AUTOMATITZACIÓ DE TESTS PER A UNA APLICACIÓ J2EE

Memòria del projecte d'Enginyeria en Informàtica realitzat per **Iván Vargas Checa** i dirigit per **Joan Borrell Viader** i **Alberto Costoya Leizán**.

Bellaterra, juny de 2009

El sotasignat, **Alberto Costoya Leizán**
de l'empresa, KINAMIK DATA INTEGRITY S.L.

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat en l'empresa sota la seva supervisió mitjançant conveni 08/0183/BT firmat amb la Universitat Autònoma de Barcelona.

Així mateix, l'empresa en té coneixement i dóna el vist-i-plau al contingut que es detalla en aquesta memòria.

Barcelona, juny de 2009

Signat: Alberto Costoya Leizán

Agraïments

Vull agrair a l'empresa Kinamik Data Integrity S.L. l'haver-me brindat l'oportunitat de realitzar aquest projecte tant important per a mi, però en especial vull donar les gràcies a dues persones, Alberto Costoya Leizán i Xavier López Barquiel, per haver-me assessorat, supervisat, recolzat i aguantat les meves preguntes durant tot aquest temps de manera desinteressada i sense els quals aquest projecte no hagués estat possible.
Per descomptat, agrair a la meva família la paciència que han tingut amb mi, aguantant els meus nervis i mals moments, donant-me suport moral. Finalment, donar les gràcies a Joan Borrell Viader, professor de la UAB i supervisor d'aquest projecte, per haver-me ajudat a portar aquest projecte endavant.

A tots ells, moltes gràcies.

Índex

Índex de figures

Capítol 1

Introducció

1.1 Motivacions personals

La meva empresa, **Kinamik Data Integrity**, és una empresa de desenvolupament de software relativament jove, però en constant creixement i disposada a fer-se un lloc en el mercat. Concretament, desenvolupa un producte enfocat a la seguretat i la integritat de dades anomenat *Secure Audit Vault*, el qual encara està en fase beta però aviat sortirà al mercat.

Secure Audit Vault és una aplicació J2EE composada de tres elements (kFeeds, kAuditor, kSecure) que permet recol·lectar, notaritzar i auditar dades per tal d'assegurar la seva integritat i validesa. Els *kFeeds* s'encarreguen de recollir les dades de les aplicacions client i d'enviar-les a *kSecure*, el qual les encripta amb un algoritme d'encriptació híbrid (utilitza encriptació simètrica i asimètrica a la vegada) i en calcula el hash. Aquestes dades es poden consultar i es pot validar la seva integritat utilitzant *kAuditor*, al qual s'accedeix mitjançant una interfície via Web.

La meva trajectòria a l'empresa comença al Setembre de 2008, incorporant-me al departament de Qualitat com a becari. És un departament petit però amb molta importància dins de l'empresa, ja que qualsevol bon producte que surti al mercat ha de passar uns controls i més tractant-se d'un producte orientat a la seguretat, on qualsevol *bug* pot ser molt perillós. Em van oferir incorporar-me a l'equip com a becari per anar-me instruint mica en mica i en un futur incorporar-me com a

empleat de l'empresa.
Les meves tasques a l'empresa són les de testejar l'aplicació i a la vegada (i és aquí on sorgeix el meu PFC) automatitzar el procés de test.

1.2 Descripció del problema

L'aplicació que desenvolupem a Kinamik es revisa (surt una nova versió) aproximadament cada mes. Això implica que aproximadament cada mes el producte s'ha de tornar a testejar per a provar les seves funcionalitats i el seu comportament envers diversos escenaris de treball. La cosa es complica cada vegada més, ja que l'aplicació va creixent en funcionalitat a cada nova versió, el que implica que en el mateix període de temps (unes dues setmanes) s'ha de testejar tot el que hi havia abans més les noves funcionalitats afegides. Com es pot veure, això podria arribar a ser infinit.

És en aquest punt on sorgeix la idea d'aquest PFC, amb la intenció de millorar el procés de testing del producte, intentant automatitzar gran part del procés de testing que és invariant, estalviant així una gran quantitat de temps. Així doncs, bàsicament aquest projecte tracta de millorar el procés de testing del producte, automatitzant i/o facilitant l'execució d'alguns tests específics.

D'altra banda, el departament de desenvolupament de l'empresa es troba sempre amb un problema. Ells acaben el desenvolupament d'una nova versió del software i la passen al departament de qualitat per a que sigui testejada, mentre ells continuen treballant en la propera versió. Això implica que fins que l'equip de qualitat no testegi l'aplicació (això mínim són un parell de dies), l'equip de desenvolupament està treballant en una nova versió basada en la versió anterior, sense saber si aquesta versió anterior presenta algun error greu. Es per això que requereixen d'alguna eina que els permeti fer d'una manera ràpida i automàtica un test bàsic (també anomenat test de *smoke*) per a assegurar que l'aplicació funciona amb un mínim de garanties. Així doncs, aquest serà un altre punt a tractar en aquest PFC.

1.3 Objectius

Realment, els objectius els podríem dividir en dos tipus. D'una banda, un dels objectius del projecte és fer recerca, veure les opcions que hi ha i donar la millor solució possible a la problemàtica existent d'automatització de tests. Això serien els objectius "teòrics".
D'altra banda, tenim la part més important del PFC, els objectius "tècnics", pels quals abans s'han hagut d'assolir els objectius "teòrics". Concretament, aquests objectius es resumeixen en implementar una aplicació per a la creació/execució de tests automàtics que permeti:

1. Configurar l'entorn d'execució dels tests (crear audit trails, configurar els clients i els *kFeeds*)
2. Posar en marxar l'entorn
3. Executar els diferents tests (de *smoke*, de funcionalitat, d'estrès, etc.)
4. Obtenir resultats (test OK o no OK, què ha fallat, etc.)

1.4 Estudi de viabilitat

L'objectiu principal d'aquest PFC, com he comentat abans, és el de crear una aplicació que sigui fàcilment ampliable, ja que automatitzar tots els tests és inviable, però sí que ha de ser una bona base per on començar. Així doncs, el PFC no anirà enfocat a automatitzar tot el procés de testing, sinó que anirà enfocat a millorar el procés, automatitzant algunes parts (les que s'han de repetir a cada versió), deixant definida una eina que pot ser fàcilment ampliada segons les necessitats.

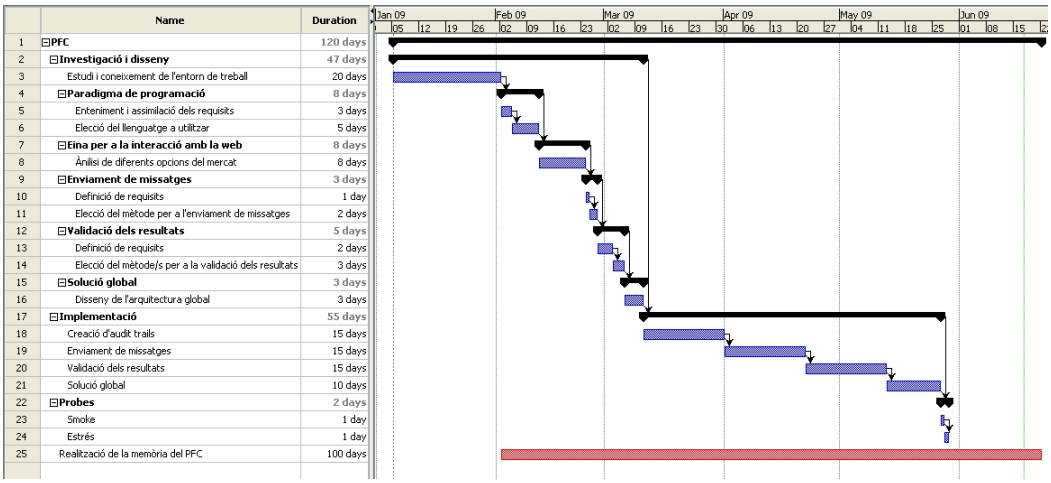
Tot el maquinari i software necessari per a la realització d'aquest PFC el tinc al meu abast a l'empresa, i és el que utilitzo habitualment en el meu entorn de treball, per tant, no hi ha cap impediment en aquest sentit. De cara a la implementació de l'aplicació, es treballarà amb eines gratuïtes i/o de codi obert, cosa que facilitarà el procés. A més, com a apunt, dir que ja fa un temps que treballo a l'empresa dins del departament de qualitat, el que fa que tingui un bon coneixement sobre

l'aplicació i el procés de testing i per tant, que em pugui centrar directament en l'automatització d'aquest procés.

