Desenvolupament d'aplicació web amb integració contínua

Diego Comas Torres
ETIG

Consultor:
Oriol Martí Girona

Juny 2015

UOC
Universitat Oberta de Catalunya

A Maria per tot el seu suport
i ajuda durant tota la carrera.
A Cristina, Laia i Roger per
recolzar-me en tot moment.

Juny 2015

Resum del projecte

Aquest projecte constarà de dos vessants que es desenvoluparan simultàniament, per una banda es vol fer una aplicació web on els usuaris puguin crear preguntes o enquestes, votar les preguntes d'altres usuaris i veure els vots de les enquestes. A l'aplicació els visitants podran enregistrar-se amb nom d'usuari i contrasenya. L'aplicació tindrà un sistema "*back-end*" que proporcionarà la persistència mitjançant un servei REST, aquest sistema serà gestionat per les tecnologies Node JS i Express JS. També s'utilitzarà un "*front-end*" codificat com aplicació web Angular JS, aquesta tecnologia desenvolupada per Google ajuda a crear aplicacions web anomenades "*single-page application*" que no necessiten actualitzar l'adreça del navegador per cada canvi de pàgina.

Per una altra banda es vol aplicar una metodologia diferent dels mètodes clàssics de desenvolupament d'aplicacions, aquest mètode està dintre de les metodologies àgils i podem anomenar-lo com DevOps. En realitat DevOps no és exactament una metodologia i és un terme poc precís però agrupa uns tipus de pràctiques que té com a objectiu fer el desenvolupament més àgil, amb integració contínua, creació de codi de més qualitat a l'incloure tests obligatòriament en el cicle de vida del desenvolupament de programari i automatització del desplegament de cada canvi de codi a producció automàticament si superar els tests.

Mitjançant aquesta metodologia agruparem uns requisits d'usuari i esbrinarem si fent ús d'aquestes pràctiques àgils podem aconseguir realitzar una aplicació web que tingui el menor nombre d'inconsistències en el seu funcionament.

Índex de continguts

Índex de figures

1. Desenvolupament àgil amb integració contínua

1.1. Introducció

Avui en dia cada cop tenen més popularitat les metodologies àgils, ara fa més de 14 anys del moment que es va publicar el manifest àgil promogut per Kent Beck a Utah. Des d'aquell moment moltes empreses i desenvolupadors han considerat beneficiós utilitzar aquestes metodologies per satisfer els clients amb el lliurament primerenc i continu de software amb valor. En aquest escrit farem un projecte enfocat principalment en la integració contínua en el desenvolupament de programari.

1.1.1. Anàlisi de la situació actual

Actualment els mètodes àgils estan agafant més protagonisme en el desenvolupament de programari. Podem observar l'evolució en el mercat laboral com cada cop es sol·liciten més professional en metodologies àgils, això es degut al fet que les empreses veuen un benefici en poder distribuir o crear el seu producte de manera més ràpida i en conseqüència abans que la competència.

Figura 1 - Evolució oferta de treball especialistes Agile

Dit això, històricament altres processos com el mètode en cascada han sigut més estesos. Els inconvenients que es troben en les metodologies clàssiques és que el desenvolupador rep els requisits inicials, després es dedica part a dissenyar com es codificarà l'aplicació, un cop fet això s'inicia la implementació del codi i posteriorment es passa a la fase d'anàlisi de qualitat del codi implementat. Un cop superat es realitzen les proves d'integració i funionals, finalment el codi passa al que s'anomena producció on l'usuari final pot utilitzar l'aplicació. Tot aquest procés pot fer que diferents equips realitzin diferents fases del cicle de vida i que hi hagi poca comunicació, perquè si una iteració del codi passa les proves, però en aquell moment el primer desenvolupador vol fer una modificació, pot provocar que amb una errada les proves que passaven abans no passin ara i el *tester* ho hagi de revisar tot un altre cop. Amb la integració contínua aquest error humà s'intenta minimitzar automatitzant fases com les proves unitàries o d'integració, ja que si un desenvolupador fa canvis aquest nou codi passarà per un servidor que revisarà automàticament que el que abans no fallava continuï funcionant.

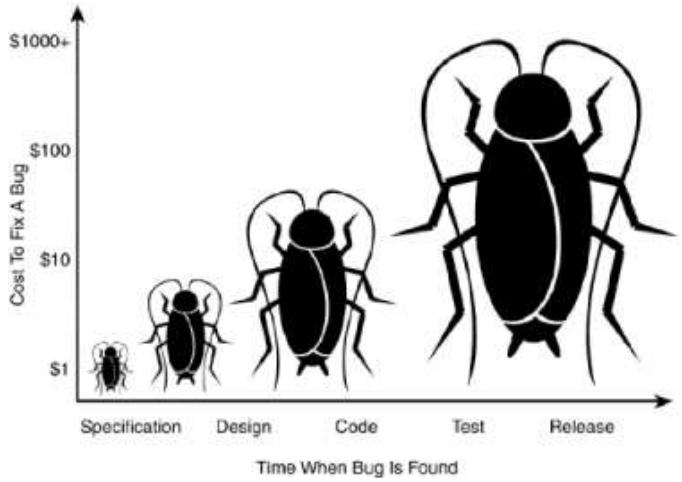


Figura 2 - Cost exponencial de trobar errades massa tard en el programari.

Per la banda de l'aplicació web és interessant fer una aplicació on la gent pugui fer votacions en línia perquè avui en dia sembla que hi ha molt interès en saber opinions de la ciutadania i valorar quines majories hi ha, ja sigui el dret a decidir, les assemblees de Barcelona en comú, temes electorals, etc.

1.1.2. Anàlisi de riscos

El principal risc que s'hi pot trobar en practicar aquesta metodologia és la complexitat que s'hi afegeix a l'intentar automatitzar tot el procés, és a dir, a causa de la necessitat de fer les proves sempre abans de desenvolupar noves funcions, això pot fer derivar en iteracions més lentes. També hi ha el risc que tot aquest procés automatitzat funcioni en un context específic amb una configuració estricta en comptes d'adaptar-se a múltiples configuracions i sistemes. Però un dels objectius de la metodologia DevOps és minimitzar els riscos mitjançant iteracions més petites.

1.1.3. Objectius del TFC.

Amb l'execució d'aquest projecte es vol analitzar i valorar quines avantatges o inconvenients pot aportar el desenvolupament d'una aplicació web mitjançant integració continua i amb metodologia DevOps. Per esbrinar això es vol crear un procés automatitzat per a poder escurçar el cicle de vida de desenvolupament de programari. Alhora també es vol crear l'aplicació web amb un enfocament de curtes iteracions incrementals adaptant-se als requisits i errors que es trobin.

1.1.4. Enfocament i mètode seguit.

La metodologia serà DevOps que inclou integració continua i desplegament continu. Això significa que s'haurà que intentar automatitzar el màxim possible tots els processos i crear un "pipeline" perquè amb una pujada dels canvis al repositori remot, aquests canvis puguin veure-s'hi reflectits a l'aplicació en línia un cop superades les proves.

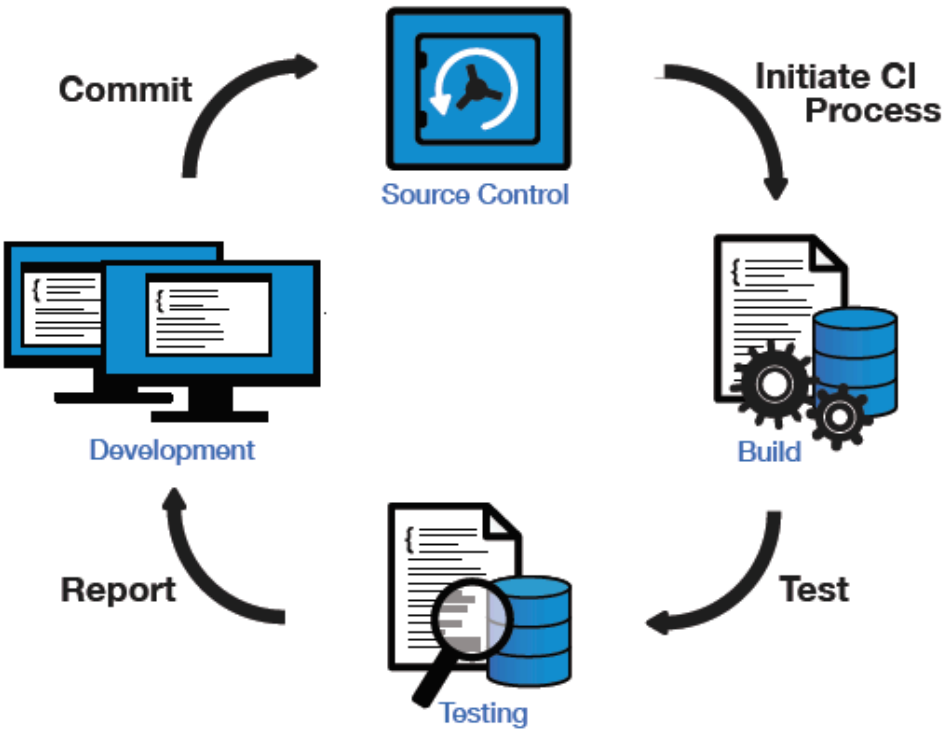
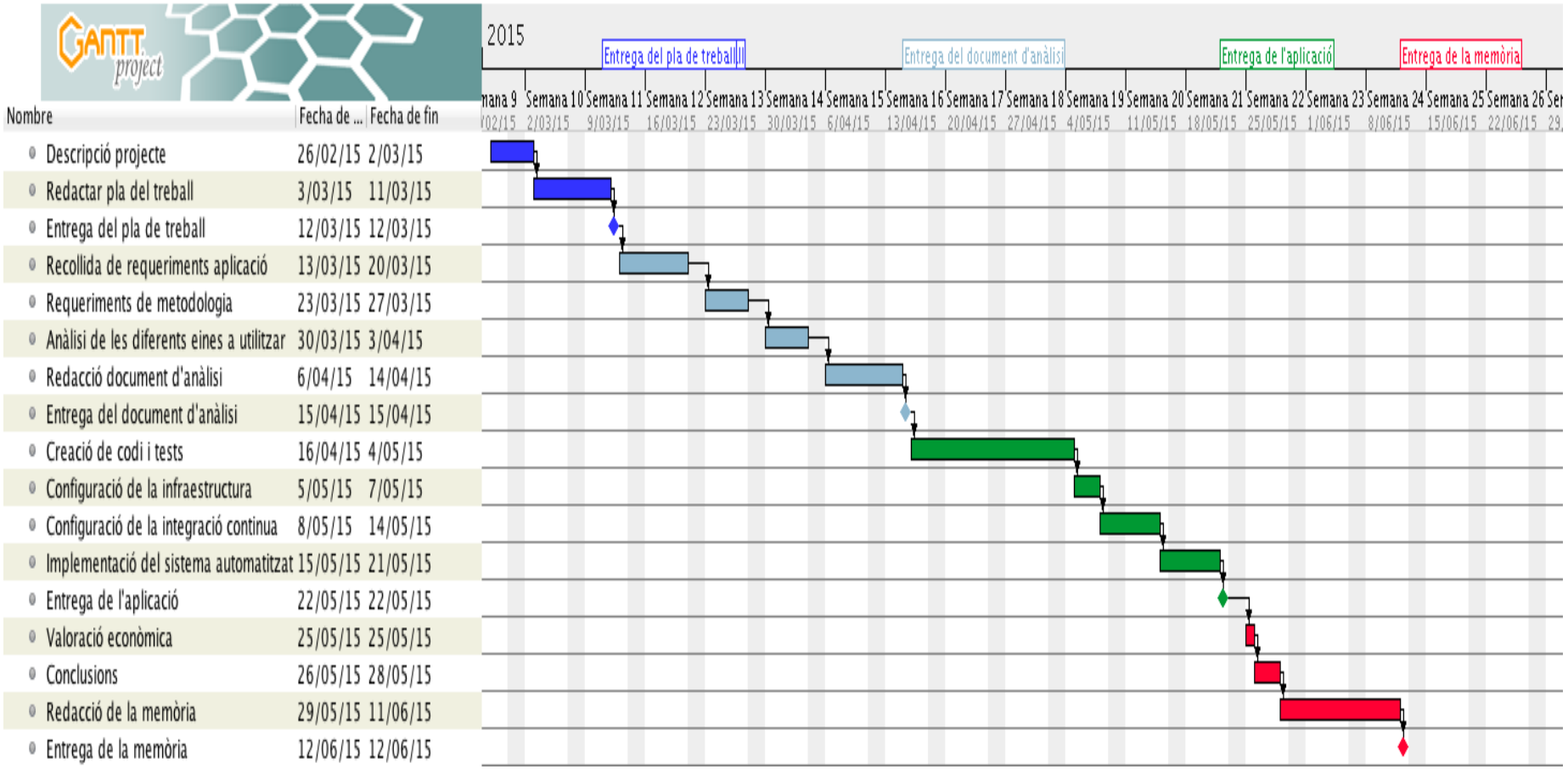


Figura 3 - Cicle de desenvolupament amb integració contínua

1.1.5. Planificació del projecte.



1.1.6. Productes obtinguts

En finalitzar aquest projecte s'haurà creat una aplicació web per fer enquestes i alhora un servidor d'integració que executarà les proves, controlarà i desplegarà tots els canvis a l'aplicació web.

