Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Miha Arh

Avtomatizirano testiranje spletnih aplikacij in storitev

diplomsko delo

visokošolski 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:
Avtomatizirano testiranje spletnih aplikacij in storitev

Tematika naloge:

Združevanje in avtomatizacija različnih vrst testiranja pomembno pripomore k učinkovitosti izvedbe testnih postopkov, predvsem pa omogoča izvedbo tovrstnih aktivnosti tudi razvijalcem, ki niso specializirani za testiranje. V diplomski nalogi zasnujte sistem za avtomatizirano testiranje spletnih aplikacij in storitev. V ta namen najprej izmed večjega števila primernih kandidatov objektivno izberite najprimernejše orodje za izvajanje zvezne integracije, orodji za testiranje spletnih aplikacij in storitev ter aplikacijski strežnik, na katerem bo sistem nameščen. Sistem vzpostavite ter ga preverite v praksi.

Zahvaljujem se viš. pred. dr. Igorju Rožancu za vso pomoč, vodenje in podporo med pisanjem diplomske naloge. Zahvaljujem se tudi mag. Gorazdu Hribarju Rajteriču za idejo in pomoč pri izdelavi diplomske naloge. Posebna zahvala pa celotni družini, prijateljem in partnerici za leta podpore, odrekanja in potrpežljivosti.

Kazalo

Seznam uporabljenih kratic

SOAP	Simple Object Access Protocol	Protokol za izmenjavo podatkov
REST	Representational State Transfer	Arhitektura za izmenjavo podatkov med spletnimi storitvami
WSDL	Web Services Description Language	Opisni jezik spletnih storitev
XML	Extensible Markup Language	Razširljiv označevalni jezik
HTTP	Hypertext Transfer Protocol	Protokol za izmenjavo nadbesedila
HTTPS	Hypertext Transfer Protocol Secure	Protokol za varno izmenjavo nadbesedila
FTP	File Transfer Protocol	Protokol za prenos datotek
LDAP	Lightweight Directory Access Protocol	Protokol za poizvedovanje in spreminjanje imeniških storitev
SMTP	Simple Mail Transfer Protocol	Protokol za prenos elektronske pošte
TCP	Transmission Control Protocol	Protokol za nadzor prenosa
HTMP	Hyper Text Markup Language	Jezik za označevanje nadbesedila
RC	Remote-Control	Daljinsko upravljanje
IDE	Integrated Development Environment	Integrirano razvojno okolje
CI	Continuous Integration	Zvezna integracija
AS	Application Server	Aplikacijski strežnik
WAR	Wab Application Archiver	Datoteka, ki vsebuje spletne storitve
BAT	Batch File	Paketna datoteka

Povzetek

Naslov: Avtomatizirano testiranje spletnih aplikacij in storitev

Avtor: Miha Arh

Cilj diplomske naloge je postavitev splošnega sistema za avtomatizirano testiranje spletnih aplikacij in storitev. Za dosego cilja smo potrebovali glavno orodje, ki bo omogočalo izvajanje zvezne integracije, orodje za testiranje spletnih aplikacij, orodje za testiranje spletnih storitev in spletni strežnik, na katerem bo sistem nameščen. Za vsako od teh orodij smo izbrali več primernih kandidatov. Orodja smo med seboj primerjali po vnaprej določenih kriterijih in izbrali tistega, ki tem kriterijem najbolj ustreza.

Pri analizi orodij za zvezno integracijo smo primerjali orodja Jenkins, TeamCity in Travis CI. Glede na definirane kriterije smo izbrali orodje Jenkins. Za testiranje spletnih aplikacij smo v ožji izbor uvrstili orodja Selenium, Katalon Studio ter TestComplete. Med vsemi je najprimernejši Katalon Studio, ker omogoča enostavno uporabo in zajem testnih scenarijev. Med orodja za testiranje spletnih storitev smo uvrstili SoapUI, Postman in Apache JMeter. Zaradi izbire Jenkinsa smo kot glavni kriterij upoštevali možnost integracije z njim, zato smo izbrali SoapUI, ki to edini omogoča. Potrebovali smo še aplikacijski strežnik, kjer smo izbirali med strežnikoma WildFly in Apache Tomcat. Bolje se je izkazal prvi.

Na koncu smo sistem sestavili v celoto. Orodju Jenkins smo dodali še funkcionalnosti za obveščanje preko e-pošte, preverjanje pravilnosti kode in prikaz rezultatov testiranja.

Celoten sistem smo preizkusili v slovenskem telekomunikacijskem podjetju, kjer se je dobro izkazal. Testiranje spletnih storitev in aplikacij na sistemu razvijalcu prihrani precej časa, poleg tega pa omogoča tudi boljši nadzor ob posodobitvah.

Ključne besede: testiranje, Jenkins, zvezna integracija, spletna aplikacija, spletna storitev.

Abstract

Title: Automated testing of web applications and services

Author: Miha Arh

The aim of this thesis is to set up an universal system for automated testing of web applications and services. To achieve this goal, we needed a master tool for continious integration, a web services testing tool, a web application testing tool, and a web server to install the system on it. Several candidates were selected for each tool. The tools were compared using predefined criteria. Finally, the ones that best suite these criteria were selected.

In case of continious integration tools, we compared Jenkins, TeamCity, and Travis CI. Based on predefined criteria, we have chosen Jenkins. Selenium, Katalon Studio and TestComplete have been shortlisted for testing web applications. Katalon Studio is the most appropriate one, because it is the easies to use. The final web service testing tools included SoapUI, Postman and Apache JMeter. Due to the choice of Jenkins, we considered integration as the main criterion, so we chose SoapUI, which is the only one that allows it. We also needed an application server, so we compared WildFly and Apache Tomcat. The first one turned out to be better.

Finally, the system is implemented and used in practice. We added functionalities for e-mail notification, verification of code and display of test results to Jenkins.

The entire system was tested in a Slovenian telecommunications company and it proved to be successful. Testing of web services and applications, saves the considerable amount of time and allows more control during updates.

Keywords: testing, Jenkins, continious integration, web applications, web services.

Poglavje 1

Uvod

Zaradi povečanega števila uporabnikov, raznolikih sistemov, obsežnejših zahtev in vse ostrejše konkurence prihaja do hujših napak in posledično večjih stroškov razvoja programske opreme [46]. Po definiciji Paula Ammanna je testiranje programske opreme praktična inženirska aktivnost, ki je ključna za izdelavo visoko kakovostnih (programskih) izdelkov [2]. Zato postaja testiranje vse pomembnejši del razvoja programske opreme v večini podjetij. Pri tem gre za testiranje na različnih ravneh, ki se delijo na testiranje enot, modulov, sistemsko testiranje in sprejemno testiranje.

Želja podjetij je hiter razvoj programske opreme ob čim manjši porabi časa za testiranje. Da bi to dosegli, se v naši nalogi posvečamo avtomatizaciji testiranja s pomočjo *zvezne integracije* (angl. Continious Integration - CI) oziroma združitve različnih testnih postopkov v celovito testiranje. Pri tem razvijalci dnevno odlagajo kodo v repozitorij (angl. repository), ta pa se brez posredovanja testerjev v določenih časovnih intervalih na različne načine testira. Rezultat je poročilo o morebitnih napakah.

Do želenega cilja bomo prišli s skrbno izbiro različnih orodij, ki so že uveljavljena pri različnih ravneh testiranja programske opreme.

Za začetek bomo analizirali orodja, ki jih potrebujemo za testiranje spletnih aplikacij. V tem delu je pomembno, da izberemo orodje, ki omogoča snemanje testnega scenarija in nato izvoz v obliki JUnit testov. Testiranje

spletne aplikacije bo izvedeno tako, da razvijalcu ne bo potrebno ročno pisati testnih scenarijev, ampak bo le-te posnel s programom in jih nato izvozil v svoj projekt. Sledi izvajanje testov v okolju Jenkins in poročilo o uspešnosti.

V nadaljevanju bomo preverili še orodja za testiranje spletnih storitev. Tu bomo uporabili protokol SOAP [53] in ogrodje WSDL [74], ki predpisuje obliko komunikacije med odjemalcem in storitvijo. Za primer bomo uporabili javansko spletno storitev, kjer bomo s pomočjo SoapUI orodja izdelali testni scenarij in ga skupaj s projektom naložili v repozitorij. V okolju Jenkins bomo nato ta scenarij v časovnem intervalu izvršili in razvijalcu omogočili vpogled v rezultate testiranja.

Cilj je postaviti sistem, ki bo omogočal testiranje poljubnih spletnih aplikacij in storitev. Pri tem je pomembno, da je sistem razširljiv in omogoča enostavno zamenjavo posameznih delov. Zadnja zahteva je pomembna zaradi novih orodij, ki prihajajo na trg in bi s časom lahko nadomestila trenutna.

Da bi opisana orodja postavili v praksi, bomo v naši nalogi na koncu prikazali še namestitev in uporabo naslednjih operacij:

- avtomatiziran prenos kode iz repozitorija,
- avtomatizirano testiranje,
- prikaz rezultatov in
- obveščanje ob napakah.

Poglavje 2

Opis problema

Naš problem je dejansko avtomatizirati testiranje programske opreme. V okviru naloge se bomo osredotočili na spletne aplikacije in storitve, ki pogosto predstavljata večino razvite programske opreme. Kljub temu pa želimo postaviti sistem, ki ga bo mogoče nadgraditi z orodji za testiranje ostalih vrst programske opreme.

Testiranje spletnih aplikacij od razvijalcev zahteva več časa kot spletne storitve. Težava pri testiranju aplikacij je predvsem možnost napak pri vnosu podatkov v vnosna polja. Razvijalec lahko v vnosno polje vnese napačen podatek in aplikacija posledično vrne nepričakovan rezultat. Da bi to preprečili želimo, da razvijalec posname interakcijo z aplikacijo in to spremeni v testni scenarij. Ko ima narejene testne scenarije lahko aplikacijo ročno testira z zagonom le-teh. Ker je tudi to časovno neučinkovito, bomo v sistemu omogočili avtomatski zagon testnih scenarijev ob vsakem prenosu kode v repozitorij.

Pri testiranju spletnih storitev je možnosti za napake razvijalca manj oziroma jih sploh ni. Večina orodij omogoča, da scenarij sestavimo samo enkrat in ga nato večkrat uporabimo. Da pa zagotovimo še krajši čas za testiranje, želimo testiranje izvajati brez posredovanja razvijalca. Naloga razvijalca bo, da enkrat sestavi testni scenarij in ga nato skupaj s projektom shrani v repozitorij. Sistem bo tako kot pri testiranju spletnih aplikacij samodejno prepoznal novo verzijo kode v repozitoriju in izvedel vse potrebne korake.

Za učinkovito izdelavo sistema smo si zadali funkcionalne in nefunkcionalne zahteve. Funkcionalne zahteve so tiste, ki opisujejo zahtevano obnašanje sistema [45]. Med nefunkcionalne spadajo tiste, ki opisujejo zahteve lastnosti sistema, kot na primer izgled [45].

Funkcionalne zahteve našega sistema so potemtakem:

- testiranje spletnih aplikacij,
- testiranje spletnih storitev,
- samodejno zaznavanje nove kode v repozitoriju,
- izgradnja poročila po koncu testiranja,
- možnost nadgradnje sistema,
- enostavna uporaba orodij in
- najcenejša rešitev.

Med nefunkcionalne zahteve sodita:

- ustrezen izgled poročila po testiranju in
- primerno za manjše projekte.

Poglavje 3

Analiza orodij

Zaradi zahtev je potrebno poiskati ustrezna orodja za izvedbo testiranja in izdelavo poročila o uspešnosti izvedenih testov. Za začetek bomo analizirali in izbrali orodje za zvezno integracijo, ki bo skrbelo za večino opravil. Sledi izbira aplikacijskega strežnika in nato orodja za testiranje spletnih aplikacij ter storitev.

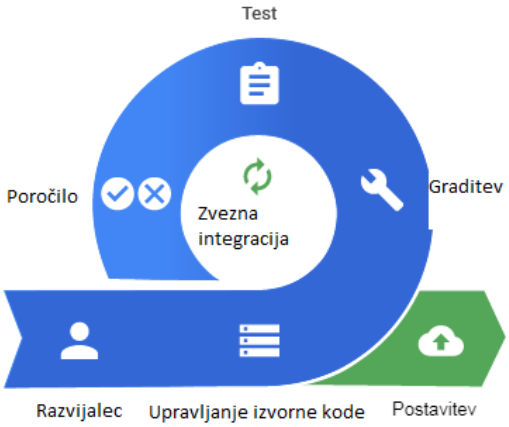
3.1 Orodja za zvezno integracijo

Zvezna integracija je postopek razvoja programske opreme, kjer razvijalci enkrat ali večkrat dnevno prenašajo novo ali dopolnjeno programsko kodo v skupni repozitorij. Po vsakem prenosu se s pomočjo vnaprej definiranih testov preveri pravilnost delovanja rešitve s ciljem čim hitrejšega odkrivanja napak [17]. Slika 3.1 prikazuje tok razvoja programske opreme z uporabo zvezne integracije, iz katere je razvidnih šest osnovnih korakov.