És un projecte viable i molt interessant, amb una forta utilitat pràctica, però que pot ser tant extens com es vulgui. Per tant, és bo definir uns objectius mínims i si es pot, ampliar-los més tard.

1.5 Planificació temporal

A la figura 1.1 es pot veure la planificació inicial del projecte, el qual es va començar a principis de gener (sense tenir en compte l'informe previ).
Aquesta planificació s'ha realitzat suposant una dedicació diària d'unes 3 hores, aproximadament, ja que és difícil preveure una dedicació constant amb hores. La intenció, bàsicament, és definir unes dates límit (fites) per portar un control, més que no pas controlar la dedicació exacta en hores.



	Name	Duration
1	**PFC**	120 days
2	**Investigació i disseny**	47 days
3	Estudi i coneixement de l'entorn de treball	20 days
4	**Paradigma de programació**	8 days
5	Enteniment i assimilació dels requisits	3 days
6	Elecció del llenguatge a utilitzar	5 days
7	**Eina per a la interacció amb la web**	8 days
8	Ánilisi de diferents opcions del mercat	8 days
9	**Enviament de missatges**	3 days
10	Definició de requisits	1 day
11	Elecció del mètode per a l'enviament de missatges	2 days
12	**Validació dels resultats**	5 days
13	Definició de requisits	2 days
14	Elecció del mètode/s per a la validació dels resultats	3 days
15	**Solució global**	3 days
16	Disseny de l'arquitectura global	3 days
17	**Implementació**	55 days
18	Creació d'audit trails	15 days
19	Enviament de missatges	15 days
20	Validació dels resultats	15 days
21	Solució global	10 days
22	**Probes**	2 days
23	Smoke	1 day
24	Estrés	1 day
25	Realització de la memòria del PFC	100 days

Figura 1.1: *Diagrama de Gantt de la planificació del projecte*

1.6 Estimació de costos

Els costos es poden dividir en dos tipus diferents. Per un costat tindríem els costos materials, que serien el hardware i el software necessari per a la realització del

projecte, i per l'altra banda tindríem els referents a hores de treball.

En aquest cas, els costos materials (hardware i software) són 0 euros, ja que totes les eines utilitzades són gratuïtes i el hardware necessari està disponible a l'empresa.
D'altra banda hi ha els costos en quant a hores de treball. Tenint en compte la planificació incial proposada, el projecte duraria 120 dies, per tant, a una mitja de 3 hores per dia, dóna un total de 360 hores de treball dedicades a la realització del projecte.

1.7 Estructura de la memòria

Aquesta memòria està dividida en 6 capítols. A continuació es mostra quins són aquests capítols i es fa un breu resum del que es pot trobar a cadascun d'ells:

- **Capítol 1: Introducció.** Es tracta d'aquest mateix capítol. En ell es presenta el problema que ha motivat aquest projecte i els objectius que es volen assolir. També inclou un petit estudi de viabilitat, la planificació temporal i un estudi de costos.

- **Capítol 2: Introducció al testing.** Aquest és un capítol teòric on es fa una breu introducció al testing, explicant alguns conceptes bàsics per tal de poder entendre una mica millor els objectius d'aquest projecte i alguns dels temes tractats en aquesta memòria.

- **Capítol 3: Requisits, entorn i eines.** En aquest capítol es detallen els requisits, s'explica quin és l'entorn amb què s'ha de treballar (es parla de l'aplicació a testejar i la base de dades utilitzada per aquesta) i es fa un petit estudi per decidir quines eines s'utilitzaran per a desenvolupar el projecte. Concretament, dins d'aquest estudi d'eines es pot veure quin serà el llenguatge de programació utilitzat, a més d'incloure una comparativa d'aplicacions per a testejar webs, la qual ens serà necessària per a fer les tasques d'usuari.

- **Capítol 4: Anàlisi i disseny.** És un dels capítols més importants i decisius, ja que aquí s'explica com s'ha plantejat l'estructura de l'aplicació, quines funcionalitats tindrà i com interaccionarà l'usuari amb ella.

- **Capítol 5: Desenvolupament i proves.** Aquest capítol tracta tot el que fa referència al desenvolupament de l'aplicació. Es donen unes nocions tècniques del paradigma de programació que s'utilitza i es fa una breu explicació a alt nivell dels scripts implementats més representatius. També es parla dels problemes trobats durant el desenvolupament i de les proves realitzades.

- **Capítol 6: Conclusions.** Aquest és l'últim capítol, on es revisen els objectius i els plantejaments inicials després d'haver finalitzat el projecte, per tal de fer una valoració final. També es proposen futures vies de continuació en les que es pot treballar per a millorar l'aplicació.

Capítol 2

Introducció al testing

En aquest capítol es fa una breu introducció al món del testing tot explicant els conceptes més importants i les diferents fases del procés, per tal d'emmarcar aquest PFC en el seu context real.

2.1 Introducció al concepte de testing

Anem a començar donant una visió ràpida i global de què és el testing i per a què serveix.

Definició de testing

Quan en el món del software parlem de "testing", ens referim a un conjunt de processos (proves) integrat dins de les diferents fases del cicle del software, el qual permet verificar i quantificar la qualitat d'un producte segons uns requisits preestablerts. Aquest procés de testing habitualment és realitzat per un grup independent de "testers", en un entorn de proves separat físicament del de producció.

Raons per fer testing

L'avanç de les noves tecnologies fa que la mida i la complexitat de les aplicacions cada vegada sigui més gran, el que repercuteix directament als costos tant de desenvolupament com de manteniment. És aquí on entra en joc el testing, verificant i avaluant la qualitat del producte en qüestió per tal de conèixer el seu estat real i trobar els "bugs" (errors) de la manera més ràpida possible, reduint així els costos de temps i millorant la imatge del producte.
D'altra banda, és important mesurar l'estabilitat, l'escalabilitat, l'eficiència i la seguretat del producte, ja que no hem d'oblidar que aquest va destinat a un usuari (habitualment client) i és important que no es trobi amb sorpreses inesperades.

M'agradaria fer notar que, en cap cas es pot considerar el testing com a substitut de la bona programació, ja que aquest pot provar l'existència d'errors, però no pas la seva absència.

2.2 Test case

Definició de test case: Un test case es pot definir com un conjunt de condicions i variables sota els quals un analista determinarà si l'aplicació compleix o no un cert requisit.

Almenys hi ha d'haver un test case per cada requisit del producte a testejar, per tal de provar que aquest requisit es compleix; aquest test case és l'anomenat *happy path* i és el test que comprova la funcionalitat bàsica d'un requisit determinat.
Encara que sembli suficient tenir només un test case per requisit, no és així, ja que s'ha d'aconseguir una àmplia cobertura per tal de garantir el seu funcionament en diferents entorns i circumstàncies.

D'altra banda, els tests encarregats de provar un requisit en circumstàncies estranyes o atípiques, s'anomenen tests de *corner case* (cas extrem) i són molt importants per veure com reacciona el producte en circumstàncies no esperades.

Per últim, afegir que, un bon test case és aquell que troba errors, ja que aquest és el propòsit amb el que es crea.