Es podrà tenir accés a l'aplicació a la següent adreça : http://comas.me:8080

Figura 4 - Pàgina inicial de l'aplicació web

També es podrà tenir accés a l'historial d'integracions en el servidor a l'adreça :

http://jenkins.comas.me:8081

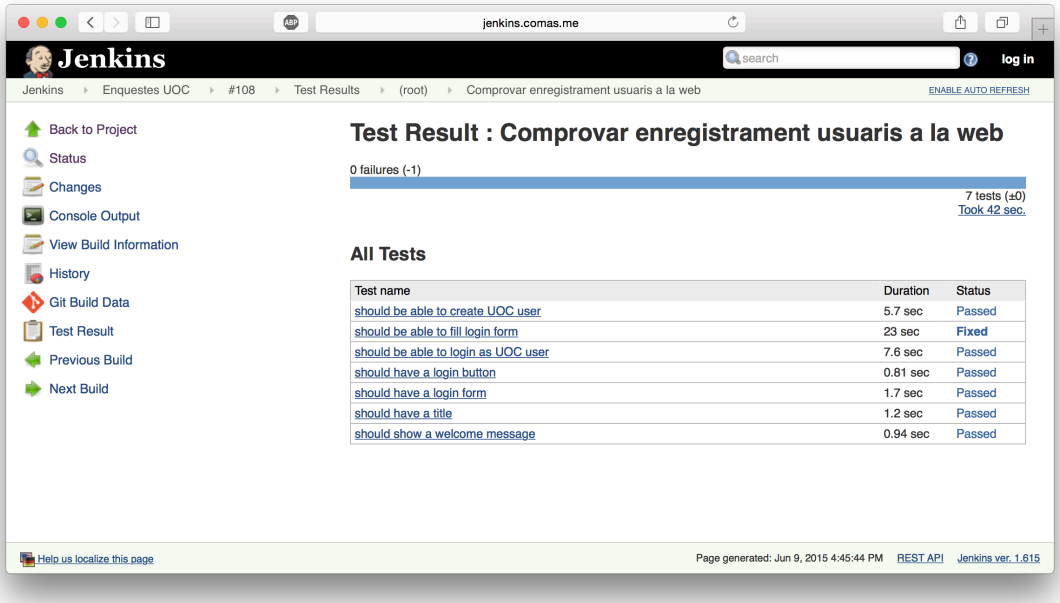


Figura 5 - Informe de resultat de proves

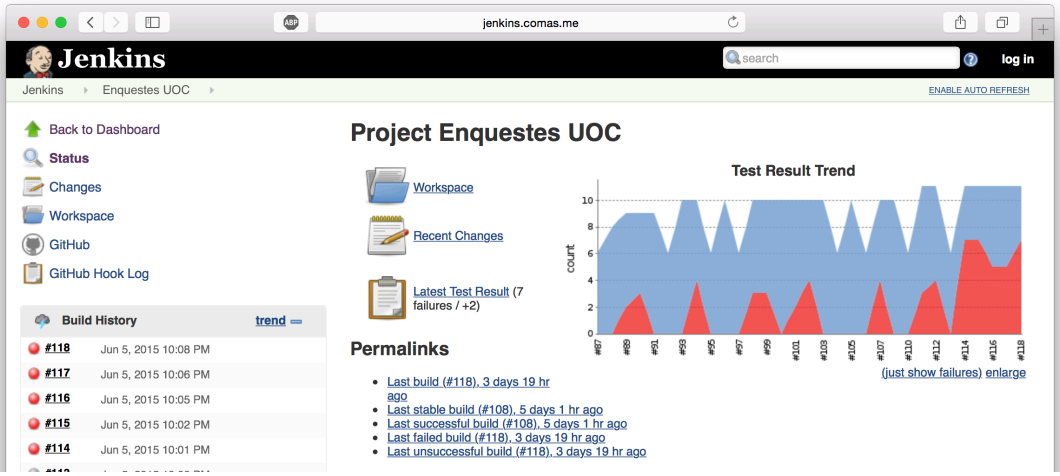


Figura 6 - Gràfic d'evolució de les proves superades

1.1.7. Breu descripció dels altres capítols de la memòria

En els següents apartats es diferenciaran dues fases principalment, a la primera part s'analitzarà en profunditat que és la integració contínua, quines són els seus principals trets característics i altres aspectes per entendre amb més claredat quins avantatges pot aportar plantejar-se l'execució d'un projecte amb aquesta metodologia. Un cop analitzat els diferents arguments i el plantejament teòric a continuació es farà un projecte web real. Aquest projecte serà la creació d'una aplicació web amb elements bàsics però indispensables per una aplicació web moderna avui en dia.

2. Desenvolupament àgil

2.1. Introducció

En aquest apartat volem introduir la metodologia de desenvolupament de software DevOps que podríem dir que és en part similar a *l'Extreme Programming* (XP) però integrant altres components com la infraestructura TI i l'anàlisi de qualitat de programari.

2.2 Desenvolupament tradicional de programari i el canvi àgil

2.2.1 Desenvolupament tradicional

Les metodologies de desenvolupament de software han evolucionat però el procés de transformació d'idees en codi encara implica diverses activitats com la recopilació de requisits, disseny, l'arquitectura, implementació i proves. Els mètodes de desenvolupament àgil de programari van sorgir a finals dels anys 90, aquests mètodes proposen un nou enfocament per organitzar aquest tipus d'activitats. En lloc de realitzar el procés en fases diferents, com la metodologia en cascada, ara succeeixen al mateix temps en iteracions curtes. Al final de cada iteració, el programari es torna més i més útil, amb noves característiques i menys errors, i l'equip decideix amb el client el que ha de ser el proper sector o funcionalitat que es vol desenvolupar.

Tan aviat com el client decideix que el programari està llest, el codi s'allibera a producció i els usuaris reals poden començar a utilitzar el sistema. En aquest moment noves preocupacions es tornen rellevants: suport, monitoratge, seguretat, disponibilitat, rendiment, facilitat d'ús, entre d'altres. Quan el programari es troba en producció, la prioritat és que segueixi funcionant de forma estable. En casos d'error o desastre, l'equip ha d'estar preparat per reaccionar amb rapidesa per resoldre el problema.

Figura 7 – Convergència DevOps

A causa de la naturalesa d'aquestes activitats, molts departaments de TI tenen una clara separació de responsabilitats entre l'equip de desenvolupament i l'equip d'operacions. L'equip de desenvolupament és responsable de la creació de nous productes i aplicacions, l'addició de funcionalitats o corregir errors, mentre que l'equip d'operacions és responsable de tenir cura d'aquests productes i aplicacions en producció. S'anima l'equip de desenvolupament a introduir canvis, mentre que l'equip d'operacions és responsable de mantenir les coses estables.

Aquesta diferència de dinàmica entre els equips que creen el programari i els que assisteixen en la resta de la cadena fa que hi puguin sorgir problemes, com per exemple de comunicació i en conseqüència un increment de canvis que aportarà un increment de risc en l'estabilitat i qualitat del producte. Això és el que passa en molts projectes quan els equips de desenvolupament, tests, control de qualitat i manteniment no es comuniquen de manera àgil. També aquests mètodes clàssics comporten un major cost a l'hora d'entregar projectes de programari si els requisits del client van canviant freqüentment, això és degut al fet que la comunicació entre equips no és immediata i un desenvolupador que revisi la qualitat de l'última versió desplegada pot està fent feina innecessària si el codi es canvia per adaptar-se als nous requisits.

2.2.2. El canvi àgil

Moltes empreses d'Internet amb èxit - com Google, Amazon, Netflix, Flickr i Facebook - es van adonar que el retard d'un desplegament de producció significa retardar la seva capacitat per competir i adaptar-se als canvis del mercat. En molts casos necessiten pujar canvis a producció més de 10 cops al dia. (wikipedia, 2015)

La línia de pensament que tracta de disminuir el temps entre la creació d'una idea i la seva aplicació en la producció també es coneix com ***Integració Contínua***, i està revolucionant el procés de desenvolupament i distribució de programari. Quan el procés d'implementació deixa de ser una cerimònia i comença a convertir-se en un lloc comú, es pot veure com els problemes de comunicació entre equips es disminueixen. L'augment de la freqüència de desplegament fa que la quantitat de canvi en cada desplegament disminueixi, reduint també el risc associat a aquest desplegament. Aquest benefici no és una cosa intuïtiva, però quan alguna cosa va malament és molt més fàcil d'esbrinar el que va passar perquè la quantitat de canvis que poden haver causat el problema és menor.

No obstant això, la reducció del risc no implica una eliminació completa dels processos entre els equips de desenvolupament i operació. El factor clau que permet la inversió del cicle és l'automatització de processos. Automatitzar el procés d'implementació permet que s'executi de forma fiable en qualsevol moment, eliminant el risc de problemes causats per errors humans. Totes aquestes pràctiques i integracions donen lloc a la metodologia DevOps, que com s'ha indicat abans també inclou altres aspectes a automatitzar o convertir en àgils, ja sigui fent test abans d'incrementar la versió (*TDD*), monitoratge, desplegament automatitzat, etc.

2.3. Integració contínua

Com volem estudiar DevOps des de la vessant de l'enginyeria de programari, analitzarem en detall un dels components que forma part d'aquesta metodologia que és la ***integració contínua***.

2.3.1. Què és la integració contínua?

Segons Martin Fowler, persona que va fer la primera proposta, la integració contínua és una pràctica de desenvolupament de programari on els membres d'un equip integren el seu treball freqüentment, habitualment una persona integra el seu treball com a mínim un cop al dia. Cada integració és verificada per un sistema d'integració automatitzat que avalua el tests i la compilació per detectar error tan d'hora com sigui possible. Molts desenvolupadors i empreses consideren que aquesta pràctica redueix els problemes d'integració en projectes de programari, i alhora els hi permet desenvolupar programari de manera més ràpida i cohesionada.

2.3.2. Quin benefici aporta la integració contínua en el desenvolupament de programari?

Reducció de riscos: A l'integrar diverses vegades al dia es poden reduir riscs en el projecte. Fent això es facilita la detecció d'errors, l'anàlisi de la qualitat i la reducció de pressuposicions de les coses fetes. En fer tests sempre automatitzats es pot mesurar quina és la qualitat i complexitat del codi creat.

