Desenvolupament d'aplicacions professionals

Qualitat de codi i integració continua

Memòria de Projecte Final de Màster

Máster en Enginyeria Informàtica

Àrea d'especialitat/Itinerari

Desenvolupament d'aplicacions web

Autor: *Juan Carlos Martín Capitán*

Consultor: *Ignasi Lorente Puchades*

11/01/2015

Abstracte (versió en català)

Les necessitats actuals en el desenvolupament de software han fet necessari que es desenvolupin tècniques per agilitzar el desenvolupament i minimitzar els problemes resultants. Per solucionar aquestes necessitats va sorgir el concepte de qualitat de codi, i la necessitat de crear un procés que ajudes en el desenvolupament de grans projectes software, anomenat integració continua.

Aquest document explica els diferents conceptes que intervenen tant a la integració continua com en la qualitat de codi i exposa, pas per pas, com implementar una solució d'integració continua, i com s'aplica en un projecte real. Veurem que el desplegament d'un servei d'integració continua no és fàcil, però que a mig termini, podem veure que ens agilitza enormement el desenvolupament del nostre projecte.

Paraules clau: *Integració continua, qualitat de codi, Java, aplicació web, desenvolupament àgil*.

Abstracto (versión en castellano)

Las necesidades actuales en el desarrollo de software han hecho necesario que se desarrollen técnicas para agilizar el desarrollo y minimizar los problemas resultantes. Para solucionar estas necesidades surgió el concepto de calidad de código, y la necesidad de crear un proceso que ayudase en el desarrollo de grandes proyectos software, llamado integración continúa.

Éste documento explica los diferentes conceptos que intervienen tanto en la integración continúa como en la calidad de código y expone, paso a paso, cómo implementar una solución de integración continúa, y cómo se aplica en un proyecto real. Veremos que el despliegue de un servicio de integración continúa no es fácil, pero que a medio plazo, podemos ver que nos agiliza enormemente el desarrollo de nuestro proyecto.

Palabras clave: *Integración continua, calidad de código, Java, aplicación web, desarrollo ágil*.

Abstract (English version)

Current needs in software development have made it necessary to develop techniques to speed development and reduce the resulting problems. To address these needs, the concept of quality code and the need to create a process that aids in the development of large software projects, called integration continues, appears.

This document explains the different concepts involved both the integration continues as the quality of code and explains step by step how to implement continuous integration solution, as applied in a real project. We will see that the deployment of an integration system is still not easy, but in the medium term, we can see that greatly speeds up in the development of our project.

Keywords: *continuous Integration, quality code, Java, web application, agile software development*.

Índex

Figures i taules

Índex d'imatges

Índex de taules

1. Introducció al treball de final de màster

> *"There are no secrets to success. It is the result of preparation, hard work, and learning from failure."*
>
> **Colin Powell**

Avui dia, el desenvolupament d'aplicacions software s'ha convertit en grans projectes d'enginyeria, els quals requereixen gran quantitat de temps i personal de diferents àrees. A més, l'impacte que tenen errors sobre aquests poden arribar als milions d'euros.

Degut a això, constantment s'han intentat crear i millorar els sistemes i processos per el desenvolupament software. En un principi, es van intentar aplicar els mateixos processos que en el desenvolupament de grans construccions. Ja que no sempre es podien aplicar, degut per exemple a la indeterminació de requisits, processos més àgils es van anar aplicant. Però els nous processos també oferien problemes, desplegaments continus per rebrà el feedback del client i fer software incremental (anar introduint funcionalitats de forma gradual) són uns d'ells.

Per exemple, a l'introduir noves funcionalitats és molt segur que es tingui que modificar codi antic. Es podrà entendre el codi antic? Com es pot estar segur que una vegada introduïda la nova funcionalitat, altres no deixaran de funcionar? I com es pot estar segur que la següent vegada que calgui tocar aquest codi continuaràs recordant que fa? O pitjor, si una tercera persona l'ha de tocar, l'entendrà?

Per solucionar els problemes que ocasionen els grans productes software respecta a la lectura i enteniment de codi, el control d'error, etc., hem de tenir cura que el nostra codi sigui de qualitat i segueixi unes certes normes. Es important que el nostre codi segueixi les normes bàsiques que tenen tots els llenguatges de programació per millorar la lectura i comprensió del codi escrit. També es molt important seguir un estricte seguiment dels tests i de la seva cobertura, ja que ofereix una gran informació sobre si el nostre programa te errors o no. Aquests conceptes es recullen en el que s'anomena la **Qualitat de codi** i ens indica si el nostra codi és de qualitat o no.

Quan parlem de desplegaments continus trobem que aquesta tasca, juntament amb altres que s'han de fer contínuament en el desenvolupament de software, poden ser molt complexes i necessitar la dedicació de molt de temps. Que podem fer per reduir la pèrdua de temps en aquestes tasques repetitives?

La solució la trobem en automatitzar aquestes tasques. D'aquí sorgeix el concepte d'**Integració continua** que ens permetrà crear i configurar tasques per automatitzar gran quantitat de processos repetitius en el desenvolupament software, des del control de qualitat, fins al desplegament a producció.

En aquest PFM explicarem amb detall aquests dos importants conceptes dels desenvolupaments software i els posarem en practica.

1.1. Context i justificació

Després d'estar un temps amb el desenvolupament web de manera tradicional, he trobat grans problemes que s'han anat solucionant de manera rudimentària. Des de fa relativament poc temps la implantació d'un sistema d'integració continua, en l'empresa on actualment treballo, m'ha obert un nou mon de possibilitats per automatitzar tasques que abans eren molt farragoses.

El control de qualitat de codi automàticament, desplegaments a servidors externs i empaquetat de codi són algunes de les funcionalitats descobertes fins ara, però només descobertes, l'afany de aprofundir en aquestes funcionalitats i veure què més es pot fer, és el que m'ha portat a pensar en aquests projecte de final de màster.

Una vegada apresos els conceptes teòric i dissenyat i implantat un sistema d'integració continua, es desenvoluparà una l'aplicació. Una aplicació que ha sorgit de la complicada decisió de seleccionar un àrea per aquests propi TFM sense estar del tot segur de quin projecte es volia fer.

1.2. Descripció

Durant els següents capítols aprendrem com els dos conceptes de qualitat de codi i integració continua ens poden ajudar a millorar el desenvolupament de software. Això no només o farem de forma teòrica, sinó, que després es dissenyarà, implantarà i configurarà un sistema d'integració continua que també ens proporcioni dades sobre la qualitat de codi dels nostres desenvolupaments.

Per posar a prova el nostra sistema, es desenvoluparà una aplicació web que utilitzi el sistema d'integració continua. L'aplicació web que es desenvoluparà serà una senzilla aplicació on professors i alumnes podran accedir, prèvia obtenció d'unes credencials per part de l'administrador, i entrar amb el seu *login* i *password.* Els professors podran generar projectes i els alumnes podran seleccionar el projecte que vulguin. Tot això tenint en compte que ha de tenir una usabilitat i funcionalitat correcte.

L'aplicació, tot i ser senzilla i de poc abast, conté molt elements bàsics de qualsevol aplicació web: Registre d'usuaris, *login* d'usuaris, administració, base de dades, etc. També dir, que l'abast s'adapta al temps establert, ja que s'ha de tenir en compte la investigació i el desplegament sobre el sistema d'integració continua que s'ha de desplegar i utilitzar.

1.3. Objectius

Els objectius que composen aquests treball de final de màster són els següents:

- Comprendre els conceptes de qualitat de codi i integració continua, els seus components i les seves funcionalitats.
- Saber configurar i desplegar, des de cero, un sistema bàsic d'integració continua i control de qualitat de codi.

- Saber configurar un projecte d'una aplicació web, per que utilitzi la solució d'integració continua desplegada, amb la qual, s'obtingui informació sobre la qualitat de codi i es realitzin desplegament en un servidor.

1.4. Metodologia

La metodologia a seguir en aquests projecte de final de màster varia una mica amb altres projectes de l'àrea d'aplicacions web. Això es degut a que cal una part prèvia d'investigació per comprendre els sistemes d'integració continua, les seves comunicacions entre components i com es configuren, així com saber quines eines tenim al mercat. També hem de tenir en compte que abans de poder començar amb el desenvolupament de l'aplicació hem de fer el desplegament i configuració del nostre sistema d'integració continua. Per aquests motiu dividirem en nostre projecte en 3 grans blocs.

A la primera part es farà una investigació per aprofundir i comprendre els conceptes de qualitat de codi i integració continua. En aquesta fase es començarà donant resposta a les principals preguntes que es fan sobre aquests conceptes: Què són? Per a què serveixen? Quins avantatges ens proporcionen? Etc. Amb els principals conceptes coberts, entrarem en més profunditat. Buscarem quins són els principals pilars de la qualitat de codi, veurem quines mesures hem d'extreure per comprovar si el nostra codi es de qualitat o no i com millorar-la. Estudiarem al complet un sistema d'integració continua, comprendrem la funcionalitat dels seus components i el cicle d'integració continua de les aplicacions software.

Després d'obtenir el coneixement necessari, ens posarem a dissenyar un sistema d'integració continua que integri les principals funcionalitats d'aquests i que, una vegada muntat, es desplegarà en un servidor per poder utilitzar-lo i comprovar tot el seu potencial. Es dissenyarà un sistema que integri, al menys, les necessitats bàsiques d'un sistema d'integració continua, es a dir, emmagatzematge, compilació, control de qualitat i desplegament. Amb el disseny, veurem les eines que tenim a la nostra disposició i farem comparatives per escollir l'ideal per el nostra sistema. Finalment farem la configuració i el desplegament i comprovarem que tot esta funcionant correctament.

Amb el nostre sistema en funcionament ja podrem començar amb el desenvolupament de l'aplicació en sí. Prèviament tindrem que fer el treball de definir els requisits, els *mockups* i diagrames varis per fer-nos a la idea de com a de ser la nostra aplicació i poder començar la programació amb les idees ben clares. Per començar a programar farem una mena de ***Sprints***, d'entre una i dues setmanes de duració aproximadament, on al final de cadascuna tindrem que tenir l'aplicació, amb les funcionalitats que pertoquin en cada moment, al servidor (a excepció del primer *Sprint* ja que encara no incorporarà la integració amb el sistema de IC).

A la planificació es té en comptes que es necessari com a mínim una setmana per fer els retocs finals a la documentació, comprovar el correcte funcionament de l'aplicació, retocar el *look&feel* i realitzar la presentació.

1.5. Planificació

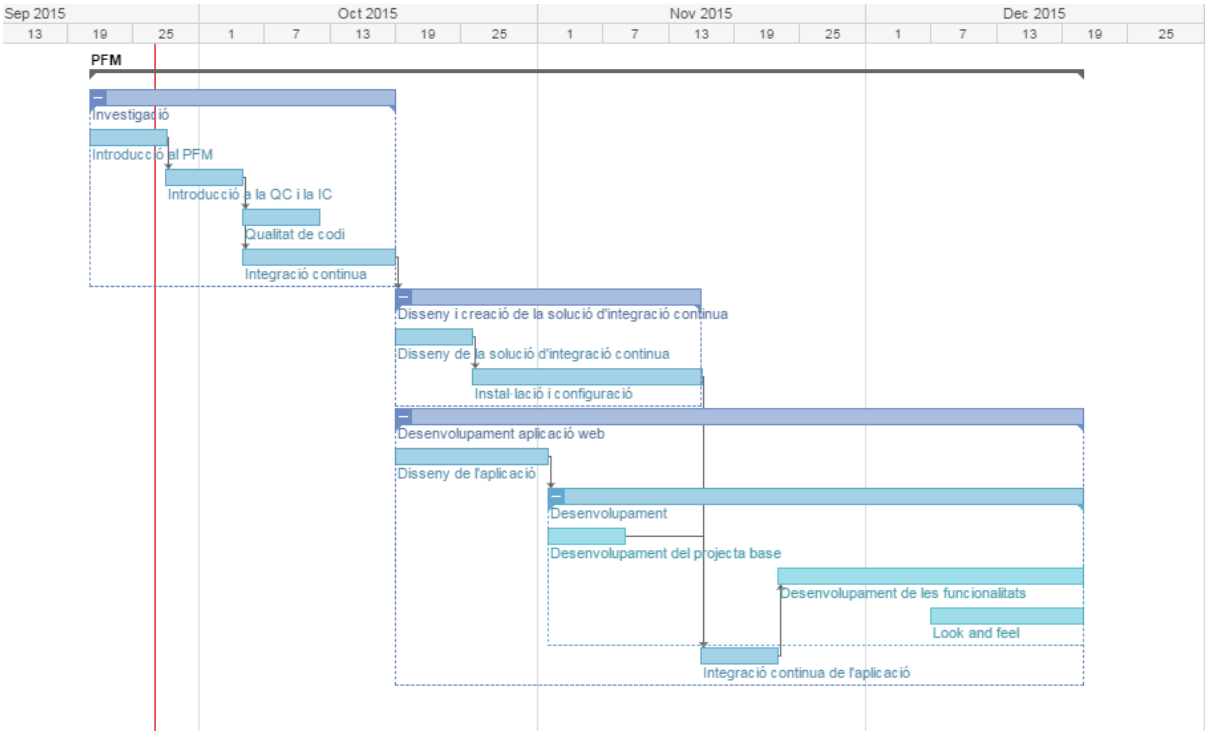
La planificació de tasques es la següent:

Tasca	Inici	Final	Duració
PFM	21/09/2015	20/12/2015[1]	92d
|- Investigació	21/09/2015	18/10/2015	28d
| |- Introducció al PFM	21/09/2015	27/09/2015	7d
| |- Introducció a la QC i la IC	28/09/2015	04/10/2015	7d
| |- Qualitat de codi	05/10/2015	11/10/2015	7d
| |- Integració continua	05/10/2015	18/10/2015	14d
|- Disseny i creació de la solució d'integració continua	19/10/2015	15/11/2015	28d
| |- Disseny de la solució d'integració continua	19/10/2015	25/10/2015	7d
| |- Instal·lació i configuració	26/10/2015	15/11/2015	21d
|- Desenvolupament aplicació web	19/10/2015	20/12/2015	63d
| |- Disseny de l'aplicació	19/10/2015	01/11/2015	14d
| |- Integració continua de l'aplicació	16/11/2015	22/11/2015	7d
| |- Desenvolupament	02/11/2015	20/12/2015	49d
| | |- Desenvolupament del projecte base	02/11/2015	08/11/2015	7d
| | |- Desenvolupament de les funcionalitats	23/11/2015	20/12/2015	28d
| | |- Look&feel	07/12/2015	20/12/2015	14d

Taula 1: Planificació de tasques

El diagrama de Gantt queda de la següent forma:

[1] Els dies que resten fins l'entrega final del projecte (08/01) seran per resoldre bugs i repassar la memòria.



Imatge 1: Diagrama de Gantt[2]

Les fites o *milestones*, amb el seu contingut quedaran de la forma següent:

- 21/09 Inici del projecte
- **30/09 PAC 1:** S'entregarà la introducció al projecte de final de màster, així com l'esquema de capítols que contindrà el projecte.
- 01/10 Inici de la investigació sobre els conceptes IC i QC
- 18/10 Finalitzar una primera versió del bloc d'investigació i inici del disseny i creació del sistema de IC
- 19/10 Inici de la recol·lecció de requeriment i disseny de l'aplicació
- 26/10 Inici del desplegament i configuració del sistema de IC
- **28/10 PAC 2:** S'entregarà la primera versió de la investigació sobre QC i IC. També s'entregarà l'esquema que tindrà el nostra sistema d'integració continua, incloent el software escollit.
- 01/11 Finalització de la recol·lecció de requisits i disseny de l'aplicació.
- **02/11 Sprint 1:** Desenvolupament del projecte base.
- **16/11 Sprint 2:** Incorporació del *workflow* d'IC.

---

[2] Aquests diagrama de Gantt a sigut dissenyat amb l'eina web Wrike (www.wrike.com)

- **23/11 Sprint 3:** Incloure les funcionalitats bàsiques de creació i visualització d'usuaris i projectes, tant per part de l'administrador com dels usuaris.
- **07/12 Sprint 4:** Incloure el *loof&feel.*
- **PAC 3 (15/12):** Inclourà l'aplicació amb les seves funcionalitats al complert i amb molta part del *look&feel* completada, a falta d'alguns retocs. La documentació estarà completa a falta d'una revisió final.
- 20/12 Finalització de la primera versió completa i inici de revisions i retocs finals, tant a l'aplicació com de la documentació.
- **11/01 Entrega final de la documentació:** Entrega final de documentació, amb els canvis proposats al PAC 3, i aplicació completament funcional, juntament amb un *log* dels diferents resultats obtinguts en el sistema d'integració continua.
- **11/01 Realització i entrega de la presentació final**

2. Introducció a la qualitat de codi i la integració continua

> *"An important part of any software development process is getting reliable builds of the software. Despite it's importance, we are often surprised when this isn't done."*
>
> **Martin Fowler**

Actualment, la necessitat de desenvolupar grans projectes software de qualitat han fet que les empreses busquin maneres de reduir el cost, el temps d'entrega i els riscos, però mantenint el mateix nivell de qualitat, es a dir, "*better, faster and cheaper*". Això no es possible sense aplicar tècniques que minimitzin els errors, problemes i tasques rutinàries, i maximitzin el rendiment dels integrants dels equips de desenvolupament del projecte.

La qualitat de codi (QC) i la integració continua (IC) són dos aspectes que actualment, i degut al seu alt impacte en els problemes anteriors, estan molt en el punt de mira dels desenvolupadors i de les empreses de desenvolupament de software en general.

La majoria de grans empreses ja incorporen en el seu treball diari la integració continua en els seus projectes, i gran part de les petites (*startups*) o mitjanes empreses dedicades al software s'han adonat que la incorporació d'un sistema d'integració continua, ja sigui en la seva totalitat o agafant part de la integració continua i adaptant el cicle a les seves necessitats, als aporta un treball més eficaç, amb la conseqüència de tenir un producte abans i de major qualitat.

Grans companyies com **Google**[3], **Facebook**[4], **LinkedIn**[5] i **Netflix**[6] són exemples de companyies que incorporen sistemes complexos d'integració continua i que són al capdavant en el que al cicle d'integració continua es refereix. Aquestes multimilionàries companyies, en molts casos, no podrien oferir el seu servei, o al menys amb un nivell alt de qualitat, si no fos per el seu control de la qualitat de codi i la integració.

---

[3] https://air.mozilla.org/continuous-delivery-at-google/ (consultada a data de 09/12/2015)

[4] http://www.infoq.com/presentations/Facebook-Release-Process (consultada a data de 09/12/2015)

[5] http://www.wired.com/2013/04/linkedin-software-revolution/ (consultada a data de 09/12/2015)

[6] http://techblog.netflix.com/2013/08/deploying-netflix-api.html (consultada a data de 09/12/2015)

Però que es la qualitat de codi i la integració continua? A continuació respondrem aquestes preguntes i les altres principals que sorgeixen al introduir-se en aquest nou mon, i explicarem com aquets dos conceptes poden ajudar en el dia a dia del desenvolupament de software.

2.1. Què és la integració continua?

El concepte d'Integració Continua (IC) és la practica o metodologia en el desenvolupament de software on els membres de l'equip integren els seus desenvolupament de forma freqüent i en cada integració es realitzen una sèrie de verificacions de forma automàtica, com poden ser: la existència d'errors, la qualitat del software, el seu rendiment, etc. Això redueix de manera significativa els problemes i temps d'integració així com augmentar la cohesió del software desenvolupat per tots els membres de l'equip.

El primer nombrament que es va fer sobre la Integració Continua va ser per Grady Booch, en el seu mètode al 1991. En aquests primers moments, Brooch no expressava la necessitat de la integració varies vegades al dia, però el concepte va ser emprat com a part del Extreme Programming (XP) que si que tenia en compte la necessitat d'aquesta integracions varies vegades al dia. Als seus inicis, el principal objectiu que complia la IC en XP era prevenir el problemes d'integració (o *"integration hell"* anomenat en angles).

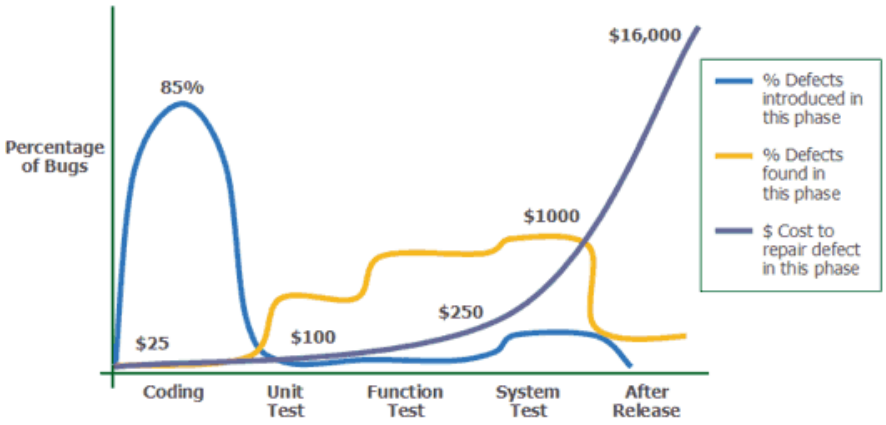
Conforme es va avançar en el temps, les seves necessitats d'integració es van anar fent més evidents. Es va fer una combinació amb la practica *"test-driven development"*, on es passaven tots els tests a la maquina de desenvolupament abans de poder compartir el teu codi amb els altres components de l'equip. Seguidament també es va adoptar el que s'anomena un *"build server"*, que automàticament executava uns tests de manera periòdica o al fer una pujada de codi, i al finalitzar reportava els resultats als desenvolupadors.

Avui dia, el concepte de IC ha sigut adoptat per gran quantitat de comunitats i empreses, i ja no està lligat únicament del concepte de XP. També a cobert altres àmbits, a part de la automatització dels tests unitaris i d'integració, com són els processos continus per el control de qualitat en general, control estàtic i dinàmic de codi, perfil de rendiment i generació de documentació, són exemples d'aquests nous processos.

2.2. Perquè es necessita un sistema de IC?

Els motius que pot portar a una empresa o equip de treball, per incorporar un sistema d'integració continua són els següents:

- La resolució de *bugs* al final del projecte és més costosa que a l'inici, per tant una eina que ens detecti el més aviat possible un error, ens pot fer reduir el temps de correcció dels errors, amb la reducció de cost que això implica.



Imatge 2: Gràfica de existències, detecció i cost de bugs al llarg del projecte[7].

- La falta de cohesió entre els membres del equip pot crear problemes (canvis incompatibles entre desenvolupaments, versions incompatibles, falta de comunicació, etc.), que si no es detecten el més aviat possible poden ocasionar uns desviaments de temps importants.
- La falta de comprovació d'uns estàndards de qualitat de forma constant pot ocasionar la mala costum de realitzar un codi de baixa qualitat (duplicitat de codi, dobles comprovacions, etc.) que pot arribar a ocasionar que el seu manteniment o millora sigui una tasca realment àrdua.
- Que tot el conjunt no tingui una visió global de l'estat del projecte (cobertura de test, versionat, etc.) pot fer que la pèrdua d'un membra de l'equip sigui catastròfica per els objectius del projecte.
- Són necessaris diferents entorns de proves i demostració de les diferents funcionalitats per ser utilitzades amb els diferents membres que infereixen en un projecte. Els desenvolupadors necessitaran provar noves funcionalitats, el client voldrà veure funcionalitat completes i provades i finalment es necessita un entorn on es trobi la última versió oberta als usuaris.

Si un grup de treball o empresa necessita cobrir aquestes necessitats (i com es pot percebre, pràcticament qualsevol desenvolupament de software necessitarà cobrir-les) és un bon punt de partida per plantejar la incorporació d'un sistema d'integració continua en el seu *workflow* de treball.

2.3. Què NO és la integració continua?

Una vegada entès què és la integració continua, és important entendrà què no és la integració continua per poder extreure tot el partit que ens ofereix i no quedar-nos només en la superfície del que es pot arribar a fer. Per aquest motiu exposem uns punts que no s'entenen com integració continua o que simplement són una ínfima part de tot aquest mon:

[7] Font: *http://www.agitar.com/solutions/why_unit_testing.html*

- ***Nighly build*:** Les *nighly builds* són aquelles que es fan en un horari fora d'oficina. Un sistema d'integració continua no és aquell que únicament fa una *build* a una hora en concret i ja esta. Els sistemes d'integració continua són molt més complexos que això, impliquen tot un *workflow* des de la pujada de codi fins el desplegament d'aquests amb tota una sèrie de comprovacions al mig.
- **Branques de desenvolupament:** La IC està dissenyada per fer desenvolupament contínuament sobre un producte i sí que existeixen branques de desenvolupament, però si deixem de banda totes les altres funcionalitats que ens aporta, deixa de ser un sistema de IC.
- ***Build* a través d'una interfície gràfica:** La integració continua està dissenyada per dissenyar i configurar un *workflow* automàtic per el desenvolupament. Amb els quals a partir de certes accions es llencen altres accions, per aquests motiu un llançament des d'una interfície gràfica elimina l'automatisme que ens aporta la IC. Moltes eines si que incorporen interfícies gràfiques per la configuració de les tasques i accepten el llançament manual però només com a opció extra.

2.4. Què és la qualitat de codi?

Segons Software Engineering Institute: "S'estima que un programador expert introdueix un defecte per cada 10 línies de codi." Suposant que es detecten un 99% dels defectes introduïts, això ens deixa un defecte per cada 1000 línies de codi. Es podria pensar que aquest volum d'errors és molt baix però si pensem en el volum de línies de codi (LOC) que tenen els principals programes que utilitzem avui dia[8]:

- La mitjana de les apps mòbils: 45.000 LOC, es a dir, 45 errors.
- Ordinador d'una llançadora espacial: 400.000 LOC, es a dir, 400 errors.
- Linux kernel versió 2.2: 2 milions LOC, es a dir, 1000 errors.
- Firefox: unes 10 milions LOC, es a dir, 10 mil errors.
- Windows 7: unes 40 milions LOC, es a dir, 40 mil errors.
- Google (amb tots els seus serveis): 2 bilions LOC, es a dir, 2 milions d'errors.

