Java ekosustav

Lučan, Martina

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Martina Lučan

JAVA EKOSUSTAV

Diplomski rad

Voditelj rada:
prof.dr.sc Robert Manger

Zagreb, srpanj, 2017

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Uvod

Tema ovog diplomskog rada je Java ekosustav. Riječ je o skupu kompatibilnih i komplementarinh alata koji služe softverskim inženjerima pri razvoju aplikacija u programskom jeziku Java. Korištenjem Java ekosustava povećava se produktivnost inženjera, olakšava se njihova suradnja na zajedničkom projektu te se podiže kvaliteta konačnog softverskog produkta. Alati iz Java ekosustava pokrivaju sve važne aktivnosti koje čine proces razvoja i održavanja softvera, dakle kodiranje, testiranje, upravljanje promjenama, upravljanje verzijama, gradnju sustava.

Rad je podijeljen u dva glavna poglavlja. U prvom poglavlju opisat ćemo ukratko svaki alat koji je bitan za pravilan tijek razvoja aplikacije, dok ćemo u drugom poglavlju prikazati jedan tijek razvoja aplikacije koji se sastoji od postavljanja razvojne okoline, upravljanja promjenama, pisanja testova te samog kodiranja.

Poglavlje 1

Općenito

1.1 Git Version Control

Git je alat koji je razvio *Linus Torvalds*, 2005.godine, kako bi mu olakšalo razvoj velikog projekta - Linux kernel-a. Linus je Git zamislio kao osnovu za druge sustave za razvijanje koda. Prema Linusovoj ideji drugi alati su trebali razvijati svoja sučelja na osnovu git-a.
No kako je git bio otvorenog koda, počeo se koristiti takav kakav jest i kasnije se razvijao s potrebama korisnika.
Nastale su mnoge platforme za spremanje projekata kao što je Github, a već postojeći su morali dodati git na zahtjev korisnika.

Neke od prednosti gita:

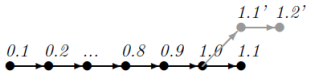
- brzina
- vrlo je lako granati te preuzimati tuđe izmjene

Git se može preuzeti na `https://git-scm.com/downloads`.

Zašto Git?

Git je alat koji olakšava izradu različitih verzija aplikacije. Recimo da smo u situaciji da radimo na nekoj web aplikaciji. Naravno, svaka aplikacija ima svoje korisnike, a svaki korisnik ima neke zahtjeve. Mi kao razvojni programeri nikada nećemo moći u potpunosti predvidjeti sve zahtjeve koje bi korisnik mogao imati, ali to nas ne može spriječiti da započnemo razvoj aplikacije. Mi napravimo početnu verziju aplikacije (*v1.0*) te je takvu damo korisnicima na testiranje, korištenje i sl. Koristeći danu verziju aplikacije korisnik će uvidjeti moguće nedostatke aplikacije te na temelju toga nam dati povratnu informaciju

što bismo trebali popraviti i kako bi to trebalo izgledati. Naša dužnost je da prihvatimo korisnikove kritike te prema njegovim željama napraviti izmjene i nakon toga mu dostaviti novu verziju aplikacije (*v1.1*). No što ako za par dana korisnik shvati da ta novija verzija (*v1.1*) nije ipak ono što bi njemu odgovaralo i dobije bolju ideju kako izmijeniti prvotnu verziju (*v1.0*)? Trebalo bi se nekako vratiti na verziju *v1.0* te iz nje započeti raditi na novim izmjenama i napraviti verziju *v1.1'*.



Slika 1.1: Verzioniranje

Također, Git nam omogućava da na jednom projektu sudjeluju više od jednog razvojnog programera. U tim situacijama postoji jedan centralni git repozitorij (npr. na GitLabu) u kojem se nalaze sve najnovije izmjene unutar projekta. Svaki od programera na tom projektu u mogućnosti je u svakom trenutku povući te izmjene i sinkronizirati ih sa svojima te svoje izmjene staviti na repozitorij kako bi i one bile dostupne drugim programerima.

Spremnici, naredbe, grananje, .gitignore

Spremnici

Imamo tri različita mjesta u kojima se čuvaju datoteke, tj. njihova stanja:

- **git repozitorij** - čuva različita stanja iste datoteke
- **radna verzija repozitorija** - stanje datoteke u našem direktoriju
- **index** - poseban međuprostor za *commit* gdje se privremeno sprema trenutno stanje datoteke prije nego li je spremimo u git repozitorij

Naredbe

Git naredbe općenito imaju sljedeću strukturu:
git naredba opcija1 opcija2 ...
Iznimno imamo naredbu *gitk* koja pokreće grafički program pomoću kojeg možemo gledati povijest projekta. Ako želimo saznati detalje o nekoj naredbi koristimo ***git help naredba***.

Sada ćemo opisati neke od osnovnih git naredbi koje će nama kasnije trebati:

- ***git status*** *- daje nam informacije o tome imamo li na projektu izmjena koje bi trebalo spremiti.*

```
mlucan@DESKTOP-OOGB3MS MINGW64 ~/Desktop/maintaining-working-hours (develop)
$ git status
On branch develop
Your branch is ahead of 'origin/develop' by 1 commit.
  (use "git push" to publish your local commits)
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   src/main/java/hr/altima/config/WebAppInitalizer.java
        modified:   src/main/java/hr/altima/config/WebConfig.java
        modified:   src/main/java/hr/altima/controllers/EmployeeController.java
        modified:   src/main/resources/config.properties

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        src/main/java/hr/altima/config/RootConfig.java
        src/main/java/hr/altima/config/WenAppInitializer.java

no changes added to commit (use "git add" and/or "git commit -a")
```

Slika 1.2: Ispis datoteka koje su modificirane a nisu spremljene

- ***git clone*** *- kopiranje projekta sa udaljenog repozitorija na lokalno računalo*
- ***git add*** *- spremanje datoteka u index, priprema za commit.*

```
mlucan@DESKTOP-OOGB3MS MINGW64 ~/Desktop/maintaining-working-hours (develop)
$ git add  src/main/java/hr/altima/controllers/EmployeeController.java

mlucan@DESKTOP-OOGB3MS MINGW64 ~/Desktop/maintaining-working-hours (develop)
$ git status
On branch develop
Your branch is ahead of 'origin/develop' by 1 commit.
  (use "git push" to publish your local commits)
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   src/main/java/hr/altima/controllers/EmployeeController.java

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   src/main/java/hr/altima/config/WebAppInitalizer.java
        modified:   src/main/java/hr/altima/config/WebConfig.java
        modified:   src/main/resources/config.properties
```

Slika 1.3: Spremanje datoteke *pom.xml* u index

- ***git commit -m*** - *spremanje svih onih izmjena koje se nalaze u indexu u lokalni git repozitorij*

```
mlucan@DESKTOP-OOGB3MS MINGW64 ~/Desktop/maintaining-working-hours (develop)
$ git commit -m "Fixing typo"
[develop 1119eed] Fixing typo
 1 file changed, 4 insertions(+), 2 deletions(-)
```

Slika 1.4: Spremanje izmjena koje su u indexu u repozitorij

- ***git log*** - *ispis svih commitova.*

```
commit b5c8d653f2a9b8231d8b066c81ee340c5f336ab3
Author: mlucan <martina.lucan@gmail.com>
Date:   Thu Jun 15 15:24:02 2017 +0200

    Changing profile id in pom

commit 181e38f295506d5cd85702f56af5f3a266f3c786
Author: mlucan <martina.lucan@gmail.com>
Date:   Fri Jun 9 13:15:02 2017 +0200

    Fixing typo

commit 5c31b12ab36b83359056786bf22c5c798c8336c5
Author: mlucan <martina.lucan@gmail.com>
Date:   Fri Jun 9 13:11:15 2017 +0200

    Adding maven ant task
```

Slika 1.5: Ispis zadnjih nekoliko commitova

- ***git checkout*** - *naredba koja nam omogućava prebacivanje na određeni commit ili granu.*

```
mlucan@DESKTOP-OOGB3MS MINGW64 ~/Desktop/maintaining-working-hours (develop)
$ git checkout adding-permission-for-users
Branch adding-permission-for-users set up to track remote branch adding-permissi
on-for-users from origin.
Switched to a new branch 'adding-permission-for-users'
```

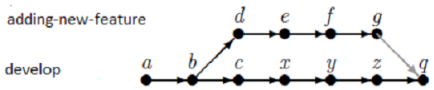
Slika 1.6: Prebacivanje u drugu granu

- ***git branch imegrane*** *- kreira novu granu u kojoj možemo raditi nove izmjene*

```
git branch adding-new-feature

ılucan@DESKTOP-OOGB3MS MINGW64 ~/Desktop/wk-hours/maintaining-working-hours (ad
ng-calendar)
 git branch
 adding-calendar
 adding-new-feature
 adding-permission-for-users
 develop
 editing-employee
 redesign
```

Slika 1.7: Nova grana **adding-new-feature**

- ***git merge*** *- naredba koja nam omogućava da izmjene napravljene u jednoj grani spojimo sa izmjenama u drugoj. Prilikom toga može doći do konflikata koji se tada moraju riješiti.*



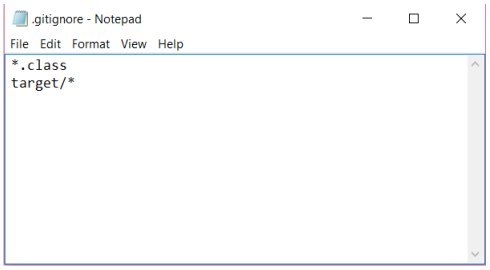
Slika 1.8: Spajanje izmjena iz grane **adding-new-feature** u **develop**

- ***git push*** *- slanje svih izmjena koje su kod nas lokalno spremljene u git repozitoriju na centralni repozitorij kako bi do njih mogli doći i ostali.*
- ***git pull*** *- povlačenje izmjena sa centralnog repozitorija u naš lokalni*

Datoteka .gitignore

U nekim situacijama neke tipove datoteka jednostavno ne želimo imati u repozitoriju. Primjerice ne želimo imati datoteke koje nastaju prevođenjem programskog koda (.class za javu, .o za C). Bilo bi dobro kada bismo mogli gitu na neki način reči : "*E mi ne želimo da nam spremaš datoteke koje nam nisu bitne za projekt.* "
Upravo to ćemo postići tako da kreiramo datoteku ***.gitignore*** u glavnom direktoriju projekta te u nju unesemo sve ono što ne želimo da bude dio povijesti projekta.

Primjerice, u povijesti projekta ne želimo .class datoteke i sve što se nalazi u direktoriju target. Na sljedećoj slici možemo vidjeti kako će ***.gitignore*** izgledati:

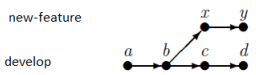


Slika 1.9: Datoteka .gitignore

Grananje

U pravilu kada razvijamo neke nove značajke za aplikaciju to ne radimo u glavnoj grani u kojoj razvijamo aplikaciju već napravimo novu i tamo nastavimo s izradom novih značajki. Kada smo gotovi sa tom izradom i zadovoljni smo sa ishodom vratimo se u glavnu granu i spojimo izmjene iz grane u kojoj je zančajka napravljena.

Naredba koja nam to omogućava: ***git branch imegrane***



Slika 1.10: Grananje iz developa: ***git branch new-feature***

1.2 Maven

Maven je alat razvijen od Apache-a. Službena web stranica se nalazi na linku: `https://maven.apache.org/`.

Upoznavanje s Mavenom

Maven je razvojni alat koji automatizira izgradnju softvera.
Maven izvršava sve potrebne ciljeve koji prethode izgradnji/instalaciji softvera. Npr. prevodi kod, izvrti sve testove, zapakira aplikaciju u jar/war datoteku.
Usvajanjem maven kovencija postižemo lakše razumijevanje strukture projekta. Dobro poznavanje strukture jednog maven projekta nam omogućava razumjevanje strukture svih projekata koji su bazirani na mavenu.
Maven se brine da osigura sve artifakte (predmete) o kojima projekt ovisi i koji su definirani u njegovoj konfiguracijskoj datoteci, prilikom svake faze u kojoj su oni potrebni.

Slika 1.11: Maven centralni repozitorij i *dependency* za JUnit

Konfiguracijska datoteka

Sve informacije o projektu koje su relevantne za maven nalaze se u *xml* datoteci koja se nalazi u baznom (root) direktoriju projekta i prema konvencijama se naziva ***pom.xml***, *eng. Project Object Model*.

U *pom-u* se definiraju informacije kao što je naziv i vrsta artifakta, osobe koje su zadužene za razvoj, artifakti o kojima projekt ovisi. Primjerice ako je za projekt potrebna neka biblioteka kao što je JUnit za unit testove, nije potrebno tražiti ***jar*** file na Internetu te ga skinuti i dodati u projekt već je dovljno pronaći njegovu ovisnost (*eng. dependency*) na centralnom maven repozitoriju i uključiti ga u konfiguracijski file. Maven će tokom faza izgradnje softvera prema potrebi povući sve ovisnosti sa repozitorija te uključiti u naš projekt.

Osnovne postavke *pom.xml* datoteke:

```
1  <project xmlns="http://maven.apache.org/POM/4.0.0"
2  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
       maven-v4_0_0.xsd">
4      <modelVersion>4.0.0</modelVersion>
5      <groupId>hr.altima</groupId>
6      <artifactId>working-hours</artifactId>
7      <packaging>war</packaging>
8      <version>1.0</version>
9      <name>Spring MVC Application</name>
10 </project>
```

Artifakt ili predmet

Artifakt predstavlja rezultat izgradnje maven projekta. Svaki maven projekt kao svoj rezultat daje artifakt. Npr. ***jar*** datoteku.

Maven repozitorij

Maven repozitorij predstavlja skladište artifakta. To skladište može biti lokalno ili na nekom udaljenom serveru. Maven skladište koje se nalazi kod nas lokalno na računalu nalazi se unutar direktorija ***.m2***.
Prilikom rješavanja ovisnosti (dependecy) maven prvo pretražuje repozitorij lokalno na našem računalu zatim ako ga ne nađe tamo spaja se na server gdje se nalazi maven repozi-

torij i povlači artifakt od tamo te ga sprema kod nas lokalno tako da idući put nebi morao pretraživati mrežno.