Reducció de processos repetitius manuals: En reduir processos repetitius les empreses i els equips de desenvolupament estalvien temps, costos i esforç. Un dels avantatges en crear un procés d'integració contínua més automatitzat és que el procés funciona a cada iteració igual amb cada canvi de codi, això permet alliberar de feina al desenvolupador evitant els processos manuals de manera que pot dedicar a desenvolupar funcions que aportin més valor en el producte.

Creació de programari preparat per anar a producció en qualsevol moment: Si s'aconsegueix crear un cicle automatitzat, cada nova funcionalitat o detall que es vulgui modificar del codi s'aplica directament al producte en producció, ja que haurà passat tots els tests automatitzats del cicle i el client o usuaris podran utilitzar abans les noves funcions o millores en comptes d'esperar dies o setmanes a què els canvis es vegin reflectits en el producte final.

Millora la visibilitat del projecte: La integració contínua aporta l'habilitat de detectar cap a on es dirigeix el projecte gràcies a mètriques i així prendre decisions efectives, alhora permet donar més facilitat a innovar perquè es perd la por a trencar el cicle de desenvolupament. Habitualment les mètriques són analitzades manualment per cada desenvolupador, però amb la integració contínua aquest procés hauria de ser automatitzat i en conseqüència ajudar a veure com un projecte falla o no segons quines funcions es van afegint o si la complexitat del codi creat aporta la mateixa estabilitat al projecte o no.

Tenir la certesa que el programari desplegat a producció funciona : Com s'ha indicat anteriorment, amb cada canvi de codi es genera un canvi en el producte si aconsegueix passar els tests, sinó no s'aplicaran els canvis a producció. Aquesta característica aporta confiança a l'equip perquè saben que si alguna cosa nova és a producció vol dir que ha passat tots els tests anteriors. Això també és gràcies al fet que un sistema d'integració contínua ben implementat pot notificar quan alguna cosa nova no assoleix els estàndards mínims de disseny.

2.3.3. Què fa que els equips no practiquin integració contínua en els seus projectes?

Manteniment d'un sistema d'integració contínua: La percepció de què s'ha de dedicar massa esforç al manteniment del sistema d'integració contínua és imprecisa, perquè si no hi ha un sistema així, s'haurien de realitzar aquestes tasques igualment, ja sigui integracions de diferents desenvolupadors, tests i inspeccions de codi. La diferència és que ho haurà de fer cada desenvolupador manualment i independentment.

Massa canvis: Pot ser un problema si es vol aplicar tots els elements de la integració contínua de cop, però la millor manera és anar afegint elements de manera progressiva. Primer es va afegint elements com la compilació diària i després s'aniran afegint els tests etc.

Massa errors de compilació: Aquest problema existeix quan els desenvolupadors no realitzant una compilació privada abans de pujar el codi al servidor de versions de

codi. Pot ser degut al fet que un desenvolupador no realitza els tests correctament o se n'ha oblidat. Aquest problema és important a l'hora d'utilitzar la integració contínua perquè es vol realitzar múltiples canvis al codi freqüentment.

Cost afegit de hardware i software : És cert que en desenvolupar programari mitjançant integració contínua es necessita un servidor per fer les integracions de codi. Però el cost d'aquest servidor serà inferior que el cost que tindrà trobar i arreglar problemes més endavant en el cicle de vida del desenvolupament del programari.

El desenvolupador hauria de fer això individualment : És cert que els desenvolupadors ja han de fer part d'aquestes pràctiques de manera privada, però quan els projectes tenen diversos desenvolupadors treballant alhora és necessari realitzar aquestes integracions en un entorn net i estandarditzat.

2.3.4. Com s'aplica la integració contínua?

Els processos esmentats a continuació s'haurien d'intentar aplicar al màxim possible d'activitats d'un projecte de programari.

1) Identificar: Primerament s'identificarà un procés que necessiti automatització, ja sigui la compilació de codi, els tests, revisió de codi, desplegament a producció o fins i tot la integració de la base de dades.

2) Construir: En segon lloc es crea el codi que permeti realitzar les tasques d'automatització, aquest codi permetrà repetir aquests processos amb consistència i sense la necessitat de crear cada vegada una nova base de codi. Un dels exemples pot ser l'eina Ant per l'automatització de compilació de projectes Java.

3) Compartir: En tercer lloc es comparteix el codi creat. En treballar en equip és important allotjar el codi en servidors de control de versió de codi, això permetrà a altres integrants de l'equip reutilitzar codi i fer que el projecte avanci més ràpid.

4) Fer-ho continu : Per últim s'ha de certificar que tot el procés és continu i s'aplica automàticament en cada canvi que es fa al producte.

2.3.5. Quan és adient la integració contínua?

És més fàcil implementar una metodologia d'integració contínua en l'inici d'un projecte que fer-ho en fases avançades. Això és degut al fet que en les fases finals de projectes els equips tenen més pressió per les dates d'entrega i són contraris a fer canvis a la metodologia de treball. Dit això, no és impossible implementar la integració contínua en fases avançades però caldrà començar amb petits canvis que seran incrementals. Per exemple es pot començar amb la compilació del codi però no aplicar l'automatització dels tests d'integració fins que l'equip estigui conforme amb el correcte funcionament del primer pas. S´ha destacar que la integració contínua no és només una implementació tècnica, també és una implementació cultural i d'organització.

2.3.6. Com es pot practicar la integració contínua?

Perquè la integració contínua funcioni de forma eficaç en un projecte, els desenvolupadors han de canviar els seus hàbits en desenvolupar el programari en el dia a dia. Els integrants de l'equip han d'enviar els canvis al repositori de codi diàriament, donar màxima prioritat a solucionar un problema de compilació abans d'enviar codi al repositori, escriure codi que passi el 100% dels tests i per últim no descarregar o pujar codi amb errades al repositori en comú.

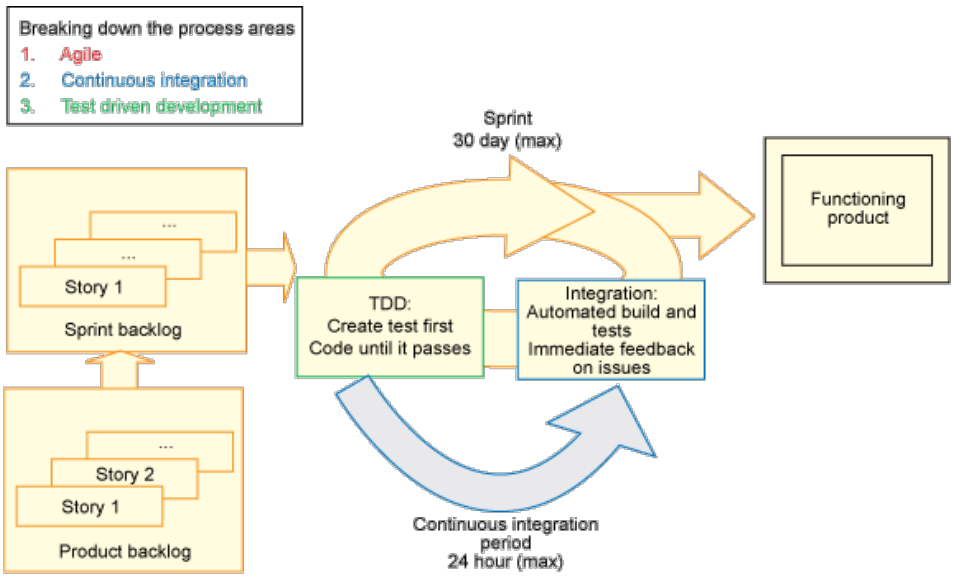


Figura 8 - Procés per desenvolupar programari amb integració contínua.

3. Creació de l'aplicació web

Com volem aplicar una metodologia de treball àgil estructurem el projecte en petites iteracions. Per definir aquestes iteracions ho farem basant-nos en diferents històries o casos d'ús representatiu, aquestes històries estan enfocades a les situacions i els requeriments que poden tenir els usuaris de l'aplicació web. Podem agrupar primerament les històries en el que s'anomena *product backlog* que ens dóna una visió genèrica de com volem distribuir les tasques necessàries per assolir aquells requeriments, quins integrants de l'equip haurien de tenir adjudicada cada tasca i les hores necessàries. També organitzem segons les històries en el *sprint backlog* per veure com els requeriments van encadenant-se i tenir una organització de com avança el projecte respecte a les iteracions.

Tenint en compte que volem valorar principalment la metodologia i no tant l'aplicació, aquesta no hauria d'exigir moltes setmanes de programació per a desenvolupar-la. Dit això l'aplicació tindrà les funcions principals que se li poden demanar a una aplicació web potent com: sistema d'enregistrament d'usuaris, sistemes d'autentificació, enregistrament de dades en base de dades, ús de crides a un *backend* que té una API exposada i també separació de les capes de presentació, persistència i controladors.

En aquest quadre mostrem com s'hauria de distribuir la planificació de les tasques segons les necessitats dels usuaris.

Product backlog

Fita	Tasca	Estat	Estimació hores	Dia 1	Dia 2	Dia 3	Dia 4	Dia 5	Revisió
Com a / visitant de la web vull veure una pàgina de inici per saber on estic accedint.	1	Obert	3	2	0	0	0	0	1
Com a / visitant de la web vull tenir la possibilitat d'enregistrar-me.	2	Pendent	13	6	6	0	0	0	1
Com a / usuari enregistrat vull poder crear una enquesta	3	Pendent	9	0	0	7	1	0	1
Com a / usuari de la web vull poder votar una enquesta.	4	Pendent	3.5	0	0	0	2	1	0.5
Com a / visitant de la web vull poder veure els resultats d'una enquesta.	5	Pendent	2.5	0	0	0	0	2	0.5

Taula 1 - Product backlog

Sprint backlog

id	Àrea	com a	voldria...	per tal que ...	afegit al sprint número	tancat al sprint número
1	Contingut de la web	Visitant de la web	Veure una pàgina inicial amb informació	Pugui observar el contingut que té el lloc web que estic accedint.	1	3
2	Personalització	Visitant de la web	Tenir la possibilitat d'enregistrar-me a la web	Pugui tenir accés a àrees que un visitant no té i personalitzar la meva interacció amb el lloc web.	2	2
3	Interacció	Visitant enregistrat de la web	Tenir la possibilitat de crear enquestes	Es pugui proposar temes de debat.	3	3
3	Interacció	Visitant enregistrat de la web	Tenir la possibilitat de votar enquestes	Es pugui enregistrar la meva opinió sobre un tema preguntat.	4	4
4	Interacció	Visitant enregistrat de la web	Veure els resultat de les enquestes	Saber quina és l'opinió de la majoria dels usuaris enregistrats en un tema específic.	5	5

Taula 2 - Sprint backlog

3.1. Primera iteració

Objectiu: Preparar entorn i crear pàgina de benvinguda

En aquest punt executarem la iteració que té com a objectiu crear una pàgina de benvinguda al lloc web, això s'identifica al *sprint backlog* amb **id 1**. Però primerament crearem l'entorn que ens ajudarà a desenvolupar l'aplicació web.

3.1.1 Entorn necessari

Per poder centrar-nos principalment en el desenvolupament i no pensar tant en la configuració de servidors web, etc. és necessari crear un procés automatitzat incloent eines que ens ajudin a detectar errors de desenvolupament de manera ràpida.