Com podem veure són una gran quantitat d'errors, però, podríem considerar que aquests errors no són importants i que ja s'aniran solucionant amb el temps. El problema és que al llarg de la historia, alguns errors, tant de software com de hardware, han resultat en unes pèrdues multimilionàries[9]:

[8] http://www.informationisbeautiful.net/visualizations/million-lines-of-code/ (ultima consulta 10/12/2015)

[9] http://archive.wired.com/software/coolapps/news/2005/11/69355?currentPage=all (ultima consulta 10/12/2015) i http://noticieros.televisa.com/economia/1506/se-cae-sistema-visas-eu-genera-perdidas-millonarias/ (ultima consulta 13/12/2015)

- **28 Juliol 1962 - La prova del coet Mariner I:** Aquesta prova va ser un fracàs, el coet es va desviar de la seva trajectòria esperada. L'error va ser degut a la mala transcripció de la formula des del paper al codi del coet.
- **1993 - Intel i la divisió del punt flotant:** Un error de només un 0.006% en la divisió del punt flotant d'alguns dels models de Intel va ocasionar la pèrdua de 475 milions de dòlars per cobrir el recanvi dels chips a totes aquelles persones que reclamessin.
- **16 Juny 2015 – Error en el sistema de visat en EU:** Un error informàtic en el sistema de concessió de vises en Estats Units va retardar l'arribada de milers de treballadors agrícoles temporals. Generant unes pèrdues que s'estimen entre 500.000 i 1.000.000 de dòlars diaris durant unes dues setmanes.

A més, si no es segueixen una sèrie de bones practiques de programació podem trobar grans problemes al fer el manteniment o a l'hora d'incrementar el software tant verticalment, amb noves funcionalitats, com horitzontalment, escalant el sistema. Cosa que es complicada cada vegada més al augmentar la complexitat del software a desenvolupar, fins al punt, que es dificulta la comprensió del codi desenvolupat i el *debugging* es fa pràcticament impossible.

La qualitat de codi no és res més que generar un codi que garanteixi les característiques de funcionament, eficiència i mantenibilitat de les aplicacions al llarg del temps. Per prevenir o minimitzar el numero d'errors que es poden generar a l'hora de desenvolupar el software, reduir el temps de resolució d'errors i manteniment, i facilitar l'escalabilitat horitzontal i vertical de les aplicacions.

2.5. Què ens aporta la qualitat de codi?

El tenir cura del nostra codi per augmentar la seva qualitat no és només un tema estètic sinó que ens aporta una sèrie d'avantatges que poden ser la diferencia entre l'èxit o el fracàs d'un projecte. A continuació llistem alguns dels seus avantatges:

- **Millora de la qualitat dels productes i serveis:** Disminueix el nombre de defectes que s'introdueixen al codi i els defectes introduïts poden ser detectats més fàcilment. Això provoca major satisfacció al client i disminueix el temps de retorn de la inversió, ja que calen menys correccions al software després del seu desplegament.
- **Escalabilitat:** Quan el codi expressa un disseny robust, exhibeix simplicitat i té un bon suport d'integració i *testing*, inserta més funcionalitats o incloure més membres a l'equip pot augmentar el valor del producte i no ocasionar un sobre-cost com succeeix si no es segueixen certs criteris de qualitat de codi.
- **Millora del manteniment:** Un codi de qualitat té un fàcil manteniment, permetent incloure o revisar funcionalitat de forma rapida i fàcil. També permet millorar i revisar altres requeriments no funcionals crítics, com poden ser el rendiment, la seguretat i l'escalabilitat.

- **Habilitat per innovar:** La innovació succeeix quan conflueixen la intel·ligència, la motivació personal i un ambient propici. Un alt nivell de qualitat de codi crea un ambient tècnic excepcional on noves idees poden ser prototipades i provades.

3. Qualitat de codi

> *"The word quality has multiple meanings. Two of these meanings dominate the use of the word: 1. Quality consists of those product features which meet the need of customers and thereby provide product satisfaction. 2. Quality consists of freedom from deficiencies. Nevertheless, in a handbook such as this it is convenient to standardize on a short definition of the word quality as "fitness for use"."*
>
> **J.M. Juran**

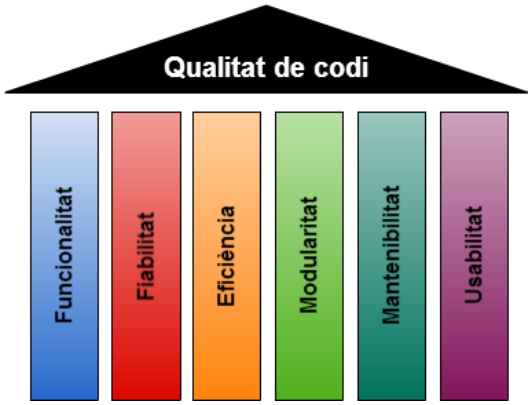
Ara que ja entenem els conceptes que treballem, és l'hora d'entrar més en profunditat en el que comporta la Qualitat de codi.

Veurem els 6 pilars bàsics de la qualitat de codi i quines són les mesures que ens proporcionen la informació necessària per comprovar la qualitat del nostra codi o sistema. Després veurem les dues tècniques que es fan servir per avaluar el codi: l'anàlisi estàtic de codi i l'anàlisi dinàmic de codi, i en relació amb això, el que ens indica el concepte de deute tècnic.

Finalment anomenarem quins són els principals estàndards ISO que inclouen conceptes referents a la qualitat de codi.

3.1. Els 6 pilars bàsics de la QC

Existeixen moltes divisions sobre el que poden ser principis de la qualitat del codi. Amb una petita revisió de varies d'aquestes divisions s'han pogut extreure 6 principis bàsics de la qualitat del codi juntament amb les mesures que ens aporten:



Imatge 3: Els 6 pilar de la qualitat de codi

Funcionalitat

- *La funcionalitat del software és la característica de qualitat associada principalment en que ha de fer el software sense tenir en compte com ho fa.*

Les mesures que ens aporten informació sobre aquest apartat són bàsicament els tests, tan unitaris com d'integració, tenint en compte la cobertura que aquests tenen. Els **tests** ens aportaran informació sobre quantes funcionalitats estem comprovant, i la cobertura és el tant per cent de línies que els tests comproven (una línia esta comprovada si al executar el test es passa per aquesta línia de codi). La **cobertura** pot tenir moltes més dades, i no només les línies que es cobreixes, altres mesures poden ser: si es cobreixen totes les opcions dels bucles, *if-else*, *switch*, etc, si es comproven els llançaments d'excepcions, etc. Tenir tests que cobreixin les funcionalitats amb una alta cobertura ens donarà una bona qualitat de codi en la seva funcionalitat.

Fiabilitat

- *La fiabilitat del software es refereix a la capacitat de mantenir uns nivells específics de rendiment sota condicions especifiques adverses com poden ser fallades del sistema.*

Existeixen principalment tres mesures que indiquen la fiabilitat del sistema, que estan estretament lligades amb el cicle de prevenció, mitigació i recuperació. La **maduresa** (*maturity*) d'un sistema es refereix a la absència de fallades del software, mentre que la **tolerància a fallades** (*fault tolerance*) està associada amb la capacitat del software de continuar amb el seu funcionament després d'haver sofert una fallada. Per últim, la **recuperabilitat** (*recoverability*) indica la quantitat d'esforç necessari per tornar a un estat de funcionament normal després d'una fallada.

Eficiència

- *L'eficiència d'un programa es la capacitat d'optimitzar el temps i recursos que utilitzen.*

Els **temps d'execució** i els **recursos requerit** per un programa sol ser un joc a dues bandes. En molt casos augmentar l'espai de memòria necessari (recursos) pot fer que el temps d'execució disminueixi, i a la inversa, més temps d'execució i menys espai de memòria necessari. El gran repte és disminuir els dos components a la vegada, o al menys mantenir un equilibri òptim. Per tant, la mesura de temps i recursos necessaris per l'execució del programa serà una mesura que indicarà com d'eficient és el nostra codi.

Modularitat

- *El programa por ser dividit en part més petites, cada qual de la forma més independent possible, tant de les aplicació com del sistema.*

La modularitat conté un àmbit més ampli que el que anomenem moltes vegades portabilitat. La portabilitat indica la **adaptabilitat** d'un software a diferents entorns (com poden ser diferents sistemes operatius), a part tenim, la **instal·labilitat** que indica com de complicat és desplegar el software dins de l'entorn i la **coexistència** amb altres programes executats en l'entorn en qüestió. La modularitat també incorpora els aspectes de **mutabilitat** de components que indica com de complicat és substituir un component del software per un altra sense que el seu funcionament s'alteri de forma substancial.

Mantenibilitat

- *Esforç que cal aplicar en una aplicació per conservar les seves funcionalitats normals durant el dia a dia.*

Quan estem interessats en la mantenibilitat del nostra codi ens fixem en 3 aspectes principalment: L'**analizabilitat** que és la capacitat d'observar el funcionament del sistema mentre està en funcionament, l'**estabilitat** del sistema ens indica si el seu funcionament és estable en el temps i durant quan de temps i la **testabilitat** és la capacitat de provar el seu correcte funcionament.

Usabilitat

- *Les persones que utilitzen el producte saben utilitzar les seves funcionalitats per complir els seus objectius.*

L'**usabilitat** és l'única característica on les seves mesures no poden ser obtinguda de forma objectiva, sinó d'una forma subjectiva pels usuaris que interaccionen amb el sistema. Entre les mesures d'usabilitat podem destacat l'**entenibilitat** que indica la facilitat de l'usuari per fer-lo funcionar i cobrir les seves necessitats, la **corba d'aprenentatge** és com de complicat és per l'usuari aprendre a fer servir el software i l'**atractiu** que indica la sensació que el software deixa.

3.2. Anàlisi del codi

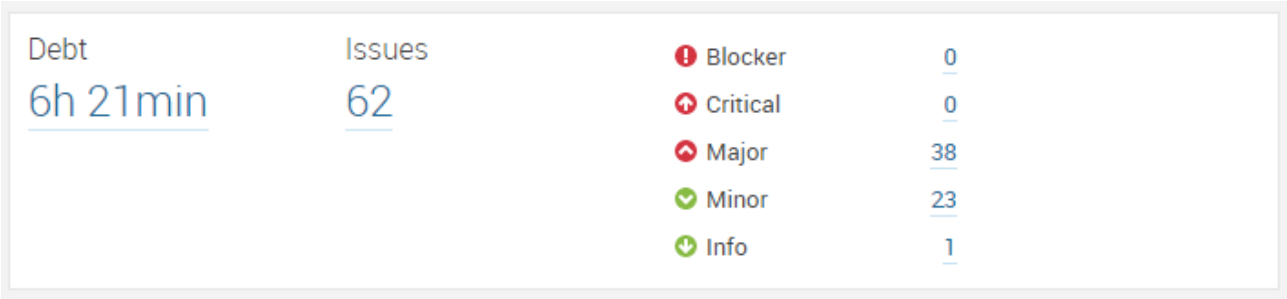
L'anàlisi del codi és la tècnica que ens serveix per extreure les mesures que hem comentat al apartat anterior. Aquestes tècniques es poden separen en 2 grans grups: l'anàlisi estàtic de codi i l'anàlisi dinàmic.

3.2.1. Anàlisi estàtic de codi

Com el seu nom indica és buscar faltes sobre el sistema en repòs, analitzant els diferents models i component del sistema en la cerca de possibles errors o variacions amb l'estàndard oficial utilitzat en el llenguatge de programació que s'utilitzi.

El que estudien aquests anàlisis estàtics són bàsicament el següent:

- **Estàndards de codificació:** Comprovar si el format de codificació (salts de línia, espais, tabulacions) concorda amb els estàndards de codificació del llenguatge utilitzat.
- **Nomenclatura:** Comprovar si els noms indicats per els diferents components (fitxers, classes, variables, etc.) concorde amb els estàndards de codificació del llenguatge utilitzat.
- **Complexitat:** Calcula el nombre de operacions que inclouen les diferents unitats de procés com poden ser funcions o classes.
- **Entrellaçament de paquets:** Comprova la interdependència entre paquets de treball. Aquesta mesura està estretament lligada amb la modularitat, ja que un alt entrellaçament indica una baixa **modularitat**.



Imatge 4: Captura d'una part de la pàgina de Sonar on es mostra la quantitat i severitat dels errors estàtics de codi.

3.2.2. Anàlisi dinàmic de codi

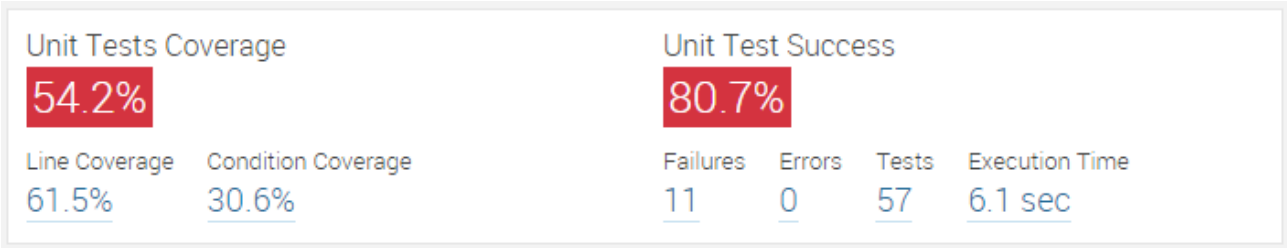
Aquests anàlisis es fan generant entrades al sistema amb l'objectiu de detectar fallades quan el sistema s'executa amb aquestes entrades. Les fallades s'observen quan es detecten incongruències entre la sortida esperada i la real.

Aquests tipus de tècniques es divideixen en 4 grups, depenent del conjunt del sistema que avaluïn a la vegada. De menys conjunt a més són:

- **Proves unitàries:** Avaluen una funció molt concreta cada vegada, tenint en compte que la resta de mòduls que utilitzen tenen un funcionament correcta[10]. Aquestes proves comproven estretament la **funcionalitat** del sistema.

[10] Normalment per no tenir en comptes el funcionament dels mòduls connectats, el que es fa es fer un *mock* dels altres components. Un *mock* el que fa és simular el comportament d'un mòdul, on tu li indiques explícitament quines sortides tindrà respecta unes entrades en concret. D'aquesta manera asegures que la sortida d'aquest *mock* sempre serà la correcte.

- **Proves d'integració:** Al igual que l'anterior, avaluen la **funcionalitat** del sistema, amb la diferencia que aquí si es té en compte el funcionament dels diferents mòduls que interconnecten, ja siguin interns o externs (pàgines web, APIs de tercers, etc.).
- **Proves de sistema:** Aquestes proves avaluen el funcionament del sistema en conjunt, per avaluar les mesures de **eficiència** i **fiabilitat** d'aquest. Els principals tests que s'utilitzen en aquesta fase són les **proves de rendiment**, on avaluem el comportament del sistema sobre uns requeriments específics, i les **proves d'estrès**, on avaluem el comportament del sistema sobre uns requeriments per sobre dels especificats pel disseny.
- **Proves d'acceptació:** Aquestes proves s'utilitzen per verificar que el software compleix amb les expectatives davant el client o els usuaris.

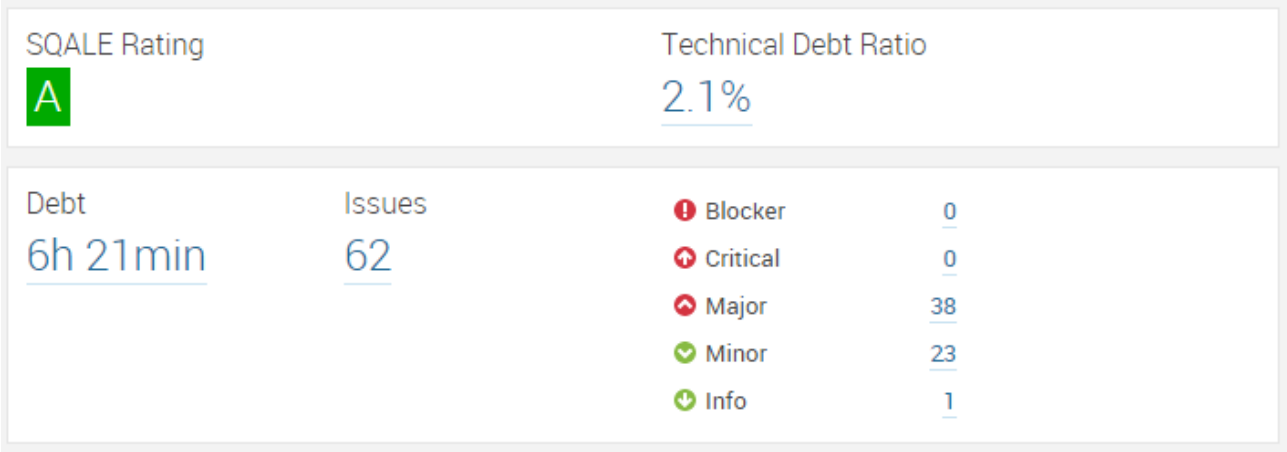


Imatge 5: Captura d'una part de la pàgina de Sonar on es mostren les dades dels test realitzats en un projecte.

3.3. Deute tècnic

El deute tècnic és un concepte, creat per Ward Cunningham, que surt a conseqüència de quantificar en temps el nombre d'errors que inclou un projecte. En concret, el deute tècnic seria el temps que caldria dedicar en un projecte per solucionar tots els errors trobats als anàlisis, tant estàtics com dinàmics.

Es podria pensar que és un valor que sempre tindria que ser zero en tots els projectes, però molts experts indiquen que no és només una mesura dels errors, sinó com un control a futur de les millores que s'han de realitzar en el projecte. Això ve de la possibilitat dels projectes de software de desenvolupar funcionalitats ràpidament, però tenint en compte que caldran canvis en el futur. D'aquesta manera, el deute tècnic ens permet avaluar el temps a dedicar per solucionar aquests problemes trobats, i nosaltres podrem avaluar la prioritat que donem als errors trobats per anar disminuint aquest deute tècnic.



Imatge 6: Captura d'una part de la pàgina de Sonar on es mostren les dades del deute tècnic.

3.4. Estàndards ISO més comuns a la QC

L'**Organització Internacional per a l'Estandardització** (*International Organization for Standardization*), coneguda com ISO, com no podia ser, ha introduït certs estàndards que comuniquen certs aspectes sobre la qualitat de codi dels projectes software o de les empreses que els desenvolupen.

Un pinzellada dels estàndards ISO més comuns a la producció de software, on comenten aspectes de qualitat de codi, són:

- **CMM (Capability Maturity Model):** Classifica les empreses per nivell de maduresa que permeten conèixer la seva maduresa en la producció de software.
- **ISO 15504 (SPICE – Software Process Improvement and Capability Determination):** Permet avaluar la capacitat i maduresa dels processos software d'una organització.
- **ISO 12207:** Modela el cicle de vida del software.
- **ISO 25000:** Estàndard internacional per l'avaluació de la qualitat del software.

4. Integració continua

> *"One machine can do the work of fifty ordinary men. No machine can do the work of one extraordinary man."*
>
> **Elbert Green Hubbard**

Explicarem en un primer moment el què és el control de versions amb els repositoris, com funcionen i unes pautes per utilitzar-los de forma correcta. Comprendrem el concepte de *environment* o entorn que ens permetrà testejar el nostra codi d'una manera adequada segons on es tingui que desplegar en cada moment. Seguidament entrarem en el cicle d'integració continua, on veurem els passos que comporta un perfecte cicle d'integració continua i veurem cada apartat de manera detallada. Seguidament veurem com quedarien tots els component de software interconnectats sobre un mapa de component. Finalment detallarem algunes bones practiques per treure un alt rendiment en aquests sistemes.

4.1. Control de versions

El control de versions és l'eina que ens permet mantenir el nostra codi emmagatzemat de manera segura i mantenir un control de diferents versions del mateix codi, ja sigui un historial de versions, diferents versions del mateix codi o ambdues a la vegada. El control de versions és una eina necessària pels grans projectes on col·laboren molts desenvolupadors. Permet mantenir una mateixa versió comuna per tots els membres de l'equip, metres que a la vegada, cadascun va fent les funcionalitats de manera autònoma fins que es finalitza la funcionalitat i es sincronitza amb tots els altres desenvolupadors. Apart d'això també permet mantenir versions pels diferents entorns i, fins i tot, recuperar versions anteriors quan sigui necessari.

Els programes encarregats d'aquesta tasca són els **repositoris de codi**. Un sistema de control de versions o repositori pot constar d'un sol servidor central o remot, o, com pot ser l'exemple del repositori de codi ***Git***, dos sub-repositoris, un anomenat **local** que com el seu nom indica està situat a prop del desenvolupador (en el cas de Git en el propi ordinador del desenvolupador) i un repositori **remot** o *origin* que normalment està ubicat en un servidor extern on normalment existeix alguna mena de còpia de seguretat per no perdre les dades que allà es troben.

Cadascun dels repositoris, tant el local com el remot o central, té cadascun un conjunt de **branques** o *branch*. Cada branca és una versió diferent del codi i cada branca pot tenir el seu propi historial de versionat per poder anar a qualsevol punt anterior de la branca.

Basant-nos en un repositori Git i podent ser extrapolat en altres, existeixen varies accions bàsiques, conegudes per el seu nom en anglès, que hem de conèixer per poder treballar i parlar sobre els repositoris:

- ***Commit*:** És l'acció de guardar el codi actual al repositori local.

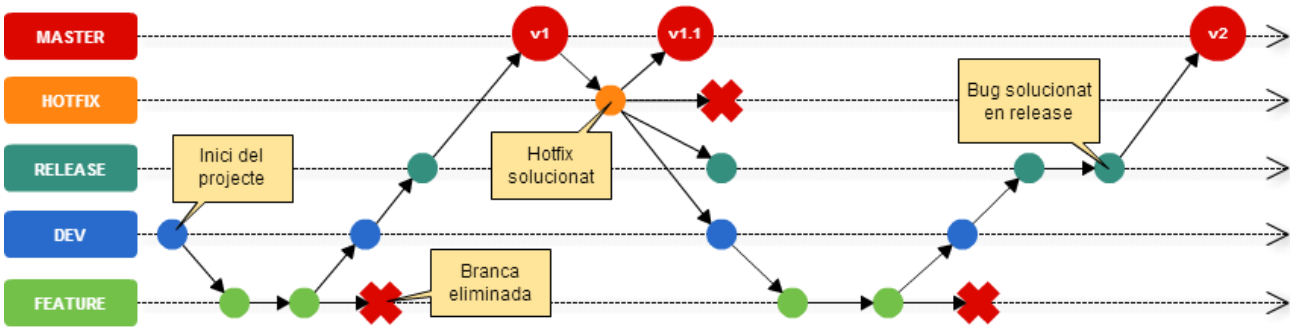
- ***Push*:** És l'acció de guardar el contingut del repositori local al repositori remot.
- ***Fetch*:** És l'acció de recuperar el contingut del repositori remot i guardar-lo al repositori local.
- ***Merge*:** És l'acció d'ajuntar el codi de dues branques en una de sola. Al realitzar aquesta acció normalment es necessari realitzar la resolució de conflictes quan existeixen canvis en les dues branques en un mateix fitxer.
- ***Rebase*:** És una acció semblant a l'anterior, amb la diferencia que en aquests cas es van incorporen els *commits* extrets, un per un, d'una de les branques cap a l'altra.
- ***Checkout:*** És l'acció de recuperar un codi d'una versió anterior en concret.

4.1.1. Bones practiques en la gestió de branques

Els principals models per la utilització de les branques creen 5 tipus de branques, cadascuna amb les seves característiques:

- La branca ***master*** és la branca principal i permanent, que contindrà el codi de producció.
- La branca ***release*** és temporal i és intermediària entre les branques de *master* i *dev*. Abans de fer un *merge* del *dev* a *master*, passarà per aquesta branca, on s'acabarà de comprovar el seu funcionament abans d'anar a producció.
- La branca ***dev*** o *develop* és la branca principal de desenvolupament i permanent que conté els últims canvis en el desenvolupament preparats per la següent entrega o actualització. Quan el codi estigui preparat per producció es sincronitzarà amb la branca *release*.
- La branca de ***feature*** o d'experimentació personal és una branca creada temporalment per desenvolupar o experimentar amb una nova funcionalitat. Quan la funcionalitat esta completa i es vol incorpora al codi, es farà un *merge* amb la branca de *dev/develop*. En cas que el codi no es vulgui incorporar, la branca s'eliminarà.
- Les branques de ***hotfix*** es creen temporalment per solucionar petits problemes concrets urgents i poden fer *merge* contra qualsevol tipus de branca.

La següent imatge resumeix la comunicació entra les diferents branques d'un repositori:



Imatge 7: Diagrama on es mostren les possibles comunicacions entre branques.

4.2. Entorns de desplegament

Al moment de desenvolupar un projecte és molt important comprovar el correcte funcionament del sistema, cosa que com hem vist al capítol de qualitat de codi, es sol fer amb tests incrementals, des de tests unitaris que proven una funcionalitat molt concreta, passant per proves d'integració on es van comunicant diferents mòduls entre sí, i finalitzant amb proves de validació del sistema complet. Durant aquestes diferents fases, també s'utilitzen diferents entorns per fer aquestes proves, entorns que compleixen certes característiques per adaptar-se a les necessitats de cada tipus de prova.