La descripció d'un test case ha d'incloure obligatòriament la següent informació:

1. Identificador únic del test case
2. Nom del test case
3. Identificador dels requeriments associats
4. Breu descripció del test case
5. Descripció detallada de com s'ha de fer la prova (entrada, configuració de l'entorn,etc.)
6. Resultats desitjats

2.2.1 Tipus de test cases

Test cases n'hi ha de molts tipus. A continuació es poden veure llistades les principals categories en què es poden classificar els test cases, acompanyada cadascuna d'una petita descripció. Notar, però, que un test case pot pertànyer a més d'una categoria.

- **Tests de caixa blanca:** Són els tests que, tenint accés al codi de l'aplicació, serveixen per comprovar el funcionament intern del software.
- **Tests de caixa negra:** Aquests tests són semblants als de caixa blanca, però sense tenir accés al codi; bàsicament es tracta de comprovar que donada una entrada s'obté la sortida esperada.
- **Tests unitaris:** Es tracta de provar el funcionament d'un mòdul concret de codi. Serveixen per a comprovar que un mòdul funciona correctament per separat.
- **Tests d'integració:** Es realitzen una vegada s'han passat els tests unitaris. Es tracta d'executar conjuntament tots els tests unitaris de l'aplicació per veure que també funcionen conjuntament.

- **Tests funcionals:** Són els tests que s'encarreguen de comprovar la funcionalitat del producte tenint en compte les especificacions del mateix.
- ***Smoke* test:** És un conjunt dels tests funcionals més crítics (els *happy path*) de l'aplicació, destinat a comprovar la funcionalitat bàsica d'aquesta. Normalment és el primer test que s'executa sobre la versió final del producte.
- **Tests d'acceptació:** S'anomenen així els tests que s'han de superar per tal de poder acceptar una certa versió de l'aplicació. Si l'aplicació no supera aquests tests, no és vàlida.
- **Tests de regressió:** Són els tests que es realitzen per a provar funcionalitats ja testejades prèviament, amb la finalitat de trobar nous bugs apareguts a causa de les modificacions realitzades per a solucionar altres bugs.
- **Tests de càrrega:** També anomenats tests d'estrès. Es tracta de sotmetre l'aplicació a una càrrega molt elevada, intentant trobar el seu límit per veure com es comporta en situacions de molta càrrega i comprovar la seva estabilitat.

2.3 Fases del procés de testing

Com tot procés, el testing ha de seguir un ordre lògic en la seva execució. Un bon procés de testing hauria de seguir els següents passos:

1. Crear un "Test Plan", on es defineixen totes les tasques a realitzar
2. Analitzar els requisits de l'aplicació a testejar
3. Fer un anàlisis dels tests necessaris
4. Dissenyar els tests que encara no existeixin
5. Executar els tests
6. Reportar els bugs trobats

7. Redactar les "Release notes", les quals informen de l'estat de l'aplicació, tot esmentant els bugs trobats

A l'hora d'executar els test cases també s'ha de seguir un ordre. Normalment, encara que hi pot haver alguna petita variació, l'ordre en què s'han d'executar és el següent:

1. Tests unitaris
2. Tests d'integració
3. Tests d'acceptació
4. Tests funcionals
5. Tests de càrrega
6. Tests de regressió
7. Tests de rendiment

Finalització del procés de testing

El procés de testing s'acaba quan s'arriba a la data límit (també anomenada *deadline*), quan es passen tots els test cases o quan el producte no passa el test d'acceptació. En aquest últim cas es crea una nova versió de l'aplicació i es comença de nou el procés.

2.4 Les tasques d'un tester

La tasca principal d'un tester (persona que es dedica a realitzar tests) és la de trobar els bugs existents el més ràpid possible. No només ha de trobar bugs, sinó que ha d'assegurar-se de que es poden reproduir, per tal de descartar que el bug sigui causat per una mala configuració de l'entorn de proves. No serveix de res trobar bugs i no informar-ne a les persones responsables de solucionar-ho, per

tant, un bon tester ha d'informar de tots els bugs que troba, encara que no semblin importants, i s'ha d'assegurar que aquests es solucionen.

D'altra banda, un tester no ha d'arreglar els bugs que troba, sinó que només n'ha d'informar de la seva existència. També hauria de ser rigorós i no donar res per suposat; per això, s'ha de regir pels requisits del producte.

Vist d'aquesta manera, sembla que el tester hagi de ser l'enemic del desenvolupador, però no és així, més aviat és al contrari. El tester no crítica la feina d'una persona, sinó que comenta els errors que conté el producte. Això beneficia al desenvolupador, ja que quan abans es trobi un error existent, abans es solucionarà el problema.

Per últim, un bug no es pot reportar de qualsevol manera, sinó que s'han de respectar certes normes. Concretament, la metodologia a seguir quan es reporta un bug és la següent:

1. Especificar exactament quin és el problema
2. Donar els passos necessaris per a reproduir-ho
3. Dir quins eren els resultats esperats (correctes)
4. Indicar la severitat del bug i la prioritat amb què s'ha de tractar
5. Ser precís, no donar més informació de la necessària

Capítol 3

Requisits, entorn i eines

3.1 Requisits

Aquest PFC, com ja he comentat abans, sorgeix d'una proposta de la meva empresa, per tant, hi ha uns requisits a complir donats per la mateixa, els quals detallo a continuació:

1. L'aplicació a desenvolupar ha de ser flexible, "amigableï ha de permetre ampliar el catàleg de tests de manera fàcil i ràpida.

2. Ha de ser compatible amb la base de dades utilitzada per *Secure Audit Vault*.

3. Ha de ser compatible amb el Cruise Control[1].

4. Ha de ser compatible tant amb Windows com amb Linux.

5. Tot el software necessari per a desenvolupar aquest projecte (i que no estigui disponible a l'empresa) ha de ser gratuït.

[1]El Cruise Control és un framework que serveix per a fer compilacions automàtiques. En el cas de la meva empresa és utilitzat per a compilar l'aplicació cada nit, amb la finalitat de compilar el codi amb tots el canvis que hi ha hagut durant el dia per saber si hi ha errors i així tenir una base consistent amb la que treballar el proper dia.

3.2 Entorn de treball

3.2.1 Aplicació a testejar (*Secure Audit Vault*)

L'aplicació a testejar i a la qual està enfocat aquest projecte, s'anomena *Secure Audit Vault* i està destinada a la seguretat de dades. Més concretament, s'encarrega d'emmagatzemar les dades (logs) que li arriben i de garantir la seva integritat, de manera que es pot saber si han estat modificades, quin dia, què és el que s'ha modificat i el més important, si aquestes dades són fiables (ja que també pot ser que s'hagin afegit dades il·legítimes).
Secure Audit Vault és un sistema una mica complex i està format per tres mòduls ben diferenciats: *kFeeds*, *kSecure* i *kAuditor*. A part d'això, per tal de comunicar-se entre tots ells, s'utilitza una base de dades. A la figura 3.1 hi trobem un esquema de l'arquitectura de l'aplicació.

