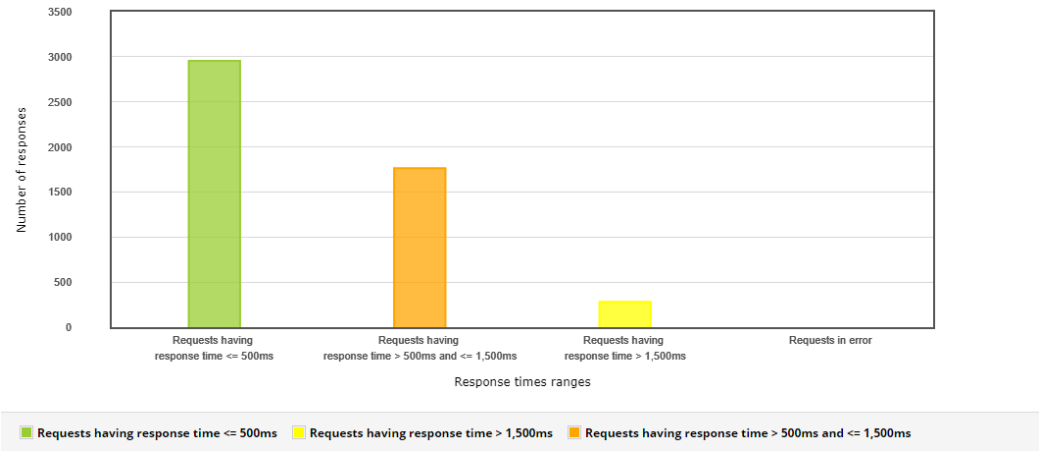
Testiranje je tudi pokazalo, da je 60 % zahtevkov odgovor prejelo v času krajšem od 500 ms. Manj kot 6 % je prejelo odgovor v času, ki je bil daljši kot 1500 ms (slika 4.9). Ker med vsemi zahtevki nismo prejeli napake, sta tako API kot strežnik primerna za potrebe delovanja naše Android aplikacije.

4.6 Testiranje funkcionalnih zahtev in načrta aplikacije

Po končanem načrtovanju aplikacije je sledilo še testiranje *načrta aplikacije s funkcionalnimi zahtevami*. Načrtovalec je pripravil načrt glede na specifikacijo aplikacije, pred samo implementacijo pa je bilo potrebno preveriti, ali so v načrtu napake. Pri testiranju so sodelovali programer, projektni vodja in načrtovalec.

Namen

S tem testiranjem se izognemo nepotrebnim spremembam načrta med razvojem, kar bi vzelo več časa za odpravo napak. Pri tem je pomembno, da so v načrtu definirani koraki povezani z dejanskimi funkcionalnimi zahtevami (kot na primer, kateri zaslon se prikaže ob kliku na gumb *Prijava*).



Slika 4.9: Število odgovorov glede na čas pri testiranu pridobivanja oblačil.

Potek testiranja

Pri testiranju smo preverjali pravilnost po naslednjih korakih:

- *Koraki ob kliku na gumbe:* v načrtu smo preverjali, ali imajo gumbi v navigaciji definirane prehode med različnimi zasloni (slika 3.5). Poleg tega preverjamo tudi prikaz opozorilnih oken ob registraciji, izposoji oblačil in registraciji.