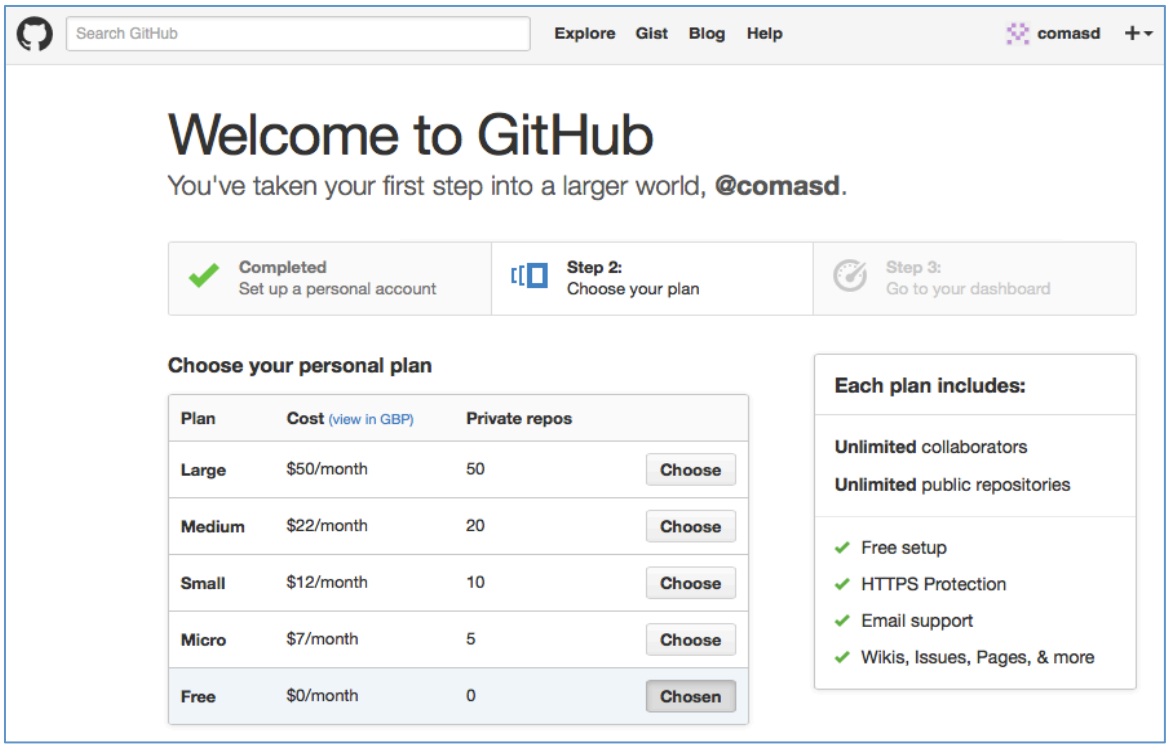
3.1.1.1 Control de versions

Per una banda necessitarem un repositori de codi per guardar totes les versions que anem creant a cada iteració diària. Avui en dia el més comú és utilitzar git com a control de versions d'arxius, podem crear el nostre propi repositori en un servidor privat però també és habitual utilitzar un servei com Github or Bitbucket que tenen versions gratuïtes per guardar els nostres projectes en el núvol.

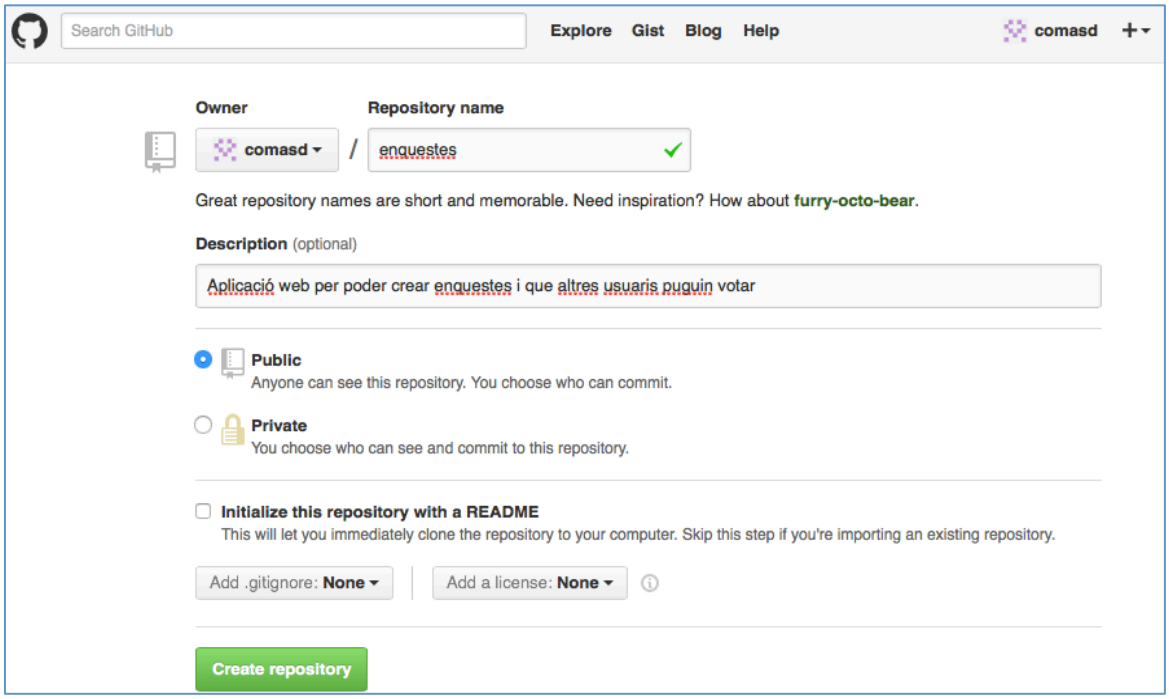
En aquest cas utilitzarem Github, a continuació esmentem els passos per crear un compte gratuït en aquest servei i crear el primer repositori de codi per l'aplicació web.

1) **Enregistrar-se**

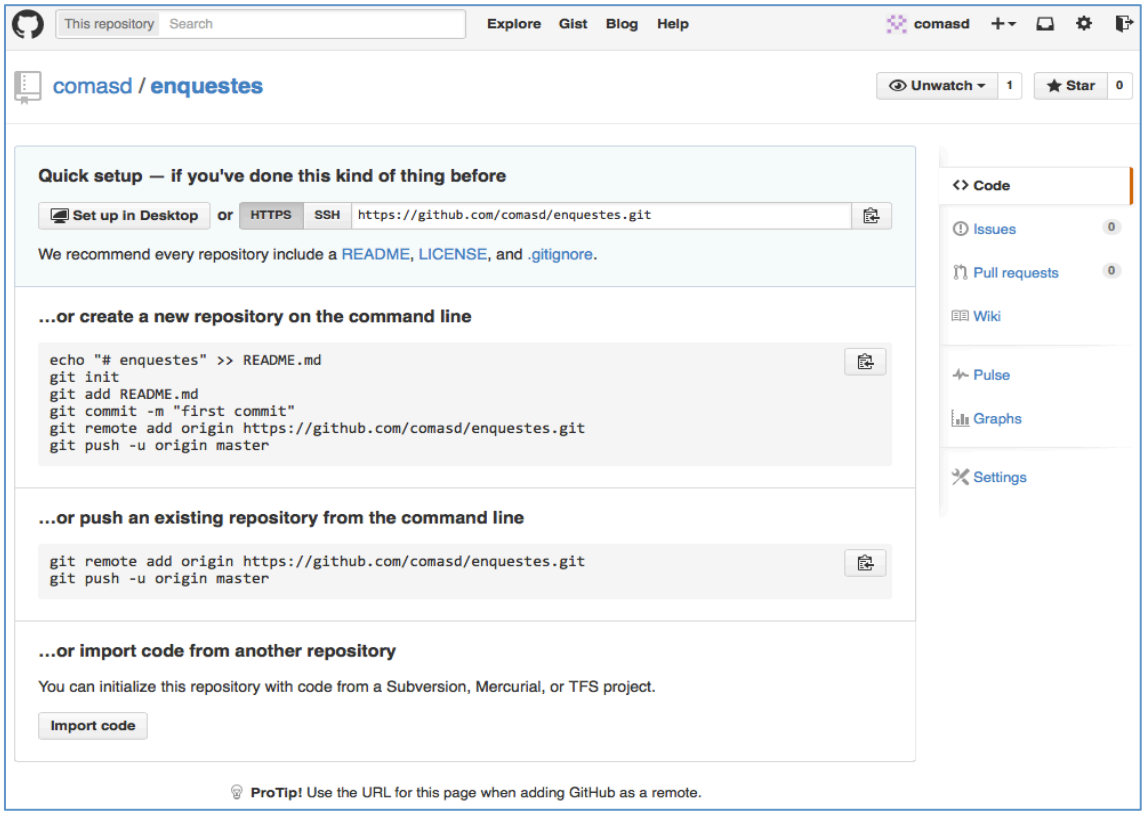
   Accedint a l'adreça www.github.com veurem una pàgina on podem crear un usuari gratuïtament:



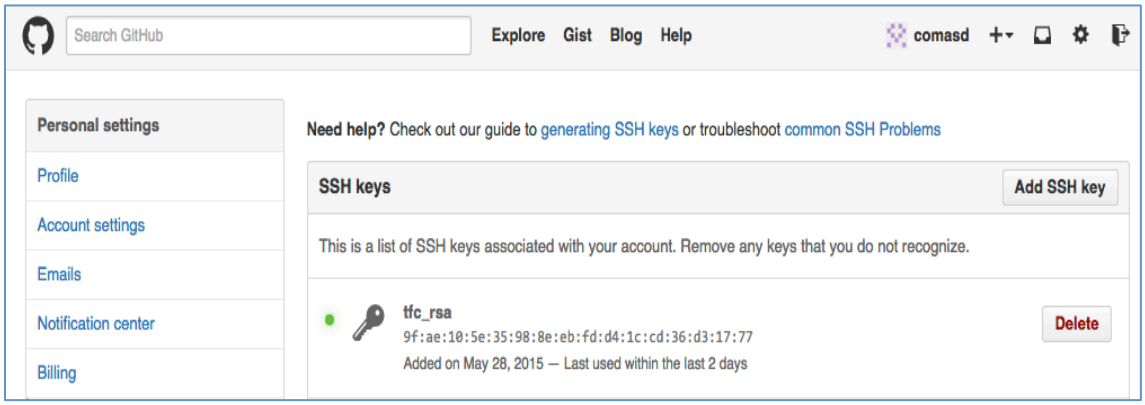
2) Crear un nou repositori



3) Crear repositori local i enllaçar-ho amb el repositori remot.

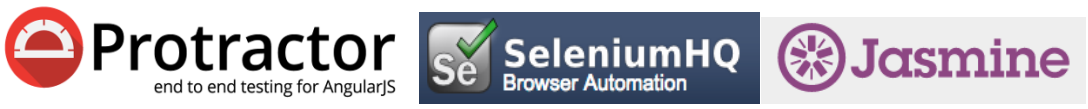


Per evitar el procés d'introduir la contrasenya cada cop que connectem amb el repositori remot, crearem unes claus SSH i copiem la clau pública de la nostra màquina a la configuració del nostre compte github.



3.1.1.2. Eines i arxius que executin les proves a l'aplicació.

Es necessita crear proves per cada funció que volem implementar al producte prequè és una de les bases de la integració contínua. Per això sempre hi haurà un arxiu que sigui la prova o prova abans de crear la funcionalitat i un cop superada la prova creant el codi adient, tindrem la seguretat que les modificacions són consistents i no afecten el treball anterior. Existeixen moltes eines de proves de codi però en aquest cas utilitzarem ***Protractor***, aquesta eina es diferencia perquè pot realitzar el que s'anomena proves "end to end" que vol dir que no només comprova el que fan les funcions internament sinó que va més enllà i amb l'ajuda de ***Selenium,*** realitza comprovacions en un navegador real com si fos un usuari final. En aquesta eina es defineixen totes les especificacions de les proves en arxius que anomenarem "spec". També cal dir que es necessita l'eina ***Selenium webdriver*** per realitzar la simulació real a navegadors web i el mòdul ***Jasmine*** per l'estructura del codi de les proves realitzades al codi Javascript.



En el cas inicial es pot crear un arxiu que retorni la resposta d'na pàgina de benvinguda.