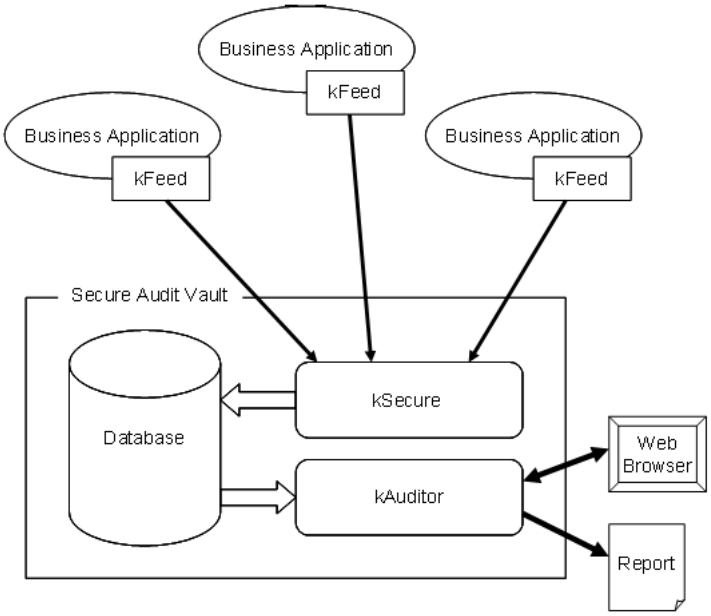


Figura 3.1: *Arquitectura de Secure Audit Vault*

Després de veure de manera molt general què és i per a què serveix *Secure Audit Vault*, anem a veure un per un tots els components que conformen aquesta aplicació per tal d'entendre el seu funcionament.

kFeed

L'element *kFeed* podríem dir que és un element extern al sistema, similar a un agent, el qual s'associa a una o més aplicacions clients[2] per tal de recollir la informació generada per aquestes i enviar-la a *kSecure*. Així doncs, un *kFeed* es pot considerar com un apèndix de *kSecure*. La informació (o logs) que es pot recollir mitjançant un *kFeed*, pot ser de quatre tipus diferents:

1. Missatges Log4j
2. Missatges Syslog
3. Text pla
4. Events d'auditoria de Weblogic (per a poder utilitzar un kFeed d'aquest tipus s'ha de tenir instal·lat *Weblogic Server*)

De *kFeeds* n'hi poden haver tants com es vulgui, però cada *kFeed* només pot recollir un tipus d'informació, és a dir, que hi ha quatre tipus diferents de *kFeed* (un per a cada tipus de dades), encara que com ja he comentat, un kFeed pot estar associat a més d'una aplicació client.

kSecure

kSecure és l'element que juga el paper de "notary" dins de *Secure Audit Vault*. Més concretament, *kSecure* s'encarrega de rebre la informació enviada pels diferents *kFeeds*, notaritzar-la (la signa mitjançant un Hash) i emmagatzemar-la a la base de dades centralitzada, per tal de poder mostrar i assegurar la integritat d'aquesta informació més tard. Es podria dir que *kSecure* és l'element central de *Secure Audit Vault*.

kAuditor

kAuditor, com el seu propi nom indica, s'encarrega d'auditar les dades prèviament guardades per *kSecure* a la base de dades i és qui confirma la validesa o no

[2]Dins d'aquest context, una aplicació client és aquella que genera la informació que es vol auditar.

d'aquestes dades. A diferència dels altres elements de *Secure Audit Vault*, *kAuditor* disposa d'una interfície gràfica en forma de plana Web (figura 3.2), la qual permet que l'usuari interaccioni amb l'aplicació. Mitjançant aquesta interfície gràfica es poden crear i mantenir els diferents audit trails[3], consultar les dades contingudes dins d'un o més audit trails, comprovar la integritat d'aquestes dades, etc. Es podria dir que *kAuditor* és qui posa cara i ulls a *Secure Audit Vault*.

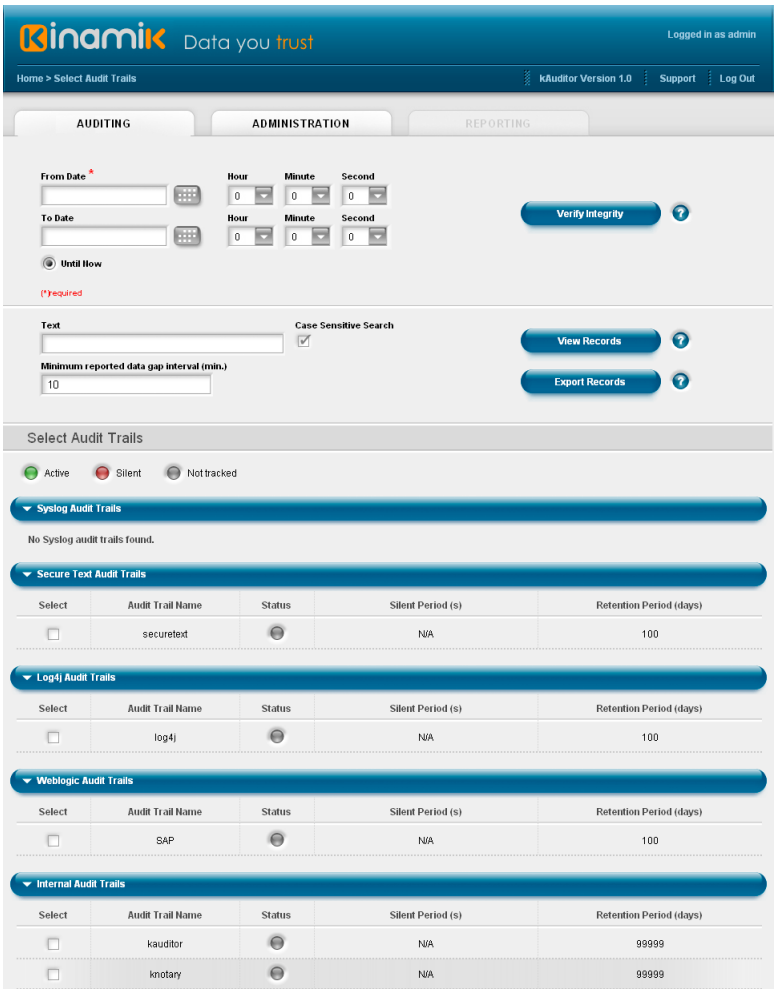


Figura 3.2: *Pàgina principal de kAuditor*

[3]Un audit trail (en català, pista d'auditoria) és una seqüència de registres d'auditoria, registres que normalment són creats com a conseqüència d'execucions del sistema o transaccions. En el cas de la nostra aplicació, quan es parla d'audit trails es refereix a les entitats que contenen aquests registres, el que en llenguatge pla es podria anomenar contenidor. Així, quan un *kFeed* envia missatges a *kSeucre*, aquests es guarden en audit trails prèviament configurats.

3.2.2 Base de Dades

L'aplicació a testejar (*Secure Audit Vault*) utilitza una base de dades per a emmagatzemar tota la informació enviada pels clients, senyals de control i algunes propietats del sistema. Al començament d'aquest PFC la base de dades utilitzada era Oracle, però hi havia rumors de que es migraria a MySQL, ja que aquesta és software lliure i Oracle no. Doncs bé, finalment, aquests rumors es van fer realitat i es va migrar a MySQL; bé, exactament no es va migrar, sinó que es van passar a suportar els dos tipus. Això va fer canviar els requisits i que a partir d'aquest moment s'haguessin de suportar les dos bases de dades. En seccions posteriors es pot veure com es va solucionar aquest problema.

3.3 Eines de desenvolupament

El primer que necessitem per a desenvolupar l'aplicació, és definir i tenir clar quin serà el llenguatge de programació que utilitzarem per a l'automatització dels tests, encara que a la seva vegada utilitzem altres eines. Així doncs, a la secció 3.3.1 es pot veure quin és el llenguatge escollit i les seves característiques.

D'altra banda, el software per al qual es vol fer l'automatització, té una interfície gràfica en forma de web. És per això que necessitem una eina de test de planes web per tal de poder fer la gran majoria de proves i simular les accions que pot realitzar un usuari.
Després d'una recerca de les eines que hi ha disponibles al mercat, he seleccionat unes quantes que s'emmotllen al que necessitem (basant-me en els requisits), per tal de fer una comparativa i arribar a decidir quina és la més adequada. Aquesta comparativa es pot veure a la secció 3.3.2.

3.3.1 Llenguatge de programació

El llenguatge de programació escollit per a realitzar aquest projecte està supeditat pel requisit imposat de que l'aplicació a crear ha de ser compatible amb *Cruise Control*, per tal de poder automatitzar la compilació.