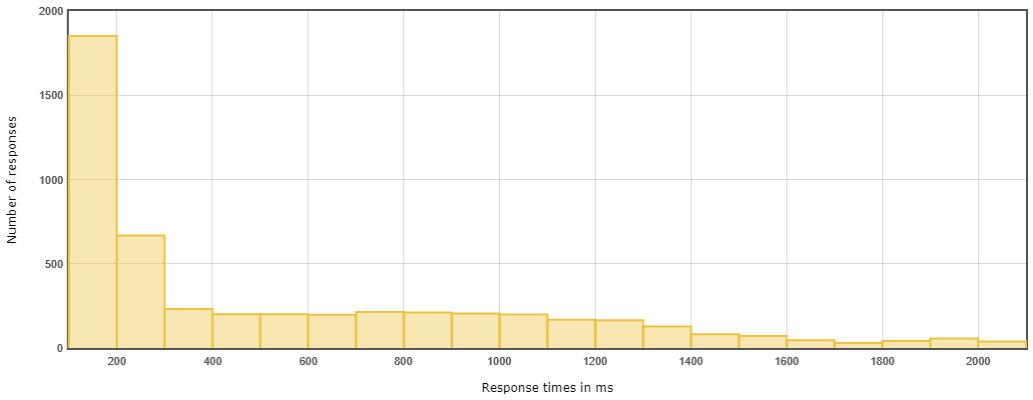
- *Uporabnost in oblika načrta:* pri načrtu smo preverjali tudi enostavnost in preglednost aplikacije. Pomembna je bila velikost gumbov in pisave v aplikaciji, ter prikaz ključnih informacij za uporabnika. Pri posameznih zaslonih smo preverjali tudi to, kako so elementi razporejeni po zaslonu in pa popolnost vseh informacij, ki so zapisane v specifikaciji.

Rezultati testiranja

Rezultati testiranja so sledeči:

1. Gumbi v navigaciji aplikacije niso imeli določenih prehodov na zaslone.

2. Barva aplikacije je bila preveč svetla, barve med zaslonom in elemeti pa so si bile preveč podobne, zato je bila aplikacija težko uporabna.



Slika 4.10: Število vrnjenih zahtevkov glede na čas trajanja.

Prav tako so imeli zasloni različne barve, kar je bilo potrebno popraviti in določiti vsem zaslonom enake kontraste.

3. Pisava na zaslonih je bila premajhna, zato bi bila na manjših napravah neberljiva.

4. Manjkal je načrt za opis aplikacije in pomoč uporabnikom.

4.7 Testiranje načrta aplikacije

Drugi del testiranja *načrta Android aplikacije* se je začel po zaključeni implementaciji. Testiranje je izvedla testna ekipa, ki je bila osredotočena na iskanje napak v načrtu na različnih mobilnih napravah.

Namen

Namen testiranja je bil preverjanje prilagodljivosti aplikacije glede na različne mobilne naprave. Pomembno je bilo testiranje načrta po implementaciji na napravah z različnimi zasloni in različnimi mobilnimi operacijskimi sistemi. Na ta način zagotovimo, da se aplikacija pravilno prilagaja na različne naprave.

Potek testiranja

Testiranje je potekalo še brez izvedbe določenih funkcionalnih zahtev. Med testiranjem smo preverjali sledeče:

- prilagajanje oblike glede na različne velikosti zaslona in na različico mobilnega operacijskega sistema,
- velikost in pravilno postavitev besedila,
- pravilno postavitev vseh elementov na zaslonu ter
- vidnost in prekrivanje elementov.

Testne naprave

Testiranje je obsegalo sledeče naprave, sisteme in specifikacije:

- 10 različnih mobilnih telefonov,
- operacijski sistemi Android verzije: KitKat (4.4)[1], Lollipop (5.0 in 5.1)[2], Marshmallow (6.0)[3] in Nougat (7.0)[4],
- mobilni telefoni znamk: Samsung, LG, HTC, Sony in Huawei,
- velikost zaslona: 4.0", 4.6", 4.7", 5.0" in 5.2".

Rezultati testiranja

Pri testiranju so bile odkrite naslednje napake:

- na napravah z zaslonom velikosti 4.0" je bilo besedilo premajhno in neberljivo. Pisava na gumbih je bila prevelika, zato ni bila vidna v celoti,
- slike na napravah z zaslonom, manjšim od 4.6", niso bile vidne v celoti,
- naprave z operacijskim sistemom Android KitKat niso imele vidne navigacije,
- gumbi na napravah velikosti 5.2" so bili preveliki.

4.8 Testiranje Android aplikacije

Testiranje Android aplikacije se je izvajalo vzporedno s samim razvojem. Testiranje smo izvajali programerji pri razvoju in si pri tem pomagali z orodjem *Espresso Test Recorder* [32].