Životni ciklus izgradnje softvera

Prilikom izgradnje softvera oslanjamo se na standardni redoslijed akcija (ciljeva) (npr. priprema, generiranje koda, prevođenje, testiranje, pakiranje, instalacija).
Maven definira određeni broj faza koje se u nizu izvršavaju i koje definiraju ciljeve koje je potrebno ispuniti.
Također maven definira i redosljed kojim će se faze izvršiti pa ako zadamo fazu koju želimo izvršiti, maven će izvršiti sve faze od prve do zadnje uključujući i zadanu. Životni ciklus izgradnje softvera sastoji se od 23 faza.

Phase	Handles	Description
prepare-resources	resource copying	Resource copying can be customized in this phase.
compile	compilation	Source code compilation is done in this phase.
package	packaging	This phase creates the JAR / WAR package as mentioned in packaging in POM.xml.
install	installation	This phase installs the package in local / remote maven repository.

Slika 1.12: Neke od faza izgradnje softvera

Ako želimo izvršiti fazu compile tada se koristimo naredbom: ***mvn compile***. Prije samog izvršavanja *compile* faze izvršit će se sve faze prije. Akcija koja će se izvršiti u fazi *compile* je prevođenje koda.

1.3 Ant

Apache Ant je alat za izgradnju softvera od Apache Software Foundation-a. Ant dolazi od riječi *Another Neat Tool*. Kreirao ga je James Duncan Davidson. Koristio se kao alat za izgradnju Tomcat-a. Promoviran je kao neovisan projekt 2000.godine.
Zadnja verzija Ant-a je iz 2014. godine. Ant je dobar alat za automatiziranje kompliciranih zadataka. Dolazi s velikom listom već definiranih zadataka, ali i pruža mogućnosti za izgradnju vlastitih zadataka.
Kasnije ćemo pokazati kako Maven i Ant mogu zajedno "surađivati".

Ant skripte

Ant skripte pisane su u *xml-u*. Neformalna konvencija je da se takve skripte nazivaju *build.xml* i da se nalaze u baznom (root) direktoriju projekta ali nije pogrešno nazvati ih drugačije ili spremiti ih u neki drugi direktorij.

Sada ćemo na jednostavnom primjerima pokazati koja je sintaksa pisanja Ant skripte te kako je pokrećemo.

```
1 <?xml version="1.0"?>
2   <project name="Ant Project" default="info">
3     <target name="info">
4       <echo>Hello World - Welcome to Apache Ant!</echo>
5     </target>
6 </project>
```

Prije *xml* deklaracije nesmijemo imati praznih linija inače ćemo dobiti informaciju o greški.

```
mlucan@DESKTOP-OOGB3MS MINGW64 ~/Desktop/ant-example
$ ant
Buildfile: C:\Users\mlucan\Desktop\ant-example\build.xml

BUILD FAILED
C:\Users\mlucan\Desktop\ant-example\build.xml:3: The processing ins
truction target matching "[xX][mM][lL]" is not allowed.

Total time: 0 seconds
```

Slika 1.13: Prazni razamci prije *xml* deklaracije

Unutar *xml* elementa ***project*** zadajemo ciljeve koje želimo da se izvrše. Ciljeve označavamo *xml* elementom ***target***. Ciljevi su kolekcija zadataka koje želimo pokrenuti kao jednu cjelinu. U jednoj datoteci možemo imati više različitih ciljeva koji mogu biti ali i ne moraju

povezani, tj. ovisni o nekim drugim ciljevima. Unutar ***project*** elementa moramo imati barem jedan element ***target***.

Attributes	Description
name	The Name of the project. (Optional)
default	The default target for the build script. A project may contain any number of targets. This attribute specifies which target should be considered as the default. (Mandatory)
basedir	The base directory (or) the root folder for the project. (Optional)

Slika 1.14: Atributi koje ima element ***project***

Svojstva

Ant skripte pisane su u ***xml***-u. Prema tome one nam ne omogućavaju deklaraciju varijabli kao što nam to omogućavaju programski jezici, ali bilo bi korisno kada bismo mogli na neki način deklarirati varijable koje bi sadržavale neku informaciju kao što je npr. ime direktorija, ime projekata itd.

Sretna vijest je da nam to Ant zaista omogućava!
Ant koristi ***property*** element koji mu omogućava spremanje svojstava.
Ako želimo definirati neko svoje svojstvo to onda radimo na sljedeći način:

```
1 <?xml version="1.0"?>
2 <project name="Ant Project" default="info">
3    <property name="buildTool" value="Ant"/>
4    <target name="build-tool">
5      <echo>Najbolji alat je ${sitename} </echo>
6    </target>
7 </project>
```

```
mlucan@DESKTOP-OOGB3MS MINGW64 ~/Desktop/ant-example
$ ant
Buildfile: C:\Users\mlucan\Desktop\ant-example\build.xml

best-tool:
     [echo] Najbolji alat je Ant

BUILD SUCCESSFUL
Total time: 0 seconds
```

Slika 1.15: Izvršavanje cilja **build-tool**

Properties	Description
ant.file	The full location of the build file.
ant.version	The version of the Apache Ant installation.
basedir	The basedir of the build, as specified in the **basedir** attribute of the **project** element.
ant.java.version	The version of the JDK that is used by Ant.
ant.project.name	The name of the project, as specified in the **name** atrribute of the **project** element.
ant.project.default-target	The default target of the current project.
ant.project.invoked-targets	Comma separated list of the targets that were invoked in the current project.
ant.core.lib	The full location of the Ant jar file.
ant.home	The home directory of Ant installation.
ant.library.dir	The home directory for Ant library files - typically ANT_HOME/lib folder.

Slika 1.16: Unaprijed definirana svojstva

Tipovi podataka

Ant nam pruža veliki broj tipova podataka, no nemojmo se zbuniti s tim nazivom, to nisu tipovi podataka koje susrećemo u programskim jezicima. Ove tipove podataka možemo smatrati više kao skupinu servisa koji nam olakšavaju pisanje ant skripti.

Jedan od tipova podataka je ***Fileset***. On predstavlja kolekciju datoteka, a omogućava nam da iz nekog skupa datoteka izvučemo samo one koje nam trebaju prema nekom ključu kojeg smo zadali.

```
1 <fileset dir="${basedir}">
2 <include name="*.jar"/>
3 </fileset>
```

Element **fileset** ima i atribut **dir** koji označava direktorij iz kojeg uzimamo datoteke koje nam trebaju, u ovom slučaju samo **jar** datoteke.

1.4 Kontinuirana integracija

Na projektima u kojima sudjeluje više programera jedina opcija koja omogućava paralelan rad na istom projektu je korištenje centralnog repozitorija u kojem će se nalaziti programski kod te preko kojeg će se svi učesnici na tom projektu sinkronizirati.
Svaka nova izmjena programskog koda potencijalno može narušiti stabilnost aplikacije koja se razvija.

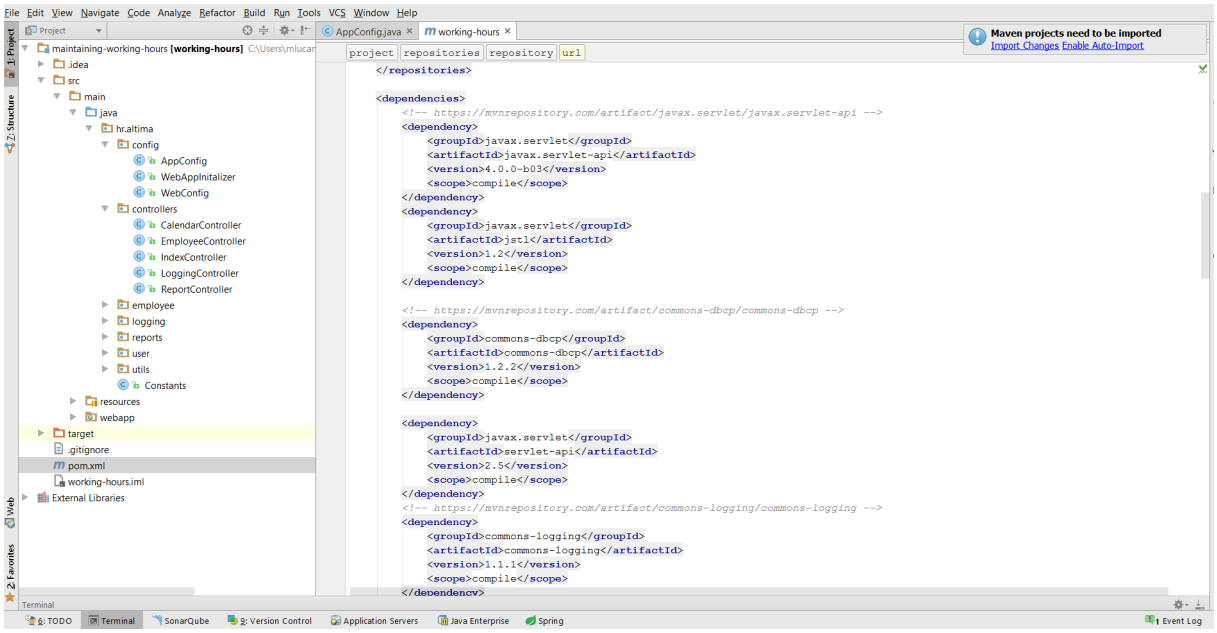
Kontinuirana integracija se svodi na svakodnevnu praksu da programeri automatizirano integriraju svoje promjene na centralni repozitorij, pri čemu se svi problemi mogu detektirati nakon svakog *commita*, a ne tek kad se aplikacija postavi na aplikacijski poslužitelj. Provjera ispravnosti svih promjena u pravilu se provodi osnovnim testovima (*unit* testovima - testovima kojima tesiramo djelove aplikacije) i integracijskim testovima koje programeri moraju pisati prilikom svake izmjene programskog koda. Bilo bi dobro kada bi svaki programer lokalno kod sebe izvrtio testove prije nego li svoje izmjene stavi na centralni repozitorij kako nebi narušio stanje aplikacije. Ponekad se zna dogoditi da programer zaboravi ili nema vremana lokalno pokrenuti testove te svejedno stavi izmjene koje je napravio u centralni repozitorij. Tu ulazi u igru integracijski poslužitelj (*eng.**integration server***) kao što je **Jenkins** koji će zatim pokrenuti sve testove te o ishodu testova poslati informacije programerima kako bi se pravovremeno ispravili bugovi.

Jenkins

Jenkins je javno dostupan automatizacijski server pisan u Javi. Koristi se za automatizaciju pojedinih zadataka kao što su izgradnja softvera, testiranje te za postavljanje softvera na aplikacijski poslužitelj.
Moguće ga je preuzeti u WAR paketu i pokreće se naredbom ***java -jar jenkins.war***.
Kasnije ćemo pokazati kako ga koristimo za automatizirano pokretanje svih testova.

1.5 IntelliJ IDEA

IntelliJ IDEA je integrirano razvojno okruženje za razvoj softvera. Proizvedena je od JetBrains-a. Prva verzija bila je puštena u siječnju 2001.godine i to je bio prvi dostupni alat za razvoj softvera koji je omogućavao napredno navigiranje po kodu te refaktoriranje koda.
Podržava i alate za kontrolu verzioniranja kao što su ***GIT***, ***Mercurial***, ***Perforce*** i ***SVN***. Također, osim za ***Javu***, ima podršku i za neke druge programske jezike : ***Groovy***, ***Python***, ***Kotlin*** *itd*.



Slika 1.17: IntelliJ

1.6 Tomcat

Tomcat je aplikacijski poslužitelj razvijen od Apache-a koji omogućava izvršavanje Java servleta i prikazivanje jsp stranica. Mi ćemo ga koristiti kako bismo pokretali našu aplikaciju lokalno iz IntelliJ-a.
Tomcat je skinut sa `https://tomcat.apache.org/download-70.cgi`.

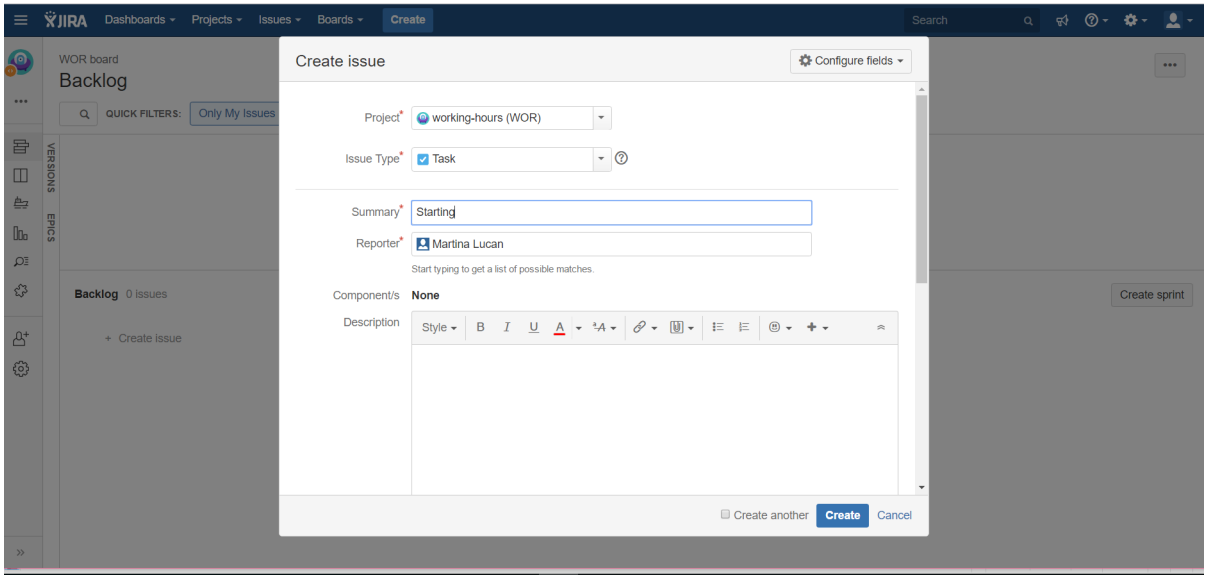
1.7 Jar/War datoteka

Jar (Java ARchive) je format komprimirane datoteke koja sadržava prevedene Java klase (*.class) i potrebne resurse koje mogu sačinjavati slike, tekstovi itd.