Es poden fer moltes divisions depenent de les necessitats, però qualsevol entorn correcte a de disposar, al menys, amb 4 entorns principals:

- **Desenvolupament (DEV):** L'entorn de desenvolupament és on les noves funcionalitats o canvis de software són desenvolupats. La majoria són simplement els ordinadors dels desenvolupadors o alguna mena d'escriptori remot. Aquest entorn difereix molt amb l'entorn final de desplegament del software a desenvolupar, pot ser que el software a desenvolupar no sigui per un ordinador de sobretaula (pot ser un software destinat per mòbil o sistemes encastats), poden desenvolupar sobre altres versions de llibreries i incorporar totes les eines de desenvolupament necessàries en el propi entorn.

  En aquest punt, una vegada realitzat una funcionalitat, el repositori només deixa passar al següent entorn si els test unitaris passen de forma correcte, sinó s'han de solucionar, per poder incorporar la nova funcionalitat a l'entorn d'integració.

- **Integració (INT):** L'entorn d'integració és on s'interconnecten tots els components (que poden haver sigut desenvolupats de manera individual) i es realitzen tots els test d'integració per comprovar que la comunicació i el funcionament entre components es el correcte.

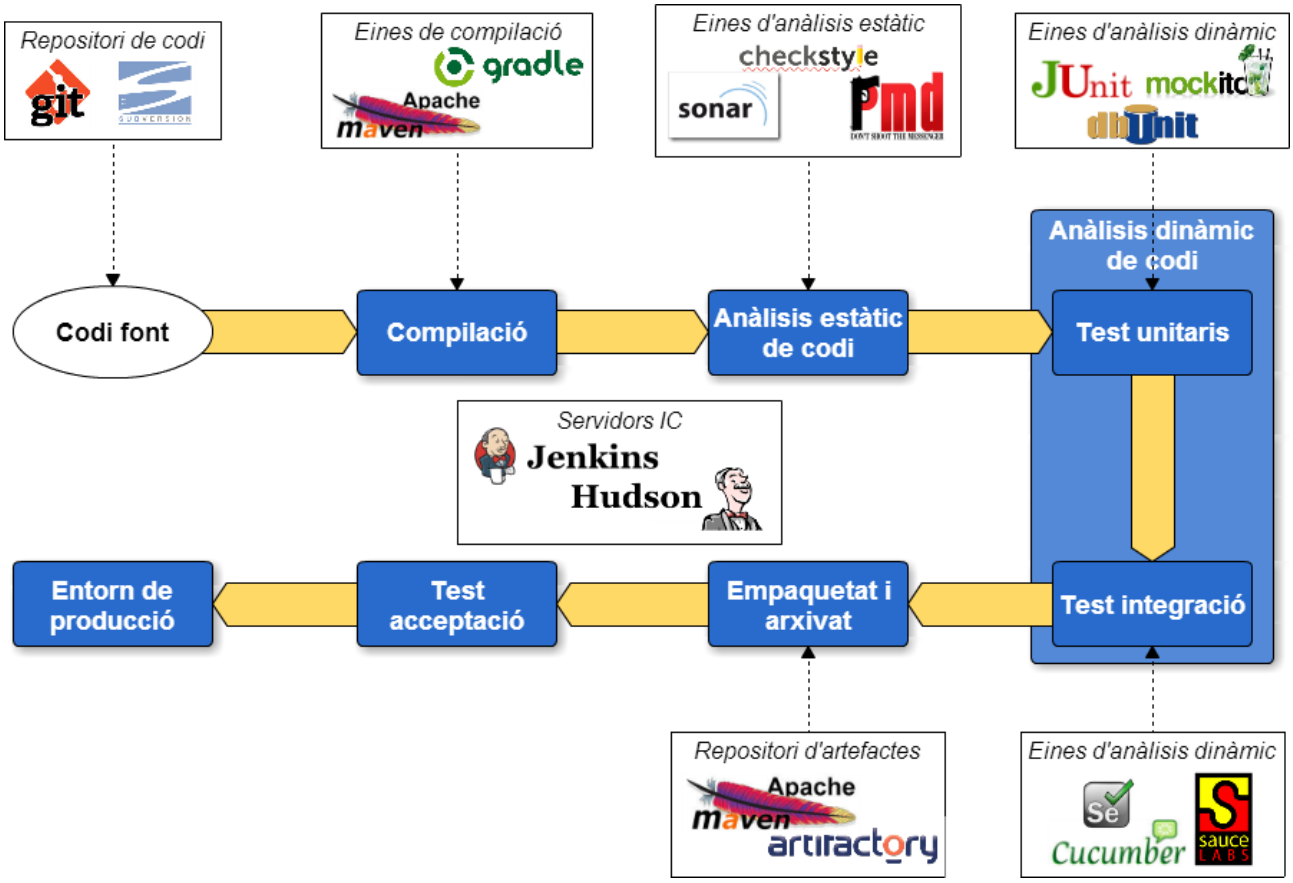
- **Pre-producció (PRE):** Aquest entorn és l'últim entorn abans del desplegament en l'entorn final. Aquí es realitzen les ultimes verificacions, com poden ser la validació per part del client o propietari del software. És important que l'entorn de pre-producció sigui una copia exacta de l'entorn de producció per evitar detectar incompatibilitats una vegada ja feta la pujada a producció.

  En aquest entorn també es possible aplicar els test d'estrès i de rendiment per comprovar com respon el servidor a aquests estímuls extrems.

- **Producció (PRO):** És l'entorn final i es on interactuaran directament els usuaris del software i per tant el pas més delicat. Un codi en aquest entorn ha d'haver passat per l'entorn de preproducció, on s'ha verificat el seu correcte funcionament i s'ha solucionat qualsevol error detectat. Un problema detectat en aquest entorn ha de ser solucionat de manera urgent i incloure les eines necessàries al entorn de preproducció per verificar que es detectin els problemes similars abans d'arribar a producció.

4.3. Cicle d'integració continua de software

El principal objectiu de la integració continua és realitzar certes tasques de forma automàtica, normalment aquestes tasques consten de l'obtenció de la qualitat de codi i del desplegament del artefacte en algun servidor. El cicle d'integració continua engloba una sèrie d'etapes que s'han de cobrir per obtenir una integració continua òptima. No existeix un únic cicle d'integració continua, ja que en part s'ha d'adaptar a les necessitats de cada empresa, però la imatge següent mostra les principals fases que qualsevol sistema d'integració continua hauria de cobrir per que es consideri complet:



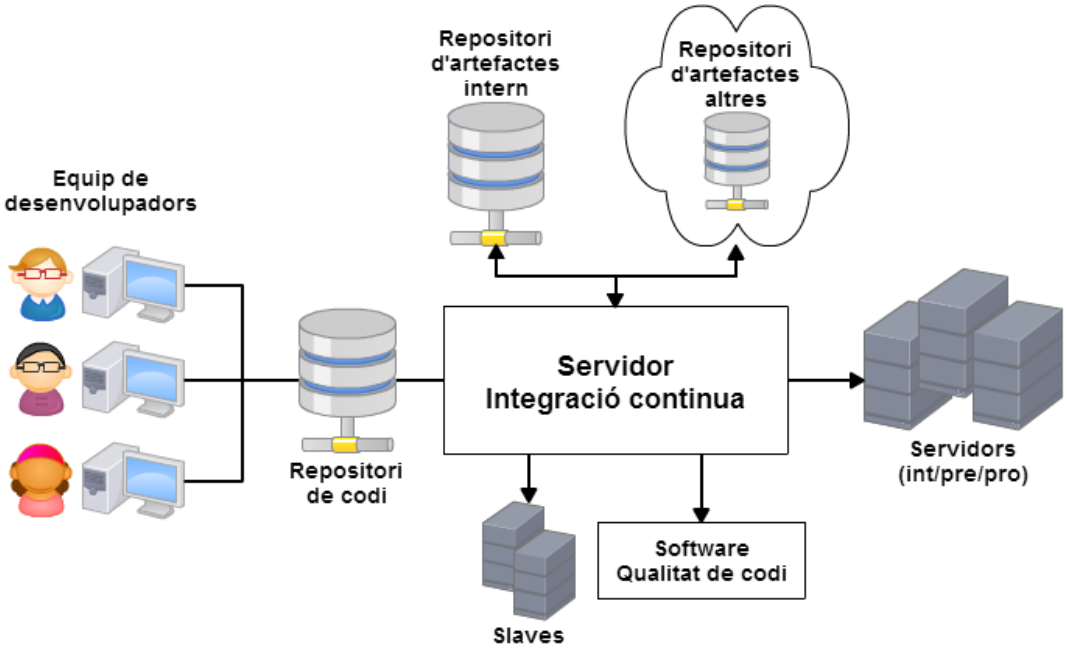
Imatge 8: Cicle ideal d'integració continua

Com és lògic, tot comença amb el codi de l'aplicació. El primer pas és la compilació del codi, per obtenir alguna mena d'arxiu que pugui ser executat. Seguidament existeixen 3 etapes on s'avalua la qualitat de codi. Primerament un anàlisis estàtic de codi on veurem quin grau de compliment tenim dels estàndards de codificació del llenguatge que utilitzem i seguidament passarem els test unitaris per veure si al nostra codi te un funcionament esperat a nivell unitari. Si els tests unitaris són correctes, es pot fer un desplegament a un servidor d'integració per realitzar les proves d'integració i de sistema. Si passa totes aquestes fases de manera adequada, aquesta versió es marcada com a correcte i pot ser empaquetada i guardada en algun repositori d'artefactes. Quan arribi el moment es podrà realitzar un desplegament en un entorn de pre-producció perquè es realitzi una avaluació per part del client, qui indicarà si la versió és adequada per fer el desplegament a producció o no. En cas positiu, ja sigui de forma manual o automàtica, es realitzarà un desplegament d'aquesta ultima versió als servidors de producció.

Depenent de les diferents polítiques de qualitat de software que s'utilitzin, en cadascun dels passos pot ser que la versió que s'està avaluant no les compleixi i per tant quedi com a rebutjada i no continuï amb el cicle. Per exemple, si falla la compilació ja no es continua amb les següents fases, si l'anàlisi de codi estàtic o dinàmic no compleix uns certs límits de qualitat es poden rebutjar aquestes versions i marcar-les com errònies. Si finalment arriba als tests d'acceptació, però el client no veu el software correcte, aquesta versió no continuarà fins a producció.

4.4. Arquitectura d'un sistema d'integració continua

Per complir amb el cicle de integració continua és necessari tenir un sistema que el suporti i ofereixi aquesta funcionalitat als diferents membres del equip. La arquitectura bàsica per desplegar aquest sistema descrit anteriorment seria la següent:



Imatge 9: Arquitectura d'un sistema d'integració continua

El codi o projecte estaria emmagatzemat de forma segura al repositori de codi, juntament amb una copia als ordinadors dels desenvolupadors per anar desenvolupant les noves funcionalitats. El servidor d'integració continua és al centre de tot i com indiquem, és qui orquestra totes les diferents funcionalitats. Aquest servidor estarà connectat amb els diferents softwares que ens aportaran la qualitat de codi, amb els diferents entorns (entorn d'integració, entorn de preproducció i entorn de producció) per desplegar els artefactes i finalment connectat amb el repositori d'artefactes intern, per poder guardar els artefactes generats en aquest repositori, i externs, per poder obtenir les dependències externes que calgui. El *build* dels projectes es fa a través dels esclaus o *slaves* i no a través del servidor d'integració continua, d'aquesta manera, un sol servidor d'integració continua pot controlar l'execució de varis projectes simultàniament als esclaus.

Cal mencionar que és important que les diferents parts puguin escalar de manera individual, com és el cas de l'execució de la *build* dels projectes als esclaus, ja que les necessitats de certes parts poden necessitar una aportació extra de còmput o emmagatzematge. En concret, podem veure que la necessitat de memòria dels dos repositoris (codi i artefactes) serà molt més important que al servidor d'integració continua. També necessitarà una aportació extra de memòria el sistema de qualitat de codi, degut a la quantitat de memòria que emmagatzema de les diferents mesures de la qualitat de codi. Per contra el servidor d'integració continua i els *slaves* no necessiten una quantitat de emmagatzematge molt gran però en canvi la compilació i execució dels diferents tests dels projectes requereix una capacitat de còmput molt major. Es per aquests motiu que es distribueix el còmput de l'execució del projectes en un sistema de varis servidors esclaus, on el màster va distribuint les diferents tasques als diferents esclaus.

4.5. Bones practiques

Molts autors han expressat com utilitzar els sistemes d'integració continua i com automatitzar les tasques de forma correcte. A continuació es nombren les bones practiques nombrades per Martin Fowler, ja que resumeixen de manera clara i resumida com utilitzar de manera correcte els sistemes de IC.

Mantenir un únic repositori de codi

En la practica és necessari mantenir un únic punt on es trobin els diferents codis desenvolupats. Tots els artefactes necessaris per la construcció del projecte han de ser inclosos al repositori, això inclou: test, arxius de propietats, esquemes de base de dades, inicialitzacions de base de dades, llibreries de tercers, documentacions, etc.

El control de branques en el repositori també és important. S'ha de mantenir en tot moment el nombre de branques al mínim. Per norma general es tindrà la branca principal o de desenvolupament que tots els desenvolupadors utilitzarà. També es poden tenir altres branques raonables, com poden ser la de producció o experimentals.

Automatització de *builds*

Només una sola comanda a de ser necessària per la construcció de tot el sistema. Eines per la construcció d'artefactes han existit des de fa molt de temps, com *make* i Ant, i actualment existeixen d'altres més noves, com Maven i Gradle. Aquestes eines permeten automatitzar la construcció a partir de la configuració de comandes especials. A part d'això, ja permeten fer altres tasques juntament amb la construcció d'artefactes com poden ser la construcció de documentació.

Build self-testing

Una vegada que la construcció del artefacte es feta, això implica que el programa executarà, però no implica que faci les coses que ha de fer de forma correcte. Per aquest motiu, s'han d'aplicar els test necessaris, de forma automàtica, per verificar que el comportament es l'esperat. Els test no asseguren completament que el comportament serà perfecta, però es poden identificar molts dels problemes a partir d'aquests.

Tothom puja a la branca principal cada dia

La integració és principalment comunicació entre els diferents membres del equip de forma continuada. La comunicació freqüent dels canvis permet que l'equip els conegui i que el codi sigui guardat de manera segura. Per aquests motiu és necessari que es facin *commits* a la branca de desenvolupament principal de manera freqüent, i es proposa com a mínim un al dia. Per fer un *commit*, la *build* a de passar tots els tests disponibles i resoldre els conflictes amb la versió existent a la branca amb la que es vol fer el *merge*.

Fer *commits* amb només unes hores de diferencia, ajuda en la detecció d'errors ràpidament, abans de que siguin massa grans. Ajuda a facilitar el *merge* de codi entre diferents desenvolupadors, ja que el codi modificat és menor. Finalment ajuda a la pròpia planificació diària, ja que obliga als desenvolupadors a dividir el treball en petites parts.

Després d'un *commit* s'ha de fer una *built*

És important que la *build* estigui contínuament en un estat correcte, és a dir, que al fer una *build* a la branca que conte el codi principal, aquesta passarà tots els test de forma correcte. Per aquests motiu és important fer una *build* i comprovar si passa els tests, abans de fer una pujada a la branca principal.

Això es pot realitzar de dues maneres diferents: utilitzant una *build* manual o a partir del servidor d'integració continua. L'aproximació manual és senzilla, cada desenvolupador és l'encarregat de fer els tests i, quan cregui convenient, el *commit*. La segona és més complexa però molt més elegant, una vegada es fa un *commit*, el servidor d'integració continua fa la *build* i comprova si passa els tests. En cas negatiu informa al desenvolupador i fa enrere el *commit*, en cas afirmatiu fa el *commit* a la branca principal i el dona com correcte.

Mantenir una *build* rapida

El principal avantatge de la integració continua és rebre un *feedback* de manera ràpida, per tant, la construcció d'artefactes ha de ser rapida per tal de poder rebre la informació ràpidament i detectar els problemes el més aviat millor.

Fer els test en un entorn clonat al de producció

L'objectiu principal del *testing* és detectar i solucionar qualsevol problema que es pugui trobar en producció i una part significant d'això es el sistema de l'entorn de producció. Si es realitzen tests en un entorn diferent al real (producció) els resultats podrien arribar a ser diferents, amb el risc que això comporta, detectar errors en producció que no succeeixen en els tests.

Per aquest motiu és totalment necessari que al menys un dels entorn, recomanable el de preproducció, sigui una copia exacta de l'entorn final de producció. Això vol dir, utilitzar el mateix hardware (o maquina virtual), la mateixa base de dades i llibreries utilitzades, això inclou mantenir la mateixa versió de tot el software utilitzat en els dos entorns.

Això te algunes limitacions. Si el producte a desenvolupar és una aplicació d'escriptori, no serà possible replicar totes les combinacions d'entorns que els usuaris poden utilitzar. En aquests cas l'objectiu hauria de ser duplicar l'entorn de producció el millor que sigui possible, i entendre que en aquests cas, s'assumeixen alguns riscos per la diferencia entre els entorns reals que poden existir i el de test.

Tothom pot veure que està succeint

Un dels components principals de la integració continua és la comunicació, per tant, és important que tots els integrant de l'equip puguin veure en tot moment què està passant al projecte i quins són els últims canvis que han succeït.

Com exemple es podria posar, l'estat de l'ultima *build*. Si la *build* ha resultat en un error, tots els components de l'equip han de saber-ho, perquè possiblement és un problema que s'ha de resoldre entre més d'un membre de l'equip.

Existeixen moltes tècniques per mostrar aquesta informació. Des de grans calendaris murals per veure quins dies falla la *build*, passant per pilots lluminosos verds o vermells, fins a pantalles amb una diapositiva dinàmica on es mostra l'estat, entre altres dades, de l'ultima *build* i que poden veure tots els membres de l'equip mentre treballen.

Desplegaments automàtics

Degut a la integració continua, és necessari la coexistència de múltiples entorns, l'entorn de tests unitaris, l'entorn d'integració, de preproducció, producció, etc. Això, juntament amb la quantitat de versions que conté un desenvolupament software abans de la seva versió final, implica que el desplegament d'un projecte es farà una gran quantitat de vegades. Per aquest motiu, hem de mantenir un desplegament automàtic, que permet simplificar el desplegament, reduint el temps i els errors que es poden produir.

Per poder evitar problemes després de realitzar un desplegament automàtic és important tenir algun sistema de *rollback*, que asseguri poder tornar a un estat estable en qualsevol moment. En els clústers, degut a conviure en varis servidors es necessari, a part del *rollback* simple, aplicar altres mètodes per evitar un estat

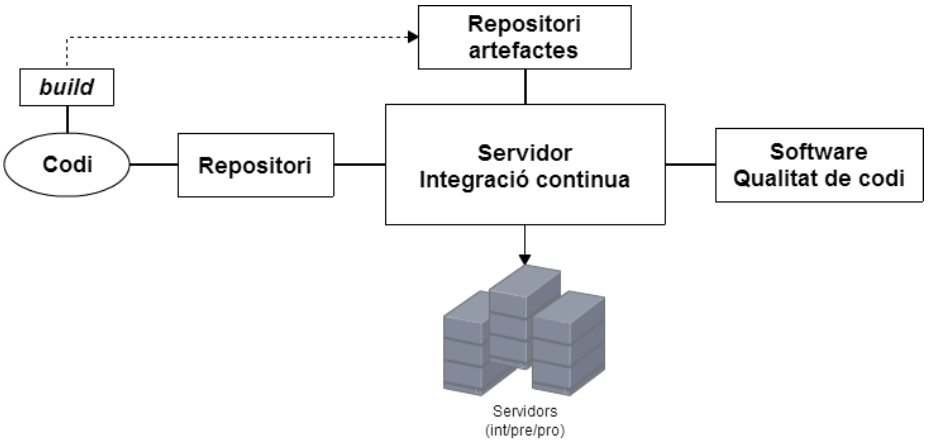
erroni de tot el sistema. Per això s'apliquen dos mètode: fer un desplegament gradual dels servidors a la nova versió o fer el nou desplegament per un nombre reduït d'usuaris on, després d'un temps prudencial sense cap problema, s'obrirà a la resta d'usuaris.

5. Disseny i creació d'una solució d'integració continua

"640K ought to be enough for anybody."

Bill Gates, 1981

En els capítols anteriors, hem pogut comprendre com és el funcionament d'un sistema d'integració continua i quines mesures podem obtenir per comprendre si el nostre codi és de qualitat o no. Amb aquesta informació podem dissenyar el nostre sistema d'integració continua bàsic, que quedaria de la manera següent:



Imatge 10: Arquitectura d'un sistema d'integració continua

El nostra **codi** seria la part inicial del sistema. Aquest codi seria emmagatzemat en un **repositori** i seria generat (*build*) a partir d'alguna **eina de construcció** que podria agafar artefactes des del nostra **repositori d'artefactes** així com d'altres repositoris extern. Al centra tenim el **servidor d'integració continua** que cada vegada que es faci una pujada de codi podrà compilar aquest codi i amb el software de qualitat de codi generar un correu que ens proporcioni les mesures de qualitat de codi com poden ser: complexitat de les funcions, nombre de test, tant per cent de test passats, cobertura de *testing*, interdependència entre paquets, etc. Si les mesures de qualitat superen uns certs límits, el nostra sistema d'integració continua podrà fer la pujada, del artefacte creat, al servidor.

Aquest servidor el tractarem com a integració (INT) degut a que és el primer servidor que es fa la pujada del codi i a més la fem de forma automàtica[11]. Com expliquem en els capítols anteriors, en cas d'un correcte funcionament en INT es podria fer una pujada a pre-producció (PRE), i finalment si supera les proves a PRE, fer una pujada a producció (PRO) de forma manual[12].

Ara, amb aquest esquema, hem de posar noms concrets del software a utilitzar, però per això és necessari veure quines eines ens ofereix el mercat.

5.1. Software al mercat

A continuació farem un resum de les principals característiques que ens ofereixen els principals softwares que existeixen actualment al mercat en cadascun dels blocs necessaris en el nostra sistema d'integració continua: una eina de construcció (*build*), un repositori de codi, un servidor d'integració continua, un sistema per mesurar la qualitat de codi i un repositori d'artefactes.

5.1.1. Eines de gestió i construcció de projectes

Les eines de gestió i construcció de projectes són les que ens permeten obtenir diferents accions que es poden aplicar en un projecte, com podria ser generar documentació, generar els desplegables, pujada en un repositori d'artefactes, etc., en un ordre concret i sobre unes condicions concretes, a partir d'una comanda.

Existeixen un munt d'eines de construcció disponibles depenent del llenguatge utilitzat i de les prestacions que necessitem. En el nostra cas, per a projectes Java, existeixen dues eines destacables: Maven i Gradle.

Maven

Maven és una eina de construcció automàtica de projectes que treballa en dos bandes diferents, la primera és com s'ha de fer la *build* i la segona són les dependències necessàries per realitzar-les. Un XML descriu aquestes dues fites. Maven de forma automàtica descarrega les dependències necessàries en un repositori local des d'on les utilitza per construir els projectes. Maven també pot ser utilitzat per generar projectes escrits en C#, Ruby o Scala entre d'altres llenguatges.

---

[11] Recordem que no es recomanable que una acció automàtica com un *push* del repositori faci una pujada automàtica directament als servidor de PRO.

[12] Quan parlem de "forma manual" en aquests sentit és que un responsable faci el llançament de la tasca de pujada de PRE a PRO.

Gradle

Gradle és una eina que es basa en Maven i comparteix moltes de les seves funcionalitats. La gran diferencia resideix en com s'especifica la construcció i dependències del projecte. Maven utilitza un XML, mentre que Gradle opta per Groovy. Gradle incorpora un sistema de graf acíclic direccional per determinar l'ordre d'execució de les tasques i va ser dissenyat per generar *build* en multi-projectes de forma coherent.

Altres eines de gestió i construcció de projectes

Altres eines de construcció (*build*) de projectes: *make, Apache Ant, Apache Buildr, MSBuild, Rake*.

5.1.2. Repositoris

Com hem explicat amb detall anteriorment, aquestes eines ens permeten tenir un control de versions, important per el desenvolupament de mitjans-grans projectes. Podríem destacar 3 repositoris de codis diferents.

Git

Git és un software de control de versions pensat en l'eficiència i la confiabilitat en el manteniment d'aplicacions amb un gran volum d'arxius de codi font. En un principi, es va pensar com un sistema de baix nivell on altres podrien implementar una interfície d'usuari, però amb el temps s'ha convertit en un sistema de control de versions amb una funcionalitat completa. Entre les seves funcionalitats podem destacar la seva ràpida gestió de múltiples branques al repositori i la gestió distribuïda (cada usuari té una versió local del repositori al complet). Git té una llicencia *GNU General Public License* v2 que permet la seva utilització en projectes tant personals com comercials, sempre que el software resultant continuï amb la mateixa llicencia.

Apache Subversion

Apache Subversion (abreviat freqüentment com SVN) és una eina de control de versions *open-source*, amb una llicencia Apache, al qual s'assembla molt al funcionament d'un sistema de fitxers. Utilitza un sistema de revisions per mantenir els canvis (només emmagatzema el conjunt de modificacions i no tot el fitxer duplicat), optimitzant així l'ús de la memòria i fa que la creació de noves branques mantingui una complexitat constant. Permet als usuaris borra, crear i copiar arxius i carpetes amb la mateixa flexibilitat que es podria fer en un disc dur.

Github

L'últim repositori que veurem és una mica diferent als altres dos. Tant Git com SVN són repositoris que han de ser instal·lat en un servidor o ordinador per que puguin ser utilitzats, en canvi, **GitHub** es una plataforma d'Internet de desenvolupament col·laboratiu de projectes a partir d'un sistema de control de versions. Per tant, no es només un software de control de versions, sinó que incorpora altres funcionalitats com són *wikis* per cada projecte, gràfiques de desenvolupament i seguiment, funcionalitats per les principals xarxes socials, etc. Al ser una plataforma col·laborativa, pot ser utilitzada de forma gratuïta sempre que el codi emmagatzemat sigui públic, també disposa la possibilitat d'emmagatzemar projectes de forma privada amb la creació d'un compte de pagament.