V prvem koraku razvijalec ustvari nov projekt ali prenese obstoječ projekt iz skupnega repozitorija.

Med kodiranjem večkrat preveri pravilnost delovanja lokalno na svojem sistemu in ob pozitivnem rezultatu kodo naloži na skupni repozitorij, ki ga imenujemo tudi sistem za upravljanje izvorne kode (angl. subversion).

Ob vsaki spremembi kode se na sistemu za zvezno integracijo sproži avto-



Slika 3.1: Tok zvezne integracije.

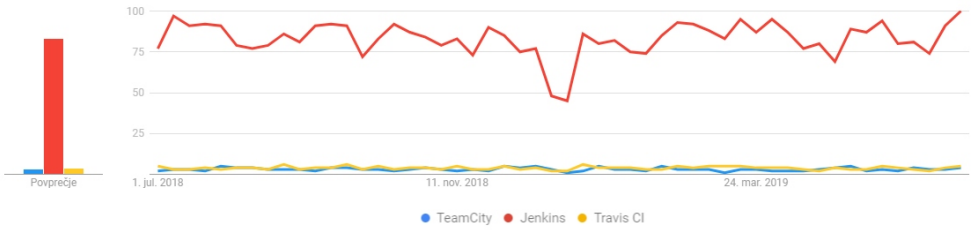
matiziran postopek, ki je definiran v sistemu. Sistem začne s prevajanjem in graditvijo kode, kjer lahko pride do napake pri prevajanju ali pa ne. Napaka v tem delu povzroči takojšnjo ustavitev postopka in zabeleži integracijo kot neuspešno, kar se vidi v končnem poročilu.

Če sta prevedba in graditev kode uspešna, se začne testiranje z vnaprej definiranimi testnimi scenariji. Testi so največkrat na ravni enot, za kar se največkrat uporablja ogrodje JUnit [26]. Če so testi funkcionalni, potrebujemo še testno okolje, na katerem se testira programska koda.

Naslednji korak je priprava poročila o uspešnosti testov, kjer lahko glede na naravo napake sprožimo nadaljnje korake. V primeru napake se pošlje sporočilo razvijalcem, da lahko hitro najdejo in odpravijo napako, kar je ena glavnih lastnosti zvezne integracije.

Če med procesom ni prišlo do napak ali zaustavitev, je zadnji korak postavitev kode v produkcijsko okolje.

Na trgu obstaja veliko orodij za zvezno integracijo, kot na primer Buddy [9], Bamboo [8], GitLab CI [18], Circle CI [12], Codeship [14], primerjali pa bomo tri največje Jenkins [23], TeamCity [60] in Travis CI [66]. Glede na statistiko Google Trends [68] je največje zanimanje za orodje Jenkins, kar prikazuje slika 3.2.



Slika 3.2: Trend iskanja orodij za zvezno integracijo. [68]

3.1.1 Kriteriji

Orodje za zvezno integracijo bo glavni del našega sistema, zato bomo za primerjavo preverili naslednje kriterije:

- uporaba (kakšen je izgled uporabniškega vmesnika in enostavnost uporabe),
- razširljivost (ali je mogoče dodajati oziroma zamenjati orodja),
- skupnost (ali ima orodje veliko odjemalcev),
- cena (kakšna je cena orodja) in
- vtičniki (ali orodje ponuja veliko vtičnikov).

Vsako orodje bomo po zgoraj naštetih kriterijih ocenili s pomočjo Likertove lestvice [30], kjer ocena 1 predstavlja najslabšo, 5 pa najboljšo oceno.

3.1.2 Jenkins

Jenkins je odprtokodno orodje za zvezno integracijo, ki je bilo razvito na Java platformi [23].

Orodje se je prvotno imenovalo Hudson. Razvil ga je Kohsuke Kawaguchi v podjetju Sun. Ko je orodje postalo znano, so se v podjetju leta 2008 odločil za sistematičen razvoj orodja. Leta 2009 je podjetje Oracle kupilo Sun. Ker je med razvojem prihajalo do nesoglasij, so se razvijalci leta 2011 odločili

preimenovati projekt v Jenkins in razvoj se je nadaljeval v dveh ločenih projektih [35].

Jenkins je najbolj razširjeno orodje, ki ga uporabljajo zelo velika podjetja, kot so denimo Microsoft, RedHat, Atlassian in JFrog. Glavna prednost je podpora široki paleti programskih jezikov in tehnologij. Med temi so .Net, Ruby, Groovy, Grails, PHP in Java.

Orodje ponuja veliko različnih možnosti, zato je pomembno, da so razvijalci naredili enostaven spletni uporabniški vmesnik. Žal je ta z današnjega vidika nekoliko zastarel.

V spletnem vmesniku lahko med drugim naložimo tudi vtičnike, ki razširijo ali dodajo nove funkcionalnosti sistemu. Vtičniki so razviti s strani razvijalcev v podjetju Oracle ali s strani posameznikov, ki lahko svoj vtičnik objavijo na spletu [23].

Ocena po kriterijih:

Kriterij	**Ocena**
Uporaba	4/5
Razširljivost	4/5
Skupnost	4/5
Cena	5/5
Vtičniki	3/5
Skupaj	**20/25**

Tabela 3.1: Ocena Jenkins po kriterijih

Orodje Jenkins smo ocenili z oceno 20/25 točk. Gre za brezplačno orodje, ki je enostavno za uporabo, vendar izgubi eno točko pri uporabi zaradi zastarelega uporabniškega vmesnika. Orodje je mogoče razširiti z različnimi orodji, kljub temu pa ne omogoča integracije čisto vseh orodij. Skupnost uporabnikov je velika, kar pripomore k lažjemu učenju in uporabi orodja. Največja pomanjkljivost so vtičniki, ki niso razviti s strani razvijalcev v podjetju in zato vsebujejo slabo dokumentacijo ali pa te sploh ni.

3.1.3 TeamCity

TeamCity je produkt podjetja JetBrains, ki je prvo verzijo izdala leta 2006 [60]. Šele konec leta 2018 so izdali prvo stabilno verzijo. Orodje še ni tako znano, zato je trenutna implementacija in uporaba takšnega orodja zahtevna [60]. Podobno kot Jenkins tudi TeamCity temelji na platformi Java.

TeamCity je na voljo brezplačno z vsemi funkcionalnostmi, vendar pa je omejeno s številom konfiguracij gradnje in agenti za gradnjo. V primeru enega agenta in 10 konfiguracij stane 299 USD, v primeru licence za poslovne uporabnike pa letna članarina znaša že med 1.999 USD in 21.999 USD [61].

Glavne lastnosti orodja so možnost sinhronega testiranja na različnih platformah in okoljih, integracija z IDE orodji Eclipse [15], IntelliJ IDEA [22] in Visual Studio [70], ter podpora platformam .NET, Java in Ruby [60].

Ocena po kriterijih:

Kriterij	**Ocena**
Uporaba	5/5
Razširljivost	4/5
Skupnost	3/5
Cena	2/5
Vtičniki	4/5
Skupaj	**18/25**

Tabela 3.2: Ocena TeamCity po kriterijih

TeamCity ima moderen vmesnik, ki omogoča enostavno dodajanje opravil in v splošnem uporabo sistema. Orodje je razširljivo in ima veliko vtičnikov že implementiranih, vendar nima podpore za obveščanje preko e-pošte. Ker je orodje na trgu med novejšimi, je skupnost manjša od ostalih. Zaradi manjše skupnosti je zelo pomembno, da so med razvojem naredili dobro dokumentacijo vmesnikov. Glavna slabost orodja je cena, ki orodje za manjša podjetja naredi nedostopno. Končna ocena orodja TeamCity je tako 18/25 točk.

3.1.4 Travis CI

Travis CI je orodje, ki je namenjeno testiranju programske kode, ki je objavljena na portalu GitHub. Slednje zelo omeji njegovo uporabnost. Za razliko od orodij Jenkins in TeamCity je Travis CI napisan v jeziku Ruby. Gre za eno najstarejših rešitev na področju zvezne integracije, zato je skupnost uporabnikov že zelo velika [67].

Travis CI omogoča brezplačno testiranje vseh odprtokodnih projektov na GitHubu preko gostitelja `travis-ci.org`. Tudi zasebni projekti se testirajo preko `travis-ci.org`, kjer je potrebno mesečno plačilo med 69 USD in 489 USD [66].

Da omogočimo testiranje projekta, je potrebno projektu dodati konfiguracijsko datoteko `.travis.yml` formata YAML [65]. V datoteki navedemo uporabljeni programski jezik, okolje za testiranje in ostale potrebne parametre. Travis CI podpira več kot 30 programskih jezikov, med njimi so Java, C++, C, Groovy in Python [67].

Ocena po kriterijih:

Kriterij	**Ocena**
Uporaba	4/5
Razširljivost	1/5
Skupnost	4/5
Cena	3/5
Vtičniki	2/5
Skupaj	**14/25**

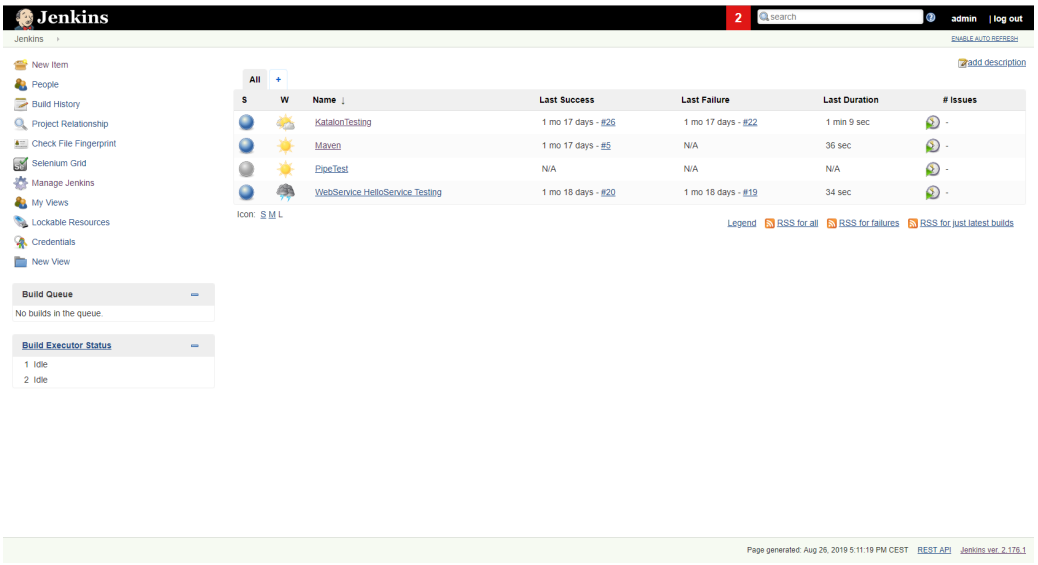
Tabela 3.3: Ocena Travis CI po kriterijih

Travis CI je namenjen predvsem za uporabo pri odprtokodnih projektih, ki so naloženi na GitHub repozitoriju. Orodje je enostavno za uporabo, vendar pa ga zaradi omejitve pri izbiri repozitorija ne moremo uvrstiti med razširljive. Prav tako ni veliko vtičnikov, ki bi razširili uporabo. Ker gre za eno starejših rešitev, je skupnost uporabnikov velika. Cenovno se nahaja med

Jenkinsom in TeamCity orodjem. Končna ocena Travis CI je 14/25 točk.

3.1.5 Izbira orodja za zvezno integracijo

Orodja so si po uporabi in funkcionalnostih zelo podobna. Glavne razlike se pojavljajo pri naboru vtičnikov in ceni. Naš cilj je cenovno ugodno orodje z veliko različnimi vtičniki. Če cena ne bi bila ovira, bi lahko uporabili orodje TeamCity, ki bi predvidoma lahko postalo vodilno orodje na področju zvezne integracije. Ker pa želimo slediti izbranim kriterijem, smo se odločili za uporabo orodja Jenkins. Nadzorno ploščo orodja prikazuje slika 3.3.