3.1.1.3. Servidor web

En aquest cas s'utilitzarà un servidor virtual amb sistema operatiu Linux amb la distribució Ubuntu. Aquest servidor serà l'encarregat de tenir la versió de producció de l'aplicació web i publicar-la obertament a Internet. Per tenir tot automatitzat els canvis no es faran manualment sinó que tot el codi i scripts seran enviats des del servidor d'integració, un dels elements indispensables de la integració contínua.

Com s'ha fet en el cas del servidor de repositori, es crearan unes claus SSH per fer l'autentificació entre el servidor d'integració i el servidor web. Per l'aplicació s'ha

escollit un servidor virtual que ofereix Digital Ocean, hi ha milers d'alternatives però el preu és el més competitiu amb un rendiment correcte. Com en altres proveïdors de hosting, es pot escollir les aplicacions pre-instal·lades en el servidor, això ens estalvia temps i possibles errors d'instal·lació. En aquest cas hem seleccionat per instal·lar el bloc ***MEAN*** (Mongo DB, Express JS, Angular JS i Node JS).

3.1.1.4 Servidor d'integració

Un cop executats i havent superat les proves locals, es necessita un servidor que executi les proves per cada canvi enviat al repositori remot. Aquest servidor també pot aplicar els canvis al servidor web de producció per automatitzar tot i tenir un *feedback* més ràpid. Aquest és un element clau de la integració contínua, ja que serem més àgils si podem detectar els errors més ràpidament i en conseqüència millorar la consistència i qualitat del codi i en definitiva del producte final. Avui en dia el més comú és utilitzar un servidor amb Jenkins com programari d'integració. Jenkins incorpora un servidor web que publica un lloc web d'administració on es configura i controla cada projecte a integrar.

3.1.2 Arquitectura de l'aplicació

L'aplicació web del nostre cas serà desenvolupada amb un bloc MEAN, que està format per les tecnologies Mongo DB, Express JS, Angular JS i Node JS. Mongo DB és una base de dades no relacional de documents. Node JS és un entorn de programació dissenyat per escriure aplicacions d'Internet escalables. Express JS és un framework per simplificar les crides de servidor web que es fan amb Node JS. Angular JS és un framework per desenvolupar aplicacions web, majoritàriament per

aplicacions d'una pàgina dinàmica i sense forçar un refresc al navegador web al canviar de secció dins del mateix lloc web.

Aquestes tecnologies basades en JavaScript donen més agilitat a l'hora de crear un projecte web i també aconsegueixen un rendiment superior en la gestió de crides al servidor que facilita la connexió simultània d'usuaris a l'aplicació web. En contrapartida no és tan eficient en càlculs complexos d'algoritmes o funcions com podria ser aplicacions basades en C++, .NET o Java. En el cas de l'aplicació a desenvolupar no tindrà elements complexos de càlcul però si que hauria de ser ràpida a l'hora de gestionar molts clients simultanis perquè normalment les enquestes es voten en un termini breu accedint molts usuaris alhora.

3.1.3 Creació de les proves per satisfer la primera història del backlog.

Objectiu a examinar: Arribar a la pàgina d'inici.

Es crea un arxiu spec utilitzant la sintaxi de Jasmine per especificar quins elements ha de tenir l'aplicació web per assolir els requeriments definits, en aquest cas volem que s'arribi a una pàgina web a l'adreça localhost:5000, que es rebi el nom del títol del lloc web i que es mostri un missatge donant la benvinguda.

```
// spec.js
describe('Comprovar titol Enquestes UOC', function() {
    var message = element(by.css('.form-group'));

    beforeEach(function() {
        browser.get('http://localhost:5000');
    });

      it('should have a title', function() {
        expect(browser.getTitle()).toEqual('Hola UOC!');
      });

      it('should show a welcome message', function(){
        expect(message.getText()).toEqual('Benvinguts a la pagina enquestes UOC');
      });
});
```

Figura 9 - Arxiu spec primera iteració

Abans d'implementar el codi de la funcionalitat es crea la prova i es comprova que falli.

Comprovar que la prova falla:

```
server:test diek$ protractor conf.js
Using the selenium server at http://localhost:4444/wd/hub
[launcher] Running 1 instances of WebDriver
FF

Failures:

  1) Comprovar titol Enquestes UOC should have a title
   Message:
     Expected 'http://localhost:5000/' to equal 'Hola UOC!'.
   Stacktrace:
     Error: Failed expectation
    at [object Object].<anonymous> (/Users/diek/Dropbox/TFC_Codi/test/spec.js:10:36)

  2) Comprovar titol Enquestes UOC should show a welcome message
   Message:
     Expected '' to equal 'Benvinguts a la pagina enquestes UOC'.
   Stacktrace:
     Error: Failed expectation
    at [object Object].<anonymous> (/Users/diek/Dropbox/TFC_Codi/test/spec.js:14:35)

Finished in 4.395 seconds
2 tests, 2 assertions, 2 failures

[launcher] 0 instance(s) of WebDriver still running
[launcher] chrome #1 failed 2 test(s)
[launcher] overall: 2 failed spec(s)
[launcher] Process exited with error code 1
```

Figura 10 - Error confirmant proves han fallat

3.1.4. Modificació mínima del codi per que satisfaci i superi les proves.

Es crea l'arxiu ***index.html*** i aconseguim passar la primera prova.

```
<!DOCTYPE html>
<html ng-app>
<head>
    <meta charset="UTF-8">
    <title>Hola UOC!</title>

    <!-- CSS -->
    <!-- load bootstrap and our stylesheet -->
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.4/css/bootstrap.min.css">
    <style>
        body { padding-top:50px; }
    </style>

    <!-- JS -->
    <!-- load angular and our custom application -->
    <script src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.3.15/angular.min.js"></script>
    <script src="js/app.js"></script>
</head>

<!-- declare our angular application and angular controller -->
<body class="container" ng-app="firstApp" ng-controller="mainController as main">

<div class="jumbotron">

    <!-- form to update the message variable using ng-model -->
    <div class="form-group">
       <h1>Benvinguts a la pagina enquestes UOC</h1>
    </div>
</div>
</body>
</html>
```

Figura 11 - Arxiu index.html

Resultat :

Figura 12 - Missatge inicial de benvinguda

Valoració un cop finalitzada la iteració

- S'ha aconseguit fer un arxiu de proves per garantir que el codi funcioni.
- S'ha creat el producte que teníem com a objectiu en aquesta iteració.
- S'ha configurat un entorn per poder utilitzar un servidor web, un servidor d'integració i un repositori de codi remot.

- S'ha creat un procés automatitzat per enviar codi des de la màquina local al lloc web de producció, passant entremig pel repositori de codi remot i fent proves al servidor d'integració.

3.2. Segona iteració

Objectiu: Habilitar mètode per enregistrar-se a l'aplicació

Un cop finalitzada l'etapa anterior tornem a recopilar els requisits per aquesta segona iteració, necessitem crear unes proves per poder satisfer el segon sprint. En aquest cas com l'usuari necessita un mètode per enregistrar-se és obligatori crear una opció a la pàgina web per habilitar el control d'accés de visitants.

3.2.1 Creació de la prova per satisfer la segona història del backlog.

Objectiu a examinar : mètode per poder enregistrar-se a la web.

Primerament afegirem proves per comprovar si hi ha un botó anomenat botoLogin, després comprovarem si aquest botó funciona i ens obre un espai per introduir email i contrasenya de l'usuari, després comprovarem si s'hi pot fer clic a un botó de confirmació de login. Finalment també crearem un arxiu per comprovar que en introduir les dades rebem un missatge de confirmació de registre.

```
// spec.js
describe('Comprovar enregistrament usuaris a la web', function() {
    var message = element(by.id('benvingudaMessage'));
    var usuariNom = element(by.model('login.loginData.username'));
    var usuariPassword = element(by.model('login.loginData.password'));
    var confirmacioLogin = element(by.id('confirmacioLogin'));
    var botoLogin = element(by.id('botoLogin'));
    var botoConfirmaLogin = element(by.id('botoConfirmaLogin'));

    beforeEach(function() {
        browser.get('http://localhost:8000');
    });

      it('should have a title', function() {
        expect(browser.getTitle()).toEqual('Hola UOC!');
      });

      it('should show a welcome message', function(){
        expect(message.getText()).toEqual("Benvinguts a la pàgina d'enques
      });

      it('should have a login button', function() {
        expect(botoLogin.isPresent()).toBe(true);
      });

      it('should have a login form', function() {
        botoLogin.click();
        expect(usuariNom.isPresent()).toBe(true);
        expect(usuariPassword.isPresent()).toBe(true);
        expect(botoConfirmaLogin.isPresent()).toBe(true);

      });

      it('should be able to login as a user', function() {
        botoLogin.click();
        usuariNom.sendKeys("test");
        usuariPassword.sendKeys("test");
        botoConfirmaLogin.click();
        expect(confirmacioLogin.getText()).toEqual('Users NEW USER');
      });

});
```

Figura 13 - Arxiu de proves a superar a la segona iteració

Comprovar que la prova falla :

Executem les proves i confirmem que només les 3 noves fallen a la nostra aplicació web abans de realitzar canvis.

```
5 specs, 3 failures
Finished in 32.77 seconds
[launcher] 0 instance(s) of WebDriver still running
[launcher] chrome #1 failed 3 test(s)
[launcher] overall: 3 failed spec(s)
[launcher] Process exited with error code 1
```

Figura 14 - Confirmació noves proves fallen

3.2.2. Modificació mínima del codi per que satisfaci i superi les proves.

En aquest cas les modificacions són molt més amplies que a la iteració anterior. Ens trobem que podem assolir els requeriments del primer nou examen fàcilment modificant una mica de codi html per crear el botó login, però a continuació ens trobem que l'estructura de l'aplicació no està preparada per gestionar l'enregistrament d'usuaris.

Per solucionar aquest problema necessitarem una base de dades on es puguin guardar els usuaris i les contrasenyes, així podrem superar les proves. Dit això, per una altra banda necessitarem també canvis de la gestió d'accés a les diferents seccions de la pàgina web, no podrà tenir el mateix accés un visitant anònim en comparació a un usuari que s'ha donat d'alta. Això comporta que necessitem ampliar els elements de l'aplicació per gestionar aquests nous requeriments.

Com s'utilitza el conjunt d'eines MEAN, necessitarem per una banda crear diferents pàgines HTML per a servir diferent contingut dinàmic que serà gestionat pels controladors, serveis i vistes que hem de crear.