Altres repositoris de codi

Altres repositoris: *Mercurial. AccuRev, CVS, Perforce, Clearcase.*

5.1.3. Servidors d'integració continua

El servidor d'integració continua es la part central del nostra sistema i per tant es necessari veure quin ventall d'opcions ens ofereix el mercat. Existeixen gran quantitat de software considerats servidors d'integració continua, a continuació explorem els que considerem els més importants.

Bamboo Atlassian

Bamboo és un servidor d'integració continua de Atlassian[13]. Bamboo suporta *builds* des de qualsevol llenguatge que utilitzi eines de compilació com poden ser Maven, Ant o Make. Les notificacions poden ser personalitzades segons el tipus d'esdeveniment i rebudes per correu electrònic, missatge, RSS o pop-up en entorns com Eclipse o IntelliJ. Bamboo es gratuït per l'ús personal i projectes *open-source* i ofereix una versió comercial a un preu variable segons les necessitat del client.

[13] Atlassian són els creadors de JIRA, Confluence i Crowd, softwares molt coneguts en altres àmbits del desenvolupament de software.

Apache Continuum

Apache Continuum és el servidor d'integració continua d'Apache. Està estretament lligat a Apache Maven i permet programar *builds* en determinat horari, així com enviar correus electrònics amb els resultats obtinguts. Com tots els productes Apache, ofereix una llicencia Apache 2.0 gratuïta tant per utilització personal com comercial, seguint algunes normes.

CruiseControl

CruiseControl és un servidor d'integració continua basat en Java. Inclou notificacions per correu electrònic, Ant i varies eines per el control de versions, apart de que pot ser estès amb diferents *plugins*. Ofereix també una interfície web que permet executar i veure els resultats de les *builds*. Existeixen versions per .NET i Ruby anomenades CruiseControl.NET i CruiseControl.rb respectivament. CruiseControl es gratuït i *open-source* sobra una llicencia BSD.

Jenkins

Jenkins és un servidor d'integració continua *open-source*, publicat amb llicencia MIT, escrit en Java. Va sorgir a partir del projecte Hudson i va seguir evolucionant de forma independent, degut a la forta dependència que tenia Hudson sobra Oracle. Suporta gran quantitat de sistemes de control de versions com poden ser SVN, Git, i Mercurial entre molts d'altres i poden executar tant projectes Ant, Maven o Gradle. Les *build* poden ser llançades de varies maneres, ja sigui a partir d'un esdeveniment, com un *commit* al repositori, o la programació en un temps o data concrets. A part de les funcionalitats bàsiques, també disposa de gran quantitat de *plugins* per augmentar les seves funcionalitats.

Travis CI

Travis CI és un servidor d'integració continua especialment dissenyat per fer les *builds* i les proves de codi guardat a GitHub. Travis CI detecta automàticament els *commits* al repositori i intenta fer la *build* i passar els tests. Quan el procés esta complet el notifica al desenvolupador de la manera que el desenvolupador a configurat. Suporta gran quantitat de llenguatges i molts projectes importants, com poden ser Ruby on Rails, Ember.js o Node.js, utilitzen aquesta eina. Travis CI te una llicencia permissiva i el seu codi esta al complet a GitHub, pot ser utilitzat de forma gratuïta en projectes públics i, amb un cost, en els projectes privats de GitHub.

Altres servidors d'integració continua

Altres servidors d'integració continua: *Hudson* i *Team Foundation Server (Microsoft).*

5.1.4. Software per qualitat de codi

Així com dels productes anteriors hem mostrat una llista de softwares on entre ells són excloents, en aquesta part tenim una gran quantitat d'eines on cadascuna ens ofereix una part de la informació de la nostra qualitat de codi.

Per exemple eines com **CheckStyle** o **Google CodePro Analytix** són grans eines pel anàlisis estàtic de codi. **JUnit** o **TestNG**, juntament amb eines com **Mockito**, ens permeten fer anàlisis dinàmics de codi d'una manera molt completa i correcte i amb eines com **Jacoco** podem extreure la cobertura d'aquests test unitaris fets. També podem utilitzar eines com **Selenium** que ens permetran fer tests d'integració sobre una aplicació directament sobre el *frontend*.

SonarQube

La unió i visualització correcte de tota aquesta informació és la part més complicada. En aquests camp **SonarQube** és l'eina més utilitzada. SonarQube es podria considerar una eina d'inspecció de la qualitat de codi, mostra de forma visual i realment molt clara, les diferents dades que poden aportar les diferents eines de qualitat de codi que hem comentat anteriorment. Entre les seves característiques destaquem que suporta gran quantitat de llenguatges de programació, com poden ser Java, C/C++, Python, Javascript, PHP, etc., guarda les dades de les diferents mètriques per generar gràfiques històriques i té una alta integració amb eines de gestió i construcció de projectes, com Maven i Gradle, i amb eines de integració continua, com Atlassian Bamboo i Jenkins. Finalment, si tot això no es suficient, existeixen gran quantitat de *plugins* que poden ser incorporats per adaptar-lo a les necessitats dels projectes.

5.1.5. Repositoris d'artefactes

Els repositoris d'artefaces o repositoris de llibreries, són aquells repositoris que ens permetran guardar les diferents llibreries, desplegables o executables, amb les diferents versions que anem generant al llarg d'un projecte. En les empreses amb un sistema d'integració continua mínimament assentat, es important mantenir un repositori d'artefactes per emmagatzemar, de forma segura i privada, les diferents llibreries o desplegables que es van generant. Seguidament comentem 3 dels repositoris d'artefactes més habituals.

Maven 2 Central Repository

Maven 2 Central Repository és el repositori

d'artefactes públic per excel·lència de Java. Aquests repositori l'ofereix Apache i es possible pujar les llibreries pròpies en aquest repositori seguint uns certs requeriments de qualitat. Les llibreries pujades en aquest repositori es consideren que poden ser utilitzades de forma publica, sempre seguint certes limitacions segons la llicencia que mantenen.

Apache Archiva

Archiva és un repositori d'artefactes creat per Apache que pot ser desplegat per mantenir les llibreries y desplegables guardats de manera privada. Ofereix eines de control i seguretat d'accés, *proxy*, cerca d'artefactes i eines de *reporting* entre d'altres. Com la majoria de productes Apache, està construït sobre la llicencia *Apache versió 2.0*.

Artifactory

Artifactory és un projecte *open-source* de JFrog's. La gran comunitat sobra la que es sosté fa que sigui un software molt potent que no té res a envejar als softwares propietaris. Alta configuració en els diferents permisos de seguretat, una gran eficiència gràcies a mantenir localment còpies d'artefactes externs i la seva estabilitat, són els principals al·licients d'aquest software. Artifactory està sobre una llicencia *Lesser GNU General Public License 3.0* que permet la seva utilització i modificació sempre que el producte final contingui la mateixa llicencia o alguna més permissiva, així com també una versió comercial amb algunes funcionalitats extres i suport.

Sonatype Nexus

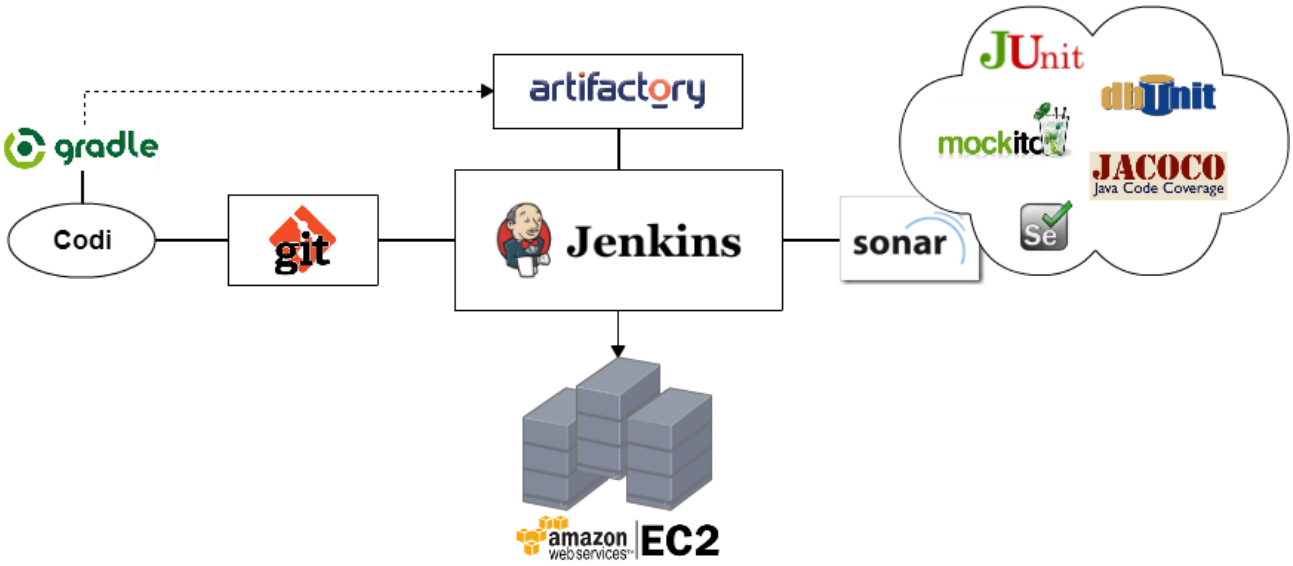
Sonatype Nexus és un dels repositoris d'artefactes que està creixent més actualment degut a la seva alta seguretat i altes prestacions. Però, els principals moviments d'aquest repository és a través de la seva llicencia comercial que incorpora tot un conjunt de funcionalitats complementaries per una integració total en un sistema d'integració continua. També cal dir, que disposa d'una versió gratuïta, bastant limitada, sobre una llicencia *Eclipse Public License 1.0* que permet el seu us, distribució i modificació seguint unes mínimes normes.

Altres repositoris d'artefactes

Altres softwares de repositoris d'artefactes: *MyGet*, *Package Drone*.

5.2. Esquema de la solució amb software

Després de veure les diferents eines que ens proporciona el mercat, i tenint en compte les funcionalitats que ofereixen, llicencies, facilitat d'instal·lació i personalització i coneixement personal, s'han escollit els següents components:

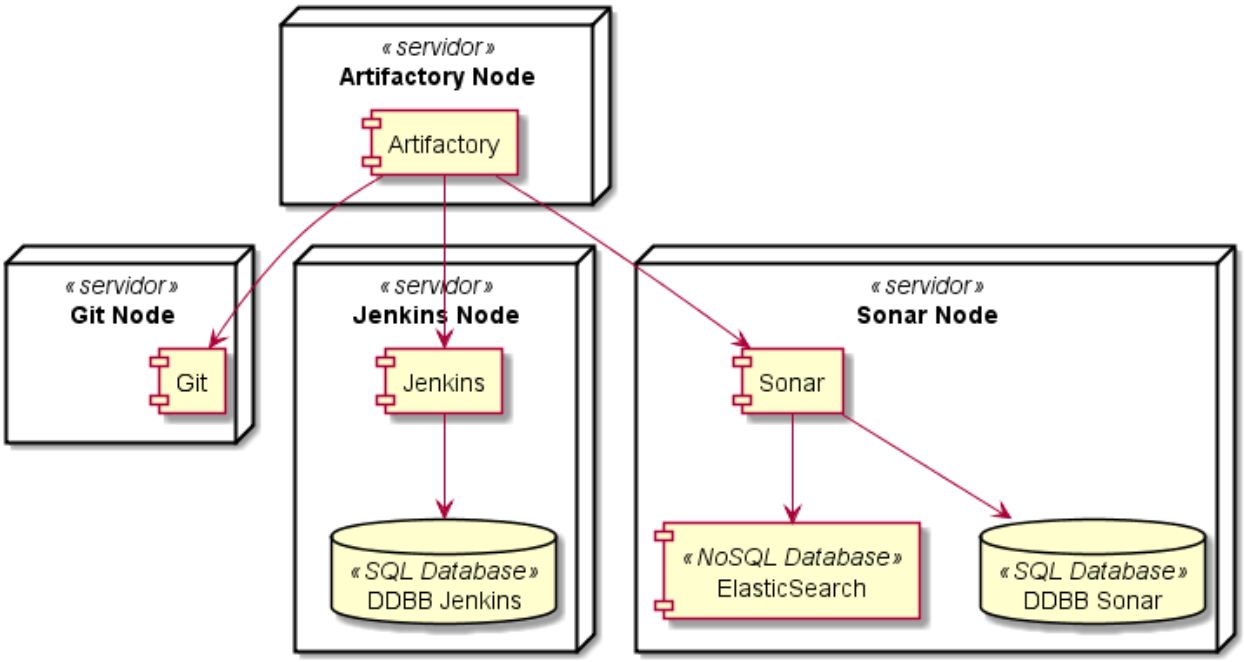


Imatge 11: Arquitectura del sistema d'integració continua a construir.

Gradle ens proporciona una forma completa, fàcil i amb un codi molt net per generar els nostres projectes i els nostres desplegables. **Git** és un repositori fàcil d'utilitzar, que ens ofereix un gran nombre d'eines i que la seva estructura d'un servidor local i un servidor remot, ens ofereix una seguretat molt alta. **Jenkins** serà el nostra servidor d'integració continua, amb el qual podrem generar tasques per cadascuna de les nostres necessitat. **Artifactory** ens proporciona un repositori d'artefactes intern per emmagatzemar els nostres JAR o WAR. Finalment, **Sonar** te una integració perfecta amb Jenkins i te una alta compatibilitat amb les eines de *testing* que utilitzarem (JUnit, DBUnit, Mockito, Jacoco, Selenium, etc). A partir d'aquestes eines, Sonar és capaç d'obtenir la informació de qualitat de codi d'una manera molt clara. Per últim, utilitzarem un **servidor d'Amazon EC2** per utilitzar-lo com a servidor, on desplegarem el nostra sistema d'integració continua i l'utilitzarem com a entorn d'integració per la nostra aplicació.

5.3. Desplegament i configuració

Una vegada seleccionat els diferents components software que volem incorporar al nostra sistema d'integració continua, farem el seu desplegament i configuració. El diagrama de desplegament idoni seria el mostrat en el diagrama següent[14], on cada node representa un servidor o màquina virtual diferent:



Imatge 12: Diagrama de desplegament idoni del sistema d'integració continua dissenyat.

Els capítols següents explicarem en detall com realitza la instal·lació i configuració de cadascun dels diferents elements del nostra sistema d'integració continua.

5.3.1. Servidor Amazon EC2

Amazon EC2 disposa d'un simple paquet gratuït amb el que és possible fer simples desplegament al *cloud* i fer proves amb les diferents eines que Amazon ens proporciona. En aquest cas em optat per escollir aquest paquet per crear una maquina virtual per desplegar el nostra sistema d'integració continua i mantenir el

[14] Degut a la falta de servidors disponibles, el desplegament dels diferents aplicatius es realitza en un mateix servidor, juntament amb l'entorn d'integració.

nostra entorn d'integració allà. Si després de realitzar els desplegaments veiem que es necessari augmentar les prestacions del nostra servidor, gràcies a Amazon, podrem fer-ho de manera totalment transparent[15].

La construcció de la màquina és bastant fàcil i està molt ben documentada per part d'Amazon. Els passos concrets es poden buscar amb facilitat però per fer un resum són els següents:

- Crear un usuari a Amazon.
- Crear una instancia amb una AMI [16] determinada. Aquí és possible la selecció de varies característiques depenent de les necessitats.
- Configura el grup de seguretat per permetre l'accés a la màquina virtual des de certes IPs (o rang de IPs).
- Descarregar el fitxer amb la clau per poder accedir per SSH.

Amb aquests passos es té creada una instancia a Amazon, que pot ser arrancada i accessible per SSH amb el fitxer de la clau[17] i des del grup de seguretat configurat.

Finalment, en aquest cas en concret, les característiques escollides són:

- Instancia: **t2.small**
- AMI: **Amazon Linux AMI 2015.09.0 x86_64 HVM GP2**
- RAM: **2 GB**
- Disc dur: **8 GB**

Al accedir per SSH a la màquina virtual podem obtenir varia informació:

```
$ cat /etc/os-release
NAME="Amazon Linux AMI"
VERSION="2015.09"
ID="amzn"
ID_LIKE="rhel fedora"
VERSION_ID="2015.09"
PRETTY_NAME="Amazon Linux AMI 2015.09"
ANSI_COLOR="0;33"
CPE_NAME="cpe:/o:amazon:linux:2015.09:ga"
HOME_URL="http://aws.amazon.com/amazon-linux-ami/"
```

Com podem veure tenim un sistema operatiu **Fedora** i amb l'eina *yum* que ofereix podem fer instal·lacions de manera molt ràpida. Però primer de tot farem un *update* de tot el sistema amb *yum*[18]:

[15] Finalment, i degut a la quantitat de recursos que necessiten les aplicacions (Sonar, Jenkins, Artifactory, etc.) en conjunt, s'ha hagut d'augmentar un nivell les prestacions del servidor en Amazon. Utilitzant un paquet no gratuït, però d'un cost inferior a 30€ al més a ple rendiment.

[16] Una AMI no deixa de ser una imatge virtual d'un sistema operatiu en concret però per el sistema d'Amazon.

[17] Depenen de l'eina utilitzada o del sistema operatiu utilitzat per accedir, és necessari seguir una sèrie de passos explicats en http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AccessingInstances.html (consultat el 23/11/2015).

[18] Això és una recomanació d'Amazon al iniciar per primera vegada una imatge.

```
$ sudo yum update
```

D'aquesta manera tenim ja el sistema completament llest per començar el desplegament del nostra sistema d'integració continua i qualitat de codi.

5.3.2. Jenkins

Per fer la instal·lació de Jenkins hem d'incloure un nou repositori a l'eina *yum* que ens ofereix el sistema operatiu, degut a que l'instal·lable de Jenkins no està inclòs en els repositoris que estan per defecte:

```
$ wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/redhat/jenkins.repo
$ rpm --import https://jenkins-ci.org/redhat/jenkins-ci.org.key
```

Amb el repositori inclòs en els repositoris de *yum* ja es possible la instal·lació de Jenkins:

```
$ yum install jenkins
```

Una vegada instal·lat disposarem d'un arxiu de Jenkins on poder configurar gran quantitat de propietats. En el nostre cas només és necessari canviar el port on Jenkins es desplegarà, perquè no hi hagi conflicte amb Tomcat (que per defecte escolta al 8080). En el nostra cas l'arxiu de configuració es troba en:

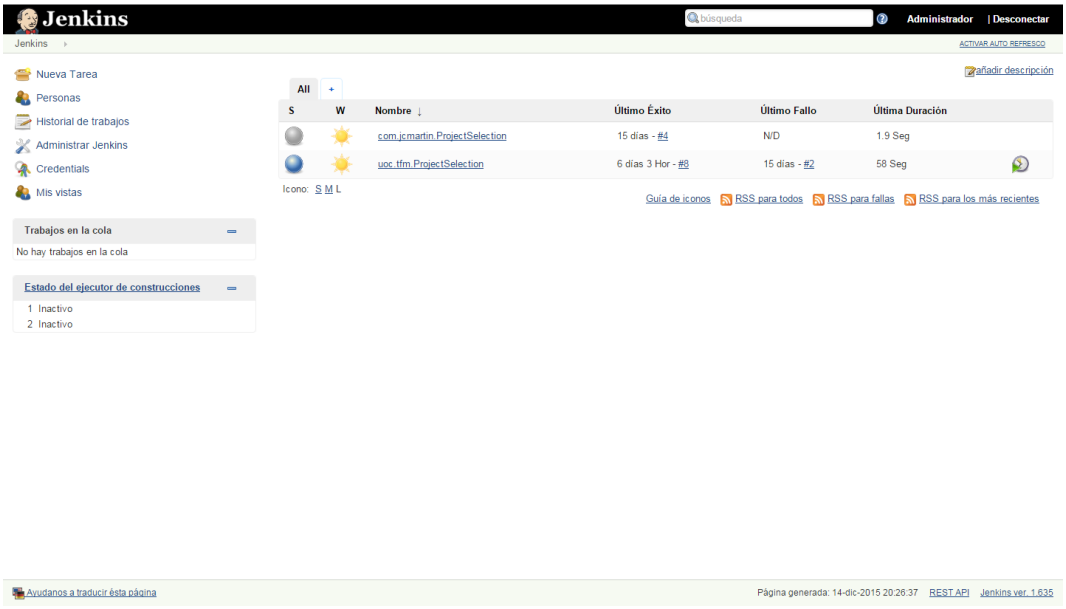
```
$ vi /etc/sysconfig/jenkins
```

Amb tot això ja es pot arrancar el servei, i per que sigui més correcte, es configura el servei de Jenkins per que s'engegui al arrancar el servidor utilitzant la comanda *chkconfig*.

```
$ chkconfig jenkins on
$ service jenkins start
```

Amb aquests passos hem de poder veure, després d'autenticar-nos, la pàgina d'inici de Jenkins en el host y el port configurat[19]:

[19] Per defecte, l'usuari administrador és *admin*, amb el mateix *password*, però pot ser configurat posteriorment.



Imatge 13: Pàgina inicial de Jenkins

5.3.3. Git

Ja que Git és distribueix com 2 repositoris, un de local i un de remot, cal fer la instal·lació de les dues parts. La instal·lació del servidor local depèn del sistema operatiu que es tingui en l'entorn de desenvolupament. Aquí ens centrarem en la instal·lació del repositori remot en el nostre servidor. Gràcies al sistema operatiu que tenim i gràcies amb la eina *yum*, la instal·lació es fa de manera fàcil, de la manera següent:

```
$ sudo yum install git
```

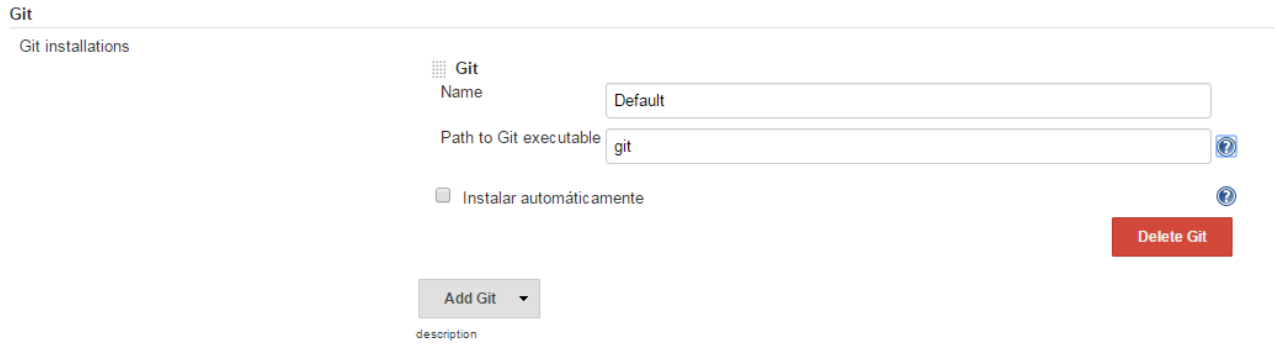
Amb això ja realitzem la instal·lació de manera automàtica de Git. Ara només és necessari crear el repositori on emmagatzarem el codi generat. En el meu cas, he escollit la carpeta `/opt/repository` i es realitza de la manera següent:

```
$ sudo mkdir /opt/repository
$ cd /opt/repository
$ sudo git init
```

Amb això disposem d'un repositori Git. A través de la direcció següent es pot accedir en aquests repositori:

```
ssh://<host>:22@<user>/opt/repository
```

Per acabar la configuració del repositori dintre del nostra sistema d'integració continua, es necessari integrar el repositori Git dintre de Jenkins. Per fer això només caldrà instal·lar el *plugin* de Git a Jenkins, a través de l'eina d'administració de *plugins*, i configurar-lo, a la configuració de sistema, de la manera següent:



Imatge 14: Configuració del plugin de Git a Jenkins

5.3.4. Sonar

Com moltes de les funcionalitats anteriors utilitzarem *yum* per fer la instal·lació principal del servidor de Sonar però al igual que Jenkins, es necessari incloure primer el repositori adequat:

```
$ wget -O /etc/yum.repos.d/sonar.repo http://downloads.sourceforge.net/project/sonar-pkg/rpm/sonar.repo
$ yum install sonar
```

Sonar utilitza dues bases de dades per emmagatzemar les dades, una base de dades no relacional i una d'altra relacional. Per defecte, la base de dades relacional bé incorporada i configurada per executar-se amb Sonar de manera correcte, aquesta no cal configurar-la si es vol mantenir el funcionament per defecte. Per la base de dades relacional cal configurar-la.