Namen

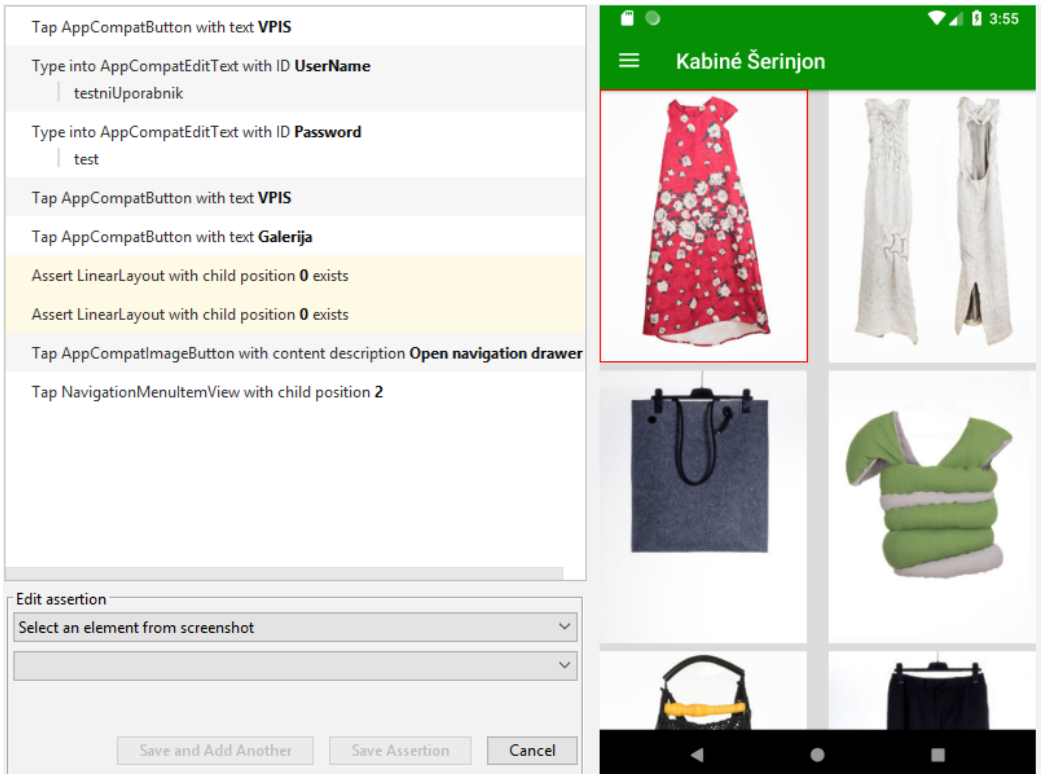
Namen testiranja je preverjanje pravilnosti med razvojem aplikacije. Programer izvaja testne primere med razvojem in preverja, ali je prišlo do napak v aplikaciji. Na ta način lahko zagotavljamo sprotno preverjanje nove in spremenjene kode. S pomočjo ustreznega orodja smo zagotovili, da so obstoječe funkcionalnosti delovale pravilno tudi po imlementaciji novih funkcionalnosti.

Orodja

Espresso Test Recorder je orodje za tvorbo testov uporabniškega vmesnika, brez pisanja kode [32]. S snemanjem testnega scenarija zabeležimo korake (slika 4.11) in jim dodamo trditve (ang. assertions). Espresso Test Recorder nato vzame shranjeni posnetek in samodejno ustvari ustrezen uporabniški vmesnik, ki ga lahko zaženemo in testiramo aplikacijo [32]. Ko shranimo testni scenarij, se nam v ozadju zgenerira koda ustreznega testnega scenarija (primer enega izmed testov na sliki 4.12)

Orodje je del programske opreme Android Studio [18] in omogoča poganjanje testov na različnih napravah. Prav tako pohitri testiranje, saj omogoča preverjanje pravilnosti delovanja brez ročnega klikanja po aplikaciji.