War (WApplication ARchive) datoteka se koristi za distribuciju web aplikacija. Slične je strukture kao i Jar datoteka koja sadrži različite datoteke u sebi. War datoteke sadrže prevedene Java klase, biblioteke za Javascript, statičke web stranice i ostale resurse koji su potrebni za aplikaciju.

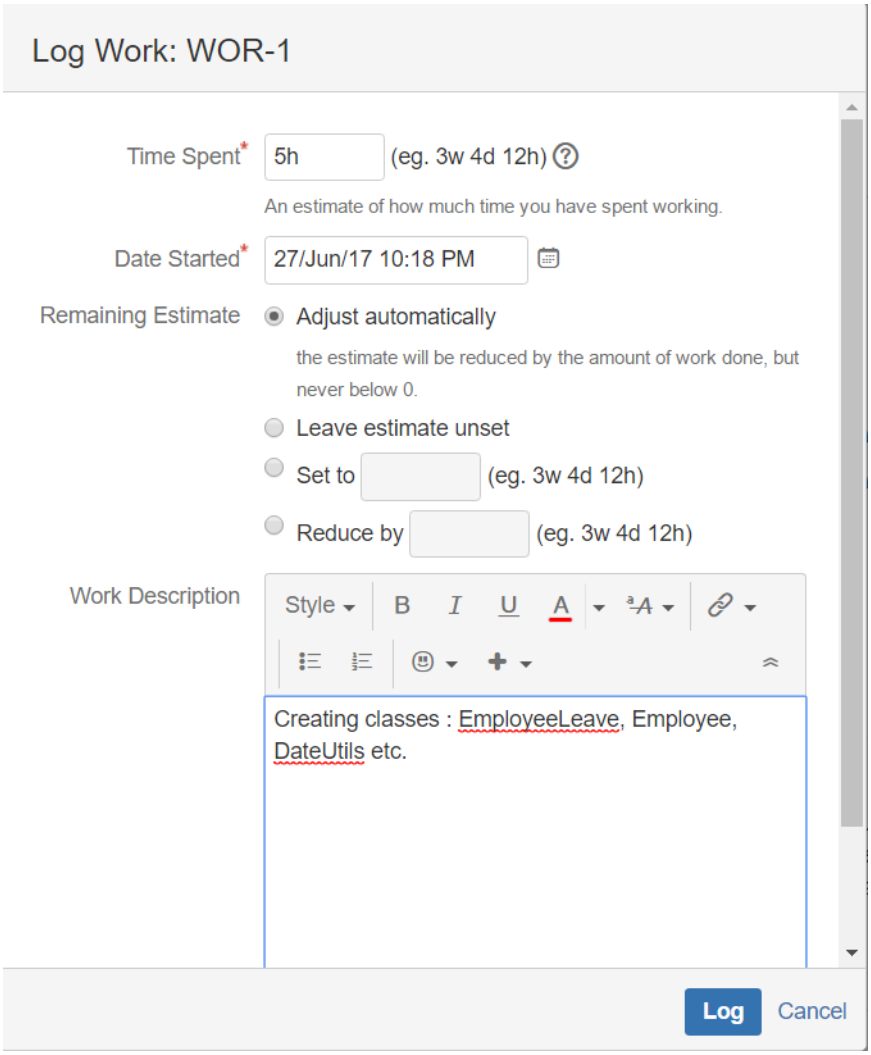
1.8 Jira

Jira je softver za praćenje novih zadataka i grešaka unutar neke aplikacije. Jira je razvijena od strane Atlassian-a. Omogućava evidenciju utrošenog vremena na pojedinim zadacima (npr. vrijeme utrošeno na razvoj neke nove značajke unutar aplikacije) te evidenciju vremena kojeg smo utrošili za prepravke određenih grešaka ili smetnji (*bugova*) koje je aplikacija imala.



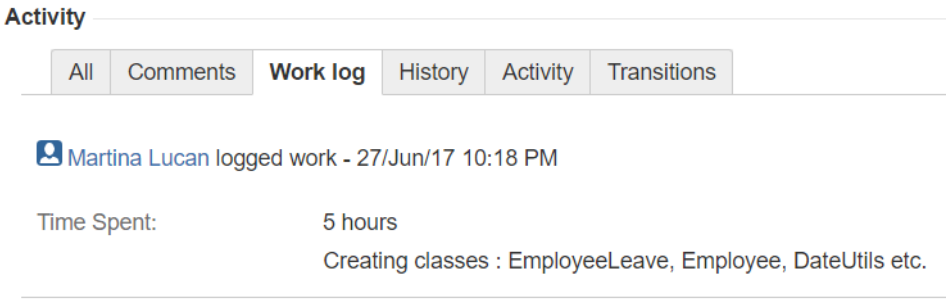
Slika 1.18: Jira- kreiranje zadatka

Na sljedećoj slici možemo vidjeti kako unosimo vrijeme koje smo utrošili za pojedine zadatke.



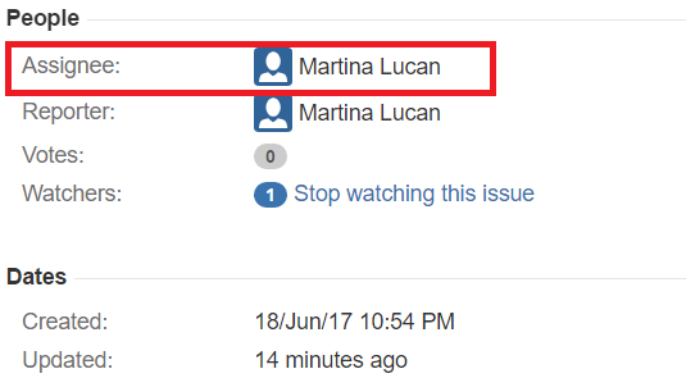
Slika 1.19: Evidencija utrošenog vremena

Ispod svakog zadatka napravljenog u Jiri možemo vidjeti koliko smo vremena potrošili na njega. Na slici ispod možemo vidjeti kako to izgleda.



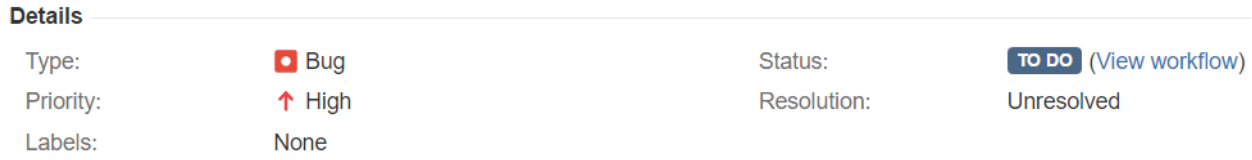
Slika 1.20: Jira - utrošeno vrijeme

Svaki zadatak u Jiri ima izvjestitelja, tj.osobu koja ga je kreirala. Ta osoba u Jiri se zove *reporter*. Osoba kojoj je zadatak namjenjen je izvršitelj tog zadatka i kada po zadatku definiramo njegovog izvršitelja u Jiri ga označimo kao *assignee*.

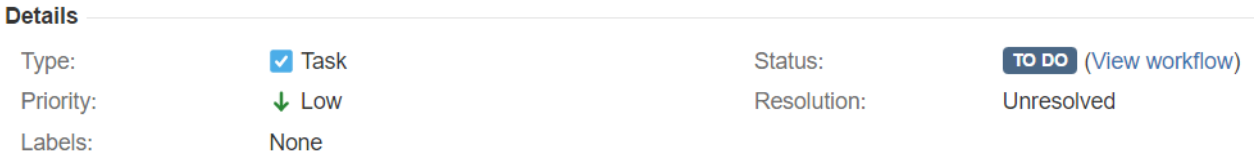


Slika 1.21: Zadatak je namjenjen Martini Lucan

Svaki zadatak ima prioritet. Nisu svi zadaci istog prioriteta. Ako imamo neku grešku u aplikaciji koja koči daljnji tijek testiranja koji prethodi postavljanju aplikacije na produkciju, tada je tu gešku potrebno što hitnije ispraviti. Jira nam omogućava da tagiramo zadatke prema prioritetu.



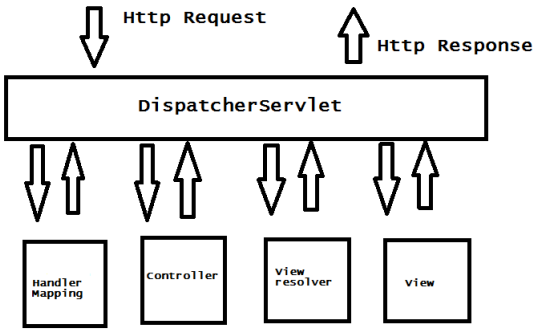
Slika 1.22: Bug je prijavljen s velikim prioritetom



Slika 1.23: Zadatak je prijavljen sa slabim prioritetom

1.9 Spring Mvc

Spring Mvc je okvir (*framework*) koji pruža gotove komponente za razvoj Web aplikacija. Spring je baziran na *Model-View-Controller* arhitekturi.



Slika 1.24: Model-View-Controller arhitektura

Prva postaja pri obrađivanju *Http upita* je *DispatcherServlet*. *DispatcherServlet* je ujedno i glavna komponenta (*front controller*). Kada *Http upit* stigne do njega on ga dalje usmjerava do ostalih kotrolera na obrađivanje. Za pronalaženja ispravnog kontrolera koji će dalje nastaviti s obradom *upita DispatcherServlet* se posavjetuje sa jednim ili više rukovoditelja mapiranja (*handler mapping*). Jednom kada je odgovarajući kontroler pronađen *upit* se dalje prosljeđuje njemu te se vrši odgovarajuća obrada *upita* (izvršava se neka logika nad tim upitom). Rezultati obrade spremaju se u tzv. model zajedno sa imenom pogleda (*view*) te tako upakirani u model dalje prosljeđuju dispečeru (*DispatcherServlet*). Dispečer se zatim konzultira sa rješavateljem pogleda (*view resolver*) koji zatim vraća odgovarajuću implementaciju tog prikaza npr. neku *jsp* datoteku kojoj će se dalje prosljediti dani model iz kojeg će se izvući potrebne informacije koje su nastale kao rezultat obrade *Http upita* što će zatim rezultirati grafičkom (user friendly) prikazu web stranice koje su čitljive i razmljive svakom korisniku.

Konfiguracija

Opisat ćemo ukratko neke osnovne elemente koje je potrebno konfigurirati kako bismo upogonili sve potrebne komponente koje su bitne za uspješno obrađivanje http upita.

Za početak potrebno je kreirati glavnu komponentu koja je ujedno i prva postaja upita kojeg trebamo obraditi. Prije se konfiguracija dispečera (*DispatcherServlet*) radila u *web.xml* datoteci koja se nalazi u direktoriju WEB-INF/. No danas imamo mogućnosti konfigurirati dispečera služeći se isključivo **Javom**.

```
import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

public class WebAppInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {
    @Override
    protected Class<?>[] getRootConfigClasses() { return new Class<?>[]{RootConfig.class}; }

    @Override
    protected Class<?>[] getServletConfigClasses() { return new Class<?>[]{WebConfig.class}; }

    @Override
    protected String[] getServletMappings() {
        return new String[]{"/"};
    }
}
```

Kreirat ćemo klasu *WebAppInitializer* koja proširuje apstraktnu klasu *AbstractAnnotationConfigDispatcherServletInitializer* te ćemo promijeniti implementaciju trima metodama. Metoda *getServletMappings()* nam omogućava da definiramo koji upiti će dolaziti do našeg dispečera. U ovom našem slučaju dolaziti će svi jer smo tako definirali mapiranje.

```
@Override
protected String[] getServletMappings() {
    return new String[]{"/login"};
}
```

Slika 1.25: Upravitelj će primati samo upite tipa: **localhost:8080/login**

Metodom *getServletConfigClasses()* smo dispečeru dali do znanja da želimo da se aplikacijski kontekst napuni *bean*-ovima koji su definirani u klasi *WebConfig*. U našem slučaju imamo samo jedan *bean*, a to je *view resolver* koji će nam omogućiti da na temelju logičkog naziva pogleda, kojeg će prosljediti kontroler u modelu kojeg će kreirati, nađemo njegovu implementaciju. Npr.u modelu ćemo poslati ime pogleda "hello" a view resolver će vratiti */WEB-INF/view/hello.jsp*.

```
@Configuration
@EnableWebMvc
@ComponentScan(basePackages = "hr.company")
public class WebConfig extends WebMvcConfigurerAdapter {
    @Bean
    public ViewResolver viewResolver() {
        InternalResourceViewResolver resolver =
            new InternalResourceViewResolver();
        resolver.setPrefix("/WEB-INF/view/");
        resolver.setSuffix(".jsp");
        resolver.setExposeContextBeansAsAttributes(true);
        return resolver;
    }

    @Override
    public void configureDefaultServletHandling(
        DefaultServletHandlerConfigurer configurer) {
        configurer.enable();
    }
}
```