El departament de desenvolupament de l'empresa utilitza una eina anomenada **Ant** per a la compilació de *Secure Audit Vault*. Aquesta eina és gratuïta, integrable amb Cruise Control i és compatible tant amb Windows com amb Linux, ja que està desenvolupada amb Java i es basa en scripts XML per a la realització de les tasques.

Ant permet executar tasques de molts tipus, com per exemple fer crides a aplicacions java, executables, scripts Bash, etc.
Com Ant és un llenguatge enfocat a la compilació, cada execució que fa és una compilació, aturant-se en cas que hi hagi algun error. Això ens facilita la feina enormement, ja que només ens haurem de preocupar de la prova a realitzar, que en cas de falla, la compilació s'aturarà i sabrem en quin punt a fallat.
Una altra de les característiques d'Ant és que s'executa molt fàcilment des de la línia de comandes, indicant quina tasca es vol executar.

Així doncs, emprar Ant sembla una bona idea per al desenvolupament de l'aplicació i per tant, és l'eina escollida.

3.3.2 Anàlisi d'eines per a testejar planes web

A continuació i per tal d'escollir-ne la millor, faig una comparativa entre totes les eines de test de webs que he seleccionat com a candidates, enumerant els pros i els contres de cadascuna.

Canoo Webtest

Canoo webtest és una eina OpenSource que permet automatitzar tests per a aplicacions web d'una manera efectiva.

- **Punts a favor**:

    1. Webtest és fàcil d'utilitzar, gràcies a que la programació es fa amb XML utilitzant una sintaxi molt intuïtiva i entenedora. Així doncs, els tests es poden fer ràpidament i són fàcilment mantenibles.

2. És ràpid en la seva execució, ja que no descarrega CSS ni imatges.
3. Proporciona reports amb molta informació, permetent entendre per què ha fallat un determinat test.
4. És una aplicació Java, per tant pot córrer a qualsevol sistema operatiu.
5. Accepta HTTP i HTTPS.
6. Els scripts de WebTest són scripts d'Ant, per tant, es poden integrar de forma directa amb eines com Cruise Control.
7. Hi ha molta documentació disponible.

- **Punts en contra**:

1. Alguns navegadors accepten html mal format, però WebTest no. Això en principi es bo, però ens pot portar problemes.
2. WebTest no suporta Javascript tan bé com un navegador "real" (com per exemple Firefox o Internet Explorer), ja que el que fa WebTest és simular un navegador real, amb les limitacions que això comporta.
3. No es poden fer verificacions d'estil, ni de colors, ni de tipus de font ni de mida de lletra, així com tampoc es poden testejar les imatges.
4. No té la funcionalitat de Record & Play (permetre gravar i executar una seqüència de passos).
5. No es poden fer bucles.

Imprimatur

Imprimatur ha estat dissenyat per a ser l'eina de testing funcional de webs més simple. Envia peticions HTTP a l'aplicació i valida les respostes utilitzant expressions regulars.

- **Punts a favor**:

1. És una aplicació Java que s'executa des de la línea de comandes, el que fa, conjuntament a que els tests es defineixen en XML, que es pugui executar amb Ant.
2. És una aplicació Java, per tant pot córrer a qualsevol sistema operatiu.

- **Punts en contra**:

1. Hi ha molt poca documentació.
2. Els tests que es poden fer són molt bàsics (té poques funcionalitats).
3. No accepta HTTPS.

ITP

ITP és una aplicació Java OpenSource molt senzilla d'utilitzar.

- **Punts a favor**:

1. Els tests es programen amb XML.
2. És fàcil d'utilitzar (es necessita poc temps per a implementar scripts).
3. Es poden fer cerques de text a la pàgina resultant.
4. És una aplicació Java que s'executa des de la línia de comandes, el que fa, conjuntament a que els tests es defineixen amb XML, que es pugui executar amb Ant.

- **Punts en contra**:

1. No hi ha documentació disponible. L'aprenentatge es fa a base de veure exemples.
2. És un projecte encara poc madur.

Sahi

Sahi és una eina per a tests d'aplicacions web, desenvolupada amb Java i Javascript. Utilitza JavaScript per a executar events al navegador. Sahi treballa com a servidor Proxy i el navegador ha d'utilitzar el servidor Sahi com el seu Proxy.

- **Punts a favor**:

  1. És una aplicació Java, per tant pot córrer a qualsevol sistema operatiu.
  2. Té la funcionalitat de Record & Play.
  3. Ofereix la possibilitat d'afegir breackpoints per a debugar.
  4. Es poden definir bucles.
  5. Els scripts es poden executar en mode batch (arxius .bat).
  6. És integrable amb ANT.
  7. Suporta HTTP I HTTPS.
  8. Hi ha molta documentació disponible, fòrums, FAQs, etc.

- **Punts en contra**:

  1. El gran inconvenient és que per a utilitzar Sahi s'ha de configurar com a proxy en el navegador, i això no sempre serà possible degut a que molts servidors no tenen interfície gràfica i per tant no es pot instal·lar cap navegador.

PesterCat

PesterCat és una eina molt completa per a testing de webs. Encara que és de pagament, m'ha semblat interessant incloure-la en aquesta comparativa, ja que realment s'adapta molt bé a les necessitats del projecte i en un futur potser es podria plantejar l'opció de la seva compra.

- **Punts a favor**:

1. Funciona en Linux, Windows i Mac OS.
2. Té la funcionalitat de Record & Play.
3. S'utilitza llenguatge XML per programar els scripts.
4. Es poden definir variables i constants als scripts.
5. Es poden definir bucles.
6. Permet fer consultes SQL.
7. Integrable amb ANT.

- **Punts en contra**:

1. És de pagament.
2. No suporta HTTPS.
3. No hi ha molta documentació lliure disponible.
4. Per a utilitzar PesterCat s'ha de configurar el navegador per a que PesterCat sigui el seu servidor Proxy.

Eina escollida

Després de definir els punts en contra i a favor de cadascuna de les eines, arriba el moment d'escollir-ne una. Dir que quan parlo de descartar-ne unes i escollir-ne una altra, no vol dir que les descartades no ens serveixin, sinó que entre les candidates n'hi ha alguna que s'adapta millor que d'altres.
Doncs bé, l'escollida és CanooWebTest, donat que s'adapta perfectament a les nostres necessitats: totalment integrable amb ANT (de fet, es basa en ANT), protocols HTTP i HTTPS, fàcil i ràpid, amb molta documentació i encara que no sembli important, un altre punt a favor és que és una eina molt popular, el que vol dir que és utilitzada per molta gent i que els possibles dubtes/malfuncionaments que podem trobar potser ja han estat tractats per algú.

Són vàries les raons que m'han fet escollir CanooWebTest: Hi ha algunes eines que necessiten ser configurades com a Proxy i per tant s'ha de configurar

el navegador per a que funcionin d'aquesta forma. Això és una restricció molt forta per al nostre cas, ja que els scripts d'automatització s'han de poder executar a qualsevol servidor de l'empresa, a qualsevol màquina local, etc. Com es pot veure, configurar el navegador de totes les màquines és molt molest i portaria molta feina i fins i tot en alguns casos es impossible ja que hi ha servidors sense interfície gràfica. Així doncs, Sahi i PesterCat queden descartats.

En quant a ITP i Imprimatur, la principal desavantatge és que tenen disponible molt poca informació i per tant, no són les més apropiades per a començar a programar sense conèixer-les. Per tant, queden descartades.
Així doncs, la última i millor opció és Canoo WeTtest.

L'opció de PesterCat potser ens podria ser molt útil en un futur, ja que cobreix pràcticament tots els requisits que tenim i utilitats que altres no tenen (consultes SQL, bucles,etc). Però no es gratuïta, i un dels requisits és que l'eina a utilitzar ha de ser gratuïta. Si algun dia canvia aquest requisit, s'avaluarà amb més profunditat i es reconsiderarà la seva utilització.
D'altra banda, ITP és un projecte poc madur, però amb molt bona pinta. Així doncs, seria interessant anar veient la seva evolució de cara al futur.