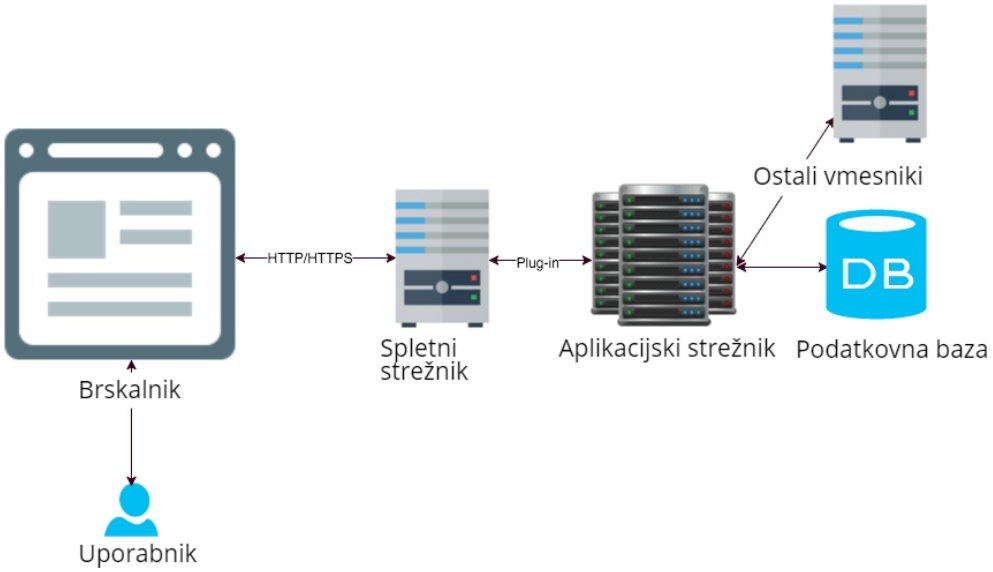


Slika 3.3: Nadzorna plošča orodja Jenkins.

3.2 Aplikacijski strežniki

Aplikacijski strežnik je programsko ogrodje, ki omogoča namestitev poslovnih spletnih aplikacij in nudi izvajalno okolje zanje [34]. Naloga aplikacijskih strežnikov je komunikacija med aplikacijo in podatki, ki je omogočena preko programskih vmesnikov. Ob zahtevi uporabnika tako aplikacija na spletnem

strežniku pridobi podatke iz aplikacijskega strežnika, ki komunicira s podatkovno bazo ali drugimi vmesniki [34]. Ta vidik uporabe prikazuje slika 3.4.



Slika 3.4: Umestitev aplikacijskega strežnika pri zahtevi uporabnika.

Namestitev orodja Jenkins na aplikacijski strežnik žal odvzame orodju nekatere funkcionalnosti, kot na primer možnost samodejnega posodabljanja in ponovnega zagona. Na ta način pa tudi omejimo možnosti napak, kar je potrebno zaradi večjega nadzora nad sistemom [47]. Tudi možnost samodejne postavitve programske opreme v produkcijsko okolje zahteva uporabo aplikacijskega strežnika.

Na trgu obstaja veliko različnih javanskih aplikacijskih strežnikov. Med pomembnejše sodijo: Jetty [25], GlassFish [19], WebLogic [33], Apache Tomcat [5] in WildFly [73]. Zaradi predhodne izbire Jenkinsa smo za primerjavo vzeli dva javanska aplikacijska strežnika, ki zavzemata največji delež trga. To sta Apache Tomcat, z 63,8% in WildFly s 13,8% [48].

3.2.1 Kriteriji

Strežnika sta si po namenu in funkcionalnostih zelo podobna. Primerjali ju bomo po naslednjih kriterijih:

- uporaba (kakšen je izgled uporabniškega vmesnika in enostavnost uporabe),
- enostavnost postavitve (koliko napora je potrebnega za namestitev),
- skupnost (ali ima orodje veliko odjemalcev) in
- cena (kakšna je cena orodja).

Strežnika bomo po zgoraj naštetih kriterijih ocenili s pomočjo Likertove lestvice [30].

3.2.2 WildFly

WildFly, poznan tudi pod imenom JBoss AS, je aplikacijski strežnik razvit s strani podjetja Red Hat [73]. Temelji na programskem jeziku Java, zato podpira izvajanje na različnih Java platformah [73].

Licenčni model je zasnovan na podlagi licence GNU LGPL, kar pomeni, da je orodje odprtokodno, poleg tega pa še brezplačno. Če potrebujemo podporo, je potrebno plačevati naročnino podjetju Red Hat [73].

Glavne lastnosti orodja WildFly so gručenje, porazdeljeno predpomnjenje, porazdeljeno nameščanje in nadomestni način delovanja.

Ocena po kriterijih:

Kriterij	**Ocena**
Uporabe	4/5
Enostavnost postavitve	5/5
Skupnost	5/5
Cena	4/5
Skupaj	**18/20**

Tabela 3.4: Ocena WildFly po kriterijih

Orodje WildFly smo ocenili z oceno 18/20. Orodje ima veliko različnih nastavitev, kljub temu pa je še vedno enostavno za uporabo. Postavitev

strežnika je ob pomoči dokumentacije hitra in enostavna. Postavitev poteka s pomočjo Batch datotek, kar lahko na začetku privede do zmedenosti. Ker je orodje uveljavljeno, ima tudi veliko skupnost uporabnikov. Glavna slabost orodja je v ceni, ker WildFly ne ponuja podpore pri uporabi brezplačne verzije.

3.2.3 Apache Tomcat

Podobno kot WildFly je tudi Apache Tomcat odprtokoden javanski aplikacijski strežnik, zato implementira večino Java EE storitev [5]. Med te sodijo:

- Java strežniški program (angl. Java Servlet),
- Java spletne strani (angl. JavaServer Pages),
- Java poenoten jezik za zapis izrazov (angl. Java Unified Expression Language) in
- spletne vtičnice (angl. WebSocket).

Poleg tega omogoča tudi izvajanje javanske kode na HTTP strežniku [5]. Orodje je razvilo podjetje Apache Software Foundation in deluje pod licenco Apache License 2.0 [5]. To pomeni, da gre za brezplačno orodje.

Ocena po kriterijih:

Kriterij	**Ocena**
Uporabe	2/5
Enostavnost postavitve	5/5
Skupnost	5/5
Cena	5/5
Skupaj	**17/20**

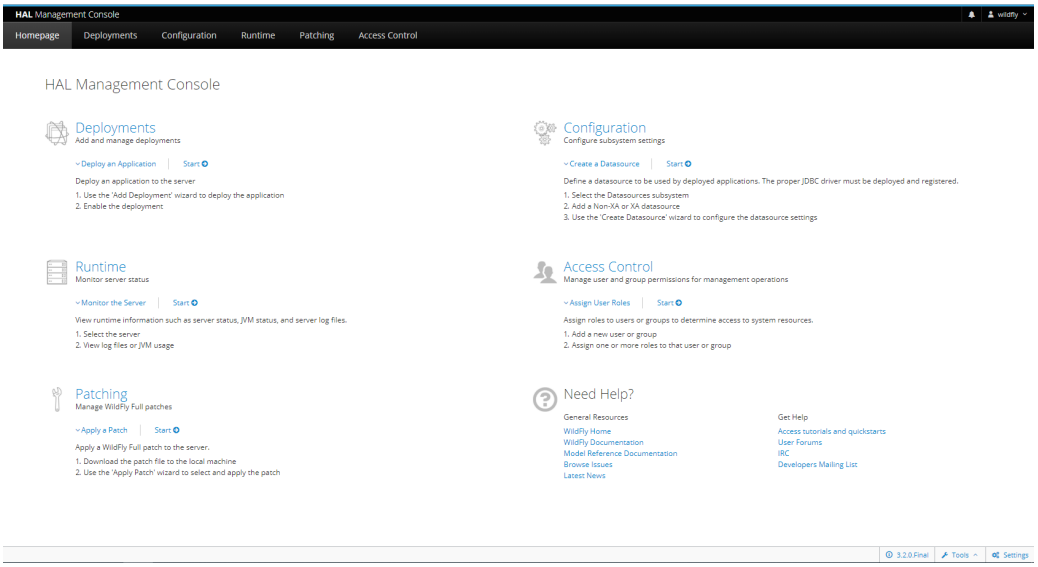
Tabela 3.5: Ocena Apache Tomcat po kriterijih

Apache Tomcat je enostavno postaviti. Uporaba ni najboljša, ker je uporabniški vmesnik zastarel in ne ponuja veliko funkcionalnosti. Skupnost upo-

rabnikov je tako kot pri orodju WildFly velika. Apache Tomcat je brezplačen, prav tako pa je brezplačna tudi podpora. Končna ocena je 17/20.

3.2.4 Izbira aplikacijskega strežnika

Strežnika se zelo malo razlikujeta in to je razvidno iz ocen, ki se med sabo najbolj razlikujejo pri uporabi. Apache Tomcat je starejši strežnik z zastarelim uporabniškim vmesnikom, zato bomo uporabili aplikacijski strežnik WildFly. Nadzorno ploščo orodja prikazuje slika 3.5. Na tega bomo namestili spletno storitev, aplikacijo in sam sistem za zvezno integracijo.



Slika 3.5: Nadzorna plošča orodja WildFly.

3.3 Orodja za testiranje spletnih aplikacij

Spletna aplikacija je program na strani odjemalca, ki omogoča izvedbo nalog na strani strežnika. Odjemalec s pomočjo brskalnika dostopa in upravlja s programom na strani strežnika. Predstavljena je kot skupek strežniških programov (angl. servlet), HTML strani, razredov in ostalih virov, ki so potrebni za izdelavo aplikacije [10].

S povečanim številom uporabnikov svetovnega spleta se povečuje tudi potreba po spletnih aplikacijah, ki vse bolj nadomeščajo namizne aplikacije. Glavna prednost razvoja spletnih aplikacij je ta, da te niso odvisne od operacijskega sistema. Posledično lahko s takšno aplikacijo dosežemo več uporabnikov z manj napora namenjenega razvoju.

Testiranje spletnih aplikacij je lahko časovno zelo potratno, če gre za več vnosnih polj, ki jih mora uporabnik izpolniti. Potreba po avtomatizaciji je tako zelo velika, zato si razvijalci pomagajo z raznimi orodji, kjer posnamejo pričakovano delovanje, ki ga orodje nato samodejno izvaja.

Za testiranje spletnih aplikacij je na voljo veliko orodij. V naši nalogi smo v ožji izbor uvrstili orodja Selenium [49], Katalon Studio [27] in TestComplete [63], obstajajo pa še orodja Ranorex [39], LambdaTest [29], HP UFT [69] in druga.

3.3.1 Kriteriji

Ker bo naš sistem namenjen tudi za testiranje spletnih aplikacij, potrebujemo orodje, kjer je ustrezna:

- uporaba (kakšen je izgled uporabniškega vmesnika in enostavnost uporabe),
- snemanje scenarija (ali omogoča snemanje interakcije uporabnika z aplikacijo),
- podpora brskalnikov (ali podpira različne brskalnike),
- Jenkins (ali omogoča integracijo z orodjem Jenkins),
- JUnit format (ali omogoča enostavno pretvorbo v JUnit format) in
- cena (kakšna je cena orodja).

Zopet bomo vsako orodje ocenili po Likertovi lestvici [30] z oceno od 1 do 5, kjer 1 predstavlja najslabšo, 5 pa najboljšo oceno.

3.3.2 Selenium

Selenium bi lahko opisali kot orodje, ki avtomatizira interakcijo uporabnika z brskalnikom. Glavni namen je testiranje spletnih aplikacij, vendar se lahko uporablja tudi za avtomatizirano interakcijo z aplikacijami. Sestavlja ga več različnih orodij z različnimi pristopi, kar omogoča razvijalcu izbiro med različnimi funkcionalnostmi, ki jih potrebuje [49].

Nabor orodij, ki jih ponuja Selenium, sestavljajo Selenium WebDriver, Selenium IDE in Selenium-Grid [49].

Selenium WebDriver omogoča testiranje z uporabo gonilnikov, kar pomeni, da za zagon testov ne potrebujemo nameščenega brskalnika [52].

Selenium IDE je orodje za pisanje testnih scenarijev. Za lažjo izdelavo testov obstaja vtičnik za brskalnika Firefox in Chrome, ki posname interakcijo uporabnika in zgradi skripto z ukazi. Skripto lahko nato izvozimo v večino programskih jezikov [49].

Selenium-Grid nam omogoča izvajanje testnih scenarijev na različnih napravah in različnih brskalnikih vzporedno. To pomeni, da lahko naenkrat testiramo aplikacijo na brskalniku Firefox in Chrome [50].

Orodje Selenium omogoča testiranje na različnih konfiguracijah, slaba stran orodja pa je, da ne deluje z nekaterimi elementi HTML.

Ocena po kriterijih:

Kriterij	**Ocena**
Uporaba	3/5
Snemanje scenarija	4/5
Podpora brskalnikom	5/5
Jenkins	5/5
JUnit format	3/5
Cena	5/5
Skupaj	**25/30**

Tabela 3.6: Ocena orodja Selenium po kriterijih

Orodje Selenium je brezplačno in vsebuje vse funkcionalnosti, ki jih potrebujemo za testiranje spletne aplikacije. Uporaba orodja ni najboljša predvsem zaradi zastarelega uporabniškega vmesnika. Omogoča izvoz testov v JUnit formatu, kateremu pa je potrebno dodati nekaj vrstic kode, da ta deluje z brskalnikom. Podpira vse večje brskalnike (IE, Chrome, Firefox) in omogoča integracijo z orodjem Jenkins. Končna ocena orodja Selenium je 25/30 točk.