Orodje splošno deluje tako, da v meniju izberemo poženi *Record Espresso Test* in izberemo napravo, na kateri želimo snemati testne scenarije. Izberemo lahko virtualno napravo, ki je del programske opreme Andorid Studio ali pa mobilni telefon povežemo z računalnikom. Če izvajamo snemanje testnih scenarjev preko mobilnega telefona je priporočljivo, da izključimo animacije. S snamanjem naših akcij pri uporabi aplikacije se nato generira testna koda. Na vsakem koraku lahko dodamo tudi preverjanje pravilnosti. Ko končamo



Slika 4.11: Generiranje testnega scenarija.

s snemanjem scenarija, se testni scenari shrani v projekt, nato pa ga lahko poženemo tako, da v meniju izberemo gumb *run selected test*.

Potek testiranja

S testnim orodjem *Espresso Test Recorder* smo posneli testne scenarije (slika 4.12). Testne scenarije smo shranili v projekt, pri tem pa se je generirala testna koda. Ob vsaki spremembi kode smo izvajali generirane teste in preverjali, ali se je pojavila napaka. Če so testi pokazali, da je v kodi napaka, smo morali preveriti spremembe v kodi in poiskativzrok napake. To orodje nam je omogočalo, da med programiranjem novih funkcionalnosti nismo naredili napake v že obstoječi kodi.

4.12).

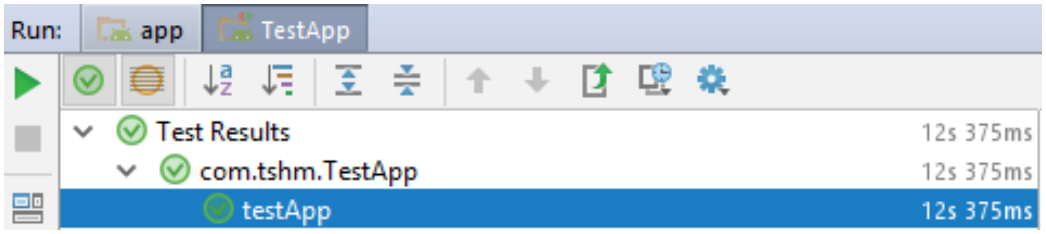
```
@Test
public void testApp() throws InterruptedException {
    ViewInteraction appCompatButton = onView(
            allOf(withId(R.id.EnterButton), withText("VPIS"),
                    childAtPosition(
                            childAtPosition(
                                    withClassName(is( value: "android.widget.LinearLayout")),
                                    position: 1),
                            position: 0),
                    isDisplayed()));
    appCompatButton.perform(click());

    ViewInteraction appCompatEditText = onView(
            allOf(withId(R.id.UserName),
                    childAtPosition(
                            childAtPosition(
                                    withClassName(is( value: "android.widget.LinearLayout")),
                                    position: 0),
                            position: 0),
                    isDisplayed()));
    appCompatEditText.perform(replaceText( stringToBeSet: "testniUporabnik"), closeSoftKeyboard());

    ViewInteraction appCompatEditText9 = onView(
            allOf(withId(R.id.Password),
                    childAtPosition(
                            childAtPosition(
                                    withClassName(is( value: "android.widget.LinearLayout")),
                                    position: 2),
                            position: 0),
                    isDisplayed()));
    appCompatEditText9.perform(replaceText( stringToBeSet: "test"), closeSoftKeyboard());
```

Slika 4.12: Primer zgenerirane testne kode s pomočjo orodja Espresso Test Recorder.

Rezultati

Rezultate testiranja (recimo na sliki 4.13) smo dobili z izvajanjem testnih scenarijev. Povedali so nam, kateri testi so se izvedli uspešno in kateri ne. S tem dobimo takojšnje rezultate o pravilnosti kode, ki so nam v pomoč pri odpravi napak.