Aspectes en contra de l'eina escollida i el seu impacte

CanooWebtest té aspectes en contra, però no són decisius.
Un d'ells és que no té la funcionalitat de Record & Play, la qual ens és molt útil per a crear scripts ràpidament. Doncs bé, aquest problema es pot solucionar amb la instal·lació del plugin *WebTestRecorder* per a Firefox. D'aquesta manera, des del nostre navegador podem fer una prova manualment i aquesta eina ens anirà dient com es tradueixen en tasques de Webtest els passos que anem fent.
Un altre dels problemes principals que hi ha és que no pot verificar imatges. Això no ens afecta gaire perquè al nostre software només hi ha una única imatge que podria ser interessant de verificar, però bé, no tot pot ser perfecte. Dir per això, que amb Webtest sí que podem validar que la imatge existeix, i la informació que ha de mostrar la podem validar amb una altra funcionalitat de l'aplicació. Per tant, aquest és un problema menor, tal vegada a tenir en compte de cara al futur.

Per últim, el tema de no poder fer bucles el podem solucionar utilitzant scripts .sh, els quals poden ser cridats amb una tasca ANT.

Capítol 4

Anàlisi i disseny

4.1 Funcionalitats que ha de tenir l'aplicació

Arribada aquesta part, es planteja el dubte de què es exactament el que es vol automatitzar. Així doncs, s'han de definir els requisits funcionals, és a dir, allò que ha de poder fer l'aplicació.

Per a decidir quines funcionalitats ha de tenir la nostra aplicació, primerament hem de pensar en les funcionalitats i el funcionament de *Secure Audit Vault* (l'aplicació a automatitzar), tenint en compte que no es pot automatitzar tot i que hem d'escollir les funcionalitats principals, a la vegada que ha de ser una implementació totalment modular i ampliable fàcilment.
Les funcionalitats bàsiques de *Secure Audit Vault* a partir de les quals s'han de definir els requisits, són principalment:

- Enviar a kSecure els missatges generats per les aplicacions client
- Validar que arriben tots els missatges
- Validar que els missatges no han estat modificats il·legítimament

Vistes les funcionalitats bàsiques de *Secure Audit Vault*, vaig decidir basar-me en les dues primeres esmentades: **enviar missatges** i **validar que tots arriben correctament**.

Aquestes dues funcionalitats poden semblar poca cosa, però no ho són, ja que de fet i de manera indirecta, aquestes afecten a moltes altres, com per exemple validar que la configuració de l'aplicació funciona, que els *kFeeds* funcionen, que es poden crear i editar audit trails mitjançant la plana web, que kSecure i kAuditor accedeixen de manera correcta a la base de dades, etc. A més, en quant al test de *smoke*, provar aquestes funcionalitats és més que suficient.

4.2 Arquitectura

Tenint doncs les funcionalitats definides, el que cal fer és aplicar el conegut lema "divideix i venceràs", tractant cada problema per separat per tal d'arribar a una solució conjunta construïda a partir de les diferents parts.
A les següents seccions es pot veure com s'ha plantejat cada cas.

Enviament de missatges a kSecure

Per a enviar missatges a kSecure es necessiten fer vàries coses:

1. Instal·lar i configurar els *kFeeds*
2. Instal·lar i configurar un o més clients per a crear els logs que seran enviats
3. Crear els audit trails on es guardaran els missatges
4. Enviar els missatges

Validació de l'arribada dels missatges a kSecure

Per a validar que els missatges han arribat a kSecure hi ha dos opcions diferents:

1. Consultar els audit trails mitjançant la interfície gràfica seleccionant els audit trails que guarden la informació, o
2. Consultar el contingut dels audit trails directament a la base de dades.

Les dues opcions per a la validació comporten saber quins són els audit trails a consultar i donarien el mateix resultat, però la segona opció és millor per dues raons: la primera raó és que és molt fàcil fer una consulta *count* en SQL per tal de saber el número de missatges totals que hi ha a la base de dades, cosa que a partir de la interfície gràfica no es pot fer directament i ens costaria una mica trobar la manera de fer-ho (s'haurien de mostrar tots els missatges i comptar el número d'elements que hi ha). La segona raó és que aquest script per a consultar la base de dades, si és prou modulable ens permetrà fàcilment fer més tipus de consultes a la base de dades només canviant els paràmetres necessaris.

Tenint ja presents els passos necessaris per a enviar missatges i per a validar que han arribat correctament, es pot intuir quina ha de ser l'estructura. D'una banda hem de tenir un script per a crear els audit trails utilitzant la plana web. D'altra banda hem de tenir un o més scripts per a instal·lar els kFeeds i els clients. També hem de tenir un script que executi els clients amb el nombre de missatges que es volen enviar i finalment hem de tenir un script que faci una consulta a la base de dades per tal de saber si el número de missatges que han arribat és l'esperat.
Més endavant, a la fase d'implementació, ja es veurà com s'ha implementat tot això.

4.2.1 Interacció de l'usuari amb l'aplicació

Ja hem vist quines són les funcionalitats que ha de tenir l'aplicació i quina ha de ser l'estructura a implementar. Ara, el que queda per veure és com ha de ser l'aplicació des del punt de vista de l'usuari, és a dir, de quina manera interactuarà aquest amb l'aplicació. Concretament, els problemes a tractar són dos: funcionalitats a les què l'usuari pot accedir de manera directa (punts d'entrada a l'aplicació) i com aquest podrà configurar els paràmetres bàsics (número de missatges a enviar, nom dels audit trails, paràmetres de la base de dades, etc.).

D'una banda, tots els paràmetres que utilitza l'aplicació es trobaran en un fitxer de propietats, el qual serà fàcilment modificable, gràcies a la inclusió de

comentaris. A part d'això, es crearà un altre fitxer de propietats més petit, el qual contindrà un subconjunt de les propietats del fitxer general amb les propietats més interessants de cara a l'usuari, ja que així serà més fàcil trobar les propietats susceptibles a ser canviades, "amagant" d'aquesta manera les propietats internes de l'aplicació. A la secció de desenvolupament d'aquest mateix document es podrà veure com s'ha implementat tot això, per tal de tenir propietats duplicades als dos fitxers sense que hi hagi conflictes.

D'altra banda, hem de definir quins són els punts d'entrada a l'aplicació. Un d'aquests punts d'entrada i a la vegada el de més alt nivell (el més general de tots), serà el del *smoke*, tasca que utilitzarà totes les altres funcionalitats (configurar els kFeeds i els clients, crear els audit trails, enviar missatges i comprovar la base de dades).
També és interessant que es puguin fer altres tasques per separat; així doncs, altres punts d'entrada seran l'execució de cada client per separat (per comprovar que s'envien missatges i per a fer tests d'estrès), la creació d'audit trails (interessant per fer altres tipus de tests) i la consulta de base de dades (permet consultar audit trails per separat, sense haver d'enviar missatges prèviament).

4.2.2 *Feedback* de l'aplicació

Com a tota eina destinada a testejar, és molt important de quina manera es donarà *feedback*, o dit d'una altra manera, de quina mena i com seran els reports que donarà l'aplicació.

Els reports han de donar informació clara i concisa del que ha passat durant l'execució del test. Per tant, s'ha de definir la millor manera de crear aquests reports per a què siguin útils i entenedors per als usuaris.

Diverses opcions per a crear reports

Bàsicament hi ha dues maneres de crear els reports: treure la informació per pantalla, o crear un fitxer de logs on hi hagi una traça del que ha succeït durant l'execució del test. Cadascuna d'aquestes opcions té els seus pros i les seves contres.