Es modifiquen els arxius html i server.js, es creen també els controladors, serveis, encaminadors i aconseguim passar el primer test.

```
5 specs, 0 failures
Finished in 7.242 seconds
[launcher] 0 instance(s) of WebDriver still running
[launcher] chrome #1 passed
```

Figura 15 - Confirmació proves superades

Resultat :

Pas 1 - Login

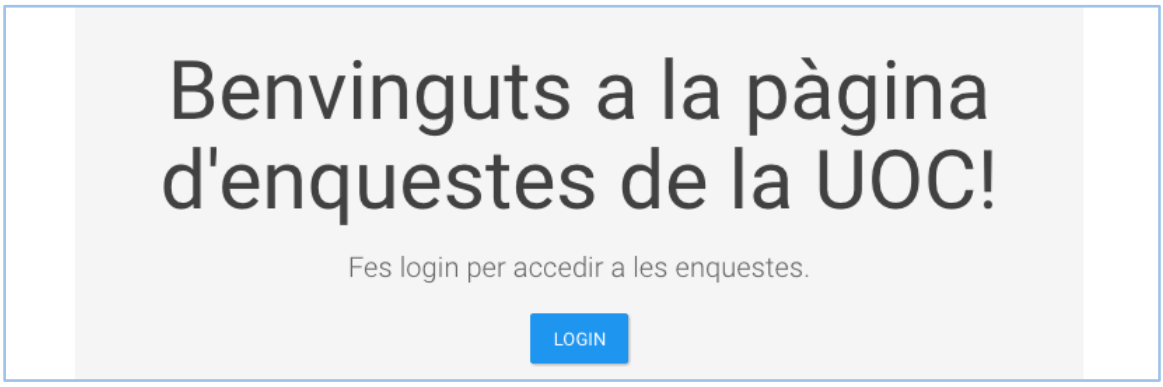


Figura 16 - Pàgina de benvinguda actualitzada amb botó Login

Pas 2 - Introduir usuari de prova

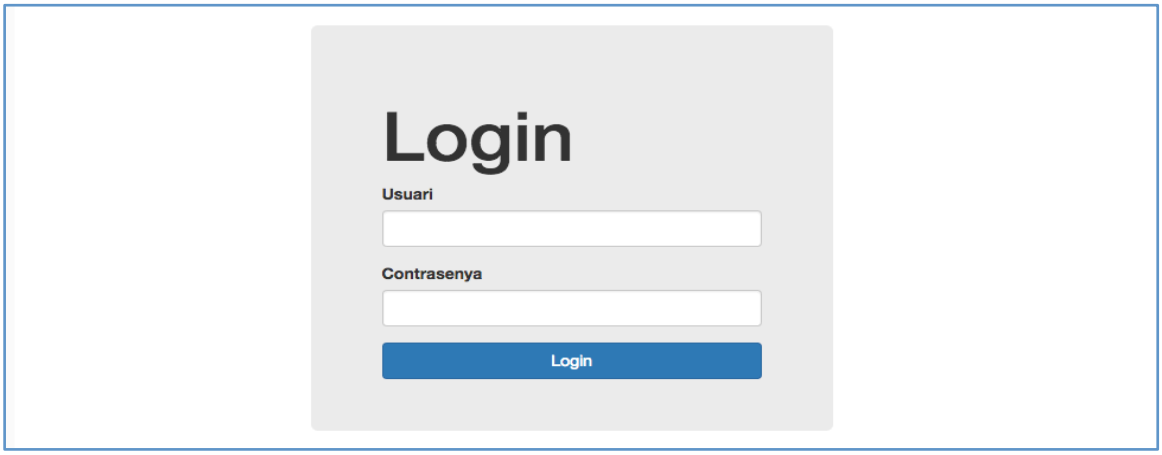


Figura 17 - Nova pàgina per introduir credencials

Pas 3 - Pàgina inicial per usuaris enregistrats correctament.

Figura 18 - Pàgina on arriven els usuaris enregistrats correctament.

Valoració un cop finalitzada la iteració

- S'ha aconseguit fer unes proves més sofisticades.
- Es pot comprovar la consistència de les noves àrees fetes de la web i no hem de patir que els nous canvis trenquin els apartats anteriors, perquè s'executen les proves de les funcions prèvies cada cop.

S'han creat noves vistes per diferenciar el que veuen els visitants del que veuen els usuaris enregistrats.

3.3. Tercera iteració

Objectiu: Habilitar la creació d'enquestes

Tornem a recopilar els requisits per aquesta tercera iteració, necessitem crear un arxiu de proves per poder satisfer el tercer sprint. En aquest cas l'usuari enregistrat necessita tenir la possibilitat de crear enquestes.

3.3.1. Creació de les proves per satisfer la tercera història del backlog.

Per tenir una millor organització anirem creant diferents arxius "spec" per les properes iteracions on es poden trobar les proves de cada àrea que volem examinar, en el seu defecte tindríem un arxiu únic "spec" però seria més difícil la comprensió del mateix al contenir tantes línies de codi.

Primerament afegirem les proves per examinar si el botó "Veure enquestes" ens porta a l'apartat on podrem crear enquestes. Com encara no s'ha de modificar el codi obtindrem que la prova ha fallat.

```
1 spec, 1 failure
Finished in 10.459 seconds
[launcher] 0 instance(s) of WebDriver still running
[launcher] chrome #1 failed 1 test(s)
[launcher] overall: 1 failed spec(s)
[launcher] Process exited with error code 1
```

Figura 19 - Confirmació d'error al executar les noves proves de la tercera iteració

3.3.2. Modificació mínima del codi per que satisfaci i superi les proves.

En aquest cas les modificacions són molt més amplies que la iteració anterior. Necessitem modificar les pàgines que ja teníem per incloure l'accés a les noves, també la generació de les pàgines que mostrin les enquestes, els nous models per guardar a la base de dades les enquestes i tota la seva informació.

Al crear les noves plantilles html i modificar els enllaços podem superar el primer examen de la iteració.

```
// spec veure i crear enquestes
describe('Comprovar acces a les enquestes ', function() {
    var message = element(by.id('benvingudaMessage'));
    var botoVeureEnquestes = element(by.id('botoLoginVeureEnquestes'));
    var usuariNom = element(by.model('login.loginData.username'));
    var usuariPassword = element(by.model('login.loginData.password'));
    var botoConfirmaLogin = element(by.id('botoConfirmaLogin'));
    var botoLogin = element(by.id('botoLogin'));

    it('should be able to login as UOC user and see polls', function() {
        browser.get('http://localhost:8000');
        botoLogin.isPresent().then(function(result){
            if (result) {
                botoLogin.click();
                usuariNom.sendKeys('UOC');
                usuariPassword.sendKeys('test');
                botoConfirmaLogin.click();
            } else {
                botoVeureEnquestes.click();
            }
        });
        expect(browser.getCurrentUrl()).toBe('http://localhost:8000/polls');
      });

});
```

Figura 20 - Nou arxiu spec amb noves proves a superar

Observem que superem la prova.

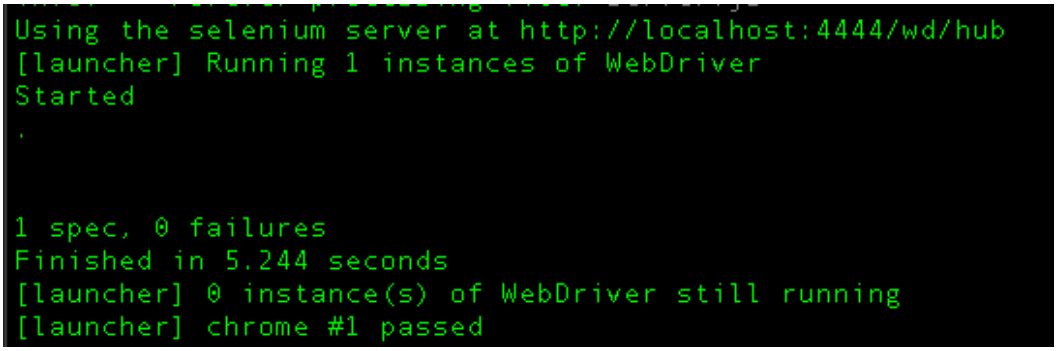


Figura 21 - Confirmació de proves superades en local

Un cop superada la prova, farem la combinació del nou "spec" amb l'anterior i podrem fer la seqüencia de totes les proves de forma automàtica.

Pàgina de creació de les enquestes

Figura 22 - Nova pàgina on arriben els usuaris al fer clic per veure enquestes.

Tot i que ja es pot accedir correctament a la pàgina on es crearan enquestes, s'ha d'actualitzar els models que es crearan a la base de dades per la informació que es vol introduir a cada enquesta. En definitiva objectes que inclouran una pregunta en format text i diverses respostes en format text també. També es necessita actualitzar els serveis per enllaçar les modificacions dinàmiques al crear elements amb el "*backend*".

Es crea una nova especificació al nou arxiu de prova. Es pot observar com es demana que es generi una nova pregunta i les seves respostes a votar. Primerament fallarà perquè hem de desenvolupar els canvis a l'aplicació. Un cop actualitzat tot executarem les proves. Succeeix un error a causa del fet que el segon examen no tenia en consideració que quan es fa el primer, l'usuari ja ha fet el login a la web. Actualitzem la prova perquè tingui aquest fet en consideració. Un cop fet això veiem que es passen tots els tests.

Alerta de què les proves han fallat

```
Using the selenium server at http://localhost:4444/wd/hub
[launcher] Running 1 instances of WebDriver
Started
.......F

Failures:
1) Comprovar acces a les enquestes  should be able to create a poll
```

Figura 23 - Error en combinar múltiples arxius spec

Arxiu de proves per crear enquesta.

```
// spec veure i crear enquestes
describe('Comprovar acces a les enquestes ', function() {
    var message = element(by.id('benvingudaMessage'));
    var botoVeureEnquestes = element(by.id('botoLoginVeureEnquestes'));
    var usuariNom = element(by.model('login.loginData.username'));
    var usuariPassword = element(by.model('login.loginData.password'));
    var botoConfirmaLogin = element(by.id('botoConfirmaLogin'));
    var botoLogin = element(by.id('botoLogin'));
    var botoCrearEnquesta = element(by.id('botoCrearEnquesta'));
    var introPregunta = element(by.id('pollQuestion'));
    var botoConfirmaEnquesta = element(by.id('botoConfirmaEnquesta'));

    it('should be able to login as UOC user and see polls', function() {
        browser.get('http://localhost:8000');
        botoLogin.isPresent().then(function(result){
            if (result) {
                botoLogin.click();
                usuariNom.sendKeys('UOC');
                usuariPassword.sendKeys('test');
                botoConfirmaLogin.click();
            } else {
                botoVeureEnquestes.click();
            }
        });
        expect(browser.getCurrentUrl()).toBe('http://localhost:8000/polls');
      });

    it('should be able to create a poll', function() {
        botoCrearEnquesta.click();
        introPregunta.sendKeys('Color favorit?');
        element.all(by.repeater('choice in poll.choices')).get(0).element(by.model("choice.text")).sendKeys('Blau');
        element.all(by.repeater('choice in poll.choices')).get(1).element(by.model("choice.text")).sendKeys('Negre');
        element.all(by.repeater('choice in poll.choices')).get(2).element(by.model("choice.text")).sendKeys('Blanc');
        browser.manage().timeouts().implicitlyWait(250);
        botoConfirmaEnquesta.click();
        expect(browser.getCurrentUrl()).toBe('http://localhost:8000/polls');
        expect(element(by.linkText('Color favorit?')).isPresent()).toBe(true);

      });

});
```

Figura 24 - Arxiu de proves actualitzat per detectar la creació d'enquestes

Totes les proves són superades de forma local.

```
Using the selenium server at http://localhost:4444/wd/hub
[launcher] Running 1 instances of WebDriver
Started
.........


9 specs, 0 failures
Finished in 18.855 seconds
[launcher] 0 instance(s) of WebDriver still running
[launcher] chrome #1 passed
```

Figura 25 - Confirmació de que les proves dels múltiples spec han sigut superades.

Proves al servidor d'integració

Un cop hem comprovat a la nostra màquina local que els canvis són correctes, enviem mitjançant un "git push" els nostres arxius al repositori de codi remot a Github. Veurem com automàticament el servidor Jenkins comença a fer tots els processos per examinar el codi i si supera les rutines especificades als arxius de proves, posteriorment si no troba cap error i pot enviar-ho a producció.

Resultats de la integració feta a Jenkins.

```
[32minfo[39m:    Forever processing file: [90mserver.js[39m
+ protractor test/conf.js
Using the selenium server at http://localhost:4444/wd/hub
[launcher] Running 1 instances of WebDriver
Started
[32m.[0m[32m.[0m[32m.[0m[32m.[0m[32m.[0m[32m.[0m[32m.[0m[32m.[0m[32m.[0m


9 specs, 0 failures
Finished in 28.24 seconds
[launcher] 0 instance(s) of WebDriver still running
[launcher] chrome #1 passed
+ ./scripts/deployLiveWeb
```

Figura 26 - Confirmació que totes les proves s'han superat al servidor d'integració.

Pàgina de creació d'enquestes amb vista actualitzada.