Slika 1.26: Klasa *WebConfig*

Anotacija @ComponentScan aktivira skeniranje paketa kojeg navedemo unutar atributa *basePackage* i tako pronalazi sve kontrolere kojima zatim napuni aplikacijski kontekst. U slučaju da nema te anotacije, potrebno je direktno kreirati instance svih servisa ili kontrolera jer inače ih aplikacijski kontekst neće sadržavati.

```
@Configuration
@EnableWebMvc
public class WebConfig extends WebMvcConfigurerAdapter {
    @Bean
    public ViewResolver viewResolver() {
        InternalResourceViewResolver resolver =
            new InternalResourceViewResolver();
        resolver.setPrefix("/WEB-INF/view/");
        resolver.setSuffix(".jsp");
        resolver.setExposeContextBeansAsAttributes(true);
        return resolver;
    }

    @Bean
    public HelloController helloController(){
        return new HelloController();
    }

    @Override
    public void configureDefaultServletHandling(
        DefaultServletHandlerConfigurer configurer) {
        configurer.enable();
    }
}
```

Slika 1.27: Umjesto anotacije @ComponentScan direktno kreiramo instance kontrolera

Java klase koje predstavljaju kontrolera anotirane sa @Controller.

```
@Controller
@Component
public class HelloController {

    @RequestMapping(value = "/hello", method = RequestMethod.GET)
    public ModelAndView hello(){
        ModelAndView modelAndView = new ModelAndView("hello");
        modelAndView.addObject("message", "Hello!");

        return modelAndView;
    }
}
```

Slika 1.28: Logičko ime pogleda je *hello*

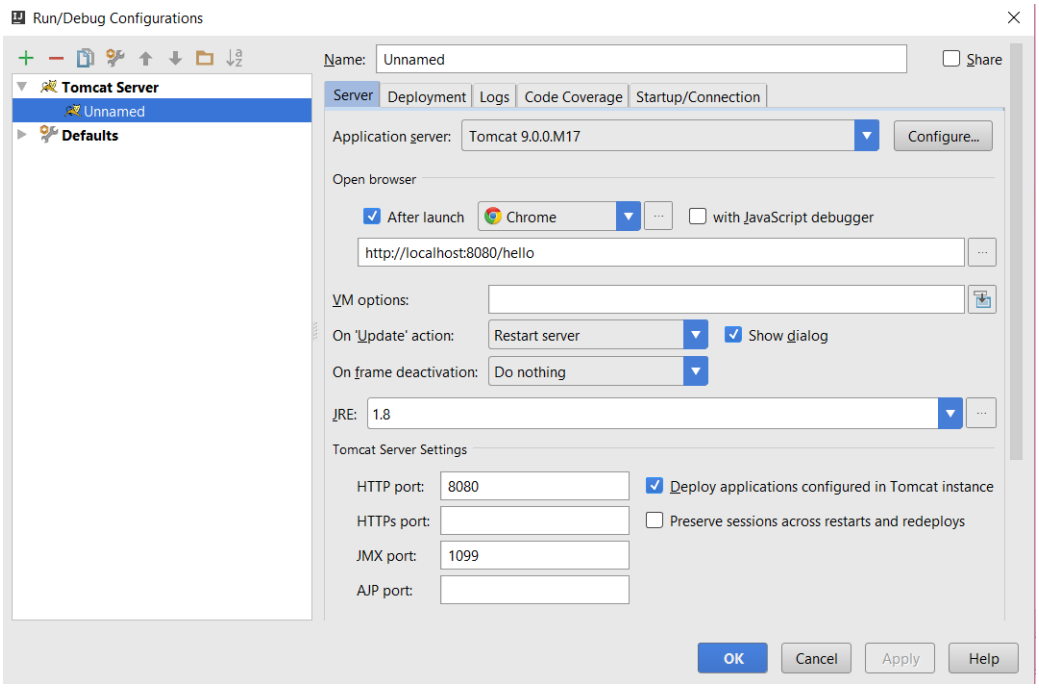
Klasa *HelloController* jednostavan je primjer kontrolera. Uočimo da je metoda *hello()* anotirana sa @RequestMapping(value = "hello", method = RequestMethod.GET). Ta anotacija govori da će se logika koja je implementirana u metodi *hello()* izvršiti nad upitima čiji **URL** izgleda otprilike ovako : ***hostname:port/hello***.
U ovom našem jednostavnom primjeru osim imena pogleda u modelu smo slali i poruku *"Hello!"* kako bismo kasnije kada pokrenemo aplikaciju vidjeli da li je naš upit došao do kontrolera te da li se pronašla ispravna **jsp** datoteka (u našem slučaju **hello.jsp**) koja će zatim korisniku ispisati poruku na web stranici.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>${message}</h1>
</body>
</html>
```

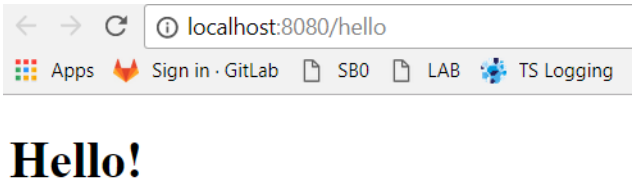
Slika 1.29: Sadržaj datoteke **hello.jsp**

Pokretanje Spring aplikacije koristeći se IntellJ-om i Tomcatom-om

Za pokretanje ove malene Spring aplikacije koristiti ćemo se IntellJ-om i Tomcat-om na način da ćemo pokrenuti aplikaciju u **debug** modu koji će u pozadini pokrenuti Tomcat. Zatim će se aplikacija instalirati na Tomcat i u Chrome-u će nam se otvoriti stranica čiji će URL biti ***localhost:8080/hello*** i prikazat će se naša poruka *"Hello"*.



Slika 1.30: Konfiguracija Tomcat-a u IntellJ-u



Slika 1.31: Krajnji rezultat: *"Hello!"*

Poglavlje 2

Razvoj web aplikacije

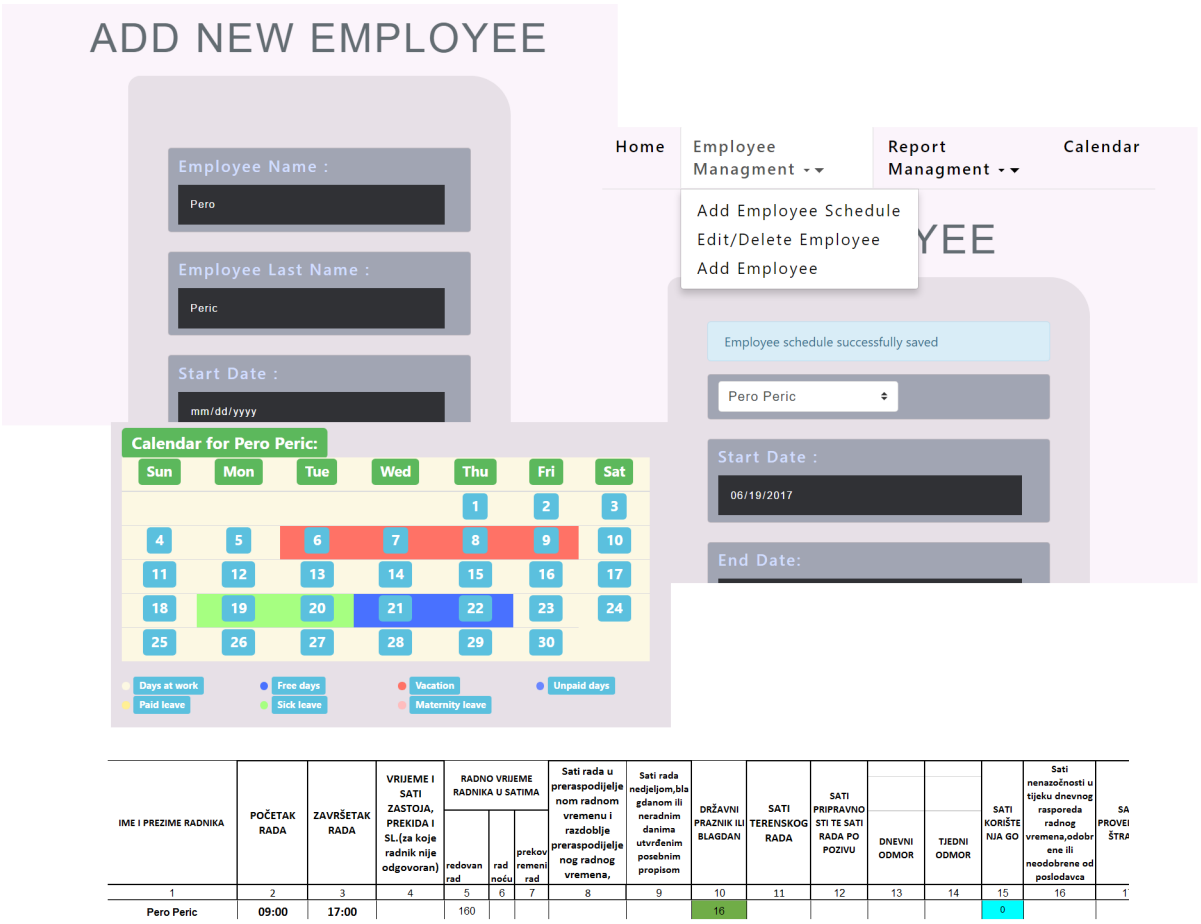
U prethodnom poglavlju ukratko smo opisali alate s kojima ćemo se u nastavku susretat, u ovom poglavlju ćemo opisati kako izgleda jedan tijek razvoja aplikacije.
Izradu same aplikacije nećemo detaljno opisivati jer bismo onda izgubili podosta vremena, već ćemo se više fokusirati na sami tijek razvoja. Prvo ćemo opisati aplikaciju kako bismo znali koja je njena svrha i za koju skupinu korisnika je predviđena, zatim ćemo opisati korake za postavljanje okoline koja nam je potrebna za razvoj a nakon toga ukratko prikazati tok razvoja aplikacije. Aplikacija je pisana u programskom jeziku ***Java*** a koristimo se i ***Spring Mvc*** okvirom.

2.1 Kratak opis aplikacije

Aplikacija je predviđena za osobe koje se bave administrativnim poslovima. Unutar velikih organizacija postoji velik broj zaposlenih ljudi. Na kraju mjeseca svaki zaposlenik prima plaću koja je predviđena za njega. Osobe koje su zadužene za isplatu plaće moraju svakog mjeseca imati uvid u sljedeće stavke za svakog zaposlenika:

- Da li je bio na godišnjem? Ako je, koliko dugo?
- Da li je koristio slobodne dane? Ako je, koliko dugo?
- Da li je bio na bolovanju? Ako je, koliko dugo?
- Da li je bio na porodiljnom dopustu? Ako je, koliko dugo?
- itd.

Na temelju informacija koje znamo o pojedinom zaposleniku potrebno je na kraju mjeseca napraviti izvještaj (report) na temelju kojeg će se isplatiti odgovarajuća plaća svakom od zaposlenika.

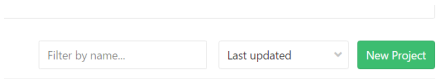


Slika 2.1: Unos zaposlenika, godišnjeg odmora i kalendar

2.2 Postavljanje razvojne okoline

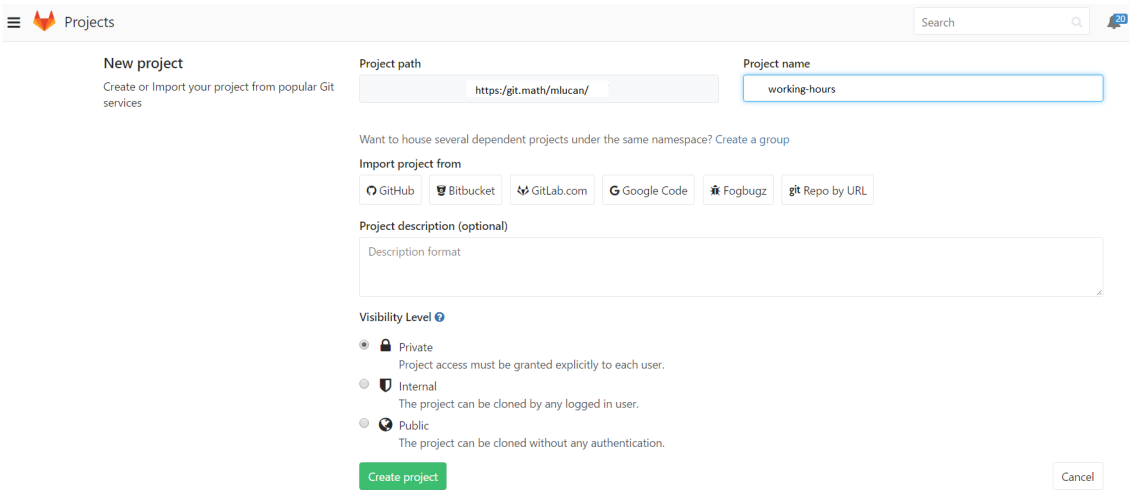
Kreiranje centralnog repozitorija na GitLabu

Na GitLabu odaberemo opciju NewProject.



Slika 2.2: Kreiranje projekta

Kliknemo li na NewProject gumb prikazan na prethodnoj slici dobivamo sljedeće:



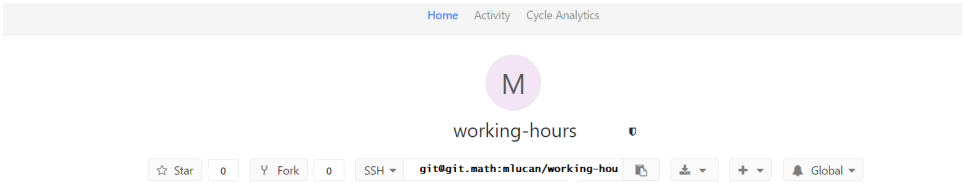
Slika 2.3: Kreiranje projekta

Kliknemo li na gumb Create project kreirali smo repozitorij u kojem će se nalaziti naš projekt.

Kloniranje projekta, .gitignore i grananje

Kloniranje projekta - kopiranje projekta sa udaljenog repozitorija

Nakon što smo kreirali centralni repozitorij na GitLabu u kojem će se nalaziti naš projekt, potrebno ga je sada povući lokalno. To radimo koristeći se git naredbom:
git clone git@git.math:mlucan/working-hours.git.
Projekt koji smo sada povukli je prazan. Grana u kojoj se zasad nalazimo je ***master***.



Slika 2.4: SSH url koji koristimo za kloniranje projekta

Datoteka .gitignore i develop grana

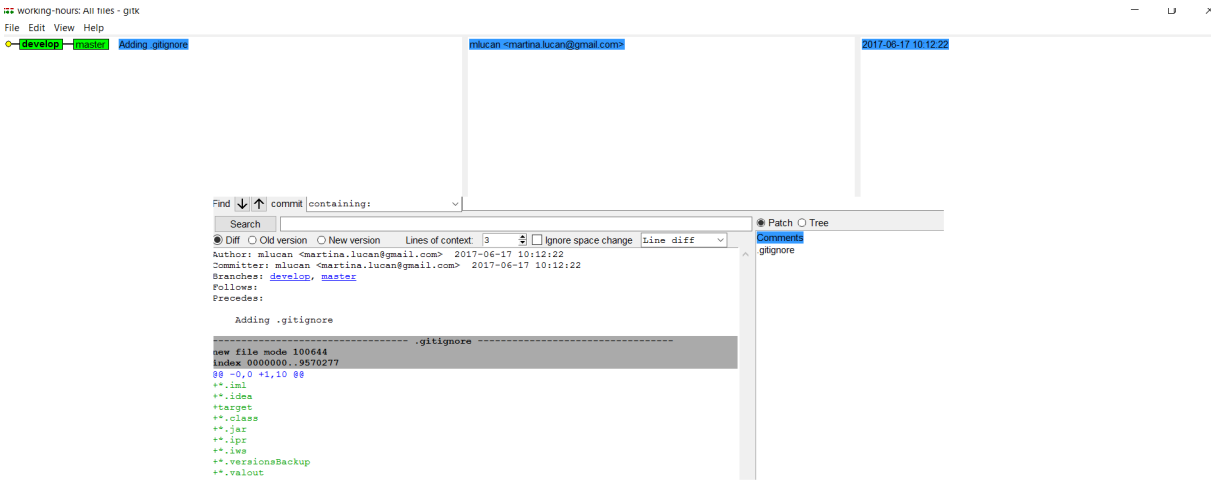
Potrebno je napraviti popis svih tipova datoteka koje ne želimo u povijesti našeg projekta, tj. ne želimo ih u git repozitoriju. Naša **.gitignore** datoteka sadžavat će:

```
*.iml
*.idea
target
*.class
*.jar
*.ipr
*.iws
*.versionsBackup
*.valout
.DS_Store
```

Nakon toga datoteku **.gitignore** spremit ćemo u ***master granu***, te iz ***master*** grane napraviti granu ***develop*** u kojoj ćemo razvijati aplikaciju.

```
mlucan@DESKTOP-OOGB3MS MINGW64 ~/Desktop/working-hours (master)
$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .gitignore

nothing added to commit but untracked files present (use "git add" to track)

mlucan@DESKTOP-OOGB3MS MINGW64 ~/Desktop/working-hours (master)
$ git add .

mlucan@DESKTOP-OOGB3MS MINGW64 ~/Desktop/working-hours (master)
$ git commit -m "Adding .gitignore file"
[master (root-commit) 7d62c48] Adding .gitignore file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 .gitignore

mlucan@DESKTOP-OOGB3MS MINGW64 ~/Desktop/working-hours (master)
$ git checkout -b develop
Switched to a new branch 'develop'

mlucan@DESKTOP-OOGB3MS MINGW64 ~/Desktop/working-hours (develop)
$
```

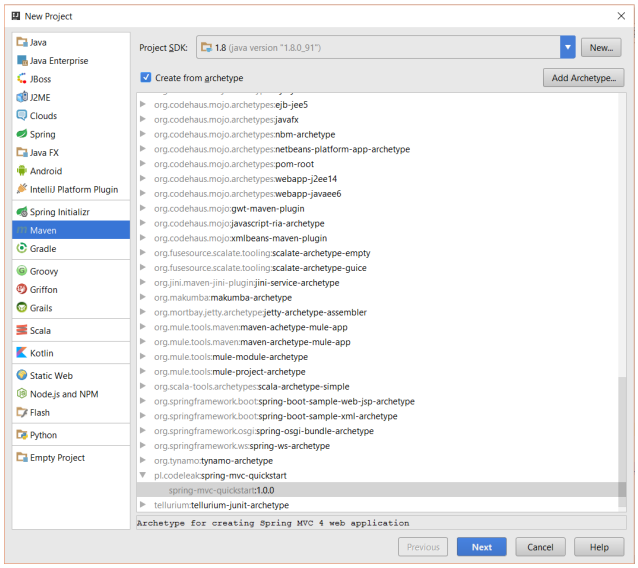
Slika 2.5: Spremanje datoteke na ***master***, kreiranje ***develop*** grane



Slika 2.6: Trenutna povijest projekta, naredba ***gitk***

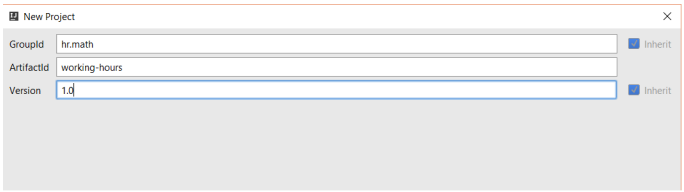
Kreiranje maven projekta koristeći IntelliJ

IntelliJ nam omogućava da kreiramo projekt prema maven konvencijama. Na slici ispod možemo vidjeti kako to izgleda.



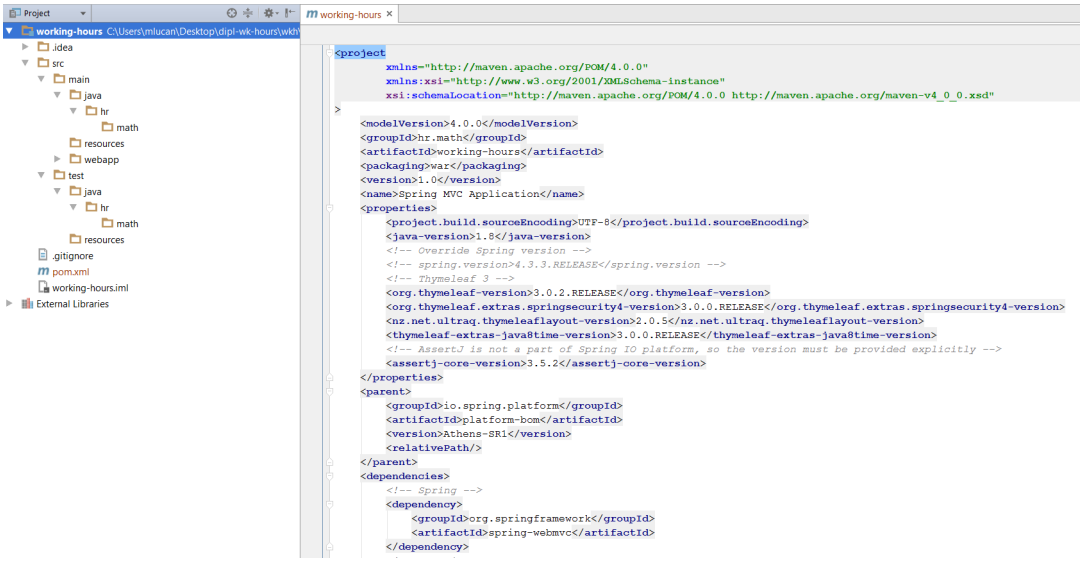
Slika 2.7: Kreiranje maven projekta + Spring MVC okvir

Omogućava nam definiranje groupId-a,artifactId-a te verzije.



Slika 2.8: GroupId, ArtifactId, Version

U konačnici dobivamo gotovu maven strukturu projekta sa pom-om koji je ispunjen osnovnim podacima bitnim za maven projekt (groupId, artifactId, version) i sadrži ovisnosti (dependencies) za Spring MVC okvir.



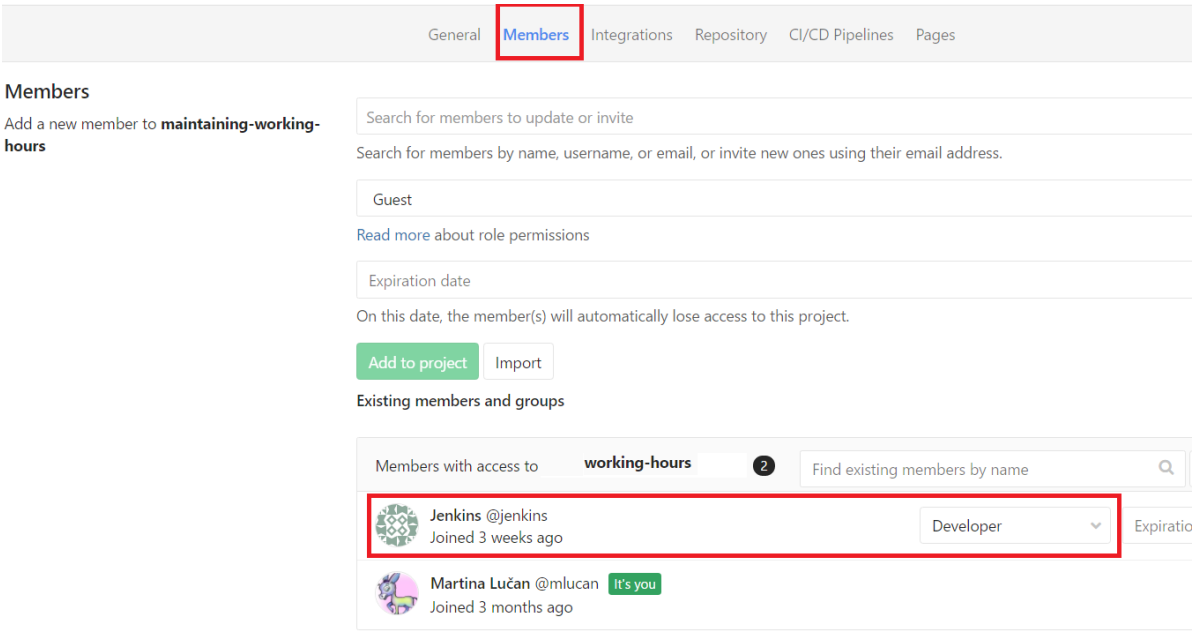
Slika 2.9: Struktura maven projekta

Integracija GitLaba i Jenkinsa

U ovom ćemo odjeljku napraviti Jenkins integraciju sa *GitLab-om*.
Drugim riječima, napravit ćemo jedan Jenkins job koji će se izvršavati svaki puta kada u ***develop*** granu na centralnom repozitoriju stavimo nove izmjene koristeći se naredbom ***git push***. Jenkins job ćemo konfigurirati tako da kad se počne izvršavati on prvo povuče projekt sa *GitLab-a*, prevodi (kompajlira) projekt, zatim pokrene sve testove, te na kraju ako sve dobro prođe zapakira aplikaciju. Ako slučajno nešto od toga ne prođe uspješno job će pasti i mi ćemo to vidjeti.

Prvi korak:

Na GitLabu moramo napraviti Jenkins korisnika te mu pridjeliti developer prava.



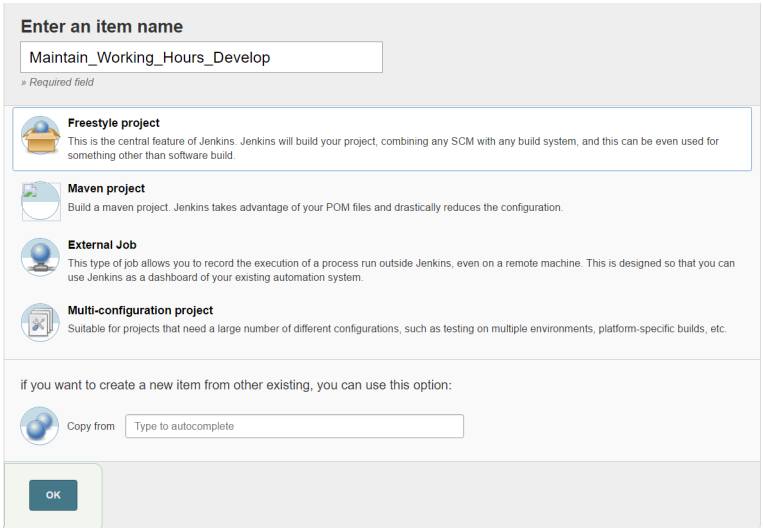
Slika 2.10: Korisniku Jenkins su dodjeljena developer prava

Drugi korak:

Kreiramo ***Jenkins job***. Na početnoj stranici Jenkinsa odaberemo link ***New item***.

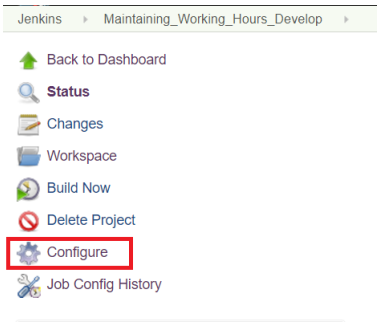
Slika 2.11: Kreiranje job-a

Zatim nazovemo *job* ***Maintain-Working-Hours-Develop*** i odaberemo opciju ***Freestyle project*** te kliknemo ***Ok***.



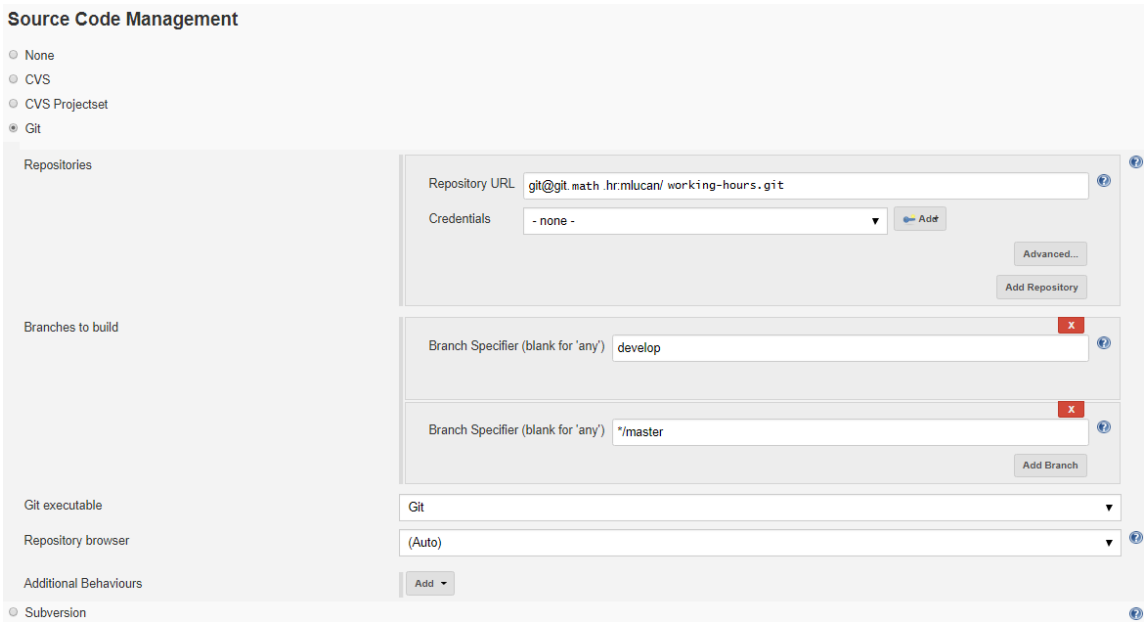
Slika 2.12: Freestyle project

Treći korak: U ovom koraku ćemo konfigurirati Jenkins job.



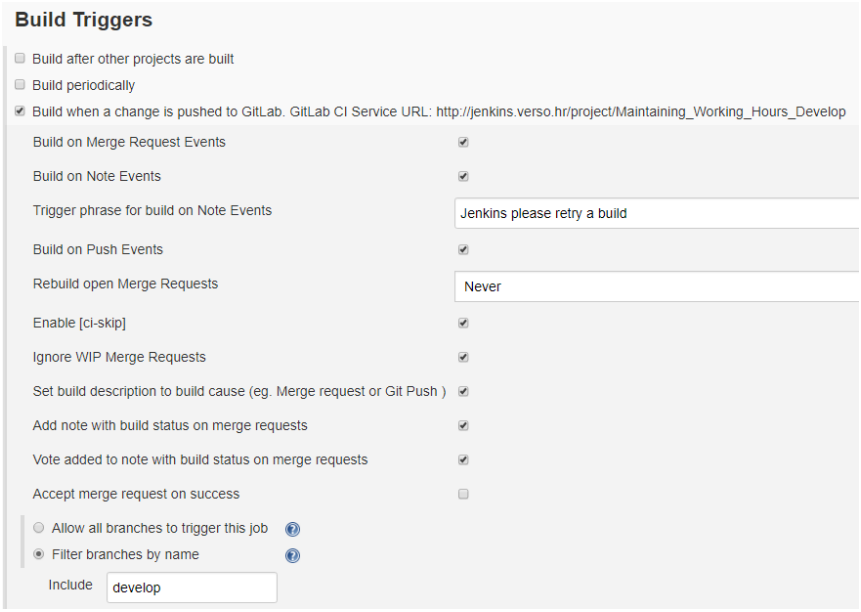
Slika 2.13: Kliknemo na ***Configure***

Za *repository URL* uzimamo isti url koji smo uzeli kada smo klonirali projekt kod sebe lokalno, tj. ***git@git.math:mlucan/working-hours.git***. Označimo da želimo izvršavanje job-a svaki puta kada se dogodi neka promjena na *GitLab-u* u grani ***develop*** našeg projekta. Na kraju u *execution shell-u* zadamo naredbu ***mvn package***. To je maven naredba koja će naš projekt upakirati u *war file* ali prije toga će se izvršiti prevođenje i pokrenuti svi testovi.



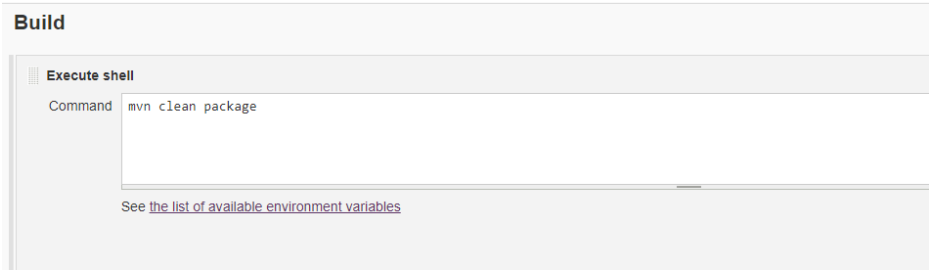
Slika 2.14: Dodavanje *repository* URL-a u konfiguraciju *Jenkins job-a*

Zatim konfiguriramo *Jenkins job* tako da se počne izvršavati nakon svake promjenu u **developu** koja je uzrokovana **git push** naredbom.