Rezultate smo dobili za vse teste vsake funkcionalnosti, predstavljeni pa so v orodju Android Studio. Pri testiranju smo odkrili največ napak



Slika 4.13: Primer rezultata testiranja Android aplikacije.

pri izposoji oblačil. Tam je največ težav povzročal postopek za izposojo oblačil. Težava je bila recimo ta, da uporabnik ni dobil obvestila o prevzetju določenega kosa oblačila. Napake so bile odkrite tudi pri rezervaciji oblačila, saj bi se moralo pokazati opozorilo, ko ima uporabnik že rezervirana ali izposojena tri oblačila.

4.9 Alfa testiranje

Alfa testiranje smo začeli izvajati, ko je bila razvita večina funkcionalnih zahtev. Testiranje je po principu črne škatle izvajala testna skupina, ki je bila sestavljena iz štirih članov.

Namen

Namen alfa testiranja je bilo vzporedno iskanje napak na Android aplikaciji. Testna ekipa je iskala napake na testni različici aplikacije in poročala o odkritih napakah v aplikaciji. S tem smo zagotovili sprotno odpravo najdenih napak, ki je običajno hitrejša in lažja od odprave napak po koncu razvoja.

Orodja

Pri testiranju so bile uporabljene sledeče naprave:

- 4 mobilni telefoni znamk: Samsung, HTC in Sony,
- operacijski sistemi Android verzije: Lollipop [2] (5.0 in 5.1), Marshmallow [3] (6.0),
- velikost zaslona: 4.6" in 5.0".

Potek testiranja

Testiranje je potekalo na specifičnih Android napravah z različnimi operacijskimi sistemi in različno velikostjo zaslona. Vsak izmed testerjev je izvajal

testiranje na eni izmed naprav in preverjal tako funkcionalne kot nefunkcionalne zahteve.

Testiranje je potekalo tako, da so po končanem razvoju specifične funkcionalnosti testerji preverili njeno delovanje. Poleg na novo razvite funkcionalnosti so preverjali še ostale, prej razvite funkcionalne zahteve. Ob zaključenem testiranju so testerji napisali poročilo, ki je vsebovalo sledeče podatke:

1. znamka mobilne naprave, velikost zaslona in verzija operacijskega sistema Androida,
2. številka verzije testne aplikacije,
3. seznam testiranih funkcionalnosti ter
4. seznam odkritih napak in postopek za ponovitev napake.

Rezultati testiranja

Rezultat testiranja so bile napake, ki so se pojavile v aplikaciji. Poročilo o napakah je bilo poslano programerjem, da jih uspešno odpravijo.

Testerji so pri testiranju aplikacije odkrili sledeče napake:

- *Nedelovanje gumba za všečkajnje oblačil:* ob kliku na gumb za všečkanje oblačil tester ni dobil obvestila, da je oblačilo všečkano. Prav tako se oblačilo ni pojavilo na osebni strani v seznamu všečkanih oblačil.
- *Napaka pri spreminjanju osebnih podatkov:* ko je tester želel spremeniti svoje osebne podatke, je dobil obvestilo, da mora izpolniti polja za spremembo podatkov, čeprav so bili podatki že vnešeni.
- *Nedelovanje gumba v navigaciji:* gumb za prehod na zaslon *pomoč* uporabniku ni deloval.
- *Težava pri izbiri osebne slike:* tester je pri registraciji poskusil za sliko izbrati fotografijo iz fotoaparata in pri tem dobil obvestilo o napaki aplikacije, ki se je povrh vsega še zaprla.

- *Po ogledu oblačil se je aplikacija ustavila:* ob pregledovanju različnih oblačil se je aplikcija večkrat ustavila.

4.10 Beta testiranje

Beta testiranje je bila zadnja faza testiranja pred objavo aplikacije. Testiranje je izvedla skupina petnajstih uporabnikov aplikacije, ki so bili predhodno izbrani. V tej fazi je bil testiran izdelek aplikacija kot celota.