3.3.3 Katalon Studio

Orodje Katalon Studio je prav tako namenjeno testiranju spletnih aplikacij. Gre za brezplačno orodje za avtomatizirano testiranje, ki so ga razvili v podjetju Katalon LLC [27]. Podjetje je Katalon Studio razvilo na osnovi odprtokodnega orodja Selenium [49] in Appium [7] s prilagojenim uporabniškim vmesnikom. V letu 2018 je orodje obsegalo 9% trga na področju avtomatizacije testiranja spletnih aplikacij [59].

Katalon Studio podpira testiranje spletnih in mobilnih aplikacij ter spletnih storitev, kar omogoča uporabnikom širok nabor funkcionalnosti. Deluje na operacijskih sistemih Windows, MacOS in Linux, poleg tega pa obstaja vtičnik za brskalnik Google Chrome, ki omogoča preprosto snemanje interakcije uporabnika z aplikacijo [1].

Delo z Katalon Studio je v primerjavi z drugimi enostavnejše in ne zahteva znanja pisanja testov razen pri zahtevnejših testnih scenarijih. Trenutno sta podprta samo programska jezika Java in Groovy, kar lahko predstavlja težavo pri zahtevnejših uporabnikih [1].

Zaradi pokritja vseh zahtev glede testiranja spletnih aplikacij smo Katalon Studio ocenili z 26/30 točkami. Uporaba je enostavna zaradi modernega vmesnika, ki ima sicer manj funkcionalnosti, vendar pokrije vse naše zahteve. Koda, ki jo izvozimo, ima vse ukaze, ki so potrebni za testiranje.

Ocena po kriterijih:

Kriterij	**Ocena**
Uporaba	4/5
Snemanje scenarija	4/5
Podpora brskalnikom	5/5
Jenkins	5/5
JUnit format	4/5
Cena	5/5
Skupaj	**26/30**

Tabela 3.7: Ocena orodja Katalon Studio po kriterijih

3.3.4 TestComplete

Tudi TestComplete je orodje za avtomatizirano testiranje spletnih aplikacij, ki ga je razvilo podjetje SmartBear. Podpira testiranje spletnih, Android, iOS in Windows aplikacij, poleg tega pa omogoča tudi testiranje zalednih sistemov (na primer podatkovne baze) [63].

Orodje je namenjeno profesionalni uporabi in je zato plačljivo. Obstajata dve plačljivi verziji, ki se razlikujeta po uporabljeni platformi. Cenejša verzija orodja omogoča uporabo samo na določenemu fizičnemu računalniku. Dražja verzija ponuja uporabo tudi na virtualnih računalnikih. Poleg platforme lahko naročnik za dodatno plačilo pridobi tudi dodatke in razširitve [64].

Orodje TestComplete omogoča enostavno uporabo in snemanje testnih scenarijev na različnih brskalnikih. Slabost orodja je ta, da ne omogoča integracije z Jenkinsom, ne omogoča pa tudi izvoza testov v JUnit obliki. Cenovno je to dražja rešitev kot ostali dve orodji, ker ne ponuja brezplačne verzije. Orodje smo ocenili z 18/30 točkami.

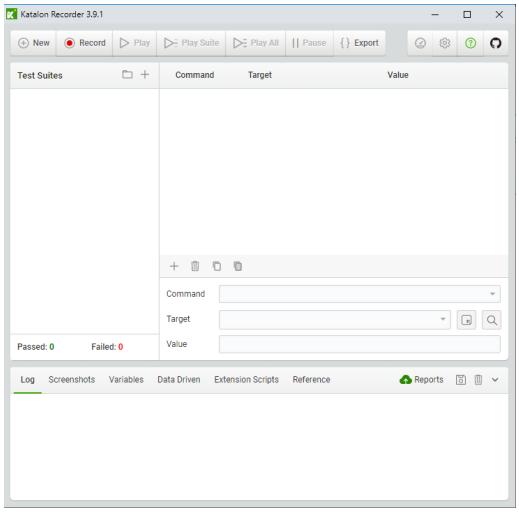
Ocena po kriterijih:

Kriterij	**Ocena**
Uporaba	4/5
Snemanje scenarija	5/5
Podpora brskalnikom	5/5
Jenkins	1/5
JUnit format	1/5
Cena	2/5
Skupaj	**18/30**

Tabela 3.8: Ocena orodja TestComplete po kriterijih

3.3.5 Izbira orodja za testiranje spletnih aplikacij

Med ocenjenimi orodji sta se Selenium in Katalon Studio najbolj približala postavljenim zahtevam. Ker se orodji skoraj ne razlikujeta, smo se zaradi lažje uporabe odločili za orodje Katalon Studio. Uporabniški vmesnik orodja je prikazan na sliki 3.6.



Slika 3.6: Nadzorna plošča orodja Katalon Studio.

3.4 Orodja za testiranje spletnih storitev

Groba definicija spletnih storitev pravi, da spletne storitve delujejo kot medij med aplikacijami na spletu, katerih glavni namen je izmenjava informacij [72]. Ta definicija je preveč splošna, zato je boljša definicija ta, da je spletna storitev vsaka storitev, ki je dostopna preko spleta, uporablja XML obliko sporočanja in ni vezana na operacijski sistem [72].

Različna orodja za testiranje spletnih storitev so API Fortress [6], SOAP Sonar [55], HP QTP [69], Postman [37], Apache JMeter [3] in SoapUI [56]. Za ožjo primerjavo smo izbrali zadnje tri.

3.4.1 Opis protokolov

Protokol je dogovor med udeleženci komunikacije glede oblike in načina komunikacije [28]. V naši nalogi bomo kot primarni protokol za izmenjavo podatkov izbrali protokol SOAP [53], za opisovanje spletne storitve pa jezik WSDL [74]. Za tak izbor smo se odločili, ker je SOAP kljub vzponu RESTa [44], še vedno ena najbolj uporabljenih tehnologij v podjetjih.

Protokol SOAP

Spletni protokol za izmenjavo podatkov (angl. Simple Object Access Protocol - SOAP) je protokol, ki temelji na XML obliki [53]. Običajno komunikacija poteka preko prenosnega protokola HTTP, kjer si dve strani izmenjata SOAP zahtevke in SOAP odgovore kot ju prikazuje slika 3.7.

Sporočilo SOAP je sestavljeno iz naslednjih elementov:

- <`envelope`> (ovojnica) - predstavlja korenski element,
- <`header`> (zaglavje) - podatki, ki so specifični za aplikacijo,
- <`body`> (telo) - podatki SOAP sporočila in
- <`fault`> (napake) del telesa, ki je namenjen opisu napak [38].

Slika 3.7: Primer SOAP zahtevka (zgoraj) in odgovora (spodaj).

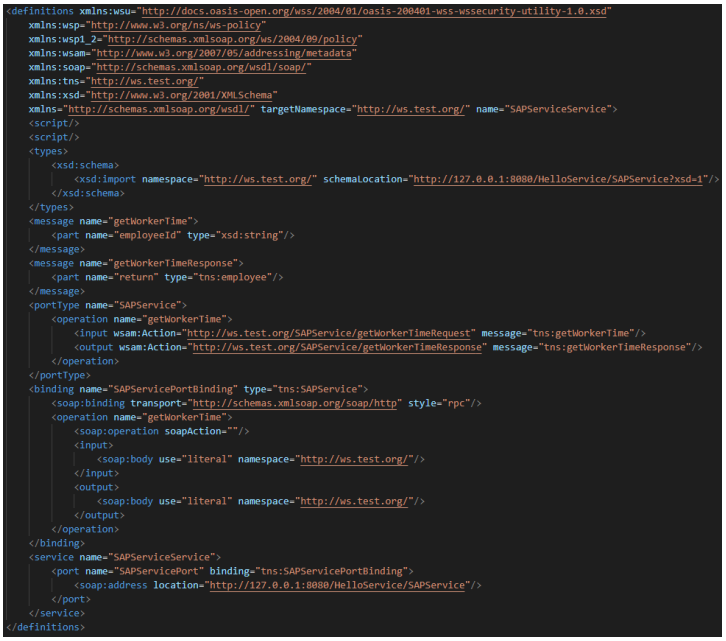
Glavne prednosti uporabe SOAP protokola so razširljivost, neodvisnost in uporaba na različnih OSI nivojih [54].

WSDL

Protokol SOAP vrača rezultate storitve v strukturirani obliki preko omrežja. Za prepoznavo spletne storitve potrebujemo protokol, ki bo storitve prepoznal. Ravno zato se je ob razvoju SOAP protokola razvil tudi jezik za opisovanje vmesnikov WSDL.

Web Services Description Language (WSDL) temelji na obliki XML in je namenjen opisovanju spletnih storitev. Posamezni elementi dokumenta opisujejo dostop do storitev in operacije, ki jih lahko nad njimi izvajamo. Največkrat se uporablja v kombinaciji s protokolom SOAP in shemo XML [74].

WSDL spletna storitev je razdeljena na tri večje elemente, ki jih lahko večkrat uporabimo in med seboj kombiniramo. To so tip (angl. type), operacija (angl. operation) in vezava (angl. binding). Primer dokumenta prikazuje slika 3.8.



Slika 3.8: Primer dokumenta WSDL.

Dokument sestavljajo še naslednji elementi:

- <definitions> - korenski element, ki definira ime storitve, domeno in vsebuje vse ostale elemente WSDL,
- <types> - podatkovni tipi, ki se bodo uporabljali pri izmenjavi sporočil,
- <message> - sestavljen iz enega ali več delov, ki opredeljujejo podatkovne elemente operacije,
- <portType> - opisuje vmesnik storitve in operacij,
- <operation> - opisuje metodo klica, kjer je možnih več vrst izmenjave sporočil,
- <binding> - določa protokol in obliko za operacije in sporočila,
- <port> - kombinacija protokolov in naslovov za komunikacijo,
- <service> - zbirka končnih točk, ki določajo lokacijo spletne storitve,

- <documentation> - namenjen dokumentaciji v razumljivem formatu in
- <import> - namenjen uvozu drugih WSDL dokumentov [38].

3.4.2 Kriteriji

Za lažjo izbiro smo si zastavili naslednje kriterije:

- uporaba (kakšen je izgled uporabniškega vmesnika in enostavnost uporabe),
- organizacija (ali omogoča enostaven pregled projektov),
- Jenkins (ali je možna integracija z Jenkinsom),
- cena (kakšna je cena orodja) in,
- WSDL podpora (ali podpira testiranje WSDL spletnih storitev).

Tudi ta orodja bomo ocenili po Likertovi lestvici [30].

3.4.3 SoapUI

SoapUI je orodje podjetja SmartBear. Namenjeno je funkcijskemu testiranju spletnih storitev tipa SOAP ali REST [56].To je pogosto uporabljeno orodje za testiranje spletnih storitev zaradi enostavnega uporabniškega vmesnika in različnih funkcionalnosti, ki jih ponuja. Podpira tako testiranje okolja, funkcijsko, regresijsko in obremenitveno testiranje. Glavna prednost orodja je, da lahko testiramo storitev še preden je ta nameščena. To pomeni, da lahko simuliramo njeno delovanje.

Orodje omogoča sestavo testnih scenarijev, asinhrono testiranje in pregledno obliko rezultatov. Slabosti orodja sta velika poraba pomnilnika in zahtevna uporaba pri večjih projektih.

SoapUI je orodje, ki ponuja veliko različnih funkcionalnosti. Uporaba nove funkcionalnosti je zato lahko zahtevna za novega uporabnika. Kljub

temu nov uporabnik ne bi smel imeti težav z enostavnimi funkcionalnostmi. Tudi organizacija projektov bi lahko bila bolj pregledna. Orodje ima podporo testiranja WSDL projektov, integracija z Jenkinsom je enostavna. Za zagon testnih scenarijev se uporablja skripta `TestRunner`, ki pa jo lahko dobimo samo v plačljivi verziji. Po naših kriterijih je orodje SoapUI dobilo 19 od 25 možnih točk.

Ocena po kriterijih:

Kriterij	**Ocena**
Uporaba	4/5
Organizacija	3/5
Jenkins	5/5
Cena	2/5
WSDL podpora	5/5
Skupaj	**19/25**

Tabela 3.9: Ocena orodja SoapUI po kriterijih

3.4.4 Postman

Postman je eno od najbolj uporabljenih orodij za testiranje in razvoj spletnih storitev na svetu [37]. Glavno področje testiranja so storitve, ki temeljijo na REST tehnologiji. Za testiranje SOAP projektov je potrebnega nekoliko več znanja. Sprva je bil razvit kot vtičnik za brskalnike Chrome, kasneje pa je bila razvita samostojna namizna izvedba, ki ponuja več funkcionalnosti. Orodje ponuja sistematičen razvoj, dokumentiranje in testiranje REST storitev.