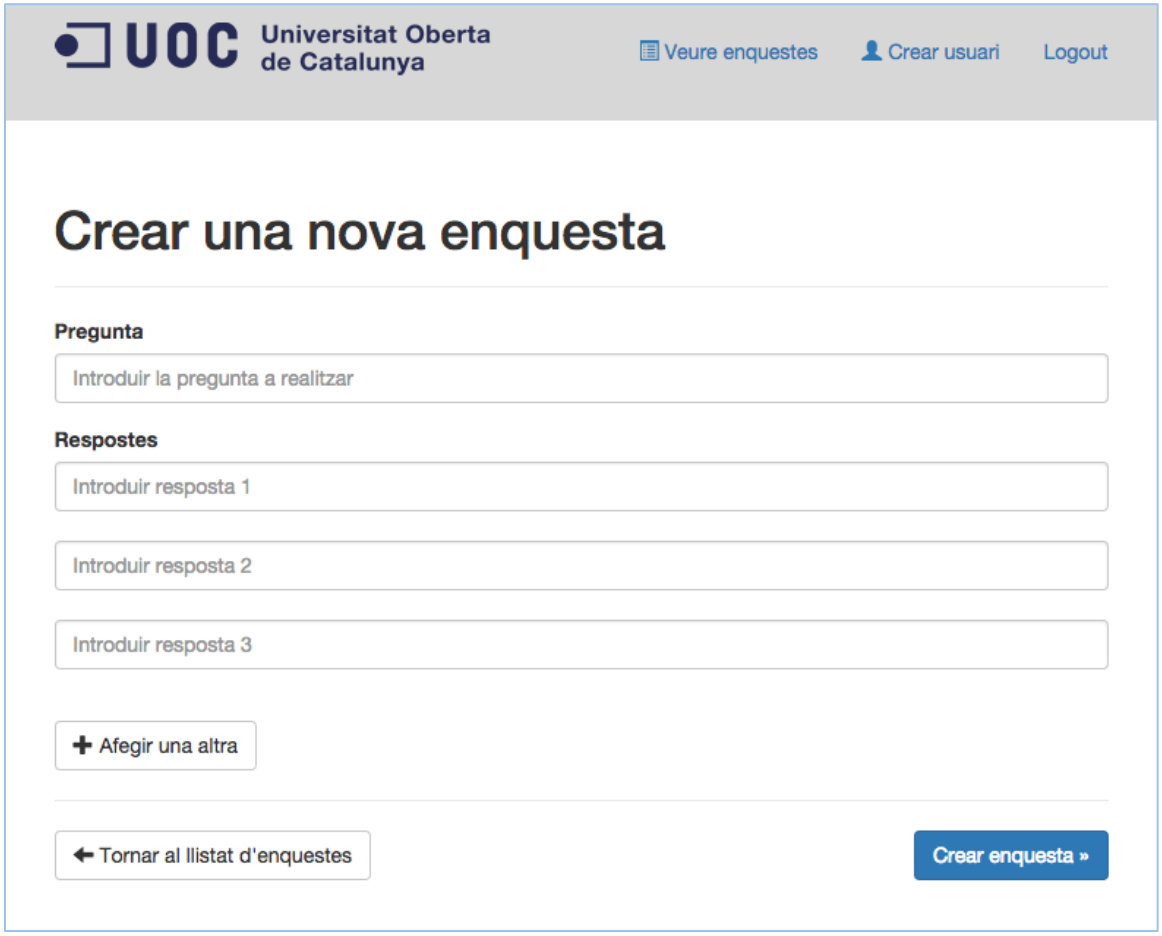


Figura 27 - Formulari per crear enquestes

Valoració un cop finalitzada la iteració

- S'ha aconseguit separar en diversos arxius les proves per tenir millor visibilitat de les especificacions.
- S'ha actualitzat el procés d'integració, és necessari netejar els documents de la base de dades perquè no hi entrin en conflicte amb la generació de dades iguals pel procés de proves.

3.4. Quarta iteració

Objectiu: Habilitar la possibilitat de votar enquestes

Tornem a recopilar els requisits per aquesta quarta iteració, necessitem crear una prova per poder satisfer el quart sprint. En aquest cas l'usuari necessita tenir la possibilitat de votar enquestes existents.

3.4.1. Creació de la prova per satisfer la quarta història del backlog

Nova especificació per provar el correcte funcionament de la votació.

```
it('should be able to vote an existing poll', function() {
    element(by.linkText('Color favorit?')).click();
    element.all(by.repeater('choice in poll.choices')).get(0).element(by.model("poll.userVote")).click();
    votarOpcio.click();
    expect(votFet.getText()).toEqual("Has votat l'opció Blau.");

});
```

Figura 28 - Nova especificació de proves per la quarta iteració.

3.4.2. Modificació mínima del codi per que satisfaci i superi els tests.

Un cop modificat el codi a l'aplicació se superen les proves.

```
server:TFC_Codi diek$ protractor test/conf.js
Using the selenium server at http://localhost:4444/wd/hub
[launcher] Running 1 instances of WebDriver
Started
..........


10 specs, 0 failures
Finished in 25.994 seconds
[launcher] 0 instance(s) of WebDriver still running
[launcher] chrome #1 passed
```

Figura 29 - Confirmació totes les proves s'han superat sense problemes.

Proves al servidor d'integració

Resultats de la integració feta a Jenkins.

```
[32minfo[39m:    Forever processing file: [90mserver.js[39m
+ protractor test/conf.js
Using the selenium server at http://localhost:4444/wd/hub
[launcher] Running 1 instances of WebDriver
Started
[32m.[0m[32m.[0m[32m.[0m[32m.[0m[32m.[0m[32m.[0m[32m.[0m[32m.[0m[32m.[0m[32m.[0m

10 specs, 0 failures
Finished in 53.254 seconds
[launcher] 0 instance(s) of WebDriver still running
[launcher] chrome #1 passed
+ ./scripts/deployLiveWeb
Deploying app to live server
```

Figura 30 - Confirmació que totes les proves s'han superat al servidor d'integració.

Pàgina de selecció de vot.

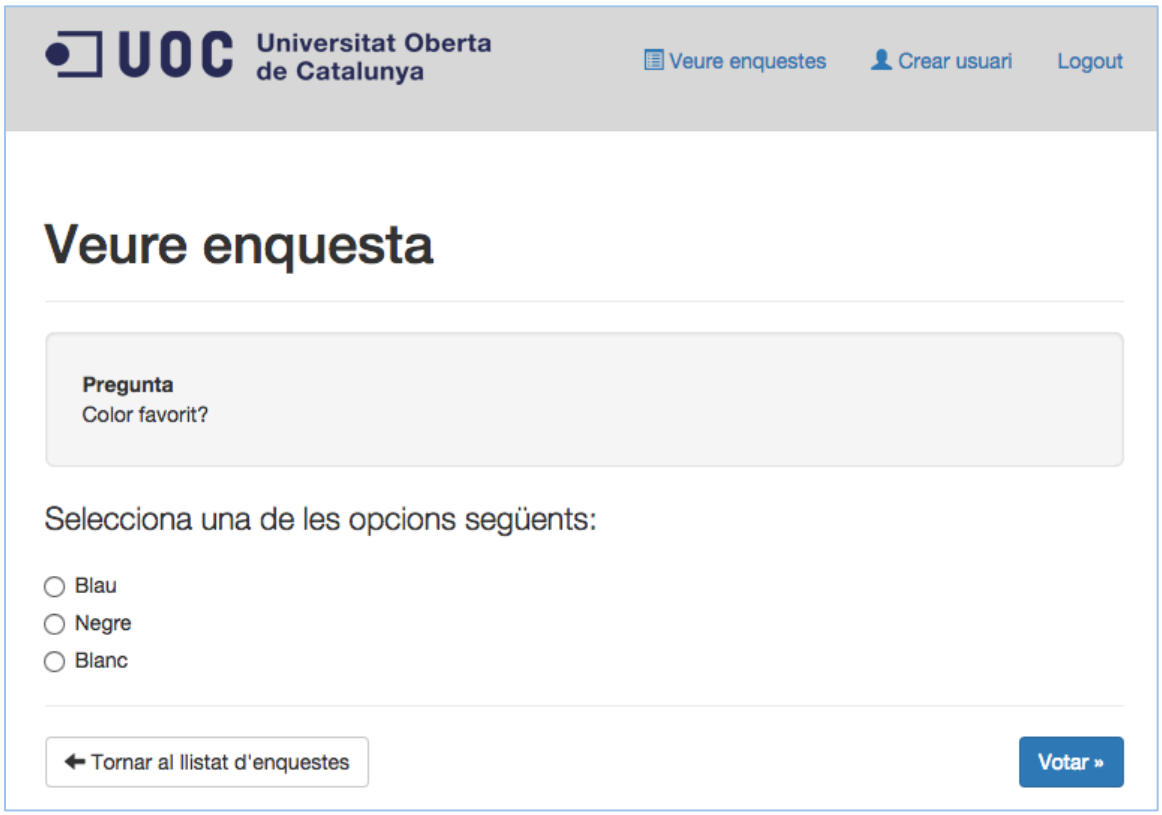


Figura 31 - Nova pàgina per votar enquestes.

Valoració un cop finalitzada la iteració

- S'ha aconseguit fer unes proves més sofisticades incloent elements nous a provar de forma dinàmica segons el renderitzat.

3.5. Cinquena iteració

Objectiu: Habilitar la possibilitat de veure resultat d'enquestes

Tornem a recopilar els requisits per l'última iteració, necessitem crear proves poder satisfer el cinquè sprint. En aquest cas l'usuari necessita tenir la possibilitat de veure els vots existents a les enquestes.

3.5.1. Creació de la prova per satisfer la quarta història del backlog.

```
it('should be able to count votes on existing poll', function() {
    expect(numVots.getText()).toEqual("1");
});
```

Figura 32 - Nova especificació per analitzar els vots.

3.5.2. Modificació mínima del codi perquè satisfaci i superi els tests.

En aquest cas no modificarem gaire, ja que amb la vista de la iteració anterior ja teníem l'estructura preparada per aquesta funcionalitat.

```
Using the selenium server at http://localhost:4444/wd/hub
[launcher] Running 1 instances of WebDriver
Started
...........


11 specs, 0 failures
Finished in 20.212 seconds
[launcher] 0 instance(s) of WebDriver still running
[launcher] chrome #1 passed
```

Figura 33 - Confirmació que totes les proves són superades en local.

Un cop modificat el codi a l'aplicació es superen les proves.

Proves al servidor d'integració

Resultats de la integració feta a Jenkins.

Les proves també són superades al servidor d'integració. A la imatge inferior ens podem fixar en totes les vegades que el servidor d'integració ha actuat. En total més de 100 vegades sumant les 5 iteracions, per tant de mitjana 25 per iteració, això està en concordança amb els mètodes d'integració contínua que promouen l'execució molt freqüent d'integracions al servidor d'integracions. Tot i que només es passa el codi al repositori remot quan el codi supera els tests locals, observem que hi ha hagut erros. Aquests errors han estat causats per problemes amb el Selenium driver que és poc estable en funcionar en un sistema sense interfície gràfica activada. En aquest cas s'ha fet una simulació d'entorn gràfic amb Xvfb.