Slika 2.15: Izvršavanje *Jenkins job-a* kada se dogodi promjena u ***developu***

Na kraju u ***shell-u*** dodamo naredbu ***mvn package***.

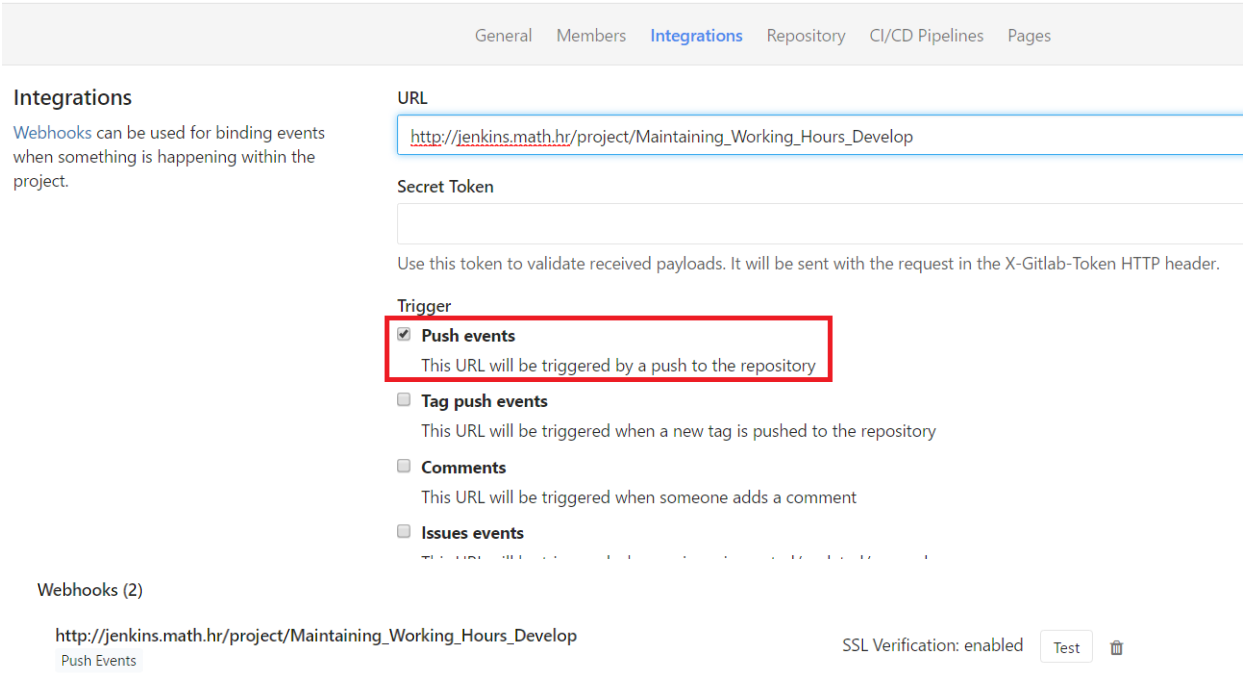


Slika 2.16: Naredba koja će se izvršiti kada se novo stanje projekta povuče sa *GitLab-a*

Četvrti korak: Ovo je zadnji korak nakon kojeg je Jenkins job spreman za izvođenje zadatka. Potrebno je još samo kopirati GitLab CI Service Url i dodati ga na GitLabu.

Slika 2.17: GitLab CI service url

Treba još samo dodati *webhook* (*HTTP callback*). Svaki puta kada se dogodi neka promjena u grani *develop* GitLab će o tome obavijestiti Jenkins. Jenkins će tada započeti izvršavanje *job-a* kojeg smo konfigurirali tako da prvo povuče najnovije stanje projekta sa *develop-a* te zatim izvrši maven naredbu: *mvn package*.



Slika 2.18: Dodavanje webhook-a

2.3 Razvoj aplikacije...

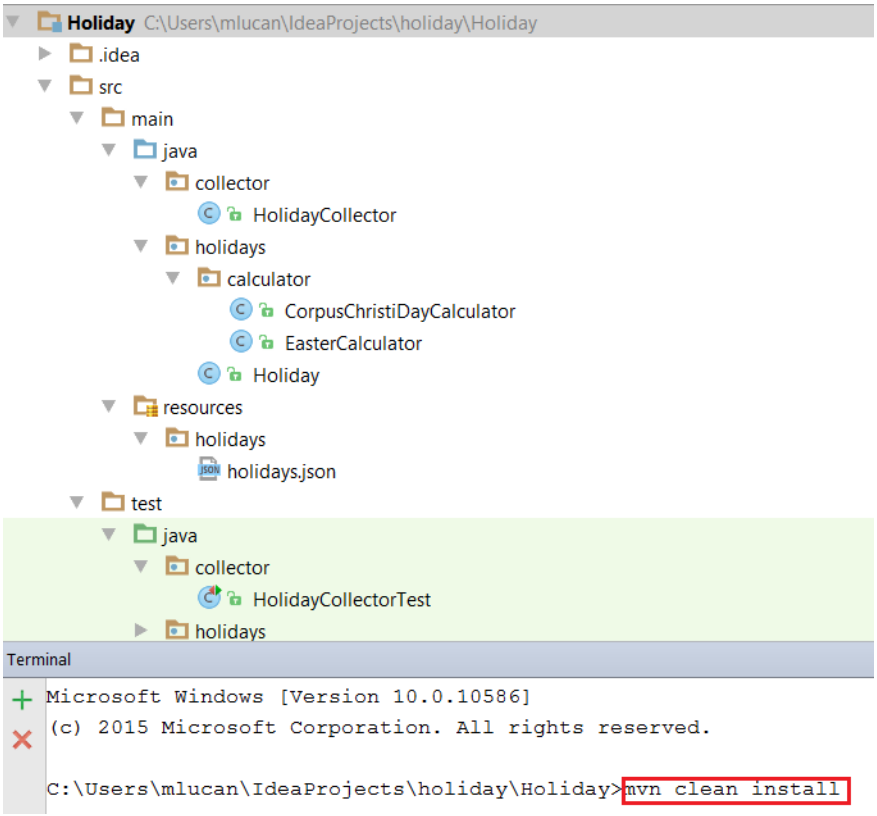
U prethodnim odjeljcima postavili smo sve što nam je potrebno kako bismo se nesmetano mogli posvetiti samom razvoju aplikacije. Napravili smo centralni repozitorij koji će nam omogućiti da na našem projektu sudjeluje više ljudi istovremeno, zatim smo kreirali maven projekt i definirali ovisnosti za *Spring Mvc okvir* u ***pom.xml*** datoteci, tj. IntelliJ nam je to sam izgenerirao. Na kraju smo napravili Jenkins job koji nam omogućava automatizirano pokretanje svih testova svaki put kad u granu ***develop*** stavimo neke nove izmjene (pushamo izmjene).
U ovom odjeljku ukratko ćemo opisati razvoj same aplikacije. Kako je aplikacija prilično velika opisat ćemo samo neke korake izgradnje. Obzirom da se ovdje radi o web aplikaciji, koju ćemo pokretati pomoću aplikacijskog servera Tomcat, automatizirat ćemo postavljanje aplikacije na Tomcat koji će se nalaziti na virtualnoj mašini. Za tu automatizaciju koristit ćemo se Mavenom i Ant-om. Kako je Ant također alat namijenjen za izgradnju softvera (*build tool*) bilo bi zgodno pokazati kako možemo koristiti Ant i Maven zajedno. Također, dobro je napomenuti kako nam i Jenkins omogućava automatizaciju postavljanja aplikacije na aplikacijski server, no s time se na žalost nećemo baviti u ovom radu.

Kreiranje osnovnih klasa

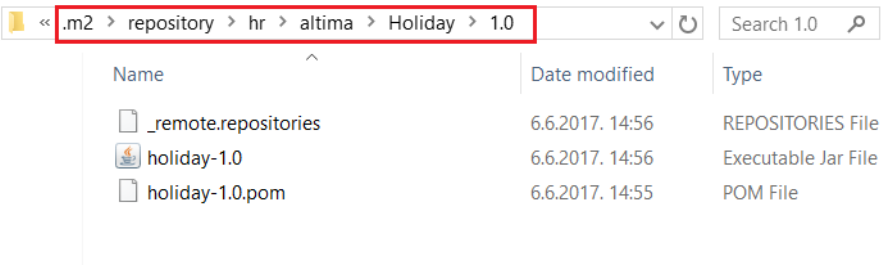
Kreirat ćemo klasu *DateUtil* koja će sadržavati metodu koja računa koliko radnih dana imamo između dva datuma. Za računanje koliko radnih dana ima između dva datuma potrebno je i imati informaciju o hrvatskim blagdanima. Iz tog razloga smo napravili maven projekt *holiday* koji smo instalirali lokalno maven naredbom ***mvn install*** tako da kada smo u našu konfiguracijsku datoteku dodali ovisnosti o biliblioteci *holiday*, Maven je pri rješavanju ovisnosti to pronašao kod nas lokalno.

Primijetimo sljedeće!
Kada će Jenkins izvršavati *job*, koji smo konfigurirali za potrebe našeg projekta, on neće moći riješiti ovisnost za biblioteku *holiday* jer ona je instalirana na našem računalu.
Stoga, da nebismo imali problema sa Jenkinsom potrebno je našu biblioteku *holiday* postaviti na neki udaljeni repozitorij. Za postavljanje biblioteke na udaljeni repozitorij koristimo se naredbom ***mvn deploy***.

Na sljedećoj stranici pokazat ćemo gdje se instalirala biblioteka *holiday* na našem računalu.



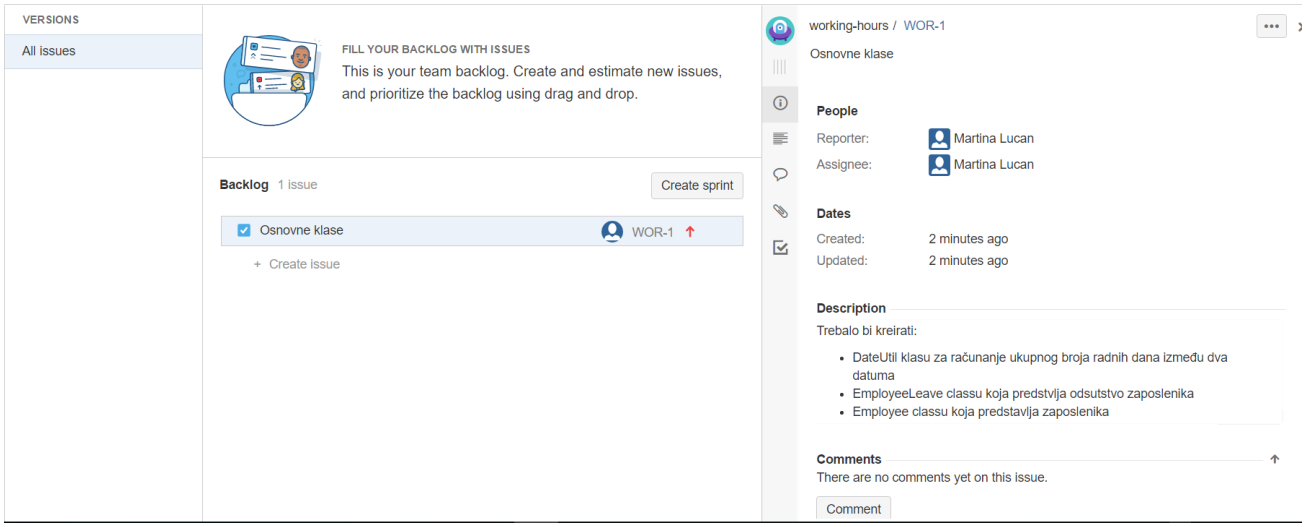
Slika 2.19: Maven struktura projekta *holiday* i njegova instalacija



Slika 2.20: Mjesto gdje je maven instalirao holiday

Nakon što smo se pozabavili sa bibliotekom *holiday* potrebno je kreirati klasu *EmployeeLeave* koja predstavlja jedno odsustvo zaposlenika, npr.godišnji odmor od *21.08.2017* do *05.09.2017*. Također, kreirat ćemo i klasu *Employee* koja predstavlja jednog zaposlenika i sadrži sve informacije o njemu.

U Jiri ćemo napraviti task koji će sadržavati plan za prve korake izgradnje aplikacije.

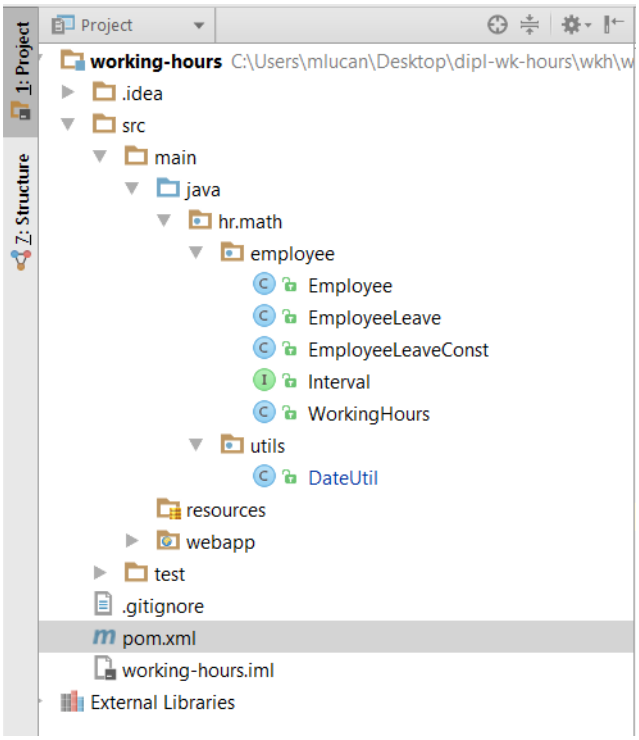


Slika 2.21: Jira task

U ***pom.xml*** smo dodali sljedeću ovisnost:

```
<dependency>
  <groupId>hr.altima</groupId>
  <artifactId>holiday</artifactId>
  <version>1.0</version>
</dependency>
```

Zatim smo kreirali klase *DateUtil*, *EmployeeLeave*, *WorkingHours*, *Employee* te sučelje *Interval*.



Sada ćemo, koristeći se git-om, te izmjene spremiti. Izmjene će biti spremljene u lokalnom git repozitoriju. Kasnije ćemo naredbom ***git push develop*** sve što je kod nas lokalno u grani ***develop*** spremiti u repozitorij na *GitLabu*.

```
mlucan@DESKTOP-OOGB3MS MINGW64 ~/Desktop/working-hours (develop)
$ git status
On branch develop
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        pom.xml
        src/

nothing added to commit but untracked files present (use "git add" to track)

mlucan@DESKTOP-OOGB3MS MINGW64 ~/Desktop/working-hours (develop)
$ git add .

mlucan@DESKTOP-OOGB3MS MINGW64 ~/Desktop/working-hours (develop)
$ git commit -m "Adding EmployeeLeave,DateUtil etc."
[develop e34fa65] Adding EmployeeLeave,DateUtil etc.
```

Za metodu *getWorkingDaysBetweenTwoDates(Date startDate, Date endDate)* iz *DateUtil* klase napisat ćemo nekoliko osnovnih (*unit*) testova. Testove je bitno pisati što više kako bismo u budućnosti, ako dođe do nekih promjena u implementaciji, bili sigurni da se funkcionalnost pojedine metode nije izgubila ili da neki proces i dalje radi ispravno.
Iz tih razloga bitno je da svaki projekt bude pokriven sa što više testova. Posebno ako se radi o izgradnji nekih većih sustava koji se stalno mjenjaju ili nadopunjuju i kod kojih stalno dolazi do nekih promjena. Iako samo pisanje testova se doima kao gubitak vremena, oni su zaista jako bitni da očuvaju stabilnost aplikacije.
Osim *unit* testova postoje i integracijski testovi. Razlika između te dvije vrste testova je ta što *unit* testovi testiraju funkcionalnost jedne metode dok integracijski jedan veći proces, tj. veći dio aplikacije.

```
public static int getWorkingDaysBetweenTwoDates(Date startDate, Date endDate)
    throws ParseException {
    Calendar calendar = Calendar.getInstance();
    calendar.setTime(startDate);
    HolidayCollector holidayCollector = new HolidayCollector();
    if (startDate.getTime() == endDate.getTime()
        && calendar.get(Calendar.DAY_OF_WEEK) != Calendar.SATURDAY
        && calendar.get(Calendar.DAY_OF_WEEK) != Calendar.SUNDAY
        && !holidayCollector.isDatePublicHoliday(calendar.getTime())) {
        return 1;
    } else if (startDate.getTime() == endDate.getTime()) {
        return 0;
    }

    if (before(endDate, startDate)) {
        return 0;
    }
    int workingDays = 0;
    while (before(calendar.getTime(), endDate) || calendar.getTime().compareTo(endDate) == 0) {
        if (calendar.get(Calendar.DAY_OF_WEEK) != Calendar.SATURDAY
            && calendar.get(Calendar.DAY_OF_WEEK) != Calendar.SUNDAY
            && !holidayCollector.isDatePublicHoliday(calendar.getTime())) {
            workingDays++;
        }
        calendar.add(Calendar.DAY_OF_MONTH, 1);
    }
    return workingDays;
}
```

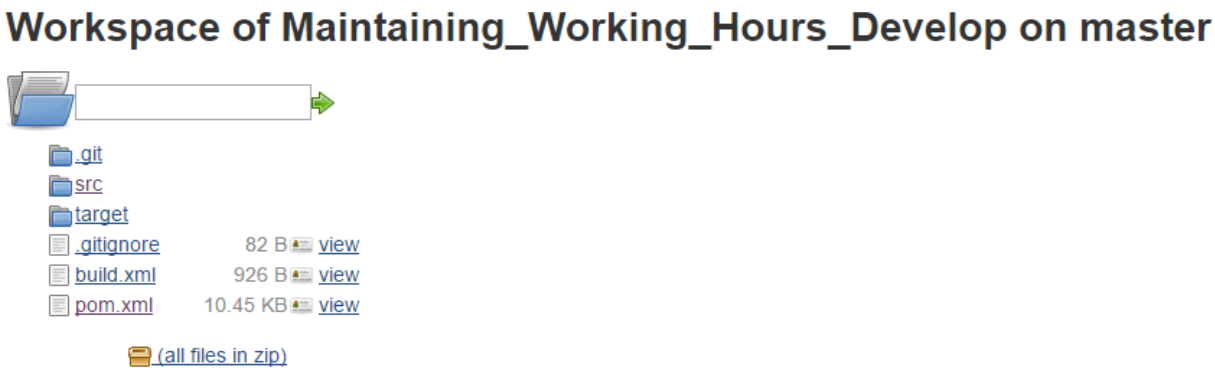
Na slici ispod imamo primjer testova napisanih za jednu metodu. Testovi nam omogućavaju da u budućnosti, bez velikog straha da ćemo poremetiti trenutnu funkcionalnost metode, mijenjamo njenu implementaciju ovisno o našim potrebama. Npr. Ako je implementacija bila poprilično nepregledna i teško razumljiva u budućnosti ćemo je možda poželjeti pojednostavniti a da pritom ne želimo izgubiti njenu funkcionalnost.

```
@Test
public void getWorkingDaysBetweenTwoDates_ResultIsOne() throws Exception {
    int wkDays = DateUtil.getWorkingDaysBetweenTwoDates(DateUtil.convertStringToDate("2017-05-31"),
                                                        DateUtil.convertStringToDate("2017-05-31"));
    assertThat(wkDays, is(1));
}

@Test
public void getWorkingDaysBetweenTwoDates_ResultIsTwo() throws Exception {
    int wkDays = DateUtil.getWorkingDaysBetweenTwoDates(DateUtil.convertStringToDate("2017-05-31"),
                                                        DateUtil.convertStringToDate("2017-06-01"));
    assertThat(wkDays, is(2));
}

@Test
public void getWorkingDaysBetweenTwoDates_ResultIsOne2() throws Exception {
    int wkDays = DateUtil.getWorkingDaysBetweenTwoDates(DateUtil.convertStringToDate("2017-05-01"),
                                                        DateUtil.convertStringToDate("2017-05-02"));
    assertThat(wkDays, is(1));
}

@Test
public void getWorkingDaysBetweenTwoDates_ResultIsTwentyFour() throws Exception {
    int wkDays = DateUtil.getWorkingDaysBetweenTwoDates(DateUtil.convertStringToDate("2017-05-01"),
                                                        DateUtil.convertStringToDate("2017-06-02"));
    assertThat(wkDays, is(24));
}
```

Preostalo nam je još samo testove spremiti lokalno na git repozitorij i na kraju sve dosadašnje promjene spremiti u granu ***develop*** na centralnom repozitoriju.

```
mlucan@DESKTOP-OOGB3MS MINGW64 ~/Desktop/dipl-wk-hours/wkh/working-hours (develo
p)
$ git add .

mlucan@DESKTOP-OOGB3MS MINGW64 ~/Desktop/dipl-wk-hours/wkh/working-hours (develo
p)
$ git commit -m "Adding tests for method getWorkingDaysBetweenTwoDates"
[develop warning: LF will be replaced by CRLF in src/main/java/hr/math/utils/Dat
eUtil.java.

mlucan@DESKTOP-OOGB3MS MINGW64 ~/Desktop/dipl-wk-hours/wkh/working-hours (develo
p)
$ git push origin develop
```

Obzirom da smo sada sve promjene stavili u ***develop*** granu na centralnom repozitoriju koristeći se naredbom ***git push***, Jenkins je započeo izvršavanje job-a kojeg smo u prethodnom odjeljku konfigurirali tako da na svaki novi push on povlači najnovije stanje projekta sa grane ***develop*** te prevodi projekt, pokrene sve testove i zapakira aplikaciju u war datoteku.



Slika 2.22: Jenkins workspace - ovdje se nalazi naš projekt koji je povučen sa GitLaba

Slika 2.23: Jenkins build

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running hr.altima.utils.DateUtilTest
Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.586 sec - in hr.altima.utils.DateUtilTest
Running hr.altima.employee.EmployeeTest
Tests run: 18, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.435 sec - in hr.altima.employee.EmployeeTest

Results :

Tests run: 23, Failures: 0, Errors: 0, Skipped: 0

[INFO]
[INFO] --- maven-war-plugin:2.2:war (default-war) @ working-hours ---
[INFO] Packaging webapp
[INFO] Assembling webapp [working-hours] in [/opt/jenkins/home/jobs/Maintaining_Working_Hours_Develop/workspace/target/working-hours]
[INFO] Processing war project
[INFO] Copying webapp webResources [/opt/jenkins/home/jobs/Maintaining_Working_Hours_Develop/workspace/src/main/webapp] to
[/opt/jenkins/home/jobs/Maintaining_Working_Hours_Develop/workspace/target/working-hours]
[INFO] Copying webapp resources [/opt/jenkins/home/jobs/Maintaining_Working_Hours_Develop/workspace/src/main/webapp]
[INFO] Webapp assembled in [286 msecs]
[INFO] Building war: /opt/jenkins/home/jobs/Maintaining_Working_Hours_Develop/workspace/target/working-hours.war
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 8.215s
[INFO] Finished at: Wed Jun 21 09:30:23 CEST 2017
[INFO] Final Memory: 26M/270M
[INFO] ------------------------------------------------------------------------
Started calculate disk usage of build
Finished Calculation of disk usage of build in 0 seconds
Started calculate disk usage of workspace
Finished Calculation of disk usage of workspace in 0 seconds
Finished: SUCCESS
```

Slika 2.24: Jenkins console output

U nastavku se više nećemo baviti opisivanjem koraka koji su prethodili samom razvoju aplikacije i doveli je do željenog cilja.
Sada ćemo se u novom odjeljku pozabaviti sa mavenom u kombinaciji sa antom. Obzirom da se ovdje radi o web aplikaciji, automatizirani deploy na aplikacijski server bio bi od velike koristi. Tu automatizaciju radit ćemo u zasebnoj grani koju ćemo kasnije kada dobijemo željni rezultat spojiti sa ***develop*** granom.