Orodje ima razvito dobro podporo za delo v skupini, kjer lahko več razvijalcev hkrati dela na projektu. Prav tako omogoča enostavno sestavljanje poizvedb.

Postman je enostavno orodje za uporabo, ker je pregleden in omogoča strukturiran pregled projektov. Žal ne omogoča integracije z Jenkinsom. Podpora za testiranje WSDL projektov je mogoča, vendar ne ponuja testira-

nja scenarijev. Orodje je na voljo brezplačno ali plačljivo. V plačljivi verziji dobimo poln dostop do vseh funkcionalnosti. Končna ocena orodja Postman je 19/25 točk.

Ocena po kriterijih:

Kriterij	**Ocena**
Uporaba	5/5
Organizacija	5/5
Jenkins	1/5
Cena	4/5
WSDL podpora	4/5
Skupaj	**19/25**

Tabela 3.10: Ocena orodja Postman po kriterijih

3.4.5 Apache JMeter

Apache JMeter je odprtokodno orodje, za testiranje spletnih storitev [3]. Za vsak testni scenarij je potrebno narediti zanko in skupino niti, kjer zanka predstavlja poizvedbo na strežnik z določenim zamikom, niti pa sočasno obremenitev sistema [32]. Z orodjem JMeter lahko testiramo aplikacije, strežnike in protokole tipa HTTP [21], HTTPS [20], SOAP [54], REST [44], FTP [43], LDAP [40], SMTP [41], TCP [42] ter Java objekte [3].

Orodje omogoča uporabo niti in ponuja veliko različnih dodatkov. Podpira različna področja testiranje, njegova slabost pa je slaba dokumentacija, ki uporabo zelo uteži.

JMeter ni tako pogosto uporabljeno orodje kot SoapUI ali Postman, vendar ima kljub temu najdaljšo prisotnost na trgu. Razvoj se je začel že leta 1998 z verzijo 1.0, danes pa je izdana že verzija 5.1. Videz orodja je za današnje čase zastarel. Orodje posledično ni enostavno za uporabo, vendar pa omogoča solidno strukturiran pregled projektov. Integracija z Jenkinsom ni možna, podpira pa testiranje WSDL projektov. Orodje smo ocenili z 16/25

točkami.

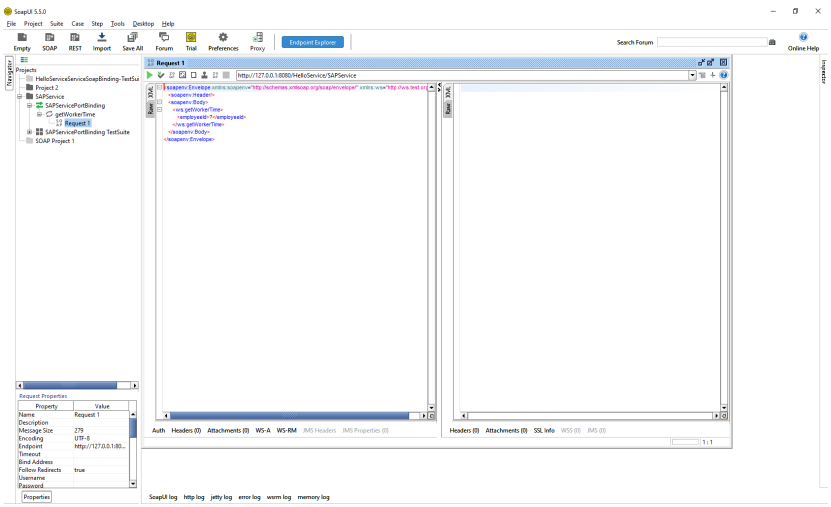
Ocena po kriterijih:

Kriterij	**Ocena**
Uporaba	2/5
Organizacija	3/5
Jenkins	1/5
Cena	5/5
WSDL podpora	5/5
Skupaj	**16/25**

Tabela 3.11: Ocena orodja Apache JMeter po kriterijih

3.4.6 Izbira orodja za testiranje spletnih storitev

Kljub plačljivi verziji in enakemu številu točk, ki jih ima Postman je orodje SoapUI najbolj primerno za našo nalogo, predvsem zaradi enostavne vključitve v Jenkins okolje. Plačljiva verzija vsebuje tudi skripto `TestRunner` za izvajanje testov. Izgled orodja prikazuje slika 3.9.



Slika 3.9: Nadzorna plošča orodja SoapUI.

Poglavje 4

Postavitev sistema

Izbrana orodja želimo sestaviti v celoto ter v praksi namestiti in preveriti celoten sistem. Za začetek bomo postavili aplikacijski strežnik WildFly, na katerega bomo nato naložili orodje Jenkins in testne aplikacije. Po namestitvi Jenkinsa je potrebno namestiti še vtičnike. Sledil bo preizkus sistema, kar bomo dosegli s testiranjem preproste spletne aplikacije in storitve.

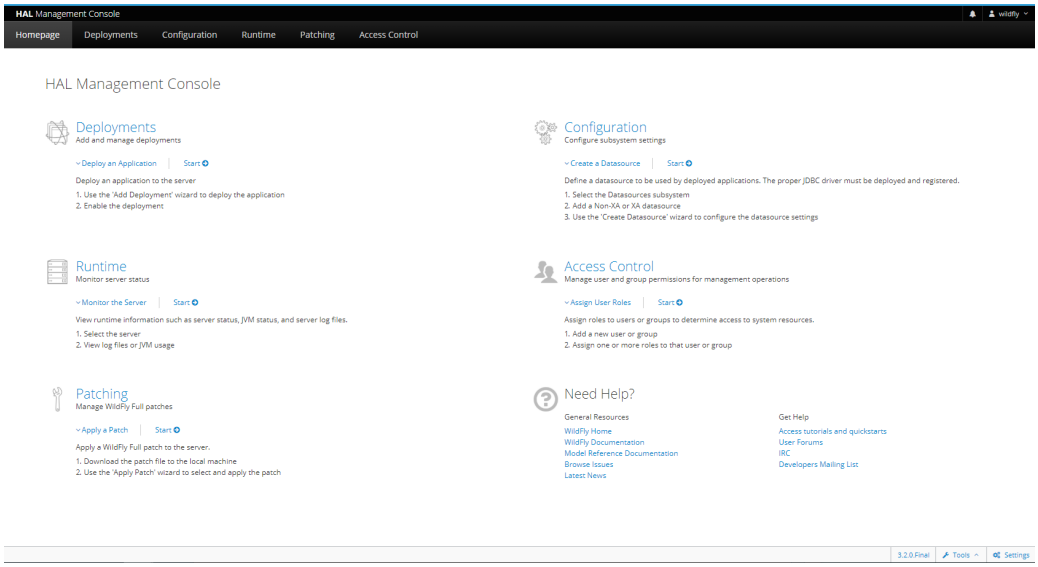
4.1 Postavitev WildFly

Iz uradnega spletnega mesta WildFly smo prenesli stisnjeno obliko aplikacijskega strežnika s polno Java EE podporo. Uporabili smo najnovejšo verzijo opreme, ki je bila izdana 03.07.2019 in sicer `WildFly 17.0.1.Final`. Postavitev strežnika je dokaj enostavna.

Najprej smo razširili preneseno datoteko na poljubno lokacijo. Za delovanje je bilo potrebno dodati uporabnika za dostop do spletne konzole. Novega uporabnika smo dodali s pomočjo zagonske datoteke `add-user.bat`, ki se nahaja v mapi `.\wildfly-17.0.1.Final\bin`. Namestili smo uporabnika `wildfly` in mu dodelili tip uporabnika `Management-User`.

Za zagon strežnika uporabimo datoteko `standalone.bat`, s čimer je strežnik pripravljen za uporabo. Do spletne konzole dostopamo preko spletnega brskalnika in sicer na naslovu `localhost:8080`. Slika 4.1 prikazuje zaslonsko

masko, preko katere smo na strežnik namestili orodje Jenkins, testno spletno aplikacijo ter testno spletno storitev.



Slika 4.1: Zaslonska maska strežnika WildFly 17.0.1.

4.2 Postavitev Jenkins

Za namestitev orodja Jenkins na aplikacijski strežnik WildFly smo iz spletnega mesta prenesli datoteko formata `.war`, ki predstavlja zbirko JAR datotek. Če v ozadju ne bi imeli aplikacijskega strežnika, bi lahko Jenkins namestili tudi neposredno na operacijski sistem Windows, Mac OS X, Ubuntu, CentOS, openSUSE, OpenBSD, FreeBSG ali Docker.

Namestitev Jenkinsa na WildFly je enostavna. V spletnem vmesniku strežnika smo v razdelku `deployments` le naložili datoteko `jenkins.war`, ki smo jo prenesli in okolje je bilo pripravljeno za uporabo.

Ob prvem zagonu smo morali vpisati geslo, ki ga je Jenkins zapisal v datoteko `~\.jenkins\secrets\initalAdminPassword`. V nadaljevanju smo izbrali še potrebne vtičnike, ki jih potrebujemo. Nekatere vtičnike nam Jenkins že samodejno predlaga, vendar jih lahko odstranimo oziroma izberemo

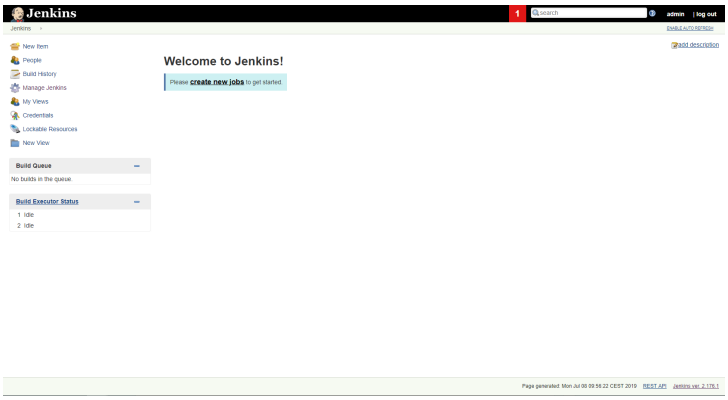
druge.

4.2.1 Opis vtičnikov

Pri iskanju vtičnikov smo veliko časa porabili s prebiranjem dokumentacije in raznih primerov uporabe. Ker so nekateri vtičniki razviti s strani različnih razvijalcev, je bilo pogosto težko najti primerna navodila za uporabo. Za naš primer smo potrebovali vtičnike za:

- preverjanje pravilnosti kode,
- obveščanje preko e-pošte,
- prikaz rezultatov testiranja in
- testiranje SoapUI projektov.

Ko se konča namestitev vtičnikov, je potrebno le-te še nastaviti, da bodo pravilno delovali. Jenkins je dosegljiv na naslovu `localhost:8080/jenkins/`. Njegov spletni vmesnik je prikazan na sliki 4.2.



Slika 4.2: Spletni vmesnik orodja Jenkins

Poleg opisanih smo namestili še dodatne vtičnike, ki pa za delovanje sistema niso ključnega pomena. Teh nismo podrobneje opisovali.

Vtičnik Warnings Next Generation

Vtičnik omogoča zbiranje podatkov ob prevajanju kode ali testiranju. Na podlagi teh podatkov zgradi vizualno poročilo [71]. Prednost vtičnika je ta, da podpira celo vrsto orodij za analizo kode, kot na primer:

- pregled standardov kodiranja (`Checkstyle`) [11],
- iskanje ponavljajoče kode (`CPD`) [71],
- pregled kode za IntelliJ IDEA razvojno okolje (`Idea`) [13],
- pregled Maven opravil (`Maven-warnings`) [71],
- pregled kode (`PMD`) [36] in
- iskanje napak v kodi (`SpotBugs`) [58].

Za delovanje je bilo potrebno v vsak projekt dodati Maven odvisnost kot je prikazano na sliki 4.3. Ta zgradi poročilo v XML obliki za nadaljnjo uporabo v okolju Jenkins.

