

Miskolci Egyetem
Informatikai Intézet
Általános Informatika Tanszék



Automatizált Tesztelési Rendszer / Környezet kialakítása az ME IIS-en

SZAKDOLGOZAT

Készítette:

Tóth Ádám
D3F1KK

Konzulens:

Dr. Tóth ZSOLT

2018.

Declaration of Authorship

I hereby certify that this essay has been composed by me and is based on my own work, unless stated otherwise. No other person's work has been used without due acknowledgement in this essay. All references and verbatim extracts have been quoted, and all sources of information, including graphs, have been specifically acknowledged.

Miskolc, 2018. május 1.

Tóth Ádám

Tartalomjegyzék

Declaration of Authorship	1
1. Bevezetés	3
1.1. Motiváció	3
1.2. Célkitűzés	4
2. Technológiák	6
2.1. Elméleti Alapok	6
2.2. Technológiák	6
2.2.1. Maven	6
2.2.2. Java, JUnit	7
2.2.3. MySQL	7
2.2.4. Virtualizáció, Hálózat	8
2.2.5. Jenkins	8
2.2.6. Nexus	8
3. Tervezés	9
3.1. Honnan hova jutunk	9
3.2. Hogyan?	9
3.3. Formális leírás	9
3.4. Lépések (logikailag)	9
3.4.1. Szerverek	9
4. Implementáció	11
4.1. Szerverek	11
5. Tesztelés	13
6. Összefoglalás	14

1. fejezet

Bevezetés

A szoftverfejlesztés a mai világunk egyik alapköve. Az informatika világa és a mi hétköznapijaink sem lennének képesek fejlődni a szorgos programozók munkája nélkül. Azonban a szoftver fejlesztés nem egy egyszerű dolog, főleg ha egy komplex programozási feladatra van szükség. Ilyenkor szinte biztosan nem egy ember fogja elkészíteni ezeket a programokat, így a munka érdemi része kevesebb feladatot ró a fejlesztőre viszont számos egyéb feladattal kell szembe néznie.

Ezek a feladatok (ha jól elő vannak készítve) nagyon könnyen vehető akadályok. A csapatban való munka fontos részét alkotja a modern szoftverfejlesztésnek.

A csapatmunkát komoly támogatási háttérrel szükséges ellátni. Gondok itt arra, hogy lehetőséget kell adni minden fejlesztő számára, hogy a fejlesztés állapotát nyomon tudják követni és a mások munkáját is könnyen feltudják használni vagy esetenként módosítani tudják azt. Fontos az is, hogy

Szoftver fejlesztés csapat munka - ugyan olyan formátumú kód - kód felhasználás - dokumentáció ennek a támogatása maga a CI fontossága

1.1. Motiváció

A Miskolci Egyetem Mérnökinformatikai szakán eltöltött éveim során próbáltam a lehető legtöbb olyan sávot választani a tantervi hálóból ami nem fejlesztéssel hanem üzemeltetéssel foglalkozik. A fejlesztés sem egy teljesen unalmas ága az informatikának, viszont számomra az üzemeltetés egy sokkal változatosabb világba varázsol el. A jövő is a nagy számítási teljesítmények (Big Data, Deep Learning) felé mutat amelyet nem titkolt célom még a technológia hajnalán elsajátítani. Úgy gondolom, hogy a mai fejlesztési feladatnak még nagyobb szüksége van a stabil, jólműködő és állandóan rendelkezésre álló supportra. Ezért is ragadtam meg a lehetőséget mikor Dr. Tóth Zsolt konzulensem felajánlott egy projekt munkát a Miskolci Egyetem Informatika Intézetében.

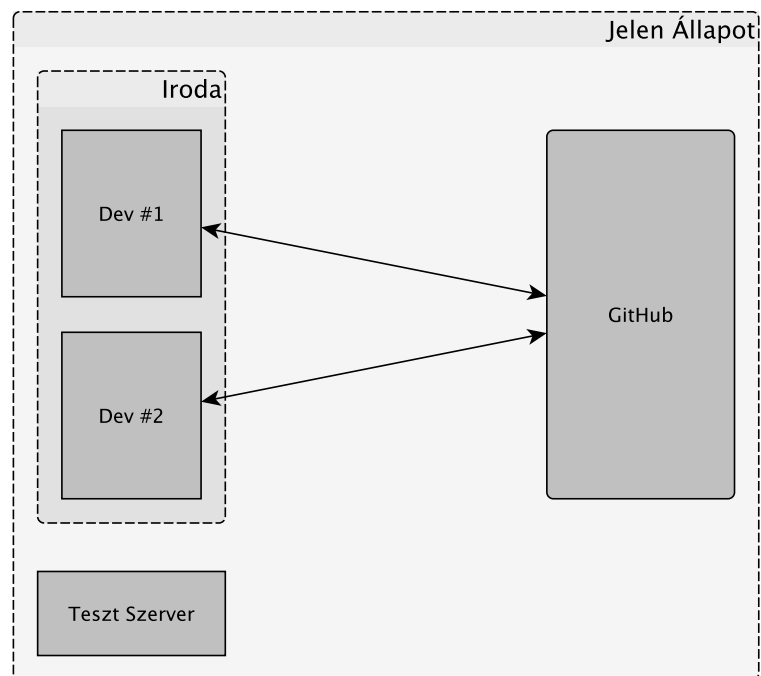
— KÉRDÉS HOGY MIT KELLENE MÉG IDE ÍRNI —

2. Amiért nekem fontos 1. Amiért a téma fontos

Ez a projekt az ILONA beltéri helymeghatározó rendszernek a fejlesztése során szükségessé vált automatizált tesztelési rendszer és környezet kialakítása volt. Mikor elkezdtem foglalkozni a témával még jobban megtetszett az üzemeltetés és a support világa és ekkor gondoltam úgy, hogy ebből szakdolgozatot fogok írni. Ezzel a dolgozattal is rengeteg újat tanultam amit az egyetemi tanórák keretein belül soha sem tanultam volna meg.

1.2. Célkitűzés

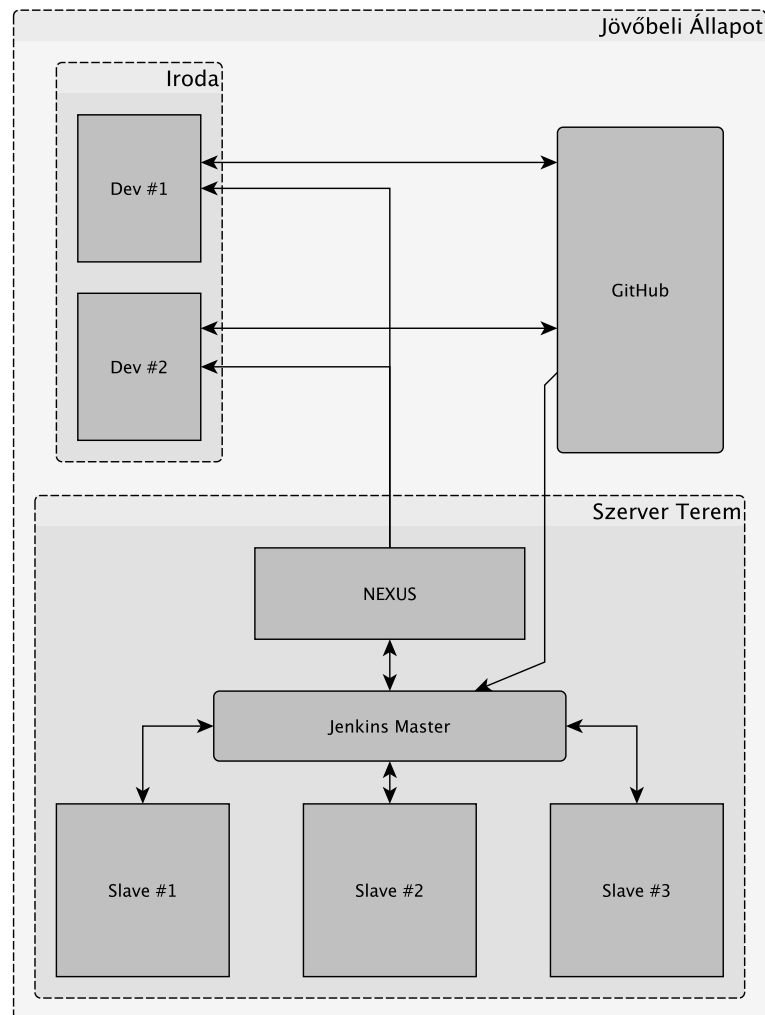
A szakdolgozat elkészítése során a célom azt volt, hogy a Miskolci Egyetem Általános Informatika Tanszékén az ILONA rendszeren dolgozó fejlesztőknek a munkáját segítsem. A fejlesztők jelenleg az építést és a tesztelést is manuálisan valósítják meg, ahogy ez a 1.1 ábrán is látható. Ez a feladatom kiindulú állapota.



1.1. ábra. Jelen Állapot

Az új rendszernek a jelenlegi állapotnak több sajátosságát meg szerettem volna tartani és ezeken felül a már meglévő szerverekkel megvalósítani a rendszer működését. Fontosnak tartottam hogy a fejlesztők kizárólag a feladataik csökkenésével szembesüljenek és ne keljen új programokat, folyamatokat megtanulniuk Kvázi ne látszódjon a háttérbe folyó változás és a jelenlegi rendszer szolgáltatás kiesésével se járjon az új rendszer bevezetése. Mivel az ILONA fejlesztése teljesen önerőből, támogatás nélkül valósul meg, ezért törekedtem arra hogy csak ingyenes, opensource alkalmazásokat használjak a feladathoz. ————— A jelenlegi rendszer alapja az egyik legnépszerűbb ingyenes internetes verziókövető rendszer a GitHub. A GitHub-ot feltétlen meg akartam tartani az új rendszerben mivel egy elterjedten használt és kedvelt környezetnek tartom és ami még mellette szól az az ingyenessége. Ez a teszt szerver Maven segítségével építi és teszteli a megírt kódokat, ezért a következő amit a meglévő rendszerből át akartam vinni az újba az a Maven. A jelenlegi rendszer nem hatékony a munkavégzés során, mivel számos munkaidőt emészt fel a folyamat amely nem a fejlesztéssel kapcsolatos. —————

A fejlesztőknek az új rendszerben lényegében csak a GitHubbal és a kész JAR fájlokat tartalmazó repository szerverrel kell dolgozniuk a többi folyamatot automatikusan látná el a rendszer. A cél tehát egy olyan Automatikus Tesztelési Környezet megvalósítása, amely képes a GitHubról letölteni a kódokat akkor ha a kódban változás történik. Ezen felül ha a tesztek és a buildelési folyamat is sikeresen végbement az elkészült fájlokat könnyen és szervezetten elérhetővé tennie egy repository szerveren. Ez a tervezett folyamat látható a [1.2 ábrán](#).



1.2. ábra. Jövő Állapot

— IDE MÉG KELL ÍRNI —

2. fejezet

Technológiák

2.1. Elméletei Alapok

Az új kiépített rendszernek olcsónak és üzembiztosnak kell lennie. Ezért törekszek a projekt alatt minél több Open Source technológiát alkalmazni. — KÉRDÉS HOGY MIT KELLENE MÉG IDE ÍRNI —

2.2. Technológiák

A projekthez felhasználandó technológiák már mind meglévő és gyakran használt eszközök. Az egész rendszer alapja a Java.

2.2.1. Maven

Az Apache Maven rövid nevén csak Maven-t szoftverfejlesztés során használjuk a projektek menedzselésére. Az Apache Ant utódja, sok tekintetben hasonlít a két szoftver, azonban az Ant lassabb és elavultabb mint a Maven. A Mavenrel az építési- és tesztelési folyamatok automatizálhatóak, így a Jenkinsel tökéletesen együtt tud dolgozni. A Maven egy új fogalmat is bevezet, ez az úgynevezett Projekt Objektummodell (angolul: Project Object Model) röviden a POM. A POM egy XML fájl amely egy építésre kész projektet és annak függőségeit írja le. Ezzel a pom.xml fájljal adjuk meg a Mavennek hogy mit is csináljon a kóddal. Előre megadott célokat tartalmaz mint a kód fordítása és csomagolása, azonban itt van lehetősége a fejlesztőknek saját célokat és teszteket létrehozni.

A Maven 2002-ben Jason van Zyl készítette el. A projektet az Apache Software Foundation fejleszti, korábban a cégnél a Jakarta Projekt részeként működött. Jelenleg a 3.3.9-es a legfrissebb verziója.

—IDÉZERT— A Maven hálózatképes, tehát szükség esetén dinamikusan is le tud tölteni komponenseket. Repository névvel illetik a különböző hosztok fájlrendszereinek azonos mappáit, ahol a letölthető komponensek találhatóak. A Maven nem csak a repository-kból való letöltést támogatja, hanem a készült szoftvercsomag feltöltését is. Ezzel az automatizálható le- és feltöltési mechanizmussal a Maven de facto szabványt próbál teremteni, de elég lassan fogadja el a Java közösség.

A Maven plugin alapú architektúrája lehetővé teszi tetszőleges parancssorból vezérelhető alkalmazás használatát. Ez elméletileg lehetővé teszi tetszőleges programnyelvekhez való pluginek készítését, de a gyakorlatban minimális mennyiségű nem javás plugin készült.

—IDÉZERT—

2.2.2. Java, JUnit

Java

—IDÉZERT— A Java általános célú, objektumorientált programozási nyelv, amelyet a Sun Microsystems fejlesztett a '90-es évek elejétől kezdve egészen 2009-ig, amikor a céget felvásárolta az Oracle. 2011-ben a Java 1.7-es verzióját az új tulajdonos gondozásában adták ki.

A Java alkalmazásokat jellemzően bájt kód formátumra alakítják, de közvetlenül natív (gépi) kód is készíthető Java forráskódból. A bájt kód futtatása a Java virtuális géppel történik, ami vagy interpretálja a bájt kódot, vagy natív gépi kódot készít belőle, és azt futtatja az adott operációs rendszeren. Létezik közvetlenül Java bájt kódot futtató hardver is, az úgynevezett Java processzor.

A Java nyelv a szintaxisát főleg a C és a C++ nyelvektől örökölte, viszont sokkal egyszerűbb objektummodellel rendelkezik, mint a C++. A JavaScript szintaxisa és neve hasonló ugyan a Java-éhoz, de a két nyelv nem áll olyan szoros rokonságban, mint azt ezekből a hasonlóságokból gondolhatnánk.

Bár a nyelv neve kezdetben Oak (tölgyfa) volt, (James Gosling, a nyelv atyja nevezte így az irodája előtt növe tölgyfáról), később kiderült, hogy ilyen elnevezésű nyelv már létezik, ezért végül Java néven vált ismertté. A Java szó a Oracle védjegye. Ennélfogva engedélye nélkül nem használható mások által kifejlesztett termékek megjelölésére; még például Java-szerű ... stb. összetételben sem, mert ez a védjegyjogosult jogaiba ütközik. —IDÉZERT—

JUnit

—IDÉZERT— JUnit egy egységteszt keretrendszer Java programozási nyelvhez. A teszt vezérelt fejlesztés (TDD) szabályai szerint ez annyit tesz, hogy a kód írásával párhuzamosan fejlesztjük a kódot tesztelő osztályokat is (ezek az egység tesztek). Ezeken egységtesztek karbantartására, csoportos futtatására szolgál ez a keretrendszer. A JUnit teszteket gyakran a build folyamat részeként szokták beépíteni. Pl. napi build-ek esetén ezek a tesztek is lefutnak. A release akkor hibátlan, ha az összes teszt hibátlanul lefut.

A JUnit a egységteszt keretrendszerek családjába tartozik, melyet összességében xUnit-nak hívunk, amely eredeztethető a SUnitből.

JUnit keretrendszer fizikailag egy JAR fájlba van csomagolva. A keretrendszer osztályai következő csomag alatt található:

JUnit 3.8-as ill. korábbi verzióiban a junit.framework alatt található JUnit 4-es ill. későbbi verzióiban org.junit alatt található —IDÉZERT—

Az ILONA rendszer a fejlesztők Java nyelven fejlesztik, ezért az új rendszernek képesnek kellett lennie a Java használatára. Emellett maga a Jenkins CI is egy Java nyelven íródott webalkalmazás ami Java VM-en fut. Így a Safranek virtuális gépre egy legfrissebb Java

2.2.3. MySQL

A MySQL egy többfelhasználós, többszálú, SQL-alapú relációs adatbázis-kezelő szerver. A MySQL az egyik legelterjedtebb adatbázis-kezelő, aminek egyik oka lehet, hogy a teljesen nyílt forráskódú LAMP (Linux–Apache–MySQL–PHP) összeállítás részeként költséghatékony és egyszerűen beállítható megoldást ad dinamikus webhelyek szolgáltatására.

A tesztrendszer számára egy elengedhetetlen kiegészítőt nyújt olyan tesztelési feladatoknál mikor a program adatbázist használ. Teszteléskor a teszt folyamat feltölti az adatbázist a teszteléshez szükséges adatokkal majd a teszt lefutása (akár sikeres, akár sikertelen) után üríti az adatbázist.

2.2.4. Virtualizáció, Hálózat

2.2.5. Jenkins

A Jenkins egy nyílt forráskódú, Java nyelven írott eszköz amely folyamatos integrációs szolgáltatást nyújt szoftverfejlesztéshez. Az Oracle Hudson folyamatos integrációs eszközének eredeti fejlesztő csapata vált ki a Hudson fejlesztéséből és valósították meg a Jenkinst. Mivel az eredeti fejlesztőcsapat jelenleg is a Jenkinst fejleszti ezért érdemes (ha a kettő közül kell választanunk) a Jenkinst választani. A Jenkins mellet szóló érv még hogy több plugin érhető el hozzá mint a Hudsonhoz, így több feladatot is egyszerűbben valósítható meg. Ezt az eszköz közvetlenül telepíthetjük a rendszerünkre, viszont választhatjuk hogy egy szervlet konténerben fusson, mint pl. az Apache Tomcat. Támogat számos SCM eszközt mint például a Git-et amely az ILONA projekt számára elengedhetetlen. Ezek mellett az Apache Ant és Apache Maven parancsait is végre tudja hajtani, így a Maven továbbra is lehetséges használni az új rendszerben. A Jenkins elsődleges fejlesztője Kohsuke Kawaguchi. A Jenkinst MIT licenc alatt adják ki és szabad szoftver.

Folyamatos Integráció

—IDÉZET— A folyamatos integráció egy olyan fejlesztési folyamat, ami hatékonyabbá teszi a csapatmunkában történő fejlesztést azáltal, hogy a fejlesztők gyakran (legalább naponként) és már a kezdetektől összeillesztik (integrálják) a kódjukat. A Continuous Integration arra a felismerésre épít, hogy a fejlesztés során a kódok integrációja a legproblematisabb fázis és hogyha ezt korán lekezeljük akkor az esetleges problémák is gyorsabban javíthatóak lesznek! —IDÉZET—

Maga a Jenkins CI látja egy a különböző elemek közti kapcsolatot, a GutHub hookjaitól egészen a Maven építési és tesztelési folyamat visszajelzéséig.

2.2.6. Nexus

—IDE MÉG NEM TALÁLTAM SEMMIT, DE MAJD TALÁLOK KI—

3. fejezet

Tervezés

Lemásolni a miskolci cégek jelenleg használt technológiák fejlesztési környezetek labor környezetben való kiépítése.

3.1. Honnan hova jutunk

Az eredeti állapot alapjait amelyet a [1.1](#) ábrán láthatunk, teljesen megtartásra kerülne. Ez a GitHub mint verziókövető rendszer és a Maven mint tesztelésre használt program. Minden szempontot figyelembe véve a [1.2](#) ábrán látható rendszer megvalósítása a

3.2. Hogyan?

3.3. Formális leírás

Táblázat milyen gép, gépek igényei és elérhetőségei
migrálható komponensek

3.4. Lépések (logikailag)

3.4.1. Szerverek

A rendszer megépítésekor már meglévő eszközök megtartásra kerülnek. Ezért az első feladat a Safranek virtuális szervergép frissítése és a szükséges függőségek telepítése. Ezen felül mivel a Jenkinst egy Tomcat fogja futtatni, ezért a Tomcat legújabb verziójának a telepítését kell megoldanunk.

- JENKINS SZERVER—
- JENKINS SZERVER ÉS A SLAVEK—
- JENKINS SZERVER ÉS A GITHUB—
- GITHUB ÉS A FEJLESZTŐK—
- JENKINS SZERVER ÉS A NEXUS SZERVER—
- NEXUS SZERVER ÉS A FEJLESZTŐK—
- EGYBE A MŰKÖSÉS—
- JENKINS SZERVER—

A fő Jenkins szervernek nincs nagy erőforrás igénye, viszont nagy tárhelykapacitással kell rendelkeznie és könnyen elérhetőnek kell lennie. Ezért a tanszéken meglévő safranek nevű virtuális szerverre esett a választásom mivel ez a szerver nem csak a belsőhálózatról

elérhető és dinamikusan növelhető tárhely kapacitással rendelkezik. Ennek a szervernek lényegében csak egy TomCat-ot és magát a Jenkins kell futtatnia számítások elvégzése nélkül.

—JENKINS SZERVER ÉS A SLAVEK—

Fontos volt a projekt megvalósítása során, hogy a rendszer könnyen bővíthető legyen és több platformra is képes legyen buildelni és tesztelni. Ezért Master-Slave rendszerrel terveztem megvalósítani a Jenkins CI-t. Ugyanis egy Jenkins Master szerver is képes több platformra tesztelni és buildelni, ha legalább egy Slave szerveren megtalálhatóak az adott platformhoz szükséges csomagok és függőségek. Első sorban azokat az igényeket kellett felmérni a jelenlegi és jövőbeli feladatok problémamentes kiszolgálása érdekében az indulásnál már meg kell lenniük—LEHET EZ IS HÜLYESÉG—.

—JENKINS SZERVER ÉS A GITHUB—

A következő lényeges feladat az automatikus GitHub pull-olás beállítása volt. Itt a lehető legnagyobb biztonság elérése és az automatizáltság miatt ssh kulcsos azonosítást szerettem volna megvalósítani. Így a fejlesztőknek a kódoláson kívül csak a GitHub-ra való szinkronizálás lenne a feladatuk.

4. fejezet

Implementáció

4.1. Szerverek

Safranek

A feladatok közül a legelső a már meglévő Safranek szerver frissítése és konfigurálása volt. A Safraneken egy Ubuntu server 16.04-es verzió futott, így nem volt szükség sem az operációs rendszer, sem pedig a disztribúció cseréjére, mivel a projektet az Ubuntu szerver ezen verziója tökéletesen kiszolgálja. A frissítéseket követően, a Jenkins CI futtatását kellett előkészíteni. Egy jenkins nevű felhasználót kellett létrehozni, hogy szeparáltan lehessen futtatni a programokat és az autentikáció során ne a root felhasználó adatait keljen felhasználni.

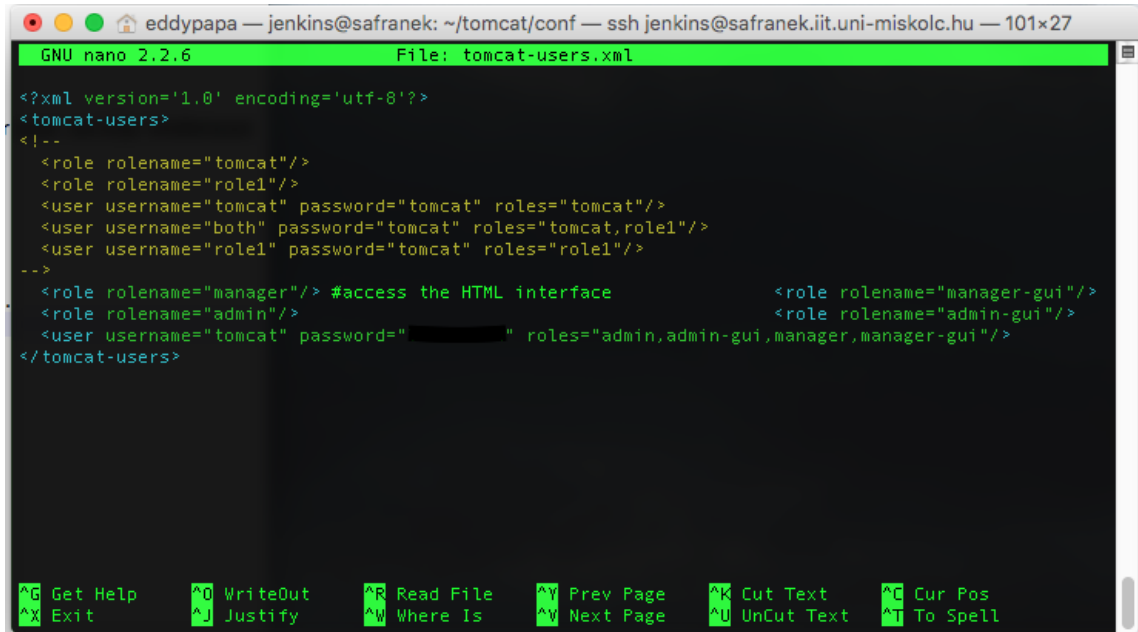
Mivel a TomCat és a Jenkins is egy-egy java alapú alkalmazás ezért a Java JRE és JDK legfrissebb verzióját kellett felkellet telepíteni. Azt a következő két parancs alkalmazásával: —"sudo apt-get install default-jre && sudo apt-get install default-jdk"——

A Jenkins CI-t szervletként futtatható verziójához telepíteni kellett az Apache TomCat-ot. A TomCat 8.5.8-as verziója volt a legfrissebb, így ezt a verzió került fel a Safranekre. A jenkins felhasználó "home" könyvtárában került elhelyezésre egy "tomcat" nevű mappába. Mielőtt a jenkins felhasználóval elindult volna a TomCat a " /tomcat/conf/tomcat-users.xml" fájlt a 4.1 ábrán láthatóan kellett módosítani, hogy a webes menedzment elérhető legyen. Ezt a Nano szövegszerkesztő segítségével lehetett elvégezni.

Következtetett a Jenkins CI telepítése. A Jenkins CI legfrissebb verziója a 2.32.2 volt. A letöltött "jenkins.war" fájlt a " /tomcat/webapps mappába kellett letölteni. A letöltés befejeztével elindíthatóvá vált a TomCat. Ezt a " /tomcat/conf" mappában található "startup.sh" scripttel kellett megtenni. A <http://safranek.iit.uni-miskolc.hu:8080/jenkins> webcímen elérhetővé is vált a Jenkins CI.

Az első felhasználó létrehozásához és a komponensek telepítése előtt egy a ".jenkins/secrets/initialAdmin" fájlban található jelszót kell bemásolni a formba a folytatáshoz. Miután ez megtörtént a Jenkins CI admin felülete tárul elének 4.2.

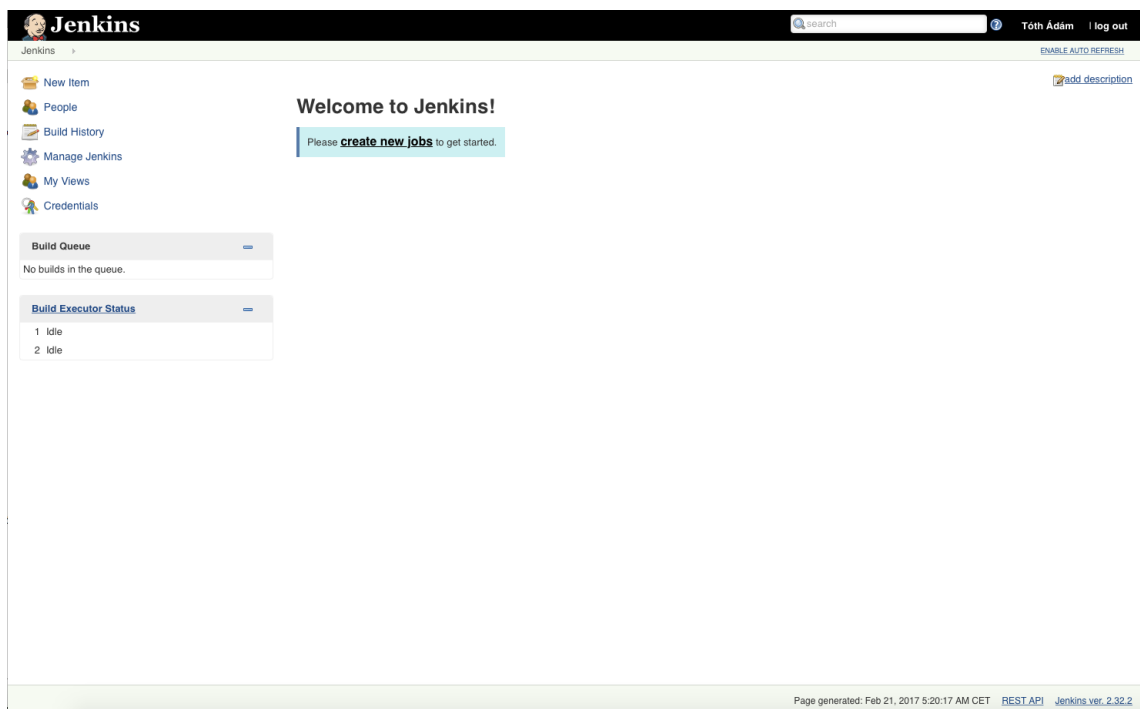
Jenkins A Jenkin CI



```
GNU nano 2.2.6 File: tomcat-users.xml

<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <!--
    <role rolename="tomcat"/>
    <role rolename="role1"/>
    <user username="tomcat" password="tomcat" roles="tomcat"/>
    <user username="both" password="tomcat" roles="tomcat,role1"/>
    <user username="role1" password="tomcat" roles="role1"/>
  -->
  <role rolename="manager"/> #access the HTML interface      <role rolename="manager-gui"/>
  <role rolename="admin"/>                                     <role rolename="admin-gui"/>
  <user username="tomcat" password="tomcat" roles="admin,admin-gui,manager,manager-gui"/>
</tomcat-users>
```

4.1. ábra. TomCat felhasználó beállítása



4.2. ábra. Első belépés a Jenkins CI felületére

5. fejezet

Tesztelés

6. fejezet

Összefoglalás

asdf [\[1\]](#)

Irodalomjegyzék

- [1] Alan Berg. *Jenkins Continuous Integration Cookbook*. Packt Publishing Ltd, 2012.