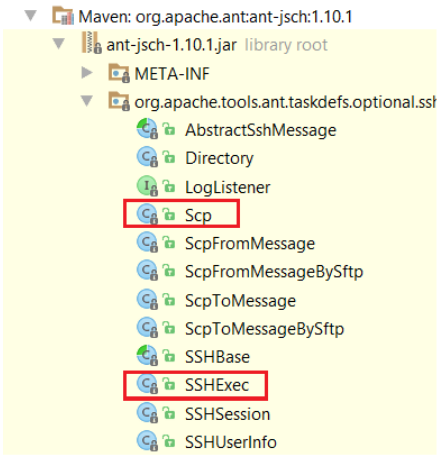
2.4 Automatizirano postavljanje web aplikacije na Tomcat i njeno pokretanje

U ovom djelu napravit ćemo automatizaciju postavljanja aplikacije na Tomcat te njenog pokretanja. Automatizaciju ćemo raditi u zasebnoj grani, nastaloj iz ***develop***-a, a nazvat ćemo je ***maven-ant-deploy-task***.

Prije nego li krenemo dalje, u maven konfiguracijsku datoteku trebamo dodati ovisnost:

```
<dependency>
<groupId>org.apache.ant</groupId>
<artifactId>ant-jsch</artifactId>
<version>1.10.1</version>
</dependency>
```

Ta ovisnost će nam povući artifakt ***ant-jsch*** unutar kojeg imamo *Scp* i *SSHExec* što će nam kasnije trebati.



Kreiranje ant skripte

Napravit ćemo ant skriptu (***build.xml***) koja će otvoriti konekciju prema aplikacijskom poslužitelju na koji želimo postaviti aplikaciju. Unutar te datoteke definirat ćemo ***target*** koji želimo da se izvrši. Unutar ***target-a*** ćemo definirati neke zadatke koje želimo da se obave, npr. želimo kopirati *working-hours.war* u */tomcat/webapps/* direktorij i nakon toga želimo restartati Tomcat kako bi se naša aplikacija uspješno instalirala.

U konačnici ***build.xml*** sadržavat će sljedeće:

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="Deploy and install Application" default="connect-to-lab" basedir="${basedir}">
 <target name="connect-to-lab">
       <taskdef id="1" name="scp"
              classname="org.apache.tools.ant.taskdefs.optional.ssh.Scp">
          <classpath refid="build.classpath.jar"/>
       </taskdef>
      <scp file="${basedir}/target/working-hours.war"
          todir="mlucan@student.math.hr:/student1/mlucan/tomcat/webapps"
          password="pass***" trust="true"/>
      <taskdef id="1" name="sshexec"
             classname="org.apache.tools.ant.taskdefs.optional.ssh.SSHExec">
        <classpath refid="build.classpath.jar"/>
      </taskdef>
     <sshexec host="student.math.hr"
            username="mlucan"
            password="pass***"
            port="22"
            command="cd /tomcat/bin; ./shutdown.sh; ./startup.sh"
           trust="true">
     </sshexec>
 </target>
 <path id="build.classpath.jar">
   <fileset dir="${basedir}/target/working-hours/WEB-INF/lib/">
     <include name="**/*.jar"/>
   </fileset>
 </path>
</project>
```

Datoteku ***build.xml*** spremit ćemo u bazni (root) direktorij projekta.

Maven-antrun-plugin i kreiranje profila

Želimo da se postavljanje aplikacije izvršava nekom maven naredbom ali pritom ne želimo da se to izvršava svaki puta kada npr. izvršimo naredbu ***mvn package***.
Zato ćemo napraviti profil koji će se aktivirati naredbom ***mvn package -P deploy-to-tomcat*** te će izvršiti sve faze do package faze, uključivo i sa package fazom, te na kraju deployat apilkaciju na Tomcat. Ako ne prođe bilo što prije toga (npr. kompajliranje), build će pasti i ništa se neće deployat.

U konfiguracijskoj datoteci ***pom.xml*** dodat ćemo profil:

```
<profiles>
   <profile>
     <id>deploy-to-tomcat</id>
     <build>
        <plugins>
          <plugin>
            <artifactId>maven-antrun-plugin</artifactId>
            <version>1.7</version>
            <executions>
              <execution>
                <phase>package</phase>
                <configuration>
                  <tasks>
                    <ant antfile="${basedir}\build.xml" target="connect-to-lab"/
                        >
                  </tasks>
                </configuration>
                <goals>
                  <goal>run</goal>
                </goals>
              </execution>
            </executions>
          </plugin>
        </plugins>
     </build>
   </profile>
</profiles>
```

Maven AntRun plugin nam omogućava izvršavanje ant zadataka preko maven-a.

Izradu nove grane u kojoj ćemo napraviti automatizaciju za postavljanje aplikacije na aplikacijski server, spremanje tih izmjena u git repozitorij, zatim ponovno vraćanje u ***develop*** granu te spajanje izmjena iz nove grane u ***develop*** vidimo na sljedećem primjeru:

```
mlucan@DESKTOP-OOGB3MS MINGW64 ~/Desktop/mtwkh/maintaining-working-hours (develo
p)
$ git checkout -b maven-ant-deploy-task
M       build.xml
M       pom.xml
Switched to a new branch 'maven-ant-deploy-task'

mlucan@DESKTOP-OOGB3MS MINGW64 ~/Desktop/mtwkh/maintaining-working-hours (maven-
ant-deploy-task)
$ git status
On branch maven-ant-deploy-task
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   build.xml
        modified:   pom.xml

no changes added to commit (use "git add" and/or "git commit -a")

mlucan@DESKTOP-OOGB3MS MINGW64 ~/Desktop/mtwkh/maintaining-working-hours (maven-
ant-deploy-task)
$ git add .

mlucan@DESKTOP-OOGB3MS MINGW64 ~/Desktop/mtwkh/maintaining-working-hours (maven-ant-deploy-task)
$ git commit -m "Maven-ant deploy task"
[maven-ant-deploy-task e0151d3] Maven-ant deploy task
 2 files changed, 13 insertions(+), 8 deletions(-)

mlucan@DESKTOP-OOGB3MS MINGW64 ~/Desktop/mtwkh/maintaining-working-hours (maven-ant-deploy-task)
$ git checkout develop
Switched to branch 'develop'
Your branch is up-to-date with 'origin/develop'.

mlucan@DESKTOP-OOGB3MS MINGW64 ~/Desktop/mtwkh/maintaining-working-hours (develop)
$ git merge maven-ant-deploy-task
Updating 736bdc2..e0151d3
Fast-forward
 build.xml | 17 ++++++++++------
 pom.xml   |  4 ++--
 2 files changed, 13 insertions(+), 8 deletions(-)
```

2.5 Zaključak

U ovom diplomskom radu susreli smo se sa alatima koji nam olakšavaju čitav tok razvoja aplikacije. Smatram da je njihova upotreba od iznimne važnosti, te da ima više prednosti njihovog korištenja nego mana. Ako bi morala navesti barem jednu manu onda bi to bilo utrošeno vrijeme na postavljanje okoline, npr. kreiranje Jenkins job-a, konfiguriranje itd. No, ako uzmemo u obzir činjenicu da nam to u kasnijim fazama razvoja bude od iznimne važnosti, prije svega za pravovremeno otkrivanje grešaka, tada ta mana i više nije mana. S druge strane, prednosti su brojne. Git nam omogućava sudjelovanje više programera na istom projektu, te olakšava izradu više verzija iste aplikacije. Maven nam omogućava dodavanje ovisnosti o potrebnim bibliotekama u maven konfiguracijsku datoteku koja se nalazi u baznom direktoriju projekta, zatim kompajliranje koda, izgradnju softvera i sl. Jenkins nam omogućava da pravovremeno otkrijemo greške u aplikaciji. Primjerice, pripremate se za puštanje zadnjih promjena u ***master***, a nakon što konačno pustite zadnje promjene, verzija aplikacije na ***master-u*** automatski se ***build-a*** te šalje korisniku. Imate rok za isporuku aplikacije a vi ste u žurbi. U cijeloj toj zbrci unesete nenamjerno kompajler greške u ***master*** granu jer niste stigli izvrtiti ni ***mvn compile***. Dogodilo se to da ste neznajući predali korisniku aplikaciju koja nije ispravna i korisnik, naravno, to reklamira a vama to sigurno nije dobra referenca. Da bismo izbjegli takve situacije Jenkins je tu da nam pomogne. U toj situaciji Jenkins će nas obavijestit o tome kako je prošlo kompajliranje, kako su prošli testovi i da li se aplikacija uspješno zapakirala.
Stoga, smatram da korištenje ovih alata uvelike pospješuje učinkovitost svakog razvojnog programera.

Bibliografija

[1] Krajina, Tomo: *Uvod u git*. `https://github.com/tkrajina/uvod-u-git`.

[2] Sierra, Kathy i Bert Bates: *Head First Java*. O'Reilly Media, 2005.

[3] Smart, John Ferguson: *Jenkins - The Definitive Guide*. O'Reilly Media,, 2011.

[4] Sonytype: *Maven-The Definitive Guide*. O'Reilly Media, February 2009.

[5] Walls, Craig: *Spring in Action*. Manning, 2014.

[]

Sažetak

U ovom diplomskom radu upoznajemo se sa Java ekosustavom.
Rad je podijeljen u dva glavna poglavlja. U prvom poglavlju ukratko opisujemo alate koji služe softverskim inženjerima pri razvoju aplikacije. Podijeljen je u nekoliko potpoglavlja:

- *Git Version Control* - alat za upravljanje promjenama, olakšava sudjelovanje više programera na projektu istovremeno
- *Maven* - automatizira izgradnju softvera
- *Ant* - alat za izgradnju softvera
- *Kontinuirana integracija* - *Jenkins* - javno dostupan automatizacijski server
- *IntellJ IDEA* - integrirano razvojno okruženje za razvoj softvera
- *Tomcat* - aplikacijski poslužitelj razvijen od Apache-a
- *Jar/War datoteke* - komprimirane datoteke
- *Jira* - softver za praćenje novih zadataka i grešaka
- *Spring Mvc okvir* - okvir koji pruža gotove komponente za razvoj Web aplikacija

U drugom poglavlju opisujemo jedan tijek razvoja aplikacije u kojem prolazimo kroz sve alate koje smo opisali u prvom poglavlju, te na kraju dajemo neki sud o svim tim alatima, tj. ističemo njihove mane i vrline.

Summary

In this thesis we are talking about Java ecosystem. The thesis is divided into two chapters. In the first chapter we are describing tools which make developers easier to develop applications. The first chapter consists of several subsections:

- *Git Version Control* - version control system for tracking changes in computer files and coordinating work on those files among multiple people
- *Maven* - tool that provides developers a complete build lifecycle framework
- *Ant* - java based build tool from Apache Software Foundation
- *Continuous Integration - Jenkins* - popular tool for performing continuous integration of software projects
- *IntellJ IDEA* - java integrated development environment for developing computer software
- *Tomcat* - application server
- *Jar/War files* - compressed files
- *Jira* - software for tracking new tasks and bugs
- *Spring Mvc framework* - provides ready components that can be used to develop flexible and loosely coupled web applications

In the second part of thesis we show how to use these tools from the first chapter when we develop web application.

Životopis

Rođena sam 06.05.1992. u Zagrebu. Jedna sam od troje djece u obitelji.
Osnovnu školu Jurja Habdelića upisala sam 1999. godine. Prva četiri razreda pohađala sam u područnoj školi u Šiljakovini, a ostala četiri u matičnoj školi u Velikoj Gorici. Nakon osnovne škole upisala sam V.gimnaziju, matematičko-informatički smijer.
Nakon završetka srednje škole, 2011.godine, upisala sam preddiplomski studij matematike na Prirodoslovnom matematičkom fakultetu u Zagrebu. Tokom studija imala sam priliku raditi sa studentima kroz demonstrature koje sam dražala iz predmeta Linearna algebra 1 i 2. Također, u svoje slobodno vrijeme, imala sam prilike raditi i sa drugom djecom kroz držanje instrukcija iz matematike. Jedan kraći period, kao student, radila sam za Profil prepravke na udžbenicima iz matematike od 1. do 4. srednje te u sklopu toga riješavala i zadatke, kako bi učenici imali gdje provijeriti svoja rješenja.
U kasnijim fazama studija počela sam se sve više interesirati za računarstvo.
Prošle godine (2016.) sredinom 7.mjeseca dobila sam i svoju prvu priliku da kao student radim u jednoj zaista dobroj firmi - Altimi, te tako započnem svoj profesionalni razvoj u struci. U trećem mjesecu ove godine (2017.) sam u istoj firimi započela i svoj prvi radni odnos.