Treure la informació per pantalla és molt senzill i es pot anar seguint l'execució "en directe", a la vegada que es pot veure exactament en quin punt de l'execució es trobava quan ha fallat. Ara bé, te alguns inconvenients: aquesta informació es volàtil, és a dir, que al tancar la finestra els missatges es perden.
D'altra banda, tenim l'opció de crear un fitxer on es guardin els reports. Aquesta opció permet guardar els reports d'una manera senzilla i es podria veure què és el que ha fallat, però no exactament el punt on es trobava a l'hora de la fallida. El principal inconvenient que comporta aquesta opció, però, és el fet de què s'ha d'escriure a disc, quan es possible que no disposem de permisos per a fer-ho o que el disc estigui ple i no es pugui escriure.

Així doncs, l'opció escollida per a crear els reports i donar feedback és la de treure els missatges per pantalla. La manera de fer-ho és molt senzilla: per defecte configurarem l'aplicació per a que s'aturi quan hi hagi un error, de tal manera que per a saber que ha fallat només haguem d'imprimir un missatge per pantalla a cada pas de l'aplicació. Quan aquesta finalitzi, Ant donarà un error de compilació i veurem quin és l'últim missatge mostrat, el qual indicarà en quin punt a fallat l'execució del test.

Un altre tema a comentar és que l'aplicació utilitzada per a crear i validar els audit trails (*Webtest*), crea els seus propis reports en forma de pàgina html i de fitxers de logs guardats a disc. Aquests reports són útils, sobretot els de html, on fins i tot surten estadístiques dels resultats, però en el nostre cas no són útils, ja que els tests s'executaran moltes vegades a servidors sense interfície gràfica i això faria que fallés l'execució dels tests. D'altra banda i per la mateixa raó abans esgrimida, guardar fitxers a disc no és adequat. Per tant, els reports que crea Webtest seran deshabilitats.

4.3 Esquema del disseny de l'aplicació

A mode de resum i per entendre una mica millor el disseny de l'aplicació, a la figura 4.1 es pot veure com s'ha estructurat la mateixa. En aquest esquema es pot apreciar la interrelació entre les diferents funcionalitats, així com quines són les

funcionalitats accessibles directament per l'usuari. També es pot veure que hi ha dos fitxers de propietats, on el de les propietats locals preval sobre l'altre.

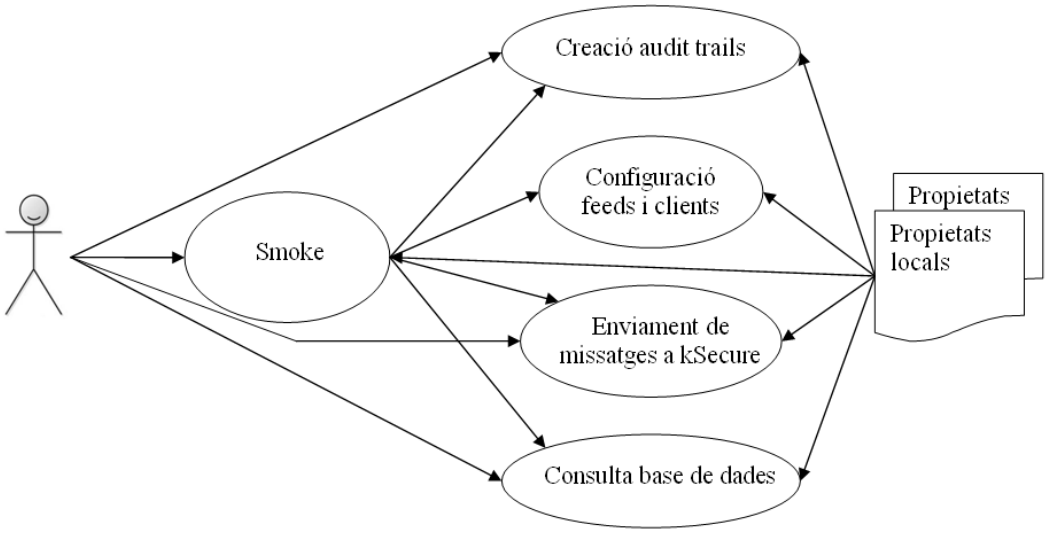


Figura 4.1: *Esquema del disseny de l'aplicació*

Capítol 5

Desenvolupament i proves

En aquest apartat es pot veure com s'han implementat els scripts de l'aplicació i com s'han solucionat els problemes plantejats a la fase de disseny.
Abans de començar a explicar més a fons com s'ha implementat l'aplicació, anem a explicar una mica quina és la filosofia d'*Ant* per tal d'entendre millor alguns raonaments.

5.1 Nocions sobre Ant

Ant és un llenguatge de programació destinat a compilar projectes de programació, el qual permet realitzar compilacions automàtiques.
Ant no és semblant a cap altre llenguatge de programació conegut, ja que té algunes particularitats, podríem dir, una mica estranyes, encara que és un llenguatge molt intuïtiu i de fàcil aprenentatge. Així doncs, introduirem alguns conceptes d'Ant que cal conèixer per entendre la seva filosofia.

La primera cosa a destacar és que no existeixen les variables, per tant, no es poden guardar valors i operar amb aquests igual que ho faríem amb un llenguatge tipus C, Java, etc. En canvi, el que hi trobem són les anomenades "propietats". Aquestes propietats venen a ser el mateix que una variable, amb la diferència de que només es poden instanciar un cop; una vegada que s'ha instanciat una propietat, aquesta mantindrà el seu valor dins de la mateixa execució d'Ant.

Les propietats es solen guardar en un fitxer de propietats, el qual es carregarà dins de cada fitxer que les necessiti (utilitzant la tasca *property*).
Això de no poder instanciar una propietat més d'un cop és molt incòmode, però es pot solucionar. La solució es basa en que encara que les propietats només es puguin instanciar un cop, aquestes només tenen valor dintre de la mateixa execució d'Ant. Així doncs, si volem fer que la mateixa propietat agafi diferents valors en diferents situacions, el que hem de fer és aïllar-la, fent una crida nova a Ant, aprofitant que Ant permet niar crides a ell mateix. D'aquesta manera, quan es surti d'una execució d'Ant, les propietats instanciades dintre d'aquesta execució perdran el seu valor.

Un altre tema molt important a ressenyar és que, com ja he comentat a l'apartat d'entorn i eines, els scripts d'Ant són XML. Això, entre d'altres coses, vol dir que no hi ha funcions ni classes. Per contra, disposem de les anomenades tasques (o targets) i de macros.
Targets i macros són molt semblants (el que es pot fer amb un target també es pot fer amb una macro), però tenen unes quantes diferències importants: els targets no poden rebre paràmetres, però les macros sí (reben atributs); les macros poden ser cridades tantes vegades com es vulgui, però les tasques no, només es poden cridar un sol cop dins d'una mateixa execució d'ant; els punts d'entrada quan es fa una crida a Ant han de ser targets, ja que les macros només poden ser cridades internament (des d'un target o una altra macro). Per cridar un target des d'un altre target, el més adequat és utilitzar el paràmetre *depends*, on es posen totes les tasques que s'han d'executar abans d'aquesta on ens trobem. A la figura 5.1 es pot veure un exemple d'utilització de *depends*, on primer s'executa el target A, després el B, més tard el C, i finalment el D.
Per a implementar el que seria una funció evitant aquestes restriccions comentades, normalment s'utilitza una macro (la podem cridar tants cops com vulguem i se li poden passar paràmetres). Llavors, si volem que aquesta macro es pugui executar quan fem una crida a Ant (és a dir, que la macro sigui un punt d'entrada), el que farem serà crear un target que simplement s'encarregui de fer una crida a aquesta macro.

```
<target name="A"/>
<target name="B" depends="A"/>
<target name="C" depends="B"/>
<target name="D" depends="C,B,A"/>
```

Figura 5.1: *Exemple de crides entre tasques fent servir depends*

Una altra característica d'Ant és que si es vol utilitzar una macro o un target que estigui en un altre fitxer, no cal fer una nova crida a Ant per aquella macro o target, sinó que es pot fer un *import* del fitxer on es troba la tasca desitjada i utilitzar tots els targets i macros d'aquest fitxer en l'altre. Per a entendre-ho millor, seria com enganxar el fitxer desitjat en el que fem el *import*. Les propietats instanciades al fitxer "pare", tenen el mateix valor per al fitxer importat. Aquesta característica ens facilita el fet d'utilitzar tasques i macros definides en altres fitxers, estalviant-nos de fer una nova crida amb el que això comporta (execució més ràpida, menys utilització de memòria, etc.).