Namen

Namen testiranja je bil preverjanje še zadnjih nepravilnosti v aplikaciji. Izbrani uporabniki so bili tudi bodoči uporabniki aplikacije, zato smo želeli pridobiti odziv na aplikacijo z njihove strani. S testiranjem smo želeli pridobiti informacijo, ali uporabniki razumejo, na kašen način deluje izposoja in rezervacija oblačil, ali je aplikacija enostavna in uporabna. Ker smo bili pri procesu testiranja do tega trenutkaomejeni na naš nabor mobilnih naprav, je bil namen tega testiranja tudi preizkus aplikacije na večji množici mobilnih naprav.

Potek testiranja

Uporabnikom aplikacije smo pred začetkom testiranja na kratko opisali namen projekta in aplikacije. Na kratko smo jim predstavili aplikacijo in postopek izposoje oblčil. Testiranje aplikacije je trajalo sedem dni. V tem času so uporabniki na svojih napravah vsakodnevno uporabljali aplikacijo. Pri uporabi aplikacije so preverjali, ali je prišlo do napake, preverjali so uporabnost in kvaliteto aplikacije. Povedali so tudi svojo izkušnjo pri uporabi aplikacije.

Rezultati testiranja

Uporabniki so nam predstavili uporabniško izkušnjo pri uporabi aplikacije. Navedli so nam vse težave pri testiranju ter moteče elemente pri uporabi

aplikacije:

1. Pri vpisovanju podatkov registracije je moteče, da je tipkovnica aktivirana, ne moreš pa se pomikati navzdol po zaslonu. Podobno v oknu *Vpis*.

2. Ko so za sliko izbrali fotografijo,se je aplikacija vrnila na okno za, registracijo, kjer so morali še enkrat pritisniti gumb *Registracija*.

3. Večkrat se lahko registriraš z enakim uporabniškim imenom.

4. Ko so naredili rezervacijo oblačila, niso dobili povratne informacije o tem, ali je bila rezervacija uspešna.

5. Aplikacija se je ustavila po sledečih dogodkih: rezervacija oblačila, rezervacija drugega oblačila (obvestilo, da to ni mogoče), klik na še eno oblačilo in nato se je aplikacija ustavila.

6. Ni vidnega seznama rezerviranih oblačil.

Poglavje 5

Sklepne ugotovitve

V diplomskem delu je opisan proces testiranja na primeru aplikacije za izposojo oblačil. S tem postopkom je bil razvoj aplikacije uspešno zaključen in aplikacija je bila dostavljena naročniku.

Pri procesu testiranja programske opreme so sodelovali testerji, naročnik aplikacije, programerji, testni uporabniki aplikacije in projektni vodja. S pomočjo izbranih orodij *JMeter, Pycharm in Espresso test recorder* smo uspešno poiskali in odpravili napake, še preden je bila aplikacija na voljo uporabnikom.

Proces testiranja je bil sestavljen iz devetih korakov (slika 5.1), s pomočjo katerih smo uspešno izpeljali celoten razvojni proces. Pred začetkom razvoja aplikacije smo si postavljali vprašanja, kako izpeljati testiranje in kako pregledati nepoznano področje glede testiranja v praksi. Pri tem smo poskušali poiskati način, s katerim bi prišli do uspeha na preprost in učinkovit način.

Zadnjo fazo testiranja smo izvedli s pomočjo izbranih končnih uporabnikov. Ti so nam bili v veliko pomoč, saj je bila aplikacija namenjena prav njim in njihovo mnenje je tisto, ki pri uporabi aplikacije šteje največ. Pri procesu testiranja smo prišli do različnih težav (npr. kdaj vključiti testiranje, katera orodja uporabiti ipd.), vendar smo jih s skupnimi sestanki uspešno reševali.

Zelo smo zadovoljni z izpeljanim procesom razvoja, saj je bil odziv uporabnikov aplikacije pozitiven in je aplikacija še vedno v uporabi.