Vtičnik Email-ext

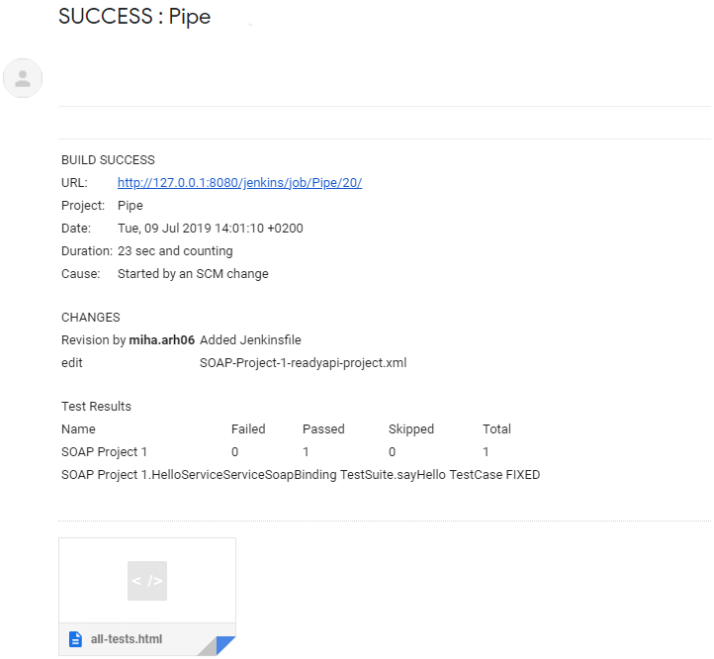
Vtičnik predstavlja razširitev za osnovni vtičnik elektronske pošte [16]. Poleg osnovnih lastnosti pošiljanja poročila po e-pošti, Email-ext omogoča še uporabo sprožilcev glede na pogoje ob izvedbi opravila, prilagojen izgled sporočila in prilagojen izbor prejemnikov sporočila [16].

V našem primeru smo uporabili sprožilec, ki ob neuspešni izvedbi pošlje sporočilo prejemnikom projekta po elektronski pošti. Za izgled sporočila smo uporabili v Groovy-ju napisano predlogo, ki smo jo prilagodili tako, da smo dobili izpis na sliki 4.4.

Za namestitev je bilo potrebno nastaviti parametre SMTP strežnika ter e-naslove razvijalcev na projektu. Težavo smo imeli predvsem z uporabo Gmail SMTP strežnika. Odpravili smo jih tako, da smo dovolili dostop manj varnim aplikacijam (kot je na primer Jenkins).

```
<reporting>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-site-plugin</artifactId>
            <version>3.7.1</version>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-project-info-reports-plugin</artifactId>
            <version>2.7</version>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-surefire-report-plugin</artifactId>
            <version>3.0.0-M3</version>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-pmd-plugin</artifactId>
            <version>3.11.0</version>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-checkstyle-plugin</artifactId>
            <version>3.0.0</version>
        </plugin>
    </plugins>
</reporting>
```

Slika 4.3: Primer `pom.xml` implementacije vtičnikov



Slika 4.4: Izgled obvestila preko elektronske pošte

Vtičnik Test Result Analyzer

Ker Jenkinsov sistem za prikaz rezultatov testiranja ni najbolj pregleden, smo se odločili uporabiti vmesnik, ki poleg rezultatov trenutnega testiranja prikaže tudi rezultate vseh testiranj do sedaj [62]. To nam omogoča večji nadzor nad projekti, hkrati pa ponuja še poročila v obliki grafov.

Glavni prednosti vtičnika sta enostaven vpogled v napake in zgrajeno poročilo. Poročilo v obliki HTML strani omogoča, da ga enostavno pošljemo kot del sporočila o napaki in tako razvijalcu prihranimo nekaj klikov.

Vtičnik deluje na več vrstah projektov, med katerimi so SoapUI, JUnit in Maven.

Vtičnik SoapUI Pro Functional Testing

Za testiranje spletnih storitev smo potrebovali še vtičnik, ki bi v ozadju izvedel teste in nato predstavil rezultate. Prav to omogoča SoapUI Pro vtičnik, vendar smo v tem primeru imeli težave z uporabo brezplačne verzije [57]. Za pravilno delovanje vtičnik potrebuje program imenovan `TestRunner`, ki ga najdemo tako v brezplačni kot v plačljivi verziji programa [57]. Ker brezplačna verzija ni delovala, smo pridobil licenčno verzijo programa SoapUI. Po namestitvi licenčnega `TestRunner` programa je testiranje delovalo tako, kot je potrebno.

S pomočjo Jenkins cevovoda smo nato naredili skripto, ki v skupnem repozitoriju pridobi SoapUI projekt, ga zažene ter izvede vse teste.

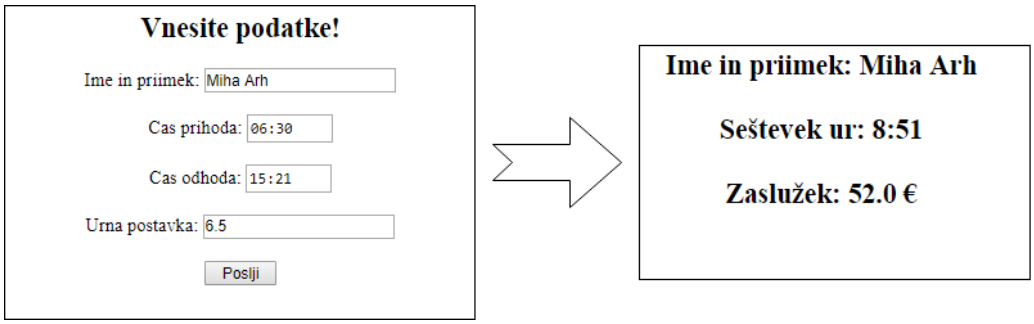
Vtičnik Selenium

Za zajem testnih scenarijev smo uporabili Katalon Studio. Ta temelji na Selenium platformi, zato je bilo potrebno na okolje Jenkins namestil še Selenium vtičnik, ki deluje kot samostojen strežnik. Selenium strežnik za delovanje potrebuje gonilnike za spletne brskalnike. Poleg tega potrebuje še nastavitev vrat in števila instanc spletnih brskalnikov [51]. Ko smo nastavili vse potrebno, smo v kodo za testiranje dodali nekaj vrstic, ki naredijo povezavo na

strežnik, izvedejo testne scenarije in prekinejo povezavo.

4.3 Testiranje spletne aplikacije

Za prikaz testiranja spletne aplikacije, smo razvili preprosto aplikacijo, ki simulira beleženje časa prihoda in odhoda iz delovnega mesta. Uporabnik vpiše ime, priimek, čas prihoda, čas odhoda in urno postavko. Aplikacija nato samodejno izračuna skupni zaslužek in vse podatke shrani v podatkovno bazo. Vpisane podatke bomo uporabili še pri izdelavi spletne storitve. Slika 4.5 prikazuje zaslonski maski aplikacije.



Slika 4.5: Vhodna (levo) in izhodna (desno) zaslonska maska aplikacije

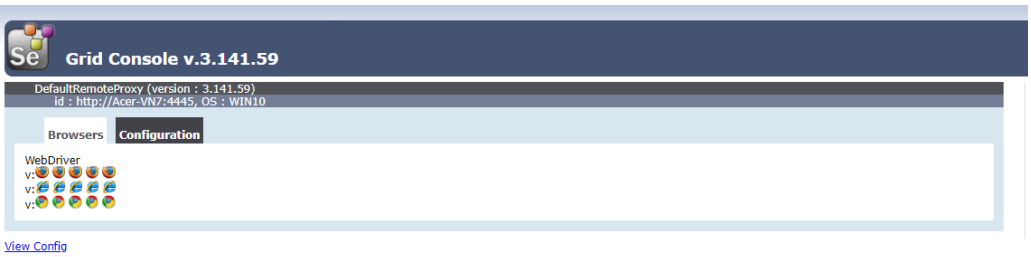
Za izdelavo aplikacije smo uporabili tehnologijo JSP, v kateri smo izdelali vmesnik in logiko. Za gradnjo `war` datoteke smo uporabili orodje Maven.

Na okolju Jenkins je pred izvedbo testiranja potrebno namestiti še spletne gonilnike za brskalnike Google Chrome, Mozilla Firefox in Internet Explorer. Selenium na svoji uradni spletni strani ponuja gonilnike za večje brskalnike, ki si jih lahko brezplačno prenesemo. V Jenkinsu se je ob namestitvi Selenium vtičnika pojavila možnost uporabe Selenium Grid, ki nam omogoča poganjanje testov na različnih brskalnikih. Vse spletne gonilnike, ki jih nameravamo uporabiti, je potrebno vnesti kot novo konfiguracijo. Ko smo dodali novo konfiguracijo, je bilo potrebno vnesti:

- ime konfiguracije,

- vozlišča, ki jih zajema,
- vrata,
- število vozlišč in
- izbrane brskalnike.

Pri dodajanju brskalnikov smo poleg poti do gonilnika morali podati še število instanc. Ko smo nastavili vse potrebno, smo preverili pravilnost nastavitev. Pravilnost smo preverili tako, da smo na naslovu `localhost:4444/grid/console` za vse tri različne brskalnike dobili pet instanci, kot je to prikazano na sliki 4.6.



Slika 4.6: Konzola Selenium Grid programa

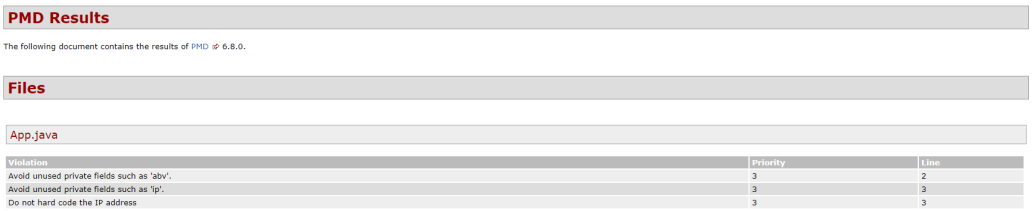
Ko smo nastavili gonilnike, smo s spletnim vtičnikom Katalon Studio naredili testni scenarij za preverjanje delovanja aplikacije. V tem orodju nato izvozimo javanske teste v formatu JUnit in jih dodamo v projekt pod `src/test/java`. Ko smo dodali datoteko v projekt, je bilo potrebno v metodi `setUp()` spremeniti del kode, ki ga uporablja Selenium. Za zagon preko Selenium Grid sistema smo gonilnik nastavili na naslov `127.0.0.1/4444/wd/hub` in dodali argument `--healdes` za zagon brskalnika v ozadju.

Teste smo poganjali z orodjem Maven, kamor smo dodali vmesnike za:

- Site [4],
- Surefire report [31],
- PMD [36] in

- Checkstyle [11].

Namen teh orodij je zagnati teste in pripraviti poročilo o testiranju, v katerem so podatki o uspešnosti testiranja in analizi kode. Maven ukaz `site` zgradi končno poročilo iz vseh orodij, ki jih navedemo v `pom.xml` datoteki. Ko se ukaz izvrši, se v mapi `target/site` izdelajo HTML strani s poročili. Primer poročila PMD prikazuje slika 4.7. Vsa poročila sistem pošlje razvijalcu preko e-pošte.



Slika 4.7: Prikaz opozoril iz analize programske kode

Zaradi raznolikosti projektov smo se odločili izvajati korake testiranja v orodju Jenkins s pomočjo cevovodov. Jenkins omogoča tudi ostale tipe projektov, med katerimi so:

- prosti projekti,
- Maven projekti,
- zunanja opravila,
- projekti z več konfiguracijami,
- mape,
- GitHub repozitoriji in
- projekti z večimi vejami.

Uporaba cevovoda je zelo razširjen način izdelave projektov. Tu lahko celoten postopek izvajanja zapišemo v datoteko imenovano `Jenkinsfile`, ki jo nato shranimo v repozitorij skupaj s projektom. Primer takšne datoteke

prikazuje slika 4.8. Za lažjo izdelavo ima Jenkins interno aplikacijo `Snipper Generator` [24], kjer v preglednejši obliki vpišemo parametre za enega od delov. Aplikacija nam samodejno zgradi skripto.

```
pipeline{
    agent any

    stages{
        stage('Checkout'){
            steps{
                checkout([
                        $class: 'GitSCM',
                        branches: [[name: '*/master']],
                        doGenerateSubmoduleConfigurations: false,
                        extensions: [],
                        submoduleCfg: [],
                        userRemoteConfigs: [[credentialsId: '75e306e8-3a4d-4902-8007-eb95d5a933d7', url: 'https://github.com/MihaArh/MavenWebApp.git']]])
            }
        }
        stage('Maven-Clean'){
            steps {
                withMaven(maven: 'maven1') {
                    bat 'mvn clean '
                }
            }
        }
        stage('Maven-TestaSite'){
            steps {
                withMaven(maven: 'maven1') {
                    bat 'mvn site'
                }
            }
        }
        stage('Checkstyle'){
            steps {
                withMaven(maven: 'maven1') {
                    bat 'mvn site'
                }
            }
        }
    }

    post{
        always{
            emailext([
                    subject: currentBuild.currentResult + " : " + env.JOB_NAME,
                    body: '${SCRIPT, template="groovy-html.template"}',
                    recipientProviders: [culprits(), developers()],
                    attachmentsPattern: '**/target/site/*'
            ])
        }
    }
}
```

Slika 4.8: Primer datoteke `Jenkinsfile`

Za testiranje spletne aplikacije smo najprej iz repozitorija prenesli kodo, ki vsebuje `Jenkinsfile` s koraki izvajanja. Celoten proces je naveden v datoteki `Jenkinsfile` in ga sestavljajo trije koraki:

1. prenos kode iz GIT repozitorija,
2. izvršitev Maven ukazov,
3. sestava poročila in
4. pošiljanje obvestila preko e-pošte.