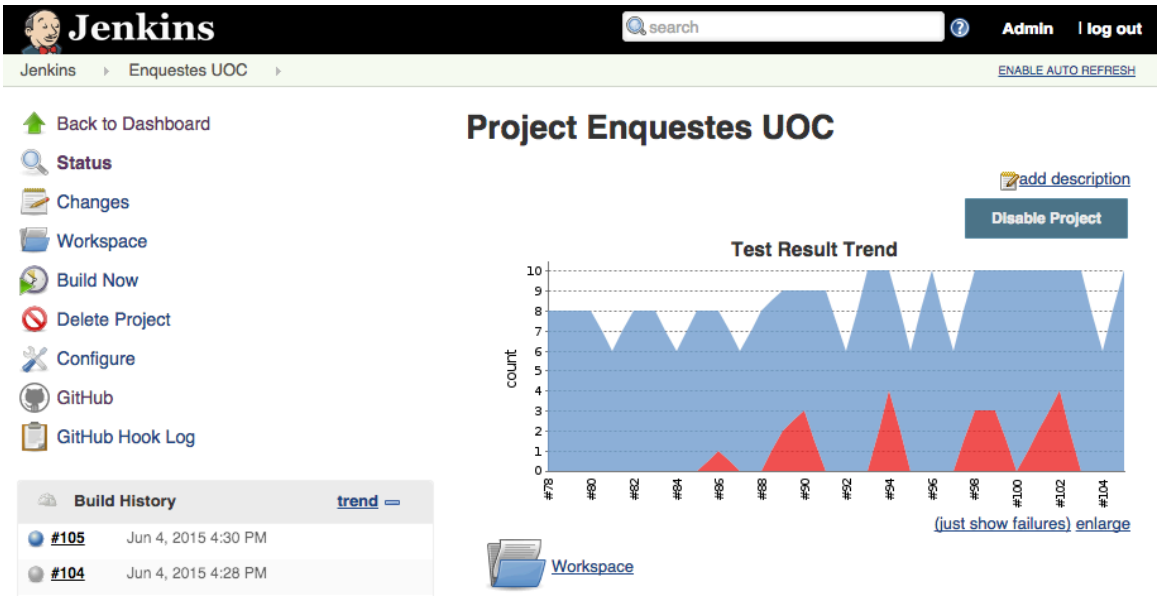


Figura 34 - Panell de control del servidor d'integració.

Vista de recompte de vots.

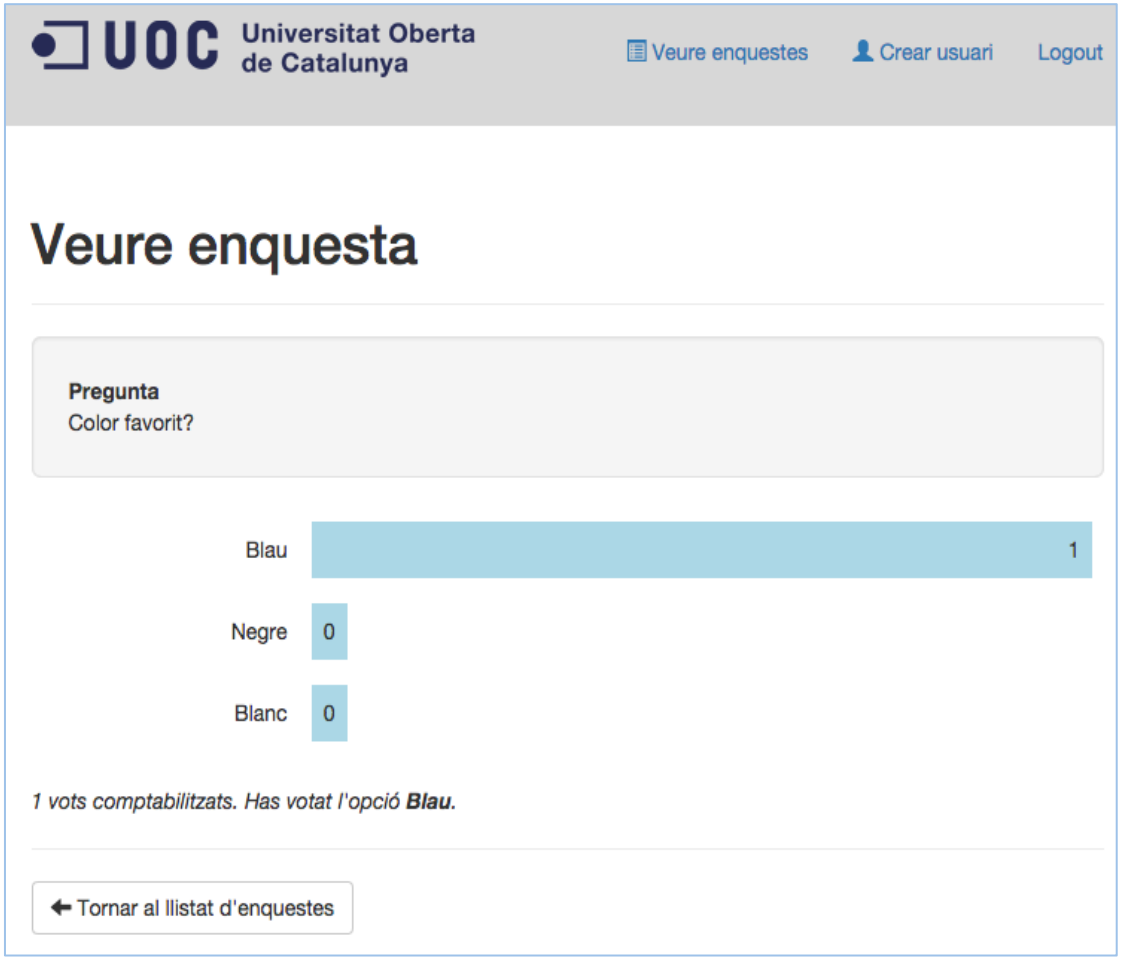


Figura 35 - Pàgina on es veuen els vots en una de les preguntes de la pàgina web.

Valoració un cop finalitzada la iteració

- S'ha aconseguit superar de manera seqüencial totes les proves que hem creat per l'aplicació. Des de la primera iteració a l'última.
- Amb les funcionalitats implementades podem satisfer totes les històries del backlog.

4. Conclusions

Un cop hem fet tot el procés de crear l'aplicació web mitjançant integració contínua, podem analitzar i valorar les avantatges i inconvenients que aporten a les diferents àrees de l'enginyeria de programari. De les etapes del cicle de vida de desenvolupament de programari en aquest cas analitzem com afecta les de proves del programari i manteniment.

4.1. Valoració respecte a les proves del programari

En aquesta àrea podem dir que durant el projecte hem creat les proves com a primer punt abans de desenvolupar qualsevol cosa, això forma part de les pràctiques àgils "TDD"(Desenvolupament dirigit per proves). El benefici que hem experimentat és que en crear primer els tests, se simplifica el disseny del programari. No hi ha desviació ni pèrdua de temps per pensar en models complexes, només es desenvolupa el que és necessari per superar els tests. En conseqüència el desenvolupament se centra en les necessitats de l'usuari final. Això també ha aportat major qualitat del programari perquè un cop desenvolupat el codi directament supera les proves unitàries i elimina possibles errors en les etapes finals del projecte.

Els beneficis que hem trobat en aquest projecte a l'utilitzar integració contínua són:

1) Major qualitat del programari creat.
2) Major simplicitat en el disseny de l'estructura del programari.
3) Disseny enfocat únicament a les necessitats de l'usuari.
4) Menys errors a resoldre en etapes finals del projecte.

En contrapartida podem dir que a vegades és difícil definir l'especificació d'una prova unitària sense saber com s'interactuarà amb l'aplicació. Per exemple en el

moment de crear enquestes amb múltiples respostes no sabem de partida com identificarem cada instància de les possibles respostes, ja que no són variables estàtiques sinó generades a voluntat de l'usuari, un usuari pot crear 3 possibles respostes i un altre 5 possibles. També és complicat crear proves per àrees que no són clau per l'aplicació, en el projecte es detecta que manca controlar situacions inesperades com quan l'usuari no fica text adequat per donar-se d'alta, Això provoca un error que no estava contemplat en les funcionalitats bàsiques.

Els inconvenients que hem trobat en aquest projecte són:

1) Complexitat per incloure tots els casos de decisions d'usuari.
2) Errors no identificats.

4.2. Valoració respecte al manteniment del programari

Per una banda en aquesta àrea podem dir que gràcies a la integració contínua és molt més fàcil ampliar l'aplicació, executar canvis o corregir errors. Ho hem comprovat clarament amb cada iteració, es podia crear noves funcionalitats a l'aplicació web sense patir per tot el codi implementat en iteracions anteriors. Això dóna molta agilitat i permet implementar canvis ràpidament al producte final. Aquesta característica té molt valor avui en dia a les empreses amb productes a Internet perquè un error que necessiti molt temps per a ser corregir a producció pot crear molta pèrdua d'ingressos*.

Els beneficis que hem trobat són:

1) Major agilitat en ampliar l'aplicació amb nous mòduls o funcionalitats.
2) Major rapidesa en corregir errors.
3) Menor cost en temps de dedicació a manteniment pels avantatges abans descrits.

En contrapartida per una altra banda una revolució d'alguna àrea ja feta pot generar un cost superior en manteniment, això és degut al fet que hi ha que reimplementar l'estructura base de l'aplicació. En el projecte això ho hem percebut en uns canvis que feien evolucionar l'accés a la creació d'enquestes, les consideracions de comprovar que l'usuari estava autentificat han necessitat més temps de reorganització del codi.

L'inconvenient principal que hem trobat és:

1) El canvi a la lògica de negoci repercuteix en el temps dedicat al manteniment de les àrees ja desplegades a producció.

* (Agileone, 2015)

5. Bibliografia

Agileone. *facebook-downtime-costs-them-more*. Retrieved 05 01, 2015, from www.agilone.com: http://www.agilone.com/blog/facebook-downtime-costs-them-more

Angula JS. *Protractor Test*. Retrieved 05 01, 2015, from angular.github.io: http://angular.github.io/protractor/#/

Canonical. *Ubuntu: The leading OS for PC, tablet, phone and cloud*. Retrieved 05 01, 2015, from ubuntu.com: http://www.ubuntu.com

Digital Ocean. *Simple Cloud Infraestructure for developers*. Retrieved 05 01, 2015, from digitalocean.com: https://www.digitalocean.com/

Fowler, M. *Continuous Integration*. Retrieved 05 01, 2015, from martinfowler.com: http://martinfowler.com/articles/continuousIntegration.html

Google. *Angular JS*. Retrieved 05 01, 2015, from Angular JS: https://angularjs.org/

IBM. *continuous-integration-agile-development*. Retrieved 05 01, 2015, from ibm.com: http://www.ibm.com/developerworks/rational/library/continuous-integration-agile-development/

Indeed.com. *Job trends*. Retrieved 05 01, 2015, from indeed.com: http://www.indeed.com/jobtrends?q=Agile&l=&relative=1

Jasmine. *Jasmine : Behaviour-Driven JavaScript* . Retrieved 05 01, 2015, from Jasmine: http://jasmine.github.io/

Jenkins-CI. *Welcome to Jenkins CI! | Jenkins CI*. Retrieved 05 01, 2015, from jenkins-ci.org: https://jenkins-ci.org/

Jesus LC. *jesuslc.com*. Retrieved 05 01, 2015, from jesuslc.com: http://jesuslc.com

Joyent. *Node JS*. Retrieved 05 01, 2015, from Node.js: https://nodejs.org/

kaizentesting. *agile-test-automation-is-incomplete-without-continuous-integration*. Retrieved 05 01, 2015, from kaizentesting.wordpress.com: https://kaizentesting.wordpress.com/2012/08/19/agile-test-automation-is-incomplete-without-continuous-integration/

Selenium HQ. *Selenium Web browser automation*. Retrieved 05 01, 2015, from seleniumhq.org: http://www.seleniumhq.org

Strongloop. *Express JS*. Retrieved 05 01, 2015, from Express JS: http://expressjs.com

Wikipedia. *Continuous_integration*. Retrieved 05 01, 2015, from wikipedia.org: http://en.wikipedia.org/wiki/Continuous_integration

wikipedia. *DevOps*. Retrieved 05 01, 2015, from wikipedia.org: http://en.wikipedia.org/wiki/DevOps

Wikipedia. *Front and back ends*. Retrieved 05 01, 2015, from wikipedia.org: http://en.wikipedia.org/wiki/Front_and_back_ends

Wikipedia. *MEAN software_bundle*. Retrieved 05 01, 2015, from wikipedia.org: http://en.wikipedia.org/wiki/MEAN_%28software_bundle%29

Wikipedia. *Single-page_application*. Retrieved 05 01, 2015, from wikipwdia.org: http://en.wikipedia.org/wiki/Single-page_application