Per configurar la base de dades cal incorporar els paràmetres següents del arxiu de configuració de Sonar anomenat *sonar.properties* dintre de la carpeta *conf*:

```
sonar.jdbc.url=<cadena de connexió a DDBB>
sonar.jdbc.username=<username>
sonar.jdbc.password=<password>
```

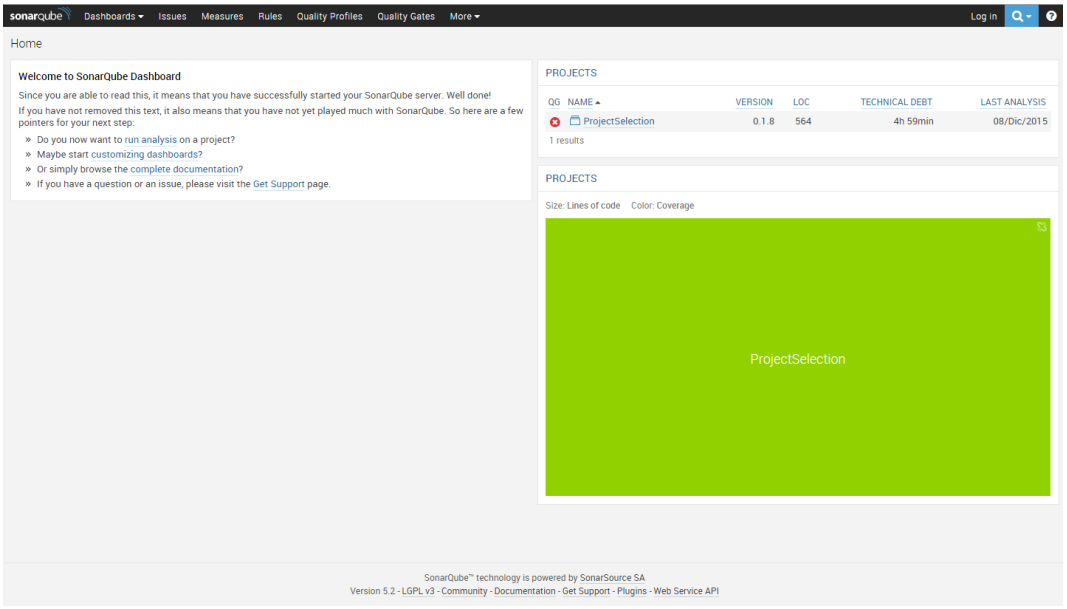
Per defecte Sonar està configurat per desplegar la seva interfície gràfica al port 9000, però, en aquest mateix arxiu de configuració, pot ser modificada amb la propietat:

```
sonar.web.port=9000
```

Amb aquesta configuració, ja pot ser arrancada la nostra eina de qualitat de codi Sonar:

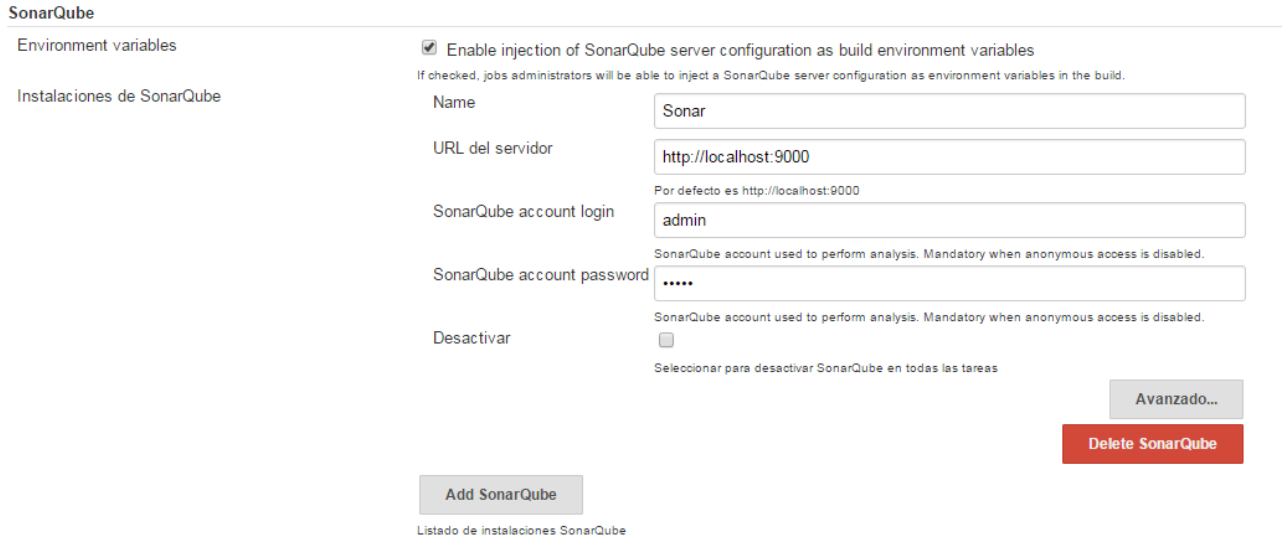
```
$ service sonar start
```

Si tot es correcte hauria de sortir una finestra semblant a la següent:



Imatge 15: Pantalla inicial de Sonar

Per incorporar sonar a Jenkins, cal seguir 3 passos. Primer cal instal·lar el *plugin* de Sonar en Jenkins, a través de l'eina d'instal·lació de *plugins*, i configurar-lo de la manera següent:

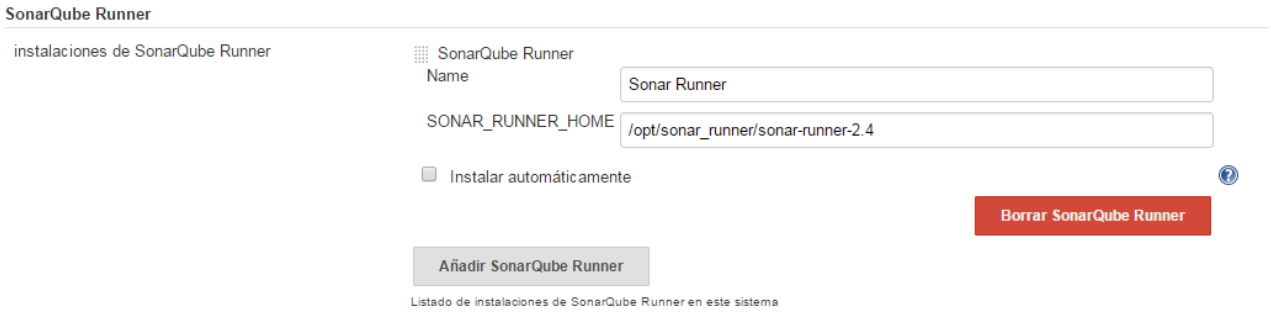


Imatge 16: Configuració del plugin Sonar a Jenkins

Aquesta configuració és en el meu cas en concret i ha de ser adaptada segons cada cas. Seguidament caldrà instal·lar l'eina que s'encarrega de fer les execucions dels tests, extreure els resultats i enviar-los a Sonar, aquesta eina és el Sonar Runner. Per instal·lar Sonar Runner el descarregarem i instal·larem de la manera següent:

```
$ wget http://repo1.maven.org/maven2/org/codehaus/sonar/runner/sonar-runner-dist/2.4/sonar-runner-dist-2.4.zip
$ unzip sonar-runner-dist-2.4.zip
```

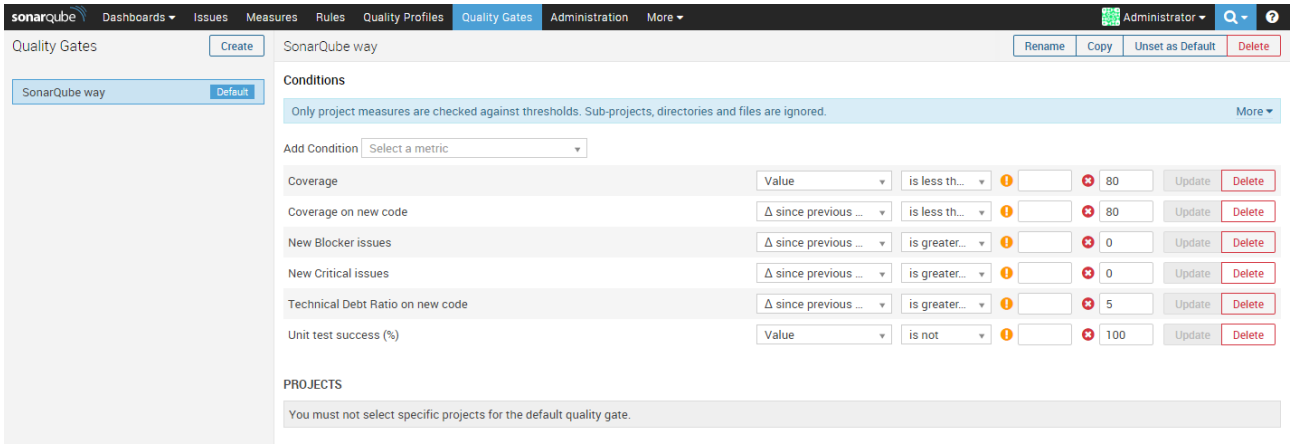
Amb això tenim un Sonar Runner que em de copiar a la carpeta on vulguem mantenir-lo. Seguidament passarem a configurar el Sonar Runner a Jenkins a través de l'eina de gestió de *plugins* i farem la configuració de la manera que es mostra a continuació:



Imatge 17: Configuració del Sonar Runner en Jenkins

En el nostra cas, el Sonar Runner es troba a `/opt/sonar_runner/sonar-runner-2.4` però caldrà adaptar aquesta opció a les necessitats pròpies.

Finalment ja tenim l'eina Sonar configurada per poder ser utilitzada en les tasques de Jenkins, però per acabar de configurar Sonar, caldrà que indiquem els criteris de qualitat propis de cada persona, empresa o projecte, els anomenats *"Quality Gates"* per Sonar. Això són simplement quins criteris donem per donar com a bona o dolenta una *build*. Criteris com el tant per cent de tests correctes o la cobertura mínima que ha de tenir un projecte per considerar-se correcte, son configurats en aquest punt. Sonar a través del menú de *Quality Gates* permet configurar de manera molt simple aquests criteris:



Imatge 18: Pantalla de Quality Gates de Sonar

En el meu cas s'han incorporat alguns criteris nous als que estan per defecte en tots el projectes, en concret en assegurar que el 100% dels test passen i que la cobertura dels test es superior a un 80%:

5.3.5. Artifactory

Instal·larem Artifactory, la versió 3.9.5 [20]. Per fer això primer descarregarem i instal·larem el fitxer d'instal·lació RPM (Red Hat Package Manager):

```
$ wget https://bintray.com/artifact/download/jfrog/artifactory-rpms/artifactory-3.9.5.rpm
...
$ yum localinstall artifactory-3.9.5.rpm
```

Amb això ja tindrem disponible el servei ***artifactory***. El problema és que per defecte el port utilitzat per *artifactory* és el 8081, on tenim allotjat el nostra Jenkins. Per aquest motiu primer modificarem el port en el que es desplega Artifactory. Per fer això podem veure que es desplega en un Tomcat, per tant, tindrem que modificar el port en el *server.xml*:

```
$ vi /opt/jfrog/artifactory/tomcat/conf/server.xml
```

On modificarem el:

```
<Connector port="8081">
```

Pel port que vulguem, en el nostra cas el 8082:

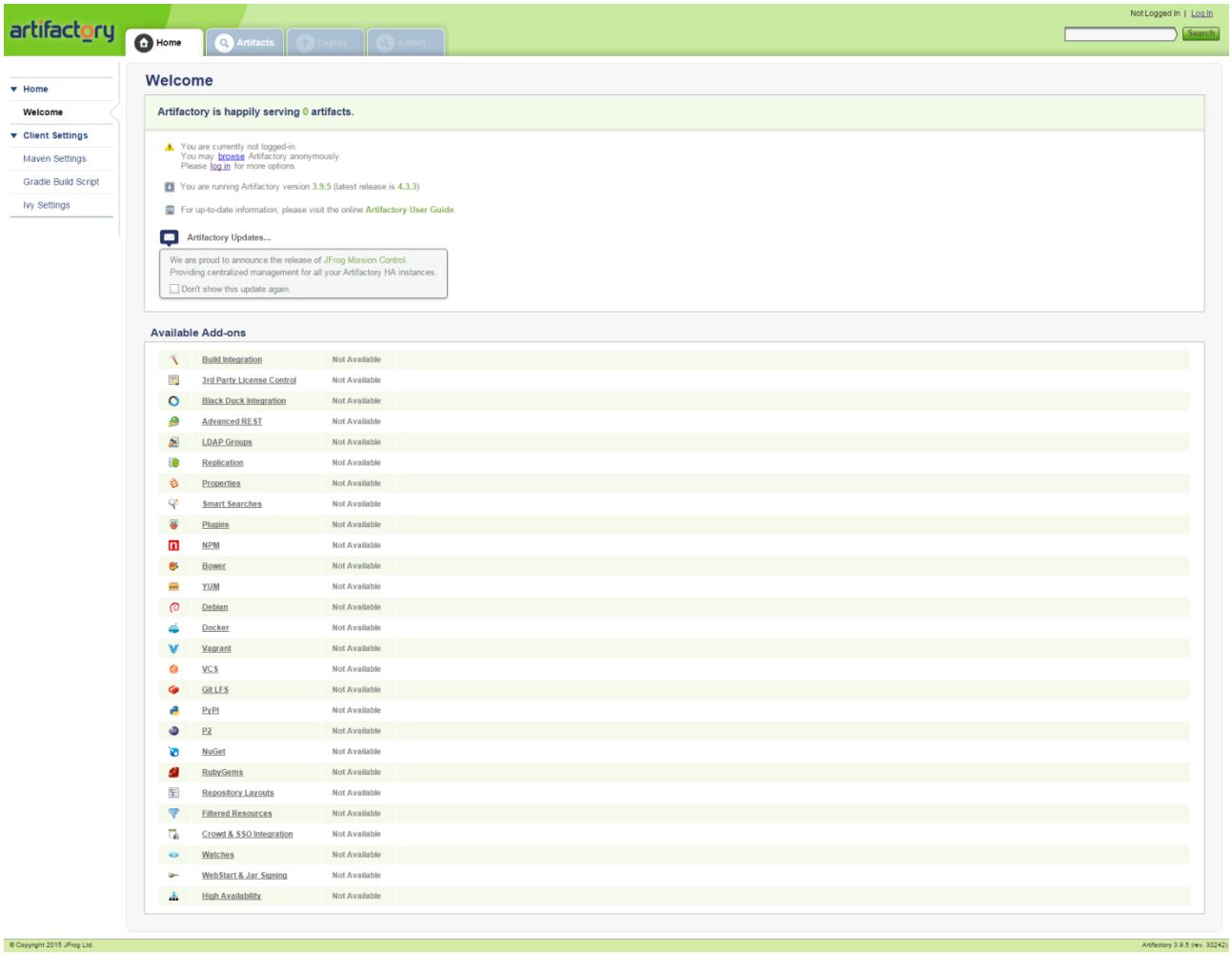
```
<Connector port="8082">
```

Y després ja podrem executar el nostra servei *artifactory*:

```
$ sudo service artifactory start
```

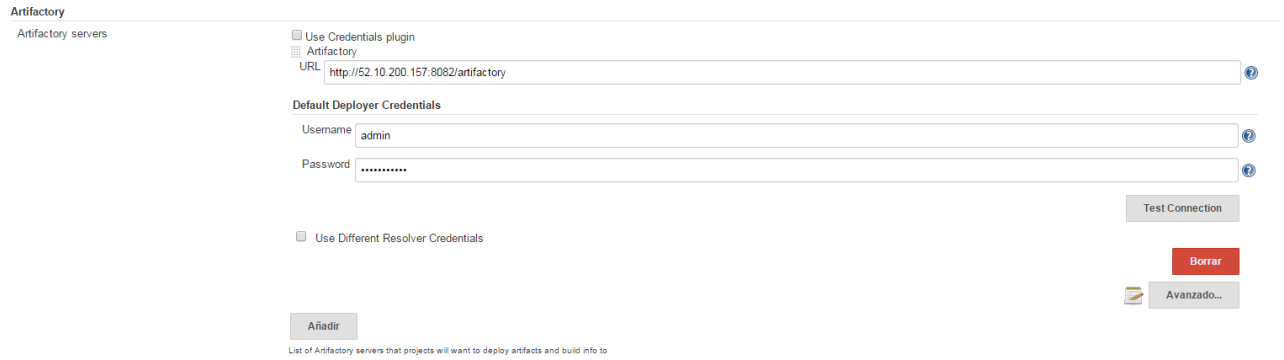
Si tot ha sigut correcte, trobarem la pàgina d'inici d'Artifactory següent al port escollit:

[20] A partir de la versió 4 d'Artifactory és necessària la versió 8 de Java. En el nostra sistema ja tenim instal·lat la versió 7 de Java i pel moment no volem modificar-la. Per altra banda la selecció d'una versió inferior d'Artifactory no causa cap problema en el nostre cas.



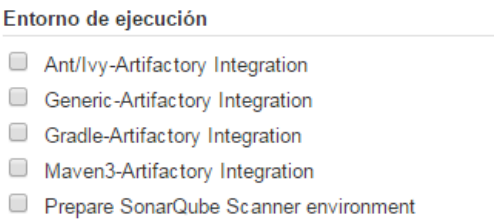
Imatge 19: Pàgina d'inici d'Artifactory

Com a últim pas, comprovarem que tenim instal·lat el *plugin* Artifactory en Jenkins. En cas de que no estigui instal·lat, serà necessari que l'instal·lem i que el configurem en l'administració de sistema:



Imatge 20: Configuració de Artifactory en la configuració del sistema de Jenkins

Una vegada instal·lat tindrem noves opcions al configurar la nostra tasca de Jenkins:



Imatge 21: Opcions que ens aporta el plugin d'Artifactory a les tasques de Jenkins

5.4. Projecció de futur

Hem aconseguit configurar un sistema d'integració continua bastant complet, però com sempre podria ser millorat en alguns aspectes si es volgués incorporar de manera professional. Els punt següents mostren projeccions de futur, que podrien ser aplicades en un futur proper:

- El pas més important i prioritari és fer un desplegament en l'arquitectura correcte. Degut a disposar d'un sol servidor, tot el desplegament s'ha realitzat en aquest servidor, amb els problemes de rendiment que això comporta. Per tenir un sistema amb un bon rendiment cal desplegar el sistema d'integració continua en un entorn idoni com s'especifica en Imatge 12: Diagrama de desplegament idoni del sistema d'integració continua dissenyat.
- Caldria tenir l'ultima versió dels diferents components que tenim. El més imminent tindria que ser Artifactory, on tindríem que utilitzar *environments* per que utilitzes una versió de Java diferent de la resta del sistema, en concret, Java 8. D'aquesta manera podríem fer l'*upgrade* a la versió 4 d'Artifactory.
- Caldrà fer un estudi de quines són les eines que s'utilitzaran en els projectes que s'utilitzaran en Jenkins. Una vegada disposat d'aquestes eines, podríem instal·lar a Jenkins els *plugins* necessaris, com podrien ser *plugins* per integrar altres repositoris de codi o artefactes, o tasques que executin altres funcions com scripts de diferents tipus, projectes Maven, etc.
- Per completar el punt anterior, caldrà fer instal·lacions d'altres *runners* que puguin executar altres tipus de projectes, com podrien ser *runners* per projectes Android, projectes PHP o projectes .NET.
- Per implementar aquest sistema en una companyia, caldria fer una revisió i configuració completa dels perfils d'usuaris que es permeten utilitzar en el sistema. Actualment existeix el rol d'administrador per cadascuna de les eines, però caldria incorporar rols de visualitzador i rols de desenvolupadors, així com possiblement alguna mena de rol híbrid pels cap d'equip, amb visualització de les dades del projecte i amb certa possibilitat de configuració d'alguns aspectes.

6. Desenvolupament d'una aplicació web amb el sistema d'integració continua

> *"The number one benefit of information technology is that it empowers people to do what they want to do. It lets people be creative. It lets people be productive. It lets people learn things they didn't think they could learn before, and so in a sense it is all about potential."*
>
> **Steve Ballmer**

L'objectiu que es desenvoluparà en aquests projecte de final de màster, és la creació d'una simple aplicació utilitzant el sistema d'integració continua desplegat anteriorment. Això ens permetrà avaluar i extreure conclusions sobre la utilitat i funcionalitat que ens aportat el nostre sistema d'integració continua i si ens ha facilitat la programació i desplegament, o per contra, ens dificulta el desenvolupament del projecte.

La aplicació que es desenvolupa a vingut donada per la pròpia necessitat d'escollir un projecte de final de màster. Molts professors tenen en ment projectes de final de grau o màster que podrien ser molt interesants, però no es duen a terme degut a que els alumnes no poden veure'ls abans d'escollir un àrea concreta pel treball. L'aplicació que es desenvoluparà solucionarà aquests problema, permetrà als professors crear una sèrie de projectes, amb una sèrie de dades, i els alumnes podran accedir, veure els treballs creats, i assignar-se a un d'ells en cas que els interessi desenvolupar-lo.

Aquestes són les bases generals del projecte, per dur-lo a terme cal primer un estudi de les funcionalitats concretes que ha de complir i veure quins requisits calen en cada cas. Seguidament es dissenyarà la solució en forma de components i com es comuniquen entre ells, per després assignar *frameworks* de desenvolupament per cadascun dels mòduls. Amb aquesta informació es podran començar a desenvolupar les funcionalitats concretes de l'aplicació, però ja que ens interessa utilitzar el nostra sistema d'integració continua, s'explicarà com configurar el projecte per integrar-lo en el nostre cicle d'integració continua, que ens permetrà veure la qualitat de codi en cada moment i realitzar els desplegaments d'una forma més automàtica.

6.1. Requisits i funcionalitats

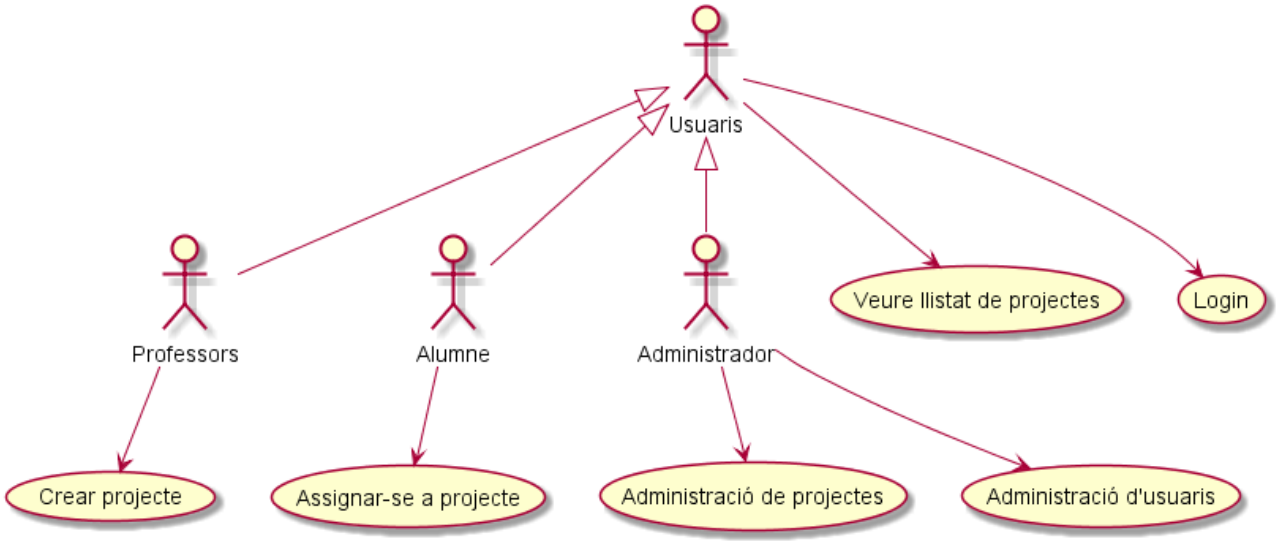
El funcionament bàsic de l'aplicació és senzill, una sèrie de professors creen projectes amb diferents paràmetres (títol, descripcions, àrea, etc.) i els alumnes es poden assignar en aquests projectes creats.

A part d'aquest funcionament bàsic, tenim altres requisits que ha de complir l'aplicació perquè sigui funcional i útil:

- Tots els usuari s'han d'autenticar amb un *login* i un *password* proporcionats de forma externa a l'aplicació.
- Els usuaris han de poder filtrar els projectes segons alguns criteris.
- Els professors poden modificar o eliminar, únicament, els seus projectes.
- Els alumnes han de poden veure en quin projecte estan assignats actualment.
- Els alumnes han de poden eliminar l'assignació feta per seleccionar un altre diferent.
- Ha d'existir un administrador que puguin crear, modificar o eliminar, qualsevol usuari o projecte.

6.1.1. Perfils d'usuari

El diagrama de casos d'us següent mostren els diferents perfils d'usuari que existeixen a l'aplicació:



Bàsicament existeixen 3 perfils d'usuaris, on tots 3 estenen d'un tipus d'usuari genèric. Tots els usuaris, com a usuaris genèrics podran accedir a l'aplicació, a partir d'un *login* i *password* proporcionats, i veure el llistat de projectes que estan actualment actius, així com filtrar-los per certs paràmetres o veure el detall d'algun d'ells.