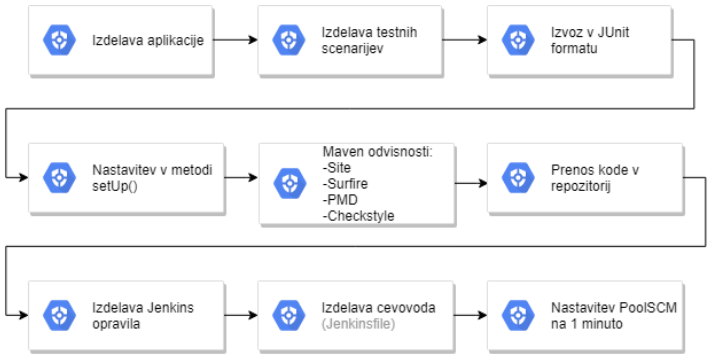
Da bi se projekt samodejno testiral med razvojem, smo v nastavitvah nastavili `Pool SCM` na vsako minuto. To pomeni, da Jenkins vsako minuto preveri, če smo v repozitorij dodali ali spremenili kakšno datoteko. Če ni sprememb, se ne izvede nič. V nasprotnem primeru se izvedejo operacije, ki smo jih izbrali.

V naslednjem koraku se Maven zažene preko ukazne vrstice z ukazom `mvn clean site`, ki nato izvede vsebino datoteke `pom.xml`.

Po zaključku izvajanja `pom.xml` skripte sledi ukaz `mvn site`, kjer se preveri pravilnost kode in izdelava poročil.

Poročilo prikazano na sliki 4.4 vsebuje URL naslova gradnje, imena projekta, datum s časom izvedbe, časa trajanja, sprožilca testiranja, podrobnosti zadnje spremembe v repozitoriju in rezultate testiranja.

Razvijalec ima ob prvem testiranju največ dela, kjer je potrebno vse pravilno nastaviti za delovanje. Slika 4.9 prikazuje tok opravil, ki jih mora razvijalec opraviti.



Slika 4.9: Tok opravil za testiranje spletne aplikacije

4.4 Testiranje spletne storitve

Za prikaz delovanja smo izdelali SOAP spletno storitev, ki ob klicu z argumentom `employeeId` vrne podatke o zaposlenemu, ki je čas prihoda in odhoda vnesel v prej predstavljeno spletno aplikacijo. Slika 4.10 prikazuje javansko kodo z metodo, ki iz podatkovne baze prebere in vrne podatke o zaposlenemu.

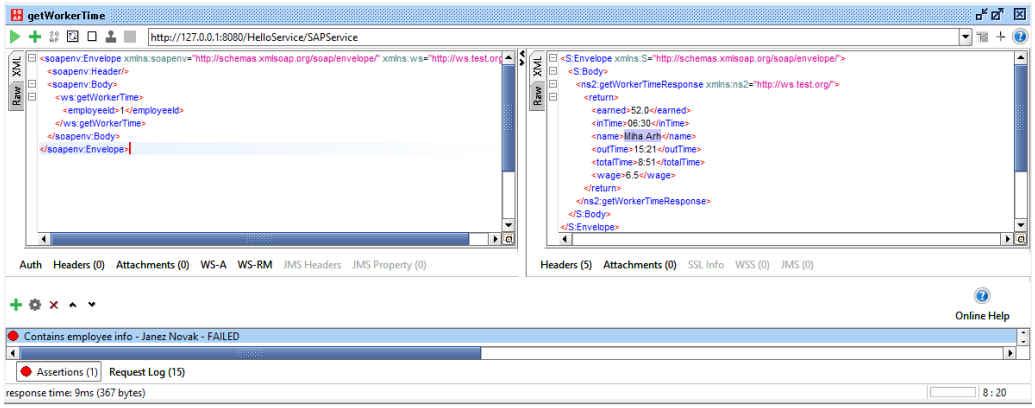
Kodo smo s pomočjo orodja Maven zapakirali v datoteko formata `war`, jo postavili na aplikacijski strežnik WildFly in pridobili povezavo do WSDL datoteke, ki se nahaja na naslovu `localhost:8080/HelloService/SAPService?wsdl`.

```java
@WebService
@SOAPBinding(style = SOAPBinding.Style.RPC)
public class SAPService
{
    @WebMethod(operationName = "getWorkerTime")
    public Employee getWorkerTimeProcess(@WebParam(name="employeeId") String employeeId){
        String query = "SELECT * FROM employee WHERE id=1";

        try {
            Class.forName("com.mysql.jdbc.Driver").newInstance();
        } catch (Exception e) {
            e.printStackTrace();
        }
        String url = "jdbc:mysql://localhost:3306/test?useLegacyDatetimeCode=false&serverTimezone=UTC";
        java.sql.Connection conn = null;
        Employee employee = new Employee();
        try {
            conn =  DriverManager.getConnection(url, user: "arhmi", password: "password");
            Statement stmt =  conn.createStatement();
            ResultSet rs;
            rs = stmt.executeQuery(query);


            while (rs.next()) {
                employee.setName(rs.getString( columnLabel: "name") + " " + rs.getString( columnLabel: "surename"));
                employee.setInTime(rs.getString( columnLabel: "intime"));
                employee.setOutTime(rs.getString( columnLabel: "outtime"));
                employee.setWage(rs.getString( columnLabel: "wage"));
                employee.setTotalTime(rs.getString( columnLabel: "totalTime"));
                employee.setEarned(rs.getString( columnLabel: "earned"));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }




        return employee;
    }
}
```

Slika 4.10: Primer preproste spletne storitve v Javi

Za testiranje storitve smo uporabili orodje SoapUI, kjer smo ustvarili nov projekt in mu podali naslov WSDL datoteke. Ob kreiranju projekta nam orodje samodejno prikaže vse metode, nad katerimi lahko izvajamo operacije. V našem primeru obstaja samo metoda `getWorkerTime`, katere SOAP zahtevek in odgovor prikazuje slika 3.7.

Ko smo naredili projekt, smo se lotili izdelave testnega scenarija. Za testiranje smo morali sestaviti testni paket (angl. test suite), v katerem smo za vsako metodo sestavili SOAP zahtevek in v polje `Assertions` vpisali pričakovan odgovor. Če testni scenarij ni prestal testiranja, se je v polju `Assertions` izpisala napaka kot je to prikazano na sliki 4.11.



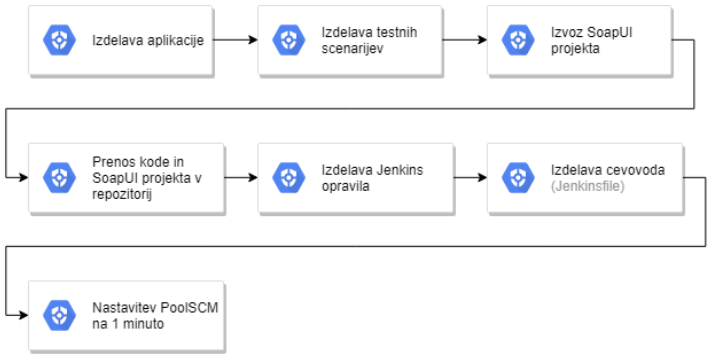
Slika 4.11: Prikaz neuspešnega SOAP testa

Testni scenarij smo nato shranili kot projekt in ga skupaj s kodo prenesli v GIT repozitorij.

`Jenkinsfile`, ki ga uporabljamo pri testiranju spletnih storitev je podoben tistemu, ki smo ga uporabili za testiranje spletnih aplikacij. Razlikujeta se samo v tem, da pri testiranju spletnih storitev uporabimo korak `SoapUI` za testiranje spletne storitve.

Testiranje se izvaja vsakič, ko v repozitorij naložimo novo verzijo spletne storitve. Tako kot pri testiranju spletnih aplikacij tudi tukaj sistem vsako minuto preverja repozitorij za spremembe.

Vse korake razvijalca pri vzpostavitvi sistema za testiranje spletnih storitev prikazuje slika 4.12.



Slika 4.12: Tok opravil za testiranje spletne storitve

4.5 Meritve

Postavljeni sistem smo na koncu še preizkusili in naredili meritve časa. Izmerili smo tri čase. Prvi čas je bil namenjen testiranju brez uporabe posameznih orodij ali sistema, drugega smo izmerili s pomočjo orodja, tretjega pa z uporabo postavljenega sistema.

Meritve smo izvajali na spletni aplikaciji, ki smo jo razvili. Za vsako meritev smo naredili tri ponovitve in nato predvidevali, da se izvajanje ponovi stokrat. Ker se razlike poznajo pri več ponovitvah izvedbe testiranja, smo čase razdelili na manjše dele.

Testiranja brez uporabe orodja ali sistema smo se lotili tako, da smo v vsaki ponovitvi morali ročno odpreti brskalnik, vpisati spletni naslov, izpolniti vnosna polja in pritisniti gumb za potrditev forme. Za takšno testiranje bi porabili 20 sekund.

Testiranje z orodjem je bilo sestavljeno iz dveh delov. V prvem smo izdelali testni scenarij, kjer smo izvedli enake korake kot pri testiranju brez uporabe orodij. V drugem delu pa smo izdelan testni scenarij izvedli. Čas za izdelavo testnih scenarijev je 20 sekund, čas za izvedbo pa 7 sekund. Pomembno je, da pri drugi ponovitvi ni časa za izdelavo, zato je skupen čas testiranja 7 sekund. Pri tem uporabnik ne dobi podrobnega poročila o testiranju, temveč zgolj rezultat o uspešnosti testov.

Testiranje na sistemu je bilo zopet razdeljeno na dva dela. Prvi je enak kot pri testiranju z orodjem, zato prepišemo rezultat 20 sekund. Drugi del pa je sestavljen iz prenosa kode v repozitorij, zagon testov in poročilo o napakah. Izmerjeni čas drugega dela je 12 sekund, kjer se poleg izvedbe testov izdela tudi poročilo, ki je nato poslano na elektronske naslove razvijalcev.

Rezultati meritev so prikazani v tabeli 4.1.

Vrsta\ponovitev	1	2	3		100	Povprečje
Ročno	20 s	20 s	20 s		20 s	20 s
Z orodjem	27 s	7 s	7 s	...	7 s	7,2 s
S sistemom	32 s	12 s	12 s		12 s	12,2 s

Tabela 4.1: Rezultati meritev

Kljub temu, da so najboljši časi pri testiranju z uporabo orodja, je potrebno upoštevati tudi možnosti kasnejšega nadzora programske opreme. Sistem lahko poleg testiranja v realnem času skrbi tudi za obveščanje o napakah pri spremembi zalednih sistemov. Ta poleg tega pošlje še poročilo, ki razvijalcem prikaže dodatne informacije o projektu.

Očitno je, da se pri manjših projektih razlike ne poznajo veliko, pri večjih pa bi bile razlike precej večje.

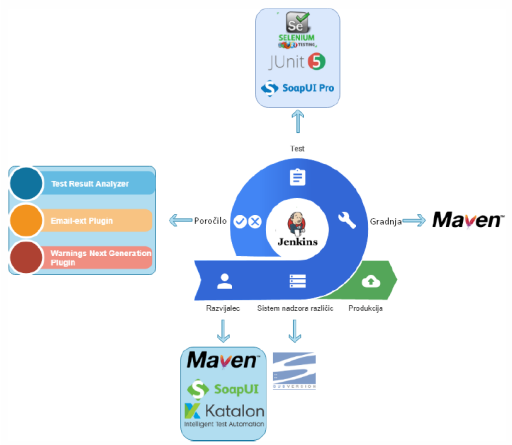
Poglavje 5

Sklepne ugotovitve

V sklopu diplomskega dela smo primerjali orodja za zvezno integracijo, aplikacijska strežnika ter orodja za testiranje spletnih aplikacij in storitev. Izbrana orodja smo sestavili v sistem za avtomatizirano testiranje, kjer razvijalcu ni potrebno ročno izvajati testov.