Korak testiranja	Testiran izdelek	Izvajalci	Namen	Rezultati	Orodja
Testiranje zahtev aplikacije	specifikacija aplikacije	- projektni vodja - programer - naročnik aplikacije	preverjanje zahtev aplikacije	odkrite napake v specifikaciji	
Testiranje načrta podatkovne baze	Načrt podatkovne baze	- programer	Preverjanje pravilnosti načrta pred izdelavo PB	Odkrite napake v načrtu PB	
Testiranje programskega vmesnika	Vmesnik API	- programer	Iskanje napak v vmesniku	Napake v posameznih enotah	- JetBrains Pycharm - Knjičnica unittest
Testiranje programskega vmesnika po namestitvi na Heroku	Vmesnik API na platformi Heroku	- programer	Preverjanje delovanja vmesnika naplatformi	- rezultati preformančnih testov v obliki grafa	- JMeter
Testiranje Funkcionalnih zahtev in načrta dizajna	Načrt dizajna in funkcionalne zahteve	- programer - dizajner - projektni vodja	Pregled dizajna v povzavi s funkcionalnimi zahtevami	- napake na načrtu dizajna	
Testiranje dizajna aplikacije	Dizajn aplikacije	- testerji aplikacije	Testiranje dizajna glede na mobilni telefon	- napake na dizajnu aplikacije	
Testiranje Android aplikacije	Android aplikacija	- programer	Iskanje napak v android aplikaciji	- napake odkrite z orodjem za testiranje	- Espresso Test Recorder
Alfa testiranje	Android aplikacija	- testerji aplikacije	Kakovost aplikacije	- povratna informacija o kakovosti in uporabnosti aplikacije	
Beta testiranje	Android aplikacija	- končni uporabnik	Kakovost aplikacije	- povratna informacija o kakovosti in uporabnosti aplikacije	

Slika 5.1: Pregled korakov testiranja.

Nadaljnje delo

- Kot naslednji korak procesa testiranja, bi želeli proces razvoja in testiranja prizkusiti še na sorodnih izdelkih. S tem bi preverili učinkovitost in zanesljivost postopka ter uporabnost izbranih orodij.
- Preveriti bi želeli testiranje s pomočjo plačljivih orodij za testiranje, prav tako pa tudi učinkovitost v primerjavi s trenutno izbranimi orodji.

Literatura

[1] Android kit-kat. Dosegljivo: https://www.android.com/versions/kit-kat-4-4/. [Dostopno 10. 9. 2018].

[2] Android lollipop. Dosegljivo: https://www.android.com/versions/lollipop-5-0/. [Dostopno 10. 9. 2018].

[3] Android marshmallow. Dosegljivo: https://www.android.com/versions/marshmallow-6-0/. [Dostopno 10. 9. 2018].

[4] Android nougat. Dosegljivo: https://www.android.com/versions/nougat-7-0/. [Dostopno 10. 9. 2018].

[5] Apache jmeter. Dosegljivo: https://jmeter.apache.org/usermanual/index.html. [Dostopno 5. 7. 2018].

[6] Dokumentacija platforme heroku. Dosegljivo: https://devcenter.heroku.com/articles/getting-started-with-python. [Dostopno 18. 8. 2018].

[7] Ftp protokol. Dosegljivo:https://tools.ietf.org/html/rfc959. [Dostopno 5. 9. 2018].

[8] Funkcionalno testiranje. Dosegljivo: https://smartbear.com/learn/automated-testing/introduction-to-functional-testing/. [Dostopno 25. 7. 2018].

[9] Http protokol. Dosegljivo:https://www.w3.org/Protocols/rfc2616/rfc2616.html. [Dostopno 5. 9. 2018].

[10] Imap protokol. Dosegljivo:https://tools.ietf.org/html/rfc3501. [Dostopno 5. 9. 2018].

[11] Jdbc protokol. Dosegljivo:http://www.oracle.com/technetwork/java/javase/jdbc/index.html. [Dostopno 5. 9. 2018].

[12] Json. Dosegljivo: https://docs.python.org/3/library/json.html. [Dostopno 18. 8. 2018].

[13] Knjižnica junit5. Dosegljivo: https://junit.org/junit5/. [Dostopno 9. 9. 2018].