Per definició, el fitxer "pare" d'un projecte Ant té com a nom *build.xml*. Així doncs, si des de línea de comandes escrivim la comanda *ant*, aquesta buscarà el fitxer *build.xml* i compilarà el projecte contingut dins d'aquest fitxer. En cas de voler executar un altre fitxer, només s'ha d'escriure la comanda *ant* seguida del nom del fitxer.
Si no especifiquem quin target del fitxer volem executar, s'executarà el target per defecte[1].
Si volem saber quins targets (punts d'entrada) hi ha definits en un projecte Ant, només hem d'afegir el paràmetre *-p* a la comanda *ant* i el nom del fitxer (opcional).

5.2 Scripts bàsics de l'aplicació

En aquesta secció enumeraré els scripts i fitxers implementats més importants, tot esmentant quins són els targets d'entrada, les macros més interessants i les particularitats més rellevants.

[1] El target a executar per defecte es defineix opcionalment com a paràmetre dins d'un projecte Ant.

genericAuditTrail.xml

Aquest script és l'encarregat de crear un audit trail.
Té un únic target, el qual inclou una tasca *webtest*, que és l'encarregada de crear l'audit trail i fer les comprovacions pertinents mitjançant la plana web de kAuditor.
És un script totalment genèric, ja que els passos a seguir per a crear i verificar els diferents audit trails són exactament els mateixos, només canvien els valors. Així doncs, tots els valors (nom de l'audit trail, tipus, caducitat de l'audit trail, etc.) que necessita el script són propietats que han d'estar prèviament instanciades, bé als fitxers de propietats o bé quan es fa la crida.

allTests.xml

En aquest fitxer hi han definides vàries macros que ens permetran crear diferents tipus d'audit trails, fent crides al *genericAuditTrail.xml* amb les propietats adequades.

La macro de més baix nivell, anomenada *create-generic-auditTrail*, serveix per a instanciar les propietats necessàries i fer la crida a *genericAuditTrail.xml* per tal de crear un audit trail.
A un nivell més alt, tenim vàries macros, concretament una per a cada tipus d'audit trail (Log4j, securetext, syslog i SAP). Cada una d'aquestes macros crida a la macro *create-generic-auditTrail* passant-li com atributs els paràmetres necessaris per a crear un audit trail d'aquest tipus.
Encara a un nivell més alt tenim la macro *create-one-auditTrail-of-each*, encarregada de fer una crida a cada una de les macros de nivell immediatament inferior per tal de crear un audit trail de cada tipus.
Per acabar, en el nivell més alt de tots, tenim dues macros, les quals permeten crear audit trails per a diferents entorns d'execució de tests. La primera macro crea els audit trails per a un entorn *bàsic* (un audit trail de cada tipus). L'altra macro crea els audit trails per l'entorn d'*estrès* (tres audit trails de cada tipus).

build-smoke-create-audit-trails.xml

Es podria dir que aquest script és una abstracció del *allTests.xml* (actua com a façana), ja que únicament conté dos targets que fan sendes crides a les macros de més alt nivell del *allTests.xml*. És a dir, és un punt d'entrada per a crear audit trails per als diferents entorns d'execució de tests (*bàsic* i *estrès*).

build-check-database.xml

Aquest script és l'encarregat de consultar la base de dades per veure si el nombre de missatges continguts per un audit trail determinat és l'esperat.
Bàsicament hi ha dos macros: la primera d'elles i més important, anomenada *macro_check-database*, és la que mira al fitxer de propietats quina base de dades estem utilitzant (Oracle o MySQL) per tal de carregar el driver adequat i fa la consulta per comparar amb el número esperat; si el número de missatges continguts a la base de dades no és l'esperat, falla l'execució del test. La segona macro, anomenada *check-database-smoke*, es troba un nivell més amunt i es va definir per a fer les crides necessàries a la macro *macro_check-database* per tal de consultar les dades dels audit trails corresponents a un test de *smoke*.

build-feedsetup.xml

És l'encarregat de copiar els *kFeeds* i configurar-los adequadament. Si ja existia una instal·lació dels *kFeeds*, l'esborra i la torna a crear.

build-clientsetup.xml

És l'encarregat de copiar els clients i configurar-los adequadament. Si ja existia una instal·lació dels clients, l'esborra i la torna a crear.

build-clientexecution.xml

Aquest script és el que executa els clients per a enviar els missatges desitjats a kSecure.

Té dos macros per a cada tipus d'audit trail (una per arrencar el *kFeed* necessari i l'altra per arrencar el client).

build-smoke.xml

Aquest és el que fa totes les crides necessàries per a executar un test de *smoke* i el que serveix com a punt d'entrada per a l'usuari.
Bàsicament té un target per a cada crida necessària a un altre script per a executar les diferents funcionalitats (creació d'audit trails, configuració de *kFeeds* i clients, enviament de missatges i verificació de la base de dades).
A part d'això, té un altre target anomenat simplement *smoke*, per tal de reunir d'una manera concisa totes les crides als diferents targets d'aquest mateix script. Així, d'una manera fàcil, es poden executar totes les funcionalitats o només la desitjada, facilitant també l'ampliació de funcionalitats en un futur.

build.properties i build.local.properties

Aquests dos fitxers són els que contenen totes les propietats necessàries per als scripts, les quals no varien durant l'execució del test. El problema és que hi ha moltes propietats i si les fiquem totes dins del mateix fitxer, es convertirà en una cosa intractable per a l'usuari.
Per tal de poder canviar les propietats de manera més amigable, s'han creat dos fitxers de propietats: al fitxer *build.properties* es troben totes les propietats (tant les que canvien per a cada usuari com les internes dels scripts), separades per tipus, per tal de millorar la seva comprensió. D'altra banda, el fitxer *build.local.properties* conté un subconjunt de les propietats contingudes al fitxer *build.properties*, amb només les propietats susceptibles a ser canviades per l'usuari.
Aquesta idea de tenir dos fitxers de propietats amb propietats duplicades, funciona gràcies a que amb Ant les propietats només es poden instanciar un cop. Això permet carregar primer el fitxer de propietats locals i després el general per acabar de carregar la resta de propietats.

5.3 Problemes trobats durant el desenvolupament

Durant la fase de desenvolupament han sorgit diversos problemes.
El primer problema de tots va ser l'haver de programar amb un llenguatge completament desconegut per a mi (Ant) i haver d'utilitzar *Canoo Webtest* per a testejar la web, també sense tenir-ne coneixements previs. Això va implicar haver de perdre temps per a conèixer la filosofia d'aquest paradigma de programació, amb la pèrdua de temps que això comporta.
Un altre problema important el vaig trobar a l'hora d'haver d'utilitzar variables. Com ja he comentat prèviament, a Ant no existeixen les variables, així que s'han d'utilitzar propietats (només es poden instanciar un cop) o atributs (dins d'una macro). Això a estat un problema, ja que hi ha molts targets que han hagut de ser convertits a macros, fent que aquestes macros no es puguin cridar directament des de la consola ni fent una crida a Ant des d'un altre fitxer.

Al principi vaig cometre un error de principiant amb Ant. Aquest