Glavno orodje sistema je Jenkins, ki je naložen na aplikacijskem strežniku WIldFly in skrbi za vsa opravila znotraj sistema. Za testiranje spletnih aplikacij in storitev smo razvili enostavno spletno aplikacijo in storitev. Aplikacijo za beleženje časa prihoda in odhoda iz dela smo testirali s pomočjo testov v orodju Katalon Studio. Za izvedbo testiranja smo projektu dodali konfiguracijsko datoteko `Jenkinsfile`. Programsko kodo, teste in konfiguracijsko datoteko smo nato shranili v repozitorij. Od tu naprej je vse potekalo avtomatsko. Sistem je pridobil kodo iz repozitorija, jo testiral in poslal poročilo o uspešnosti testov razvijalcem. Pri testiranju spletnih storitev so koraki podobni. V orodju SoapUI smo sestavili scenarij in ga skupaj s projektom shranili v repozitorij. Sistem je nato enako kot pri testiranju aplikacij pridobil kodo, jo testiral in poslal poročilo. Končen izgled sistema prikazuje slika 5.1. Meritve so pokazale, da sistem prihrani nekaj časa pri manjših projektih, pri večjih pa bi bil prihranek veliko večji. Prednost uporabe takega sistema je tudi možnost kasnejšega nadzora razvite programske opreme.

Sistem je že v uporabi v slovenskem telekomunikacijskem podjetju, kjer je



Slika 5.1: Končna slika sistema

občutno olajšal delo razvijalcev. Programske rešitve v podjetju so zahtevne in imajo ogromno zalednih sistemov. Ravno zato je bila potreba po takem sistemu velika. Občasno je potrebno posodobiti orodja, vendar je to vse, kar zadeva vzdrževanje.

Ker je sistem modularen, lahko dodajamo ali zamenjujemo orodja. Tako bi lahko v prihodnosti sistem nadgradili za testiranje namiznih aplikacij, podatkovnih baz, izgleda uporabniških vmesnikov in ostalih vrst programske opreme. Potrebno bi bilo dodati le še orodje, ki omogoča zagon takšnih testov.

Želja uporabnikov v podjetju je tudi avtomatska postavitev (angl. deploy) novo razvite rešitve v produkcijsko okolje. Tudi to bi lahko dosegli z uporabo orodja, ki podpira interakcijo z aplikacijskim strežnikom WildFly.

Da bi zagotovili še enostavnejšo uporabo, bi lahko razvili zunanjo aplikacijo za upravljanje. Razvita aplikacija bi omogočala recimo izdelavo konfiguracijske datoteke `Jenkinsfile` in nastavitev časovnega intervala za izvajanje. Uporabniku bi tako večji del sistema ostal skrit, kar bi pripomoglo k še manjši učni krivulji in možnosti za napake.

Možnost nadaljnjega razvoja je tudi dodajanja vmesne aplikacije za izdelavo opravil in cevovoda. Tako razvijalcu ne bi bilo potrebno poznavanje okolja Jenkins.

Literatura

[1] A comparison of automated testing tools. Dosegljivo: https://www.katalon.com/resources-center/blog/comparison-automated-testing-tools/. (Dostopano: 19.06.2019).

[2] Paul Ammann and Jeff Offutt. *Introduction to software testing*. Cambridge University Press, 2016.

[3] Apache jmeter. Dosegljivo: https://jmeter.apache.org/. [Dostopano: 11. 06. 2019].

[4] Apache maven site plugin – introduction. Dosegljivo: https://maven.apache.org/plugins/maven-site-plugin/. (Dostopano: 07.08.2019).

[5] Apache tomcat. Dosegljivo: https://tomcat.apache.org/index.html. (Dostopano: 05.07.2019).

[6] Api fortress. Dosegljivo: https://apifortress.com/. Dostopano: 23.08.2019.

[7] Appium. Dosegljivo: http://appium.io/. Dostopano: 23.08.2019.

[8] Bamboo continuous integration. Dosegljivo: https://www.atlassian.com/software/bamboo. Dostopano: 21.08.2019.

[9] Buddy. Dosegljivo: https://buddy.works/. Dostopano: 21.08.2019.

[10] Alex Chaffee. What is a web application (or "webapp")? Dosegljivo: http://www.jguru.com/faq/view.jsp?EID=129328. Dostopano: 27.08.2019.

[11] Checkstyle. Dosegljivo: https://checkstyle.org/. (Dostopano: 07.08.2019).

[12] Circleci. Dosegljivo: https://circleci.com/. Dostopano: 21.08.2019.

[13] Code inspections - intellij idea. Dosegljivo: https://www.jetbrains.com/help/idea/code-inspection.html. (Dostopano: 07.08.2019).

[14] Codeship. Dosegljivo: https://codeship.com/. Dostopano: 21.08.2019.

[15] Eclipse. Dosegljivo: https://www.eclipse.org/. Dostopano: 21.08.2019.

[16] Email-ext plugin. Dosegljivo: https://wiki.jenkins.io/display/JENKINS/Email-ext+plugin. (Dostopano: 05.07.2019).

[17] Martin Fowler and Matthew Foemmel. Continuous integration. *Thought-Works*, 2006.

[18] Gitlab. Dosegljivo: https://about.gitlab.com/. Dostopano: 21.08.2019.

[19] Glassfish. Dosegljivo: https://javaee.github.io/glassfish/. Dostopano: 21.08.2019.

[20] Https. Dosegljivo: https://www.w3.org/2001/tag/doc/web-https. Dostopano: 23.08.2019.

[21] Hypertext transfer protocol overview. Dosegljivo: https://www.w3.org/Protocols/. Dostopano: 23.08.2019.

[22] Intellij idea. Dosegljivo: https://www.jetbrains.com/idea/. Dostopano: 21.08.2019.

[23] Jenkins. Dosegljivo: https://jenkins.io/. (Dostopano: 01.07.2019).

[24] Jenkins pipeline. Dosegljivo: https://jenkins.io/doc/book/pipeline/getting-started/. (Dostopano: 07.08.2019).

[25] Jetty. Dosegljivo: https://www.eclipse.org/jetty/. Dostopano: 21.08.2019.

[26] Junit 5. Dosegljivo: https://junit.org/junit5/. Dostopano: 21.08.2019.

[27] Katalon studio. Dosegljivo: https://www.katalon.com/. (Dostopano: 27.08.2019).

[28] James F Kurose. *Computer networking: A top-down approach featuring the internet*. Pearson Education India, 2005.

[29] Lambdatest. Dosegljivo: https://www.lambdatest.com/. Dostopano: 23.08.2019.

[30] Rensis Likert. A technique for the measurement of attitudes. *Archives of psychology*, 1932.

[31] Maven surefire plugin. Dosegljivo: https://maven.apache.org/surefire/maven-surefire-plugin/. (Dostopano: 07.08.2019).

[32] Dmitri Nevedrov. Using jmeter to performance test web services. *Objavljeno na dev2dev*, 2006.

[33] Oracle weblogic server. Dosegljivo: https://www.oracle.com/middleware/technologies/weblogic.html. Dostopano: 21.08.2019.

[34] Joseph Ottinger. What is an app server? Dosegljivo: https://www.theserverside.com/news/1363671/What-is-an-App-Server. (Dostopano: 02.07.2019).

[35] Alan Pankratz. *Forecasting with univariate Box-Jenkins models: Concepts and cases*, volume 224. John Wiley & Sons, 2009.

[36] Pmd. Dosegljivo: https://pmd.github.io/. (Dostopano: 07.08.2019).

[37] Postman. Dosegljivo: https://www.getpostman.com/. [Dostopano: 11. 06. 2019].

[38] Marjaž Rajnar. Primerjava uporabe soap in rest za potrebe povezave mobilnih naprav s spletnimi storitvami. Diplomska naloga, Fakulteta za računalništvo in informatiko, Univerza v Ljubljani, 2019.

[39] Ranorex. Dosegljivo: https://www.ranorex.com/. Dostopano: 23.08.2019.

[40] Rfc 3712 - lightweight directory access protocol (ldap). Dosegljivo: https://tools.ietf.org/html/rfc3712. Dostopano: 23.08.2019.

[41] Rfc 5321 - simple mail transfer protocol. Dosegljivo: ttps://tools.ietf.org/html/rfc5321. Dostopano: 23.08.2019.

[42] Rfc 793 - transmission control protocol. Dosegljivo: https://tools.ietf.org/html/rfc793. Dostopano: 23.08.2019.

[43] Rfc 959: File transfer protocol. Dosegljivo: https://www.w3.org/Protocols/rfc959/. Dostopano: 23.08.2019.

[44] Leonard Richardson and Sam Ruby. *RESTful web services*. "O'Reilly Media, Inc.", 2008.

[45] Igor Rožanc. Tehnologija programske opreme. Študijsko gradivo, Fakulteta za računalništvo in informatiko, Univerza v Ljubljani, 2018.

[46] Igor Rožanc. Uvod v testiranje. Študijsko gradivo, Fakulteta za računalništvo in informatiko, Univerza v Ljubljani, 2018.

[47] Running jenkins on an application server. Dosegljivo: https://jenkins-le-guide-complet.github.io/html/sect-jenkins-app-server.html. (Dostopano: 02.07.2019).

[48] Nikita Salnikov-Tarnovski. Most popular java application servers: 2017 edition. Dosegljivo: https://plumbr.io/blog/java/most-popular-java-application-servers-2017-edition. (Dostopano: 27.08.2019).

[49] Selenium. Dosegljivo: https://www.seleniumhq.org/docs/01_introducing_selenium.jsp. [Dostopano: 13. 06. 2019].

[50] Selenium grid. Dosegljivo: https://www.seleniumhq.org/docs/07_selenium_grid.jsp#chapter07-reference. [Dostopano: 13. 06. 2019].

[51] Selenium plugin. Dosegljivo: https://wiki.jenkins.io/display/JENKINS/Selenium+Plugin. (Dostopano: 05.07.2019).

[52] Selenium webdriver. Dosegljivo: https://www.seleniumhq.org/projects/webdriver/. [Dostopano: 13. 06. 2019].

[53] Simple object access protocol. Dosegljivo: https://tools.ietf.org/html/draft-box-http-soap-00. Dostopano: 23.08.2019.

[54] Aleš Smrdel. Spletne storitve. Študijsko gradivo, Fakulteta za računalništvo in informatiko, Univerza v Ljubljani, 2018.

[55] Soapsonar. Dosegljivo: https://www.crosschecknet.com/products/soapsonar/. Dostopano: 23.08.2019.

[56] Soapui. Dosegljivo: https://www.soapui.org/. [Dostopano: 07. 06. 2019].

[57] Soapui pro functional testing plugin. Dosegljivo: https://wiki.jenkins.io/display/JENKINS/SoapUI+Pro+Functional+Testing+Plugin. (Dostopano: 05.07.2019).

[58] Spotbugs. Dosegljivo: https://spotbugs.github.io/. (Dostopano: 07.08.2019).

[59] State of testing report 2018. Dosegljivo: https://smartbear.com/resources/ebooks/state-of-testing-report-2018/. (Dostopano: 27.08.2019).

[60] Teamcity. Dosegljivo: https://www.jetbrains.com/teamcity/. (Dostopano: 01.07.2019).

[61] Teamcity license. Dosegljivo: https://www.jetbrains.com/teamcity/buy/#license-type=new-license. (Dostopano: 01.07.2019).

[62] Test results analyzer plugin. Dosegljivo: https://wiki.jenkins.io/display/JENKINS/Test+Results+Analyzer+Plugin. (Dostopano: 05.07.2019).

[63] Testcomplete. Dosegljivo: https://smartbear.com/product/testcomplete/overview/. (Dostopano: 27.08.2019).

[64] Testcomplete pricing. Dosegljivo: https://smartbear.com/product/testcomplete/pricing/. (Dostopano: 19.06.2019).

[65] The official yaml web site. Dosegljivo: https://yaml.org/. (Dostopano: 07.08.2019).

[66] Travis ci - plan. Dosegljivo: https://travis-ci.com/plans. (Dostopano: 01.07.2019).

[67] Travis ci user documentation. Dosegljivo: https://docs.travis-ci.com/. (Dostopano: 01.07.2019).

[68] Trends.google.com. Google trends. Dosegljivo: https://trends.google.com/trends/explore?q=TeamCity,Jenkins,Travis%20. (Dostopano: 07.08.2019).

[69] Unified functional testing (uft). Dosegljivo: https://www.microfocus.com/en-us/products/unified-functional-automated-testing/overview. Dostopano: 23.08.2019.

[70] Visual studio ide. Dosegljivo: https://visualstudio.microsoft.com/. Dostopano: 21.08.2019.

[71] Warnings next generation plugin. Dosegljivo: https://wiki.jenkins.io/display/JENKINS/Warnings+Next+Generation+Plugin. (Dostopano: 05.07.2019).

[72] What are web services? Dosegljivo: https://www.tutorialspoint.com/webservices/what_are_web_services.htm. [Dostopano: 05. 06. 2019].

[73] Wildfly. Dosegljivo: https://wildfly.org/. Dostopano: 21.08.2019.

[74] World Wide Web Consortium. *Web Services Description Language (WSDL) Version 2.0*. Dosegljivo: https://www.w3.org/TR/2007/REC-wsdl20-20070626/. [Dostopano: 05. 06. 2019].