[14] Operacijski sistem android. Dosegljivo: https://www.android.com/. [Dostopno 5. 9. 2018].

[15] Operacijski sistem linux. Dosegljivo: https://www.linux.org/. [Dostopno 9. 9. 2018].

[16] Operacijski sistem macos. Dosegljivo: https://www.apple.com/macos/high-sierra/. [Dostopno 9. 9. 2018].

[17] Operacijski sistem windows. Dosegljivo: https://www.microsoft.com/sl-si/windows. [Dostopno 9. 9. 2018].

[18] Programsko orodje android studio. Dosegljivo: https://developer.android.com/studio/. [Dostopno 5. 9. 2018].

[19] Programsko orodje jetbrains. Dosegljivo: https://www.jetbrains.com/. [Dostopno 18. 11. 2018].

[20] Programsko orodje jetbrains pycharm community. Dosegljivo: https://www.jetbrains.com/education/?fromMenu#lang=python&role=learner. [Dostopno 18. 8. 2018].

[21] Python knjižnica flask. Dosegljivo: http://flask.pocoo.org/docs/0.12/testing/. [Dostopno 18. 8. 2018].

[22] Python knjižnica math. Dosegljivo: https://docs.python.org/3/library/math.html. [Dostopno 9. 9. 2018].

[23] Python knjižnica unittest. Dosegljivo: https://confluence.jetbrains.com/display/PYH/Creating+and+running+a+Python+unit+test. [Dostopno 18. 8. 2018].

[24] Rest protokol. Dosegljivo:https://www.restapitutorial.com/. [Dostopno 5. 9. 2018].

[25] Smart phone market share. Dosegljivo:https://www.idc.com/promo/smartphone-market-share/os. [Dostopno 28. 11. 2018].

[26] Soap protokol. Dosegljivo:https://docs.oracle.com/cd/A97335_02/integrate.102/a90297/overview.htm. [Dostopno 5. 9. 2018].

[27] Testiranje integracije. Dosegljivo: https://www.softwaretestinghelp.com/what-is-integration-testing/. [Dostopno 1. 8. 2018].

[28] V model. Dosegljivo: https://www.softwaretestinghelp.com/what-is-stlc-v-model/. [Dostopno 5. 7. 2018].

[29] Zgodovina operacijskega sistema android. Dosegljivo: https://www.android.com/history/. [Dostopno 5. 9. 2018].

[30] Selecting development approach. Dosegljivo: https://www.cms.gov/Research-Statistics-Data-and-Systems/CMS-Information-Technology/XLC/Downloads/SelectingDevelopmentApproach.pdf, 2008. [Dostopno 20. 8. 2018].

[31] Definition of 'Alpha Testing'. Dosegljivo: https://economictimes.indiatimes.com/definition/alpha-testing, 2018. [Dostopno 26. 8. 2018].

[32] Espresso test recorder. Dosegljivo: https://developer.android.com/studio/test/espresso-test-recorder, 2018. [Dostopno 5. 7. 2018].

[33] pgAdmin 4'. Dosegljivo: `https://www.pgadmin.org/`, 2018. [Dostopno 26. 8. 2018].

[34] P. Ammann and J. Offutt. *Introduction to Software Testing*. Cambridge, 2016.

[35] A. Mili and F. Tchiere. *Managing the Testing Process*. Wiley, 2015.

[36] S. L. Pfleeger and J. M. Atlee. *Software engineering theory and practice 4th Edition*. Person, 2009.

[37] R. S. Pressman. *Software Engineering: A practitioner's approach second edition*. McGraw-Hill, 1986.

[38] K. J. Ross. *practical guide to software system testing*. K. J. Ross and Associaties Pty. Ltd, 1998.

[39] M. L. Shooman. *Software Engineering*. McGraw-Hill, 1985.

[40] F. Solina. *Projektno vodenje razvoja programske opreme*. Založba FE in FRI, 1997.

[41] I. Sommerville. *Software Engineering*. Person, 2016.