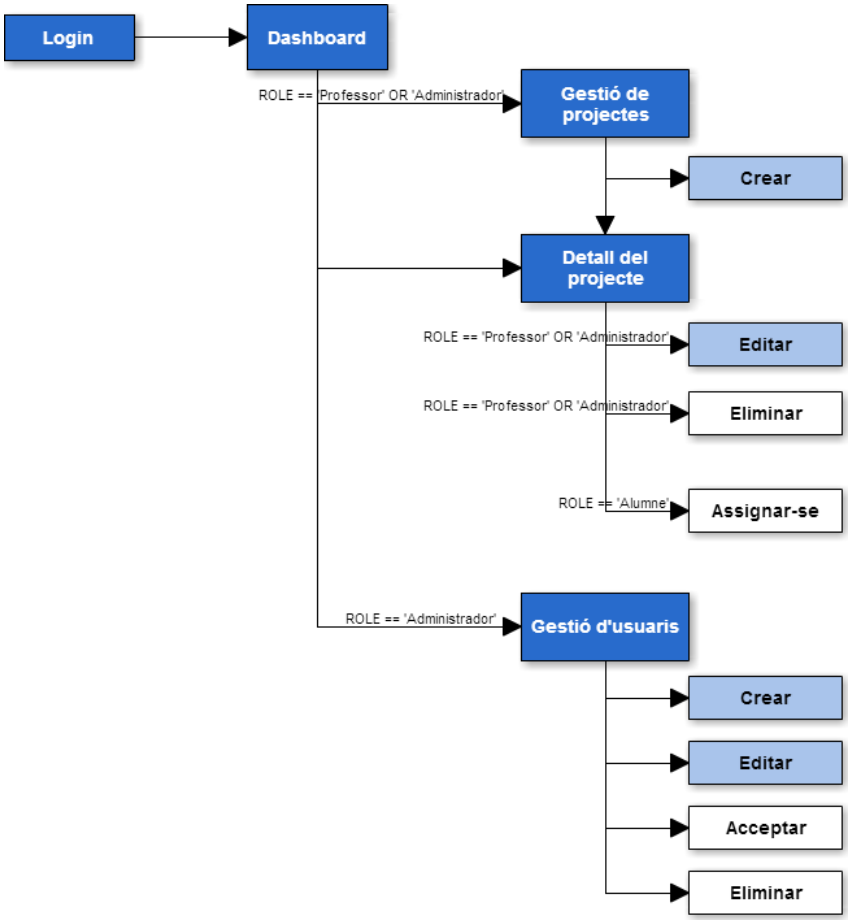
L'usuari administrador és el màxim gestos de l'aplicació web. Les accions que compleix, apart de les que li ofereix ser un usuari genèric, són les gestions d'administració, en concret, l'administració dels projectes i dels usuaris. L'administració inclou crear, editar i eliminar aquests elements.

Els professors són els encarregat de crear els projectes (apart del que podria crear un administrador). Aquests projectes que crea un mateix podran ser modificats o eliminats pel mateix usuari que els crea.

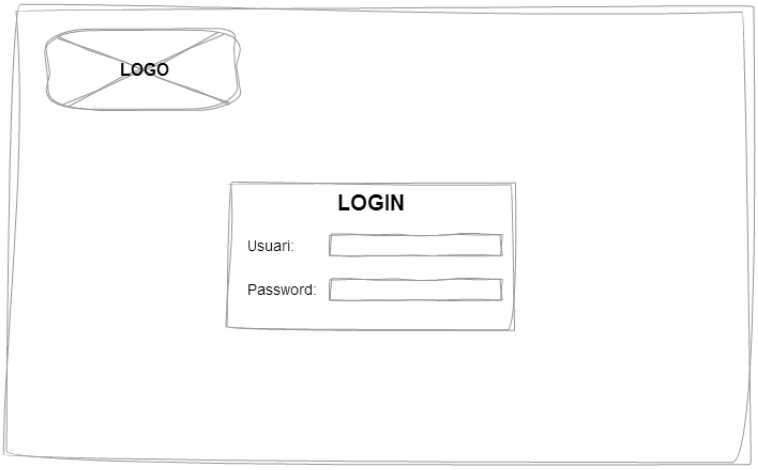
Els alumnes són els que, després de veure els diferents projectes, poden assignar-se (o eliminar l'assignació) a un projecte. Aquests projectes seleccionats mostraran que existeix un alumne ja assignat al projecte.

6.1.2. Mapa web de l'aplicació

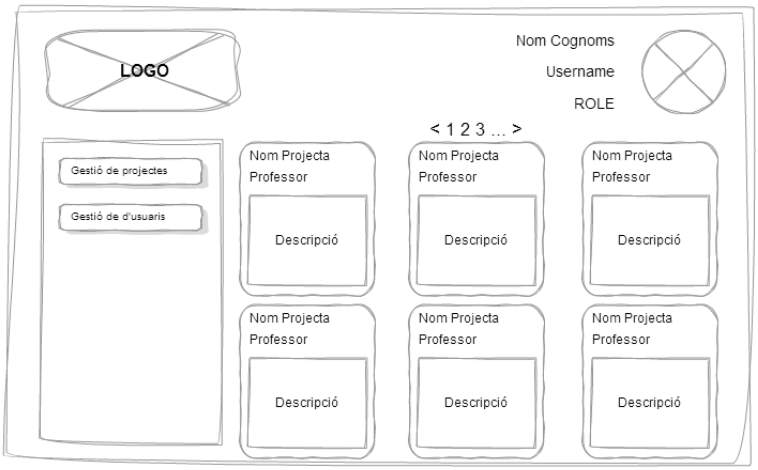
El següent mapa mostra les diferents pantalles o accions que es realitzen en l'aplicació web i alguna simple condició per poder accedir en aquestes:



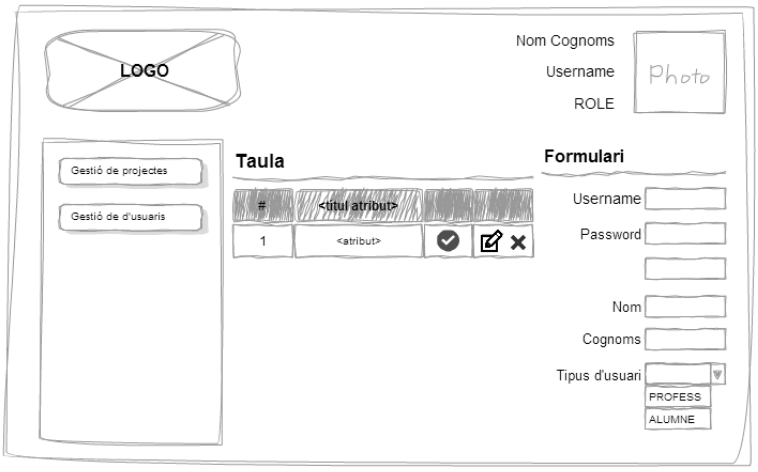
6.1.3. Mockups



Imatge 22: Mockup de la pàgina de login



Imatge 23: Mockup de la pàgina inicial



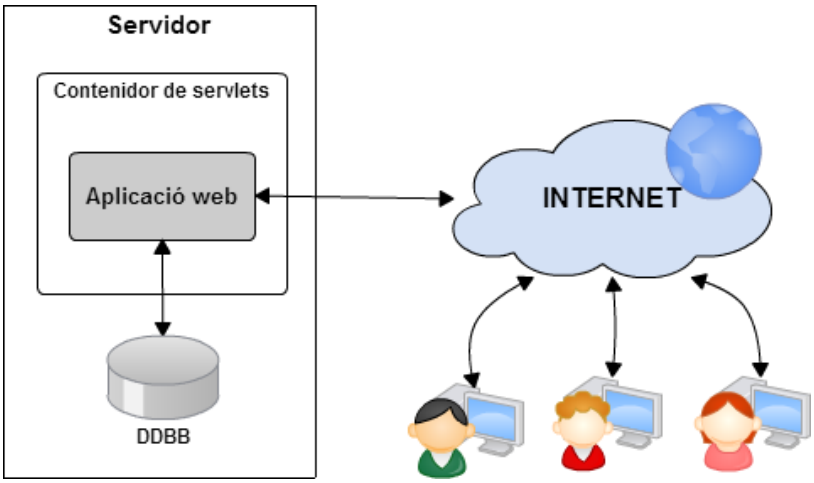
Imatge 24: Mockup de pàgines de gestió, tant d'usuaris com de projectes

6.2. Disseny i arquitectura de l'aplicació

Seguidament mostrarem l'arquitectura, tant de desplegament general com interna de l'aplicació, i el model de dades que utilitza.

6.2.1. Arquitectura de desplegament i d'aplicació

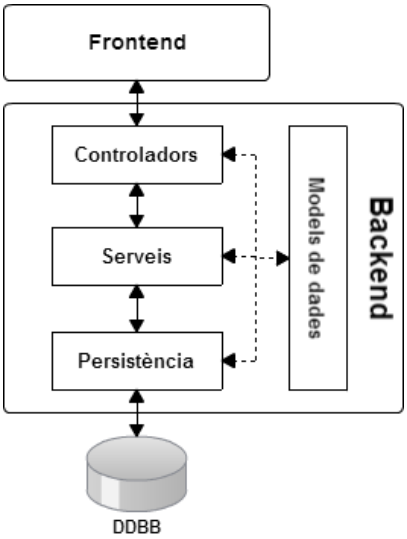
L'arquitectura de desplegament és simple, ja que només es composa d'una aplicació web senzilla. L'esquema de l'arquitectura queda de la següent forma:



Imatge 25: Arquitectura de desplegament

Com es pot veure la nostra aplicació només viu en un sol servidor ja que l'aplicació web i la base de dades que manega, no tenen un consum suficientment elevat per que sigui necessari la seva divisió en dos servidors, un que tingues únicament l'aplicació i un d'altre que únicament mantingues la base de dades.

Amb l'arquitectura de l'aplicació s'ha tingut molta més cura, i s'ha realitzat una divisió interna pensada per tenir uns bons estàndards de qualitat i que sigui el més desacoblada, escalable i mantenible possible:



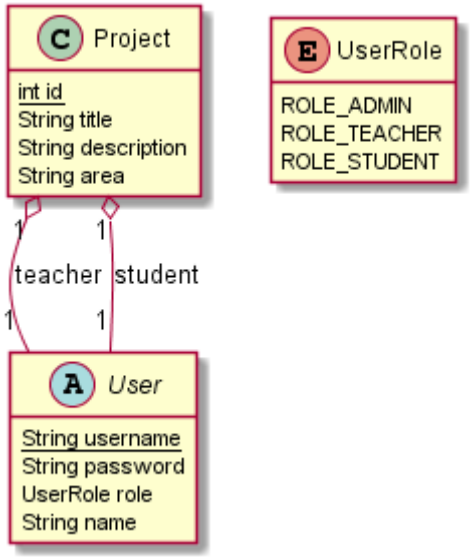
Imatge 26: Arquitectura de l'aplicació

Com és lògic la primera gran divisió és el ***frontend***, que representa tota la visualització per l'usuari de l'aplicació, i el ***backend***, que representen totes les accions que realitza la nostra aplicació per oferir les funcionalitats. La segona gran divisió, es la divisió i comunicació dintre del *backend*:

- **Models de dades:** Els models de dades són la representació dels objectes que hi haurà emmagatzemats en base de dades.
- **Persistència:** La capa de persistència o DAO (*Data Access Object*) es la que s'encarrega de treballar directament amb la base de dades. A partir dels models de dades, podrà consultar els diferents models, crear-los, modificar-los o eliminar-los.
- **Serveis:** La capa de serveis és aquella que realitza les tasques que tenen a veure amb la gestió. Si s'han de realitzar accions prèvies o posteriors al guardar un model de dades, o incloure paràmetres de control als models quan es fa una edició, aquesta és la capa encarregada d'això. Aquesta capa utilitzarà les funcions ofertes per la capa de persistència per realitzar les tasques que necessiti.
- **Controladors:** La capa de controladors és la interfície entre la visualització i les tasques de gestió. S'encarrega de recopilar les dades necessàries i transformar-les com sigui necessari per que siguin enteses per el *frontend*. Aquesta capa utilitzarà les eines que la capa de serveis li ofereixi i mai utilitzarà la capa de persistència directament.

6.2.2. Model de dades

El diagrama següent mostra el model de dades que s'utilitza:



Imatge 27: Model de dades de l'aplicació

Nom	UserRole
Tipus	Enumerador
Descripció	Indica els diferents tipus de rols que conviuen a l'aplicació.
Valors	
Nom	**Descripció**
ROLE_ADMIN	Indica el rol d'administrador del sistema.
ROLE_TEACHER	Indica el rol de professor i responsable de la creació de projectes.
ROLE_STUDENT	Indica el rol d'alumna del sistema.

Taula 2: Especificacions del model de dades de UserRole

Nom	User
Tipus	Classe
Descripció	Aquesta classe representa tots els tipus d'usuaris que existeixen el sistema i conté els paràmetres comuns a tots ells.
Atributs	
Nom	**Descripció**
username	Identificador únic amb el que es coneix i s'autentifica l'usuari al sistema.
password	Clau encriptada necessària per l'accés al sistema.
role	Rol que te l'usuari dintre del sistema.
name	Nom del usuari.

Taula 3: Especificacions del model de dades de User

Nom	Project
Tipus	Classe
Descripció	Conté els diferents paràmetres amb la informació d'un projecte.

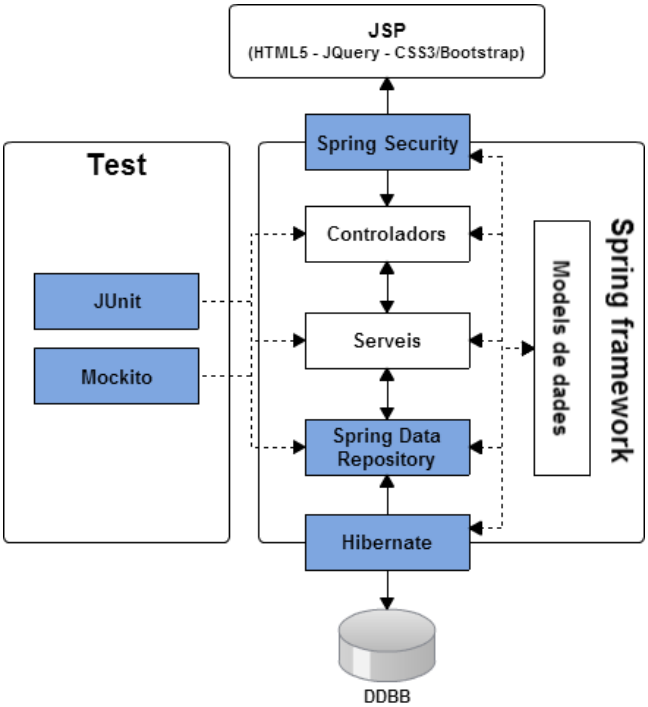
Atributs	
Nom	**Descripció**
id	Identificador únic auto-generat del projecte.
title	Títol del projecte.
description	Descripció completa del projecte.
area	Àrea a la que pertany el projecte.
teacher	Usuari creador del projecte.
student	Alumne assignat en aquests projecte. Opcional.

Taula 4: Especificacions del model de dades de Project

6.3. Plataformes i *frameworks* de desenvolupament

Una de les grans facilitats que té Java, és la gran comunitat que hi ha, per això és molt fàcil trobar eines que ens facilitin la feina a l'hora de crear qualsevol tipus d'aplicació amb Java, ja sigui una aplicació d'escriptori, una llibreria o una aplicació web com en aquest cas.

Per aquests motiu, a partir del coneixement propi, s'ha millorat l'esquema de l'arquitectura de l'aplicació web Imatge 26: Arquitectura de l'aplicació per incorporar una sèrie de *frameworks* que ens facilitaran la feina, ja sigui per temes de seguretat, persistència, configuració de *servlets* i *testing*. L'arquitectura llavors queda de la següent forma:



Imatge 28: Arquitectura de l'aplicació incloent els framework a utilitzar

A continuació s'especifica per què s'utilitzen els diferents *frameworks* escollits i mostrats en el diagrama anterior.

Spring framework

Spring framework serà la base de la nostra aplicació. Aquest *framework* es tot un ecosistema de solucions software per el desenvolupament de projectes ràpidament. Entre aquestes podem destacar Spring Boot, per crear ràpidament aplicacions web que poden ser desplegades en qüestió de minuts, Spring Cloud, que integra una sèrie d'eines per arquitectures *cloud* com poden ser l'arquitectura de micro-serveis, o Spring Security, un conjunt d'eines per securitzar les aplicacions web, entre moltes altres[21].

Hibernate

Hibernate és una eina de mapeig objecte-relacional (ORM). El que ens permet aquesta eina és abstraure la relació entre els models de l'aplicació Java, com és en el nostre cas el usuaris i projectes, amb les taules que els emmagatzemen a la base de dades i les seves relacions.

L'altra gran capacitat que ens aporta Hibernate es la independència de l'aplicació sobra la base de dades amb la que treballa. Simplement modificant un paràmetre de la configuració de Hibernate, ens permet treballar sobra una base de dades MySQL o qualsevol altra, d'una manera transparent a l'aplicació.

Spring Data Repository

Spring Data Repositoy és la llibreria de *Spring* que treballarà a la capa de DAOs (*Data Access Object*), i que treballarà sobra Hibernate. Aquesta eina ens permet simplificar, en gran mesura, la programació d'aquesta capa, fent que ens preocupem únicament de que dades em de cercar i no el com.

Spring Security

Les llibreries de ***Spring Security*** ens aportaran tota la capa de seguretat de la nostra aplicació. En primer lloc ens aporta tot el *workflow* en l'autenticació dels usuaris a partir d'un *password* xifrat a base de dades i, a partir d'aquí, assignar als usuaris els rols que li pertoquin. En segon lloc, ens aporta tota la capa de autorització, es a dir, quins rols d'usuaris poden accedir als diferents serveis de la capa de serveis i controladors.

---

[21] https://spring.io/projects (consultat per ultima vegada el 27/12/2015). Aquí podreu trobar tots els projectes que ofereix Spring.

JavaServer Pages

Els **JSP (*JavaServer Pages*)** és la tecnologia que ens permetrà desenvolupar el nostra *frontend* amb informació dinàmica procedent del nostra servidor. La sintaxis de JSP es un HTML que incorpora *tags* XML addicionals amb els quals permeten incorporar altres funcionalitats on poden intervenir dades procedents del *backend*. A part de JSP, per la part de visualització utilitzarem CSS3 i Bootstrap per la part d'estil i JQuery per aprofitar les eines que ens aporta treballant sobra Javascript.

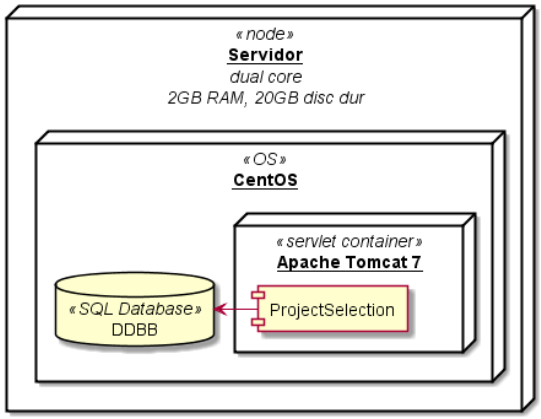
Per facilitar la feina en l'estil i visualització, i aprofitant la comunitat de Bootstrap que ofereix gran quantitat de *templates* gratuïts, s'ha escollit un *template* de Bootstrap totalment gratuït anomenat **KAdmin**[22].

Test unitaris

Finalment, i per aconseguir l'objectiu principal del nostra projecte de final de màster, les eines de *testing* que utilitzarem principalment són **JUnit i Mockito**. JUnit ens permet avaluar funcions i comprovar que el resultat esperat i el real coincideixen. Mockito el que ens proporciona es la capacitat d'aïllar funcions concretes sense tenir en compta el funcionament dels altres mòduls necessaris.

6.4. Desplegament de la solució

Degut a la mida reduïda de l'aplicació, el desplegament es pot fer en unes condicions y requisits mínims. El diagrama següent mostra el diagrama de desplegament, juntament amb els requisits bàsics, recomanats per un sol servidor:



Imatge 29: Diagrama de desplegament del projecte

[22] El *template* ha sigut descarregat de la web https://shapebootstrap.net/ que ofereix tant *templats* gratuïts com de pagament.

La base de dades podria residir en un servidor apart, però amb aquesta aplicació, que no necessita un gran consum de recursos, tindríem una pèrdua de velocitat per la comunicació amb la base de dades y no tindríem un guany en velocitat de processament.

Seguidament mostrarem com instal·lar en un servidor Fedora el software necessari per el desplegament de la nostra aplicació.

6.4.1. Instal·lació de MySQL

Amb la eina *yum* fem una instal·lació ràpida amb la comanda:

```
$ yum install mysql-server
```

Això ens instal·la MySQL amb un usuari administrador anomenat *root* sense *password*. Per configurar aquest *password*, així com altres opcions de seguretat cal executar la següent comanda y seguir les instruccions que indiqui:

```
$ sudo /usr/bin/mysql_secure_installation
```

Finalment per indicar què és un servei que ha de ser engegat al inici, amb el sistema operatiu, executarem la comanda seguen:

```
$ chkconfig mysqld on
```

Després de seguir aquests passos, i una vegada encès el servei de MySQL (o al reiniciar el servidor), podrem executar la comanda i comprovar que ens està funcionant:

```
$ mysql –u root <root_password>
Your MySQL connection id is 1257
Server version: 5.5.45 MySQL Community Server (GPL)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

A partir d'aquí serà necessari la creació d'una base de dades per utilitzar-la en la nostra aplicació i a la nostra aplicació, configurar aquesta base de dades en les propietats per l'entorn d'integració. Les propietats que estan relacionades amb la base de dades són:

- **db.hibernateDialect**: Dialecte utilitzat per Hibernate.
- **db.showSql**: Booleà que indica si es vol mostrar les peticions SQL en les traces de *log*.
- **db.hibernateAuto**: Indica l'acció a realitzar a l'inici de l'aplicació en relació a la base de dades. Els valor possibles són:
    - *validate*: comprova que la base de dades concorda amb el model de dades, sinó és correcte llença una excepció i finalitza l'aplicació.

- *update*: realitza les modificacions necessàries en la base de dades per que concordi amb el model de dades. En cas de que existeixi ja una concordança no fa cap modificació.
- *create*: crea les taules necessàries eliminant dades anteriors.
- *create-drop*: crea les taules necessàries eliminant dades anteriors i elimina aquestes taules al finalitzar l'aplicació.

En els entorns de preproducció y producció es recomana utilitzar *validate*.

- **db.driverClassName**: *Driver* a utilitzar per Hibernate per accedir a la base de dades.
- **db.url**: Cadena de connexió a base de dades.
- **db.username**: Usuari d'accés a base de dades.
- **db.password**: *Password* d'accés a base de dades.

6.4.2. Instal·lació de Tomcat7

La dinàmica per instal·lar Tomcat7 és molt semblant a la instal·lació de MySQL. Amb *yum* executem la comanda següent per fer la instal·lació:

```
$ yum install tomcat7
```

Opcionalment podem instal·lar les pàgines d'administració web:

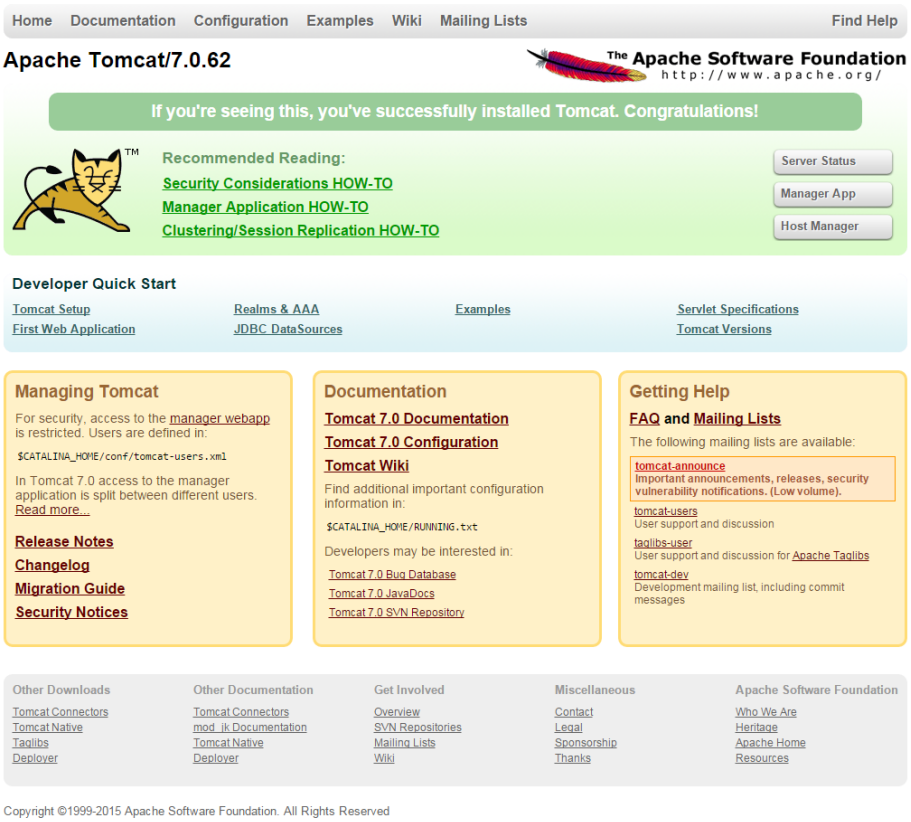
```
$ yum install tomcat7-webapps tomcat7-admin-webapps
```

Seguidament farem la configuració. Amb Tomcat7 la configuració es fa a través d'una sèrie de XML que es troben, en el meu cas, a la ruta `/usr/share/tomcat7/conf`. En el cas d'haver instal·lat la pàgina d'administració, serà necessari modificar l'arxiu tomcat-users.xml per incloure un usuari amb permisos GUI amb el qual ens podrem autenticar.

Per acabar, farem que el servei s'inicií automàticament amb el sistema operatiu amb la comanda:

```
$ chkconfig tomcat7 on
```

Una vegada seguits tots aquets passos, i una vegada engegat el servei de Tomcat, podrem veure una pàgina d'inici, de la forma següent, que ens indicarà que el servei esta corrent de forma adequada:



Imatge 30: Pàgina d'inici de Tomcat7

6.5. Configuració de la integració continua

Ara tenim disponible i configurat el software necessari per desplegar la nostra aplicació web. Però per poder-ho integrar amb Jenkins i al nostre cicle d'integració continua, hem de realitzar la configuració del projecte de forma adient.

Seguidament explicarem la configuració realitzada al projecte perquè es pugui integrar al cicle d'integració continua i com hem configurat la tasca de Jenkins per que faci un anàlisis del nostra codi i un desplegament en el nostra servidor d'integració.

6.5.1. Configuració de l'increment de versions automàtic

Per mantenir la traçabilitat de versions de forma automàtica, s'ha inclòs una petita tasca de Gradle que l'únic que fa és modificar la versió de la *build* per la introduïda per comanda i modificar aquest valor en el nostre *application.properties* per si volem accedir a la versió dintre del nostra codi. La tasca és la següent:

```
task copyVersion << {
  version = versionCode
  println "${project.group}:${project.name} [copyVersion] Project version: $version"

  ant.propertyfile(file: 'src/main/resources/application.properties') {
         entry(key: 'property.version', value: version)
```

```
  }
}
compileJava.dependsOn(copyVersion)
```

I la forma d'utilitzar-lo es simplement:

```
$ gradle build -PversionCode=0.1.2
```

Per fer aquest increment automàtic en Jenkins hem combinat això amb la variable d'entorn de la tasca `${BUILD_NUMBER}` que representa el número de *build* actual que està executant. Per tant, la comanda següent genera un numero de versió diferent i incremental cada vegada que es fa una execució de la tasca:

```
$ gradle build -PversionCode=0.1.${BUILD_NUMBER}
```

6.5.2. Configuració dels entorns

Degut a que en un sistema d'integració continua conviurem amb una sèrie d'entorns diferents, em de trobar alguna formula que ens permeti tenir les configuracions separades per entorns i poder generar *builds* per un entorn determinat cada vegada.

La solució trobada consta de 2 passos, el primer és generar una carpeta per cadascun del entorns. Així tinc una carpeta *environment* que conte dues subcarpetes *dev* (desenvolupament) i *int* (integració) que contenen els arxius de configuració propis de cada entorn.

El següent pas es la tasca de Gradle següent:

```
task copyConfiguration << {
    println "${project.group}:${project.name} [copyConfiguration] Target environment selected:
$environment"

    copy {
        from "src/main/environments/$environment"
        into "src/main/resources"
        include "*/"
    }
}
compileJava.dependsOn(copyConfiguration)
```

Aquesta tasca copia, al realitzar la *build*, tot el contingut de la carpeta del entorn seleccionat dintre de la carpeta de recursos del projecte. Per seleccionar un entorn a la *build* és tan senzill com indicar-ho com a paràmetre:

```
$ gradle build -Penvironment=dev
```

6.5.3. Configuració de Sonar

Degut a que Sonar només consulta les dades que genera Jacoco (el calculador de cobertura per els tests), el primer pas és incloure Jacoco al nostra projecte Gradle incorporant les línies següent al *gradle.build*:

```
apply plugin: 'jacoco'

jacoco {
```

```
  toolVersion = "0.7.1.201405082137"
  reportsDir = file("$buildDir/jacoco")
}

// Opcional
jacocoTestReport {
  reports {
          xml.enabled false
          csv.enabled false
          html.destination "${buildDir}/jacoco/html"
  }
}
```

Això fa que sempre que fem una *build* amb l'execució dels test, generarà un arxiu que contindrà la cobertura codificada dintre del *reportsDir* que indiquem.

El segon pas per fer la configuració de Sonar, es incorporar a la tasca de Jenkins del projecte, una execució de Sonar, i per indicar la configuració pròpia del nostre projecte podríem fer-lo a través d'aquesta execució (en el camp *Analysis properties*) però millor ens basarem en el fitxer *sonar-project.properties* que estarà inclòs al projecte i l'únic que farem serà indicar on es troba el nostre arxiu de configuració dintre del projecte:

Imatge 31: Tasca d'execució Sonar d'una tasca de Jenkins.

En el fitxer es on fem tota la configuració de la tasca de Sonar. Primerament indiquem on estan els diferents arxius tant de codi font com tests:

```
sonar.sources=ProjectSelection/src/main/java
sonar.binaries=ProjectSelection/build/classes/main,ProjectSelection/build/classes/test
sonar.tests=ProjectSelection/src/test/java
```

Seguidament indicarem la configuració de Jacoco, que es la nostra eina de cobertura de *testing*:

```
sonar.core.codeCoveragePlugin=jacoco

sonar.forceAnalysis=true
sonar.dynamicAnalysis=reuseReports
```

Degut a que fem només test unitaris per ara, és interesant excloure de la cobertura aquelles classes que no són utilitzades o no són adequades per aquesta mena de tests, ja que sinó, la cobertura mostrada ens podria donar uns resultats inadequats:

```
#Exclude unnecessary paths from test coverage
sonar.jacoco.excludes=**/model/**:**/controller/ControllerAdvice*
```

Seguidament altre configuració de Jacoco per indicar on es troben els anàlisis de cobertura que genera:

```
#Tells Sonar where the unit tests execution reports are
```

```
sonar.junit.reportsPath=ProjectSelection/build/test-results

#Tells Sonar where the unit tests code coverage report is
sonar.jacoco.reportPath=ProjectSelection/build/jacoco/test.exec
```

Finalment trobem 3 valors que indican el nom, grup i versió que rebrà a Sonar que són autogenerats cada vegada que fem la *build* a través de la tasca Gradle següent:

```
task setSonarProperties << {
  println "${project.group}:${project.name} [setSonarProperties] Sonar properties:
${project.group}:${project.name}:${project.version}"

  ant.propertyfile(file: 'sonar-project.properties') {
         entry(key: 'sonar.projectKey', value: project.group + ":" + project.name)
         entry(key: 'sonar.projectName', value: project.name)
         entry(key: 'sonar.projectVersion', value: project.version)
  }
}
processResources.dependsOn(setSonarProperties)
```

Aquesta tasca genera els valor adequats segons la configuració del projecte i les dades que genera són les següents (dintre del propi *sonar-project.properties*):

```
sonar.projectKey=com.jcmartin.projectselection\:ProjectSelection
sonar.projectName=ProjectSelection
sonar.projectVersion=0.1.0
```

Això ens funciona perfectament juntament amb Jenkins i el nostra sistema d'increment de versions automàtic, ja que controlarem en tot cas quina versió de *build* genera les dades de Sonar.

6.5.4. Configuració del repositori d'artefactes

Gràcies a Gradle la configuració del repositori es fa d'una manera realment simple:

```
repositories {
    mavenCentral()
}
```

Amb aquestes 3 línies se l'hi indica a Gradle que el repositori on ha de buscar les dependències és al **Maven 2 Central Repository**.

6.5.5. Configuració de Artifactory

En el nostre projecte ens interessaria mantenir les diferents versions dels WAR generats. Degut que el plugin de Gradle utilitzat per fer el desplegament en Artifactory a de ser configurat d'una manera bastant complexa per fer el desplegament del WAR correctament, no s'ha pogut realitzar aquesta configuració correctament, arribant únicament a la configuració de Artifactory però no en la selecció correcte del artefacte (fitxer WAR) a desplegar.

6.6. Projecció de futur

Tot i que de que s'ha aconseguit un producte mínim viable, Degut a la complexitat que incorporava la documentació i desplegament del sistema d'integració continua i del temps limitat que es proporcionava, han quedat moltes idees que podrien ser incorporades en la nostra aplicació.

Les més importants podrien ser les següents:

- Han faltat alguns requeriments per incorporar a la nostra aplicació i acabar d'incorporar-los tindria que ser el primer pas per acabar les seves funcionalitats requerides. El que més ens pot interessar seria alguna forma de cerca o filtratge de projectes, que podria ser una eina molt interesant si existeix un volum considerable de projectes.
- Degut a que seria una eina acadèmica que ja disposa d'una base de dades d'alumnes amb el seu usuari i *password*, no es desitjable que els usuaris es tornin a registrar en aquesta aplicació de nou. El camí més adequat es **fer la connexió de Spring Security amb el sistema d'autenticació que s'utilitzés**, possiblement un LDAP (*Lightweight Directory Access Protocol*). I entre la quantitat de llibreries que disposa Spring Security una d'elles es especifica per fer aquesta mena de connexions.
- Degut a que ens centrem en la integració continua, hem deixat de banda la part d'usabilitat, molt important en el món de les aplicacions web. Per aquest motiu es necessari **fer un anàlisis de la usabilitat de l'aplicació** i fer les iteracions necessàries fins a obtenir un *frontend* amb una bona usabilitat.
- Ja que el que s'espera és que la aplicació vagui destinada a una universitat, aquestes voldran, a part de la usabilitat comentada en el punt anterior, la visualització dels propis **logotips i colors corporatius**, per això és necessari fer modificacions en com estan distribuïts els estils als fitxers CSS per tenir en un sol fitxer aquells paràmetres que puguin ser modificats per canviar l'aparença bàsica (logos i colors) de l'aplicació.
- Degut a la falta de recursos, no ha sigut possible **crear els dos entorns principals que falten, un entorn de preproducció i un entorn de producció**. En cas que aquest producte vulgui ser posar en producció s'han de crear aquests entorns i mantenir el cicle d'integració continua (en preproducció només es pujaran les coses que superin els nostres límits de qualitat i a producció només es pujarà aquelles coses que hagin sigut aprovades en preproducció).
- Un altre punt que milloraria la qualitat de codi del nostre projecte seria incorporar uns **test d'integració** a través d'una eina com pot ser **Selenium**.

7. Historial de la integració continua del projecte

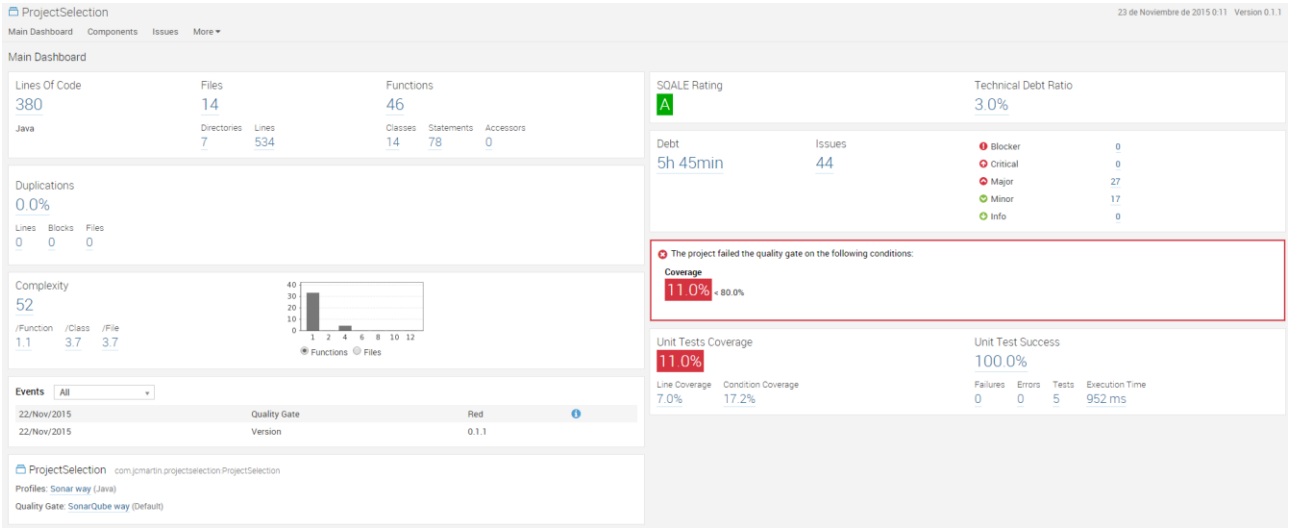
"Everybody gets so much information all day long that they lose their common sense"

Gertrude Stein

En aquest capítol es vol resumir les diferents dades que ens ha proporcionat Sonar durant el temps de desenvolupament del projecte, i com s'han anat modificant aquests valor al llarg del temps.

7.1. Dades de Sonar: Sprint 1

El dia 23/11/2015 ja estava feta la integració del projecte amb el nostre sistema d'integració continua i es va realitzar un dels primers llançaments on es feia la tasca de Sonar. Els primers resultats van ser els següents:



Imatge 32: Captura de pantalla de Sonar referent al Sprint 1

Podem veure que el projecte és bastant petit actualment, 380 LOC en 14 fitxers, ja que només conté funcions bàsiques per usuaris i projectes i poca cosa mes. Com podem veure en aquesta imatge els resultats eren molt dolents. A pesar que teníem un 100% dels test correctes, aquets només eren 5 i amb una cobertura del 11%, amb un deute tècnic del 3.0%.

Aquestes dades han de millorar al llarg del projecte ja que per ara no supera els mínims de qualitats que esperem, recordem que eren de 100% dels test correctes i superar el 80% de la cobertura en la capa de serveis (*business*), la capa de persistència (*repository*) i la capa de controladors.

7.2. Dades de Sonar: Sprint 2

Després de treballar en el projecte durant una setmana, el 29/11/2015, els resultats del projecte han millorat de forma substancial. La imatge següent els mostra:

	PROJECTSELECTION 0.1.6 29/NOV/2015
Lines of code	541
Complexity	83
Issues	56
Technical Debt Ratio	1.6%
Unit tests success (%)	96.5%
Coverage	75.9%

Imatge 33: Captura de pantalla de Sonar referent al Sprint 2 (únicament aquells paràmetres importants)

A pesar de que encara no aconseguim els nostres nivells de qualitat mínims (ja que podem veure que 2 test han fallat i la cobertura dels test està per sota del 80%) em augmentat de 5 a 57 test i la cobertura de 11% al 75.9%.

7.3. Dades de Sonar: Sprint 3

En aquest tercer *Sprint* em trobat que tant el tant per cent de test correctes i la cobertura han baixat considerablement:

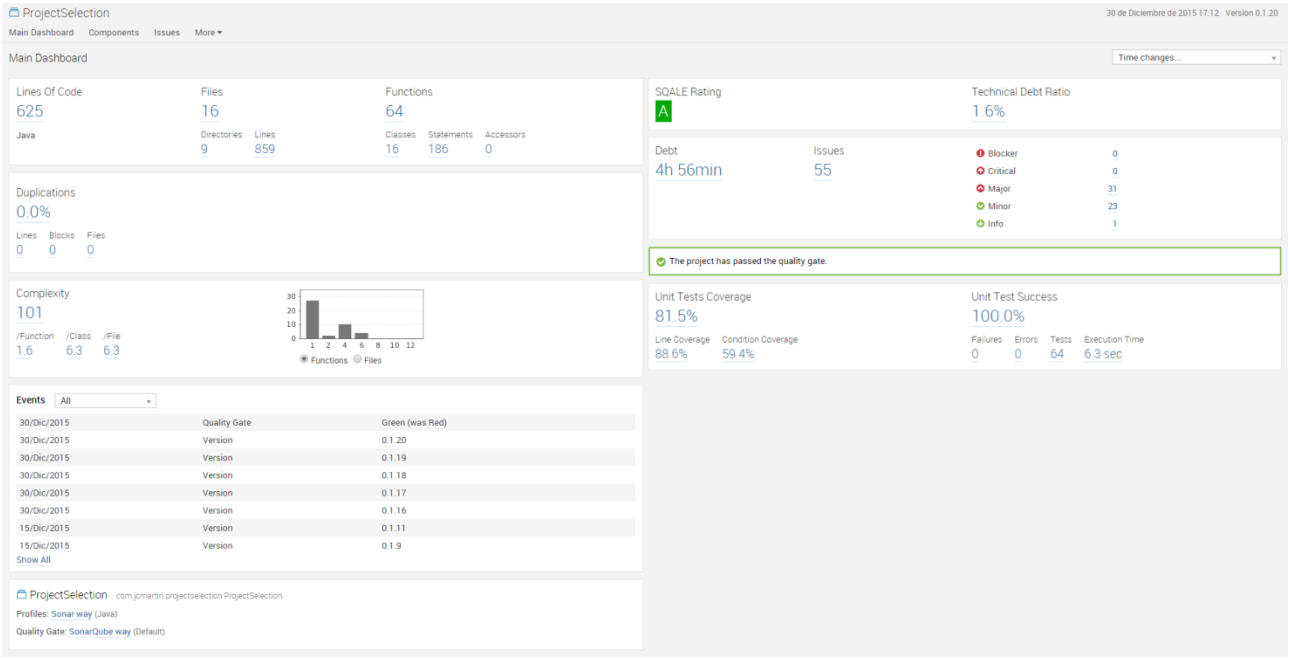
	PROJECTSELECTION 0.1.11 15/DIC/2015
Lines of code	618
Complexity	100
Issues	62
Technical Debt Ratio	2.1%
Unit tests success (%)	80.7%
Coverage	54.2%

Imatge 34: Captura de pantalla de Sonar referent al Sprint 3 (únicament aquells paràmetres importants)

Aquest descens de la qualitat de codi és degut a la incorporacions de varies funcionalitats que han afectat a altres ja creades i que fan que fallin el test. Aquest és un escenari dolent, al incorporar funcionalitats no hem mantingut la qualitat de codi i en el següent *Sprint* tindrem que solucionar aquesta baixada considerable.

7.4. Dades de Sonar: Sprint 4

Finalment després de trobar-nos que alguns canvis que baixaven considerablement la cobertura i els tests correctes, hem aconseguit superar els nostres *quality gates* que ens vam imposar (superar tots els test i més d'un 80% de cobertura de test).



Imatge 35: Captura de pantalla de Sonar referent al Sprint 4

7.5. Gràfiques històriques

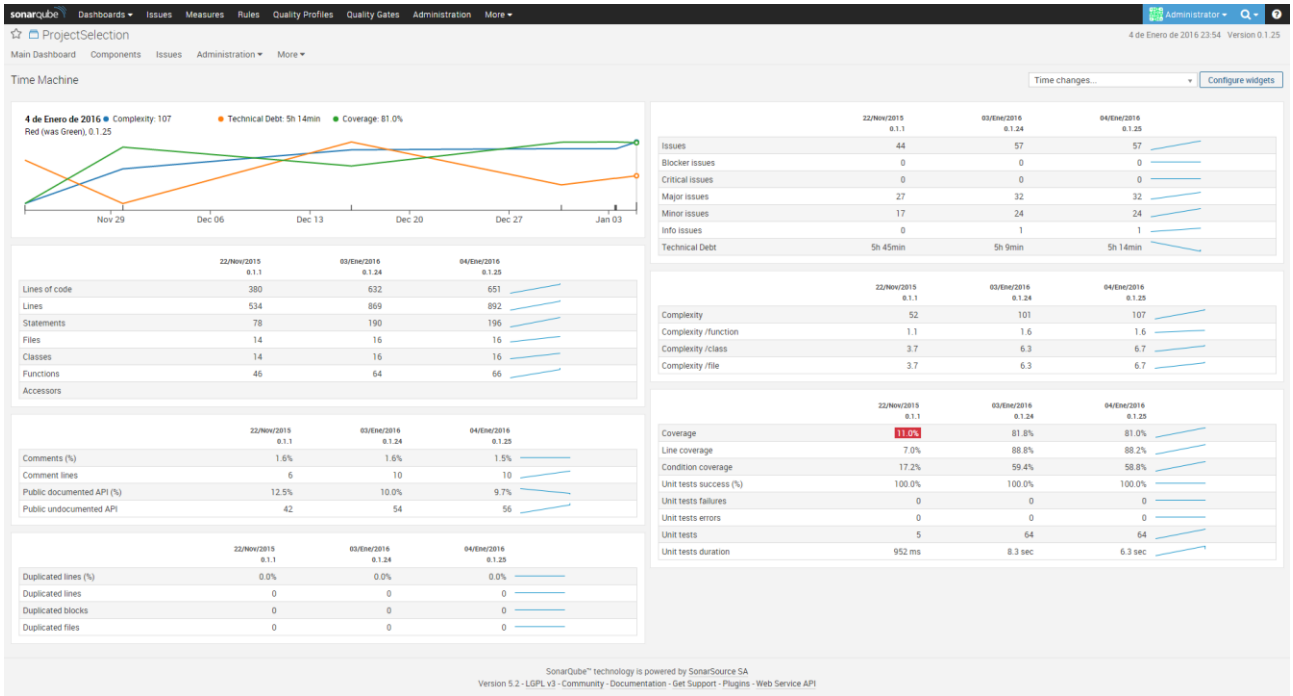
Per acabar amb la presentació de les dades finals que ens ha atorgat Sonar, voldria mostrar un parell de formes que ens aporta Sonar per mostrar gràfiques històriques, d'aquesta manera podrem veure l'evolució del nostra projecte al llarg del temps.

La gràfica, obtinguda a partir de l'opció *Compare*, mostra els 4 punts, o finals de *sprint*, que hem realitzat al llarg del temps, mostrant els indicadors que he cregut més indicatius:

	PROJECTSELECTION 0.1.1 22/NOV/2015	PROJECTSELECTION 0.1.6 29/NOV/2015	PROJECTSELECTION 0.1.11 15/DIC/2015	PROJECTSELECTION 0.1.24 03/ENE/2016
Quality Gate Status	✗	✗	✗	✓
Lines of code	380	541	618	632
Complexity	52	83	100	101
Issues	44	56	62	57
Technical Debt Ratio	3.0%	1.6%	2.1%	1.6%
Unit tests success (%)	100.0%	96.5%	80.7%	100.0%
Coverage	11.0%	75.9%	54.2%	81.8%

Imatge 36: Dades històriques importants obtingudes per Sonar en diferents versions de l'aplicació, obtinguda amb l'opció Compare.

Y la següent gràfica, obtinguda amb l'opció *Time Machine*, mostra l'evolució al llarg del temps i la comparació amb la *build* realitzada anteriorment, mostrant si l'evolució és a l'alça o a la baixa, als diferents paràmetres, de manera ràpida.



Imatge 37: Un altra visualització de l'historial de les diferents dades obtingudes per Sonar, obtinguda a partir de l'opció Time Machine.

Com podem veure, i és totalment lògic, al llarg del projecte ha hagut un augment en el nombra de línies de codi, al igual que la complexitat. Al mateix nivell, ha augmentat el nombre de problemes trobats per Sonar (*issues),* menys en l'ultima *build*, on vàrem solucionar alguns d'ells. També podem veure l'evolució de la cobertura dels tests. De la primera *build* (versió 0.1.1) a la segona (versió 0.1.6) es va realitzar una feina de *testing* important, arribant, d'un 11% de cobertura, al 75,9%. De la segona (versió 0.1.6) a la tercera (versió 0.1.11) es va produir un descens important degut a la incorporació d'algunes funcionalitats. De la tercera

(versió 0.1.11) a la quarta (versió 0.1.24) es va corregir el problema de cobertura anterior, així com el problema de tests erronis, per aconseguir, finalment, els nostres objectius de qualitat de software.

8. Conclusions

"It's fine to celebrate succés but is more important to heed the lessons of failure"

Bill Gates

Aquest projecte m'ha ajudat a adquirir uns bons coneixements sobre un tema que portava temps interessant-me com es la integració continua. No només utilitzar una petita part, sinó se capaç de fer un complet desplegament, configuració i la utilització completa per fer el desenvolupament d'un projecte.

Com he pogut comprovar, l'inici del desplegament d'un sistema d'integració continua és complex, i això que he deixat de banda cert temes de seguretat com podria ser la connexió de Jenkins amb un LDAP o la configuració completa d'usuaris en un entorn empresarial, amb els diferents rols que poden sorgir.

No només aquesta part és complexa, sinó que la configuració inicial del projecte amb un entorn d'integració continua es fa bastant complicada: configuració d'entorns, versionat, repositoris, etc. Una vegada superat aquest primer esglaó, he començat a percebre la quantitat d'avantatges que ofereix la integració continua i la qualitat de codi. Els que podria destacar són:

- El repositori de codi hem permetia mantenir diferents versions per diferents objectius, com mantenir un codi on fer les meves proves i en cas de que sorgís algun tipus d'error, tornar a una versió correcte en un moment.
- La configuració d'entorns escollida va resultat tot un èxit. Mantenir un arxiu de propietats amb cadascun dels entorn feia que el desplegament en un altre entorn fos gairebé automàtic, l'únic que tenia que fer, es escollir l'entorn a l'hora de fer la *build*, la resta era automàtic.
- La tasca de Jenkins en general és realment impressionant, ofereix una flexibilitat altíssima. *Build*, execució de Sonar únicament en cas que sigui una *build* correcta, si s'ha parametritzat amb el *flag* de *deploy* seleccionat es realitza el desplegament, etc. Totes aquestes tasques llistades són les que s'han utilitzat en fer la tasca de Jenkins, però es poden realitzar multitud més, en qualsevol ordre i condicions, i sinó, segur que existeix algun *plugin* que faci el que volem.
- El desplegament automàtic configurat per Jenkins és un dels punt que més m'ha impressionat, segurament per que era un dels punt que més esperava poder realitzar des de un principi. Una simple configuració a la tasca de Jenkins i cadascuna de les *builds* feia un desplegament automàtic al servidor, que en qüestió d'uns segons ja estava disponible.
- Possiblement la informació que es presenta de Sonar és la més rellevant, però la informació que ens proporciona és més complexa i útil del que havia arribat a veure en un principi. La informació sobre cobertura pot resultar útil per veure quins són els punts que necessiten ser revisats. Però altres informacions com podria ser la complexitat o la interdependència entre paquets arriben a ser molt útils per comprovar si cal fer alguna modificació o *refactoring* en el software.

També cal dir que degut a la limitació de temps han existit alguns aspectes que no he pogut realitzar i que m'hagués agradat realitzar al llarg d'aquest projecte:

- Ha faltat explorar altres tasques de Jenkins que podrien haver sigut molt interesants: *build* automàtica al actualitzar el repositori o enviament de correu amb els resultats de la *build* i de Sonar.
- La integració del projecte amb Artifactory no ha arribat a ser finalitzada completament. S'ha aconseguit fer el desplegament d'Artifactory i connectar Jenkins amb Artifactory, però el projecte, tot i fer pujades a Artifactory, les feia sense artefactes, quan tindria que contenir el WAR del projecte.
- M'hagués agradat realitzar uns tests més complexes, que comprovessin totes les capes del meu projecte. Finalment, només ha sigut possible comprovar fins la capa de controladors, aquesta inclosa. Hagués faltat completar el test amb una eina com Selenium, per comprovar el correcte funcionament de la capa de presentació.
- El cicle d'integració continua que s'ha realitzat en el projecte no es complet al 100%, falta tota la part de desplegament en un entorn de preproducció, proves de validació i rendiment, i un desplegament en un altre entorn de producció per completar el cicle.
- Les tasques de Jenkins poden ser parametritzades i fer que al executar una tasca es puguin introduir certs paràmetres de qualsevol tipus: texts, seleccionables entre varies opcions, *checks*, etc. Podria haver sigut interesant introduir els paràmetres del numero de versió, la branca del repositori de codi que es vol utilitzar o si es vol realitzar un desplegament o no, i modificar lleugerament la tasca de Jenkins per que els tingui en compte.
- El nostre projecte era realitzat completament en Java. Jenkins està preparat per fer-ho servir amb altres llenguatges i podria haver sigut molt interesants realitzar proves de concepte amb altres llenguatges com poden ser Python, C++ o projectes en Android.

Com a resum, la integració continua requereix un sistema complet, uns coneixements previs i una configuració prèvia del projecte amb certa complexitat. Després de tot això, ens proporciona una velocitat, fiabilitat i qualitat molt alts en comparació amb un desenvolupament ordinari, i si comparem el que suposa un primer esglaó complex amb tot un gran projecte, acaba resultant en un gran benefici a mig-llarg termini per el desenvolupament de software.

9. Annex: Lliurables del projecte

Els lliurables són els següents:

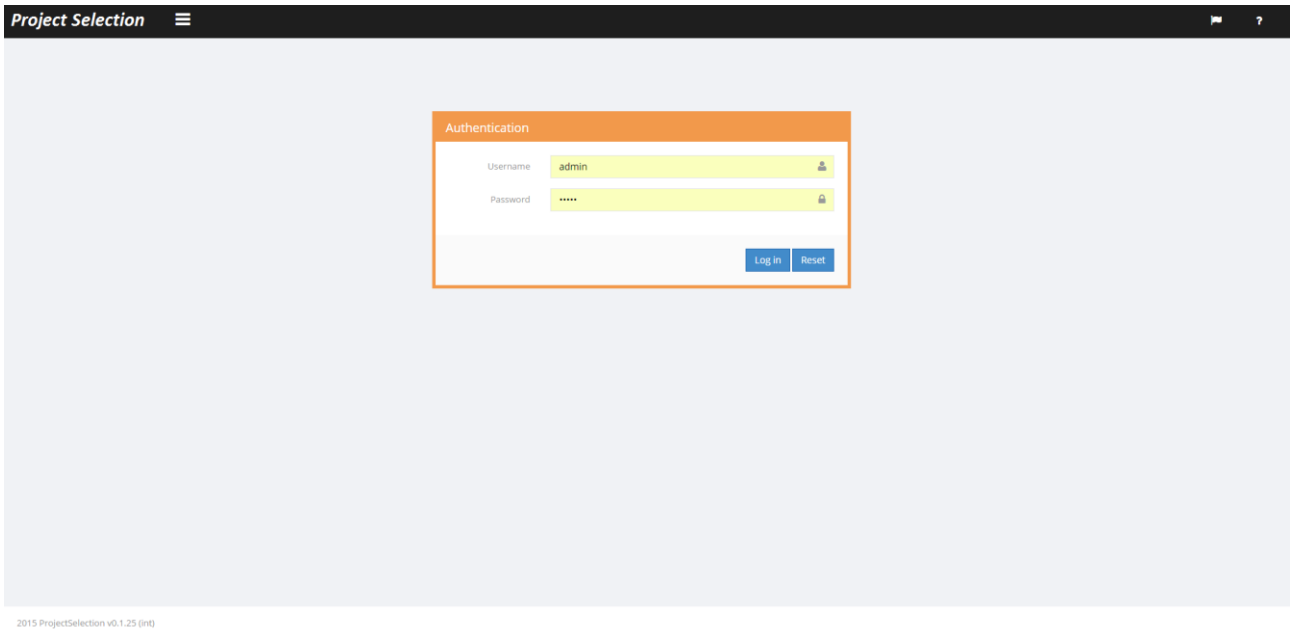
- Memòria del projecte de final de Máster.
- Aplicació web **"Project Selection"**
  - URL: **http://<host>:8080/ProjectSelection**
  - Usuari administrador[23]: **admin** (password: **admin2015**).
  - Usuari professor: **teacher** (password: **teacher2015**).
  - Usuari alumne: **student** (password: **student2015**).
- Jenkins
  - URL: **http://<host>:8081**
  - Usuari: **user** (password: **uoc2015**), únicament amb permisos de visualització.
- Artifactory
  - URL: **http://<host>:8082**
  - Usuari: No es proporciona usuari d'Artifactory.
- Sonar
  - URL: **http://<host>:9000**
  - Usuari: no cal, per defecte es permet la visualització del usuari anònim.

Actualment el host es troba a la IP **52.10.200.157** però pot ser necessària la seva modificació degut al sistema de IPs dinàmiques de Amazon. En cas de necessitar accedir a una de les web i que no estigui disponible, contactar per entregar la IP correcta.

[23] Es prega no fer grans modificacions que puguin ocasionar un mal funcionament de l'aplicació. Les credencials del administrador només s'aporten per motius acadèmics i perquè es puguin veure totes les funcionalitats que l'aplicació web ofereix.

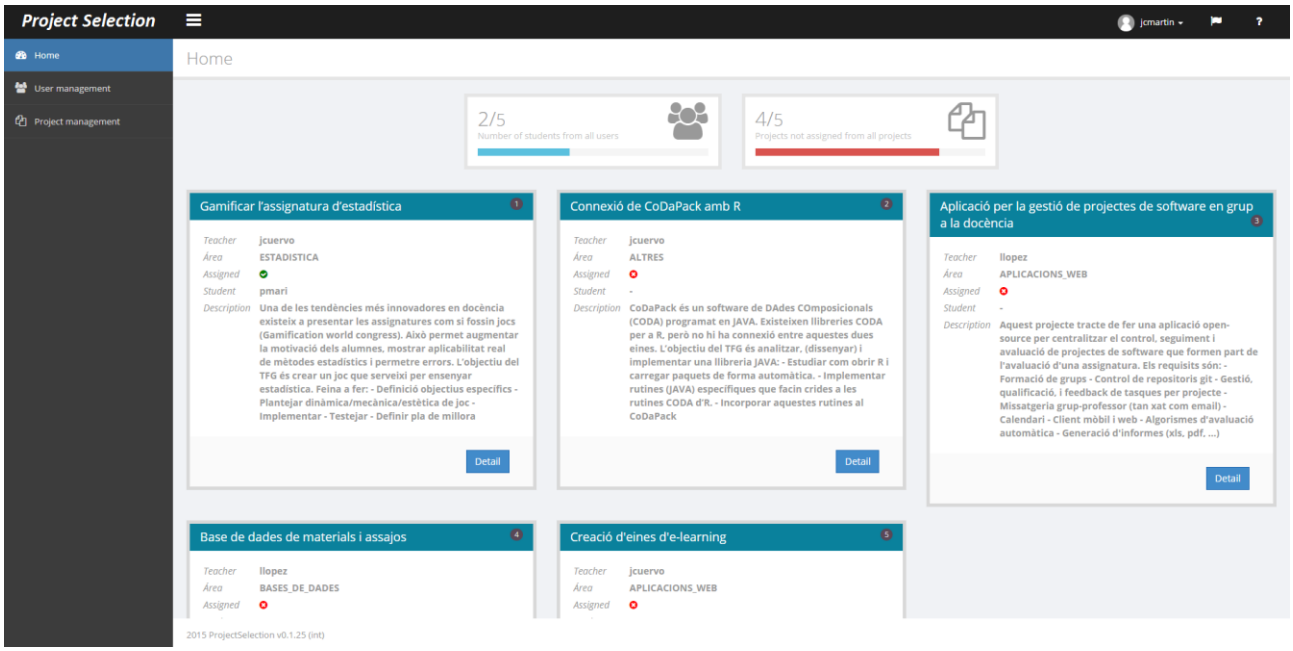
10. Annex: Manual d'usuari

La pàgina de *login* és la pàgina publica de la nostra aplicació. Aquesta pàgina ens permet autenticar-nos amb el nostra usuari i *password* i poder accedir al sistema.



Imatge 38: Pàgina de login de l'aplicació ProjectSelection

Una vegada autenticats amb el usuari/*password*, si es correcta, entrarem al sistema i se'ns mostrarà la pàgina d'inici o *home*.



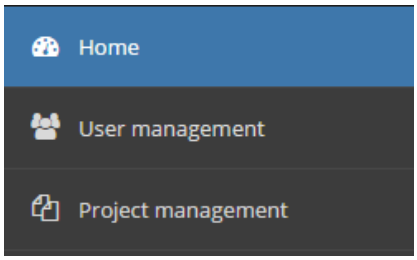
Imatge 39: Pàgina home de l'aplicació ProjectSelection

En aquesta pantalla, trobem 2 elements que sortiran a totes les pàgines i ens permetran navegar per les diferents funcionalitats, el ***menú superior*** i el ***menú esquerra de navegació***.

El menú superior ens ofereix la possibilitat de fer el *logout* del sistema, canviar l'idioma o enviar un correu als professionals de suport, per comunicar qualsevol problema o dubte que es vulgui.

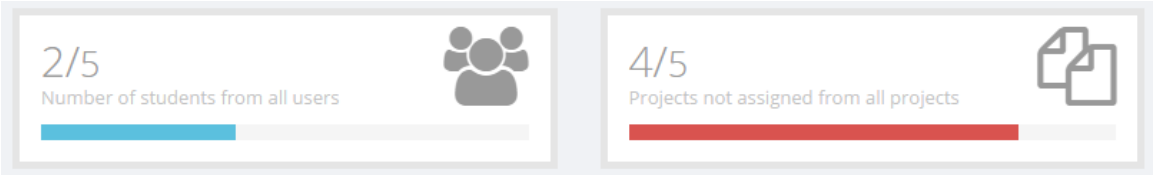
Imatge 40: Capçalera de l'aplicació ProjectSelection

El menú esquerra ens permet moure'ns per les diferents funcionalitats del sistema. Els elements que aquí es mostren dependrà del rol del usuari connectat. Així, un administrador pot gestionar usuaris i projectes, un professor únicament gestionar els seus projectes i un estudiant només tindrà accés a la pàgina d'inici (*home*).



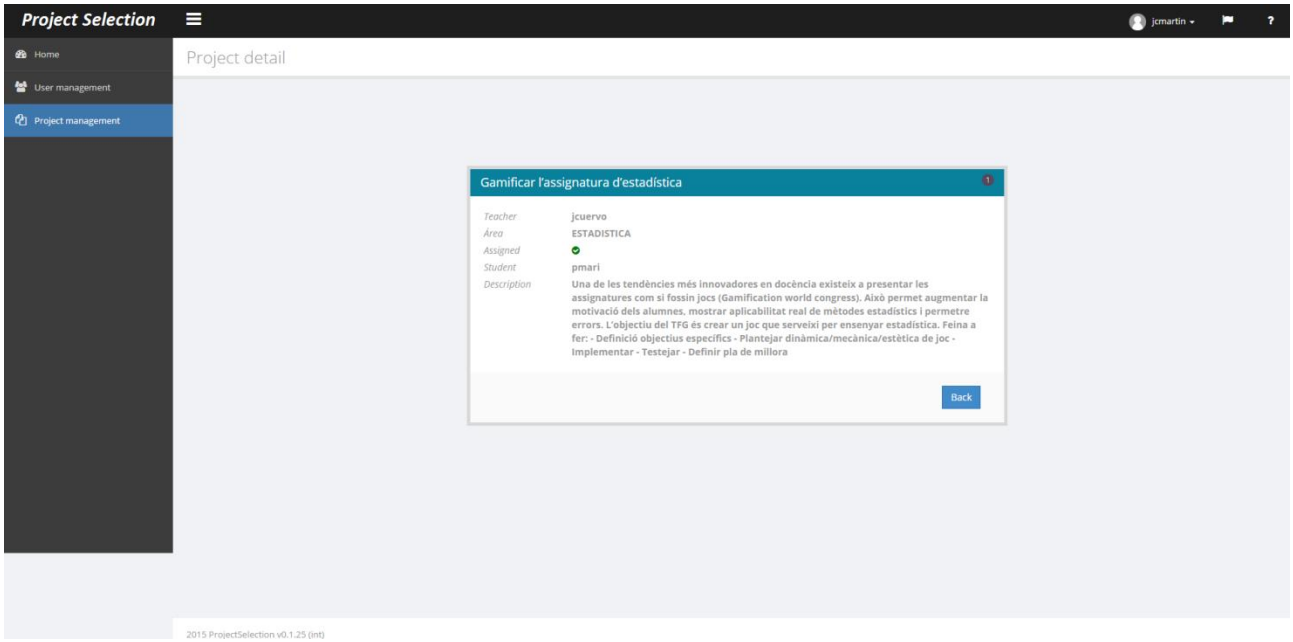
Imatge 41: Menú lateral de l'aplicació ProjectSelection

En aquesta pàgina d'inici, podem veure informació general de la aplicació, mostrats en la part superior, com són el nombre d'estudiants i el nombre de projectes totals i assignats.



Imatge 42: KPI (key performance indicator) de l'aplicació ProjectSelection

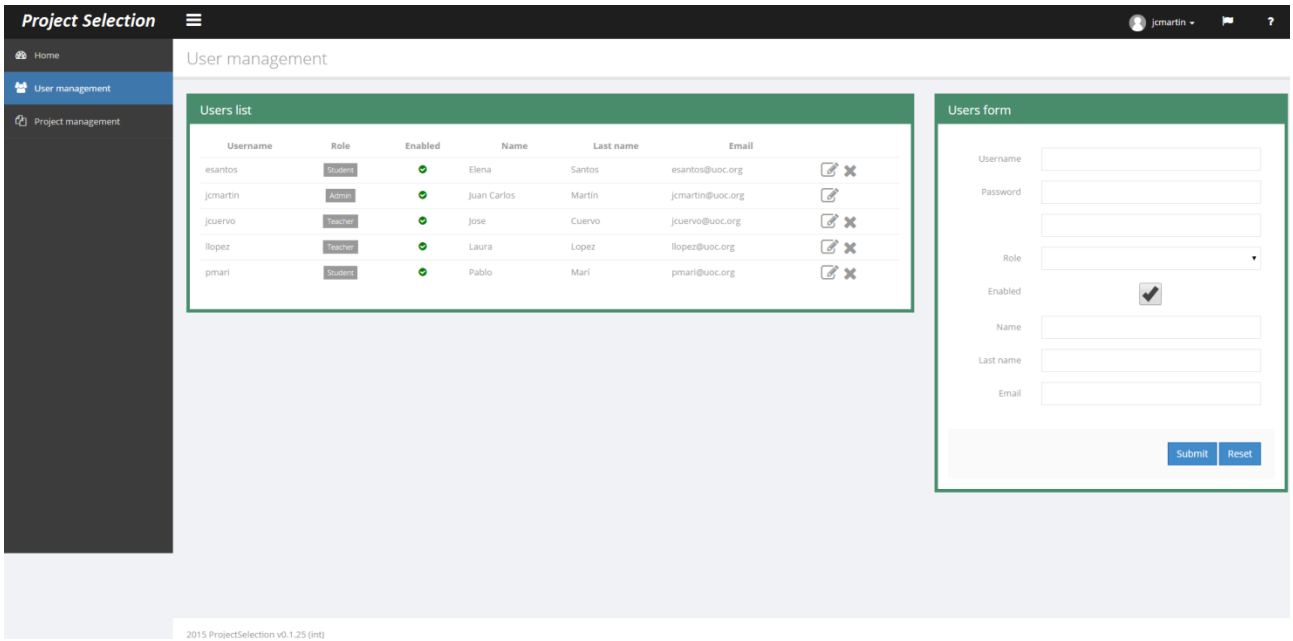
A la part central es mostraran tots els projectes que existeixen a la plataforma. En cas d'estar connectat com a estudiant i tenir un projecte assignat, aquest sortirà en primera posició i amb un color diferent, per mostrar que es el projecte seleccionat actual. A partir del boto inferior de cadascun del requadres dels projectes, podem accedir a la pàgina de detall del projecte en qüestió.



Imatge 43: Pàgina de detall de projectes de l'aplicació ProjectSelection

En cas d'estar connectat com a estudiant, en aquesta pàgina tindrem l'opció d'assignar-nos, o en cas de ser el projecte assignat, eliminar l'assignació.

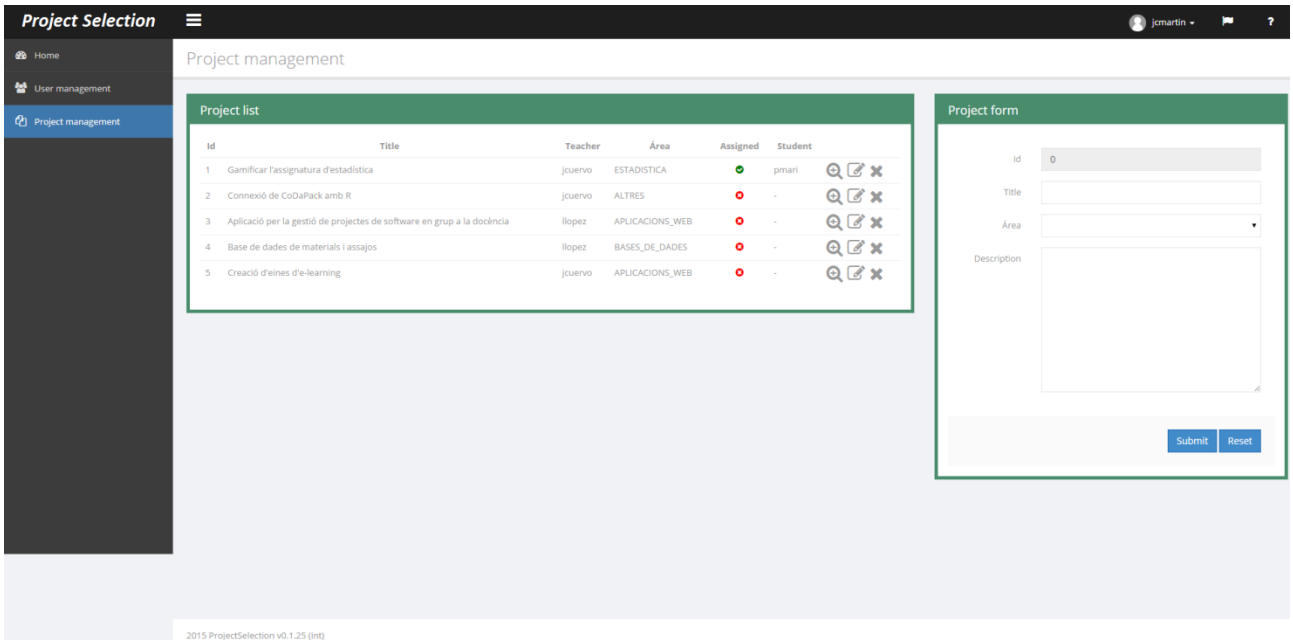
Les pàgines de gestió han sigut creades per simplificar la gestió i s'ha incorporat en una mateixa vista el formulari i la llista d'elements. La pàgina de gestió d'usuaris (que només es accessible pels administradors) queda de la següent forma:



Imatge 44: Pàgina de gestió d'usuaris de l'aplicació ProjectSelection

Aquí es permet crear (inserint dades directament a un formulari en blanc), editar (seleccionant el botó d'editar d'algun usuari de la llista) i eliminar (clicant al boto d'eliminar en un usuari de la llista) els usuaris.

Finalment, la pàgina de gestió de projectes (que només és accessible per administradors i professors) queda de la següent forma:



Imatge 45: Pàgina de gestió de projectes de l'aplicació ProjectSelection

Aquí es permet anar a la pàgina de detall del projecte seleccionat (clicant la icona de lupa), crear (inserint dades directament a un formulari en blanc), editar (seleccionant el boto d'editar d'algun projecte de la llista) i eliminar (clicant al boto d'eliminar en un projecte de la llista) els projectes. En cas de ser administrador, es mostren tots els projectes del sistema, en cas de ser professor, només es permet la gestió dels propis projectes.

11. Annex: Glossari

Amazon: Companyia de comerç electrònic i computació al núvol.

Amazon EC2: Acrònim de *Amazon Elastic Cloud Computer*. Servei d'Amazon per el còmput al núvol.

AMI*:* Acrònim de *Amazon Machine Images*. Conté la informació d'un sistema operatiu per ser llençat sobre una instància d'Amazon.

Apache Software Foundation: Gran comunitat de desenvolupament de projectes software.

***Bug*:** S'anomena *bug* a un error en el software.

***Build*:** Acció de construir alguna mena d'executable a partir del codi font.

Build server: Servidor encarregat de realitzar la build a partir d'un codi font.

Cloud: Forma de anomenar al núvol o Internet.

Debug/debugging: Acció d'executar un programa en un mode en que l'usuari pot interactuar amb l'execució, canviar paràmetre, consultar memòria, etc., conforme el programa es va executant pas per pas.

***Extreme Programming/XP*:** Metodologia de desenvolupament agil de software formulada per Kent Beck.

***Feedback*:** Representa l'acció de resposta o reacció.

***Framework*:** Representa un entorn de treball.

IC/CI: Acrònim de Integració continua o *Continuous Integration* en angles.

IDE: Acrònim de *Integrated Development Environment*. Programa per fer el desenvolupament del software.

LDAP: Acrònim de *Lightweight Directory Access Protocol*.

LOC: Acrònim de línies de codi (*lines of code*).

***Log*:** Registre d'esdeveniments durant un rang de temps.

***Look&feel*:** Disseny d'una interfície d'usuari.

***Mockup*:** Imatge esquemàtica de la visualització final d'una interfície d'usuari.

PFM: Acrònim de Projecte de final de màster.

***Plugin*:** Software que pot incloure's en un altre per ampliar les seves funcionalitats.

QC: Acrònim de Qualitat de codi.

***Refactoring*:** Revisió i modificació considerable en el codi d'una aplicació.

***Rollback*:** Acció de tornar a un punt anterior.

***Sprint*:** Divisió de treball que es realitza d'un projecte. Normalment compren el treball que es realitzarà entre 2 i 4 setmanes.

SSH: Protocol de comunicació i execució de comandes remotes entre dues màquines.

***Test-driven development*:** Procés de desenvolupament de software on inicialment es desenvolupen els test (que fallen en un principi) i seguidament es desenvolupen les funcionalitats per complir els test.

***Testing*:** Acció de fer proves en una aplicació per comprovar el seu correcte funcionament.

***Workflow*:** Es l'estudi dels aspectes operacionals d'una activitat de treball: com s'estructuren les tasques, com es realitzen, ordre, sincronització, etc.

12. Annex: Bibliografia

[1] M. Fowler, «Continuous Integration,» 01 05 2006. [En línia]. Available: http://martinfowler.com/articles/continuousIntegration.html#PracticesOfContinuousIntegration. [Últim accés: 12 10 2015].

[2] V. S. Bhalothia, «Jenkins: A complete solution,» 12 01 2014. [En línia]. Available: http://www.slideshare.net/bhalothia/jenkins-a-complete-solution. [Últim accés: 17 10 2015].

[3] F. M. G. J. M. Juran, Juran's Quality Control Handbook, New York: McGraw-Hill, 2012.

[4] D. Spinellis, Code Quality: The open Source Perspective, Addison Wesley Professional, 2006.

[5] StacksShare, «StackShare,» [En línia]. Available: http://stackshare.io/. [Últim accés: 25 10 2015].

[6] S. M. a. A. G. Paul Duvall, «Introducing continuous integration,» 21 06 2007. [En línia]. Available: http://www.javaworld.com/article/2077731/build-ci-sdlc/introducing-continuous-integration.html. [Últim accés: 24 09 2015].

[7] Y. Bugayenko, «Continuous Integration is Dead,» 08 10 2014. [En línia]. Available: http://www.yegor256.com/2014/10/08/continuous-integration-is-dead.html. [Últim accés: 24 09 2015].

[8] T. Papaioannou, «Continuous Integration,» 30 05 2008. [En línia]. Available: http://www.slideshare.net/drluckyspin/continuous-integration. [Últim accés: 24 09 2015].

[9] J. F. Smart, «Continuous Integration 101,» 20 03 2014. [En línia]. Available: http://www.slideshare.net/wakaleo/continuous-integration-101-32558174. [Últim accés: 29 09 2015].

[10] M. Brittain, «Principles and practices in continuous deployment,» 02 04 2014. [En línia]. Available: http://www.slideshare.net/mikebrittain/principles-and-practices-in-continuous-deployment-at-etsy. [Últim accés: 29 09 2015].

[11] D. Hsieh, «Continuous Integration (Jenkins/Hudson),» 11 03 2011. [En línia]. Available: http://www.slideshare.net/DennysHsieh/continuous-integration-jenkinshudson-7232207. [Últim accés: 29 09 2015].

[12] Amazon, «Connecting to Your Linux Instance from Windows Using PuTTY,» 2015. [En línia]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/putty.html. [Últim accés: 10 10 2015].

[13] MuleSoft, «A Complete Guide To Tomcat Start-Up,» [En línia]. Available: https://www.mulesoft.com/tcat/tomcat-start. [Últim accés: 17 10 2015].

[14] C. S. So, «Code Quality: Metrics That Matter,» 11 01 2015. [En línia]. Available:

http://chriskottom.com/blog/2015/01/code-quality-metrics-that-matter/. [Últim accés: 24 09 2015].

[15] Microsoft, «Writing Quality Code,» [En línia]. Available: https://msdn.microsoft.com/en-us/library/4dtdybt8(v=vs.90).aspx. [Últim accés: 24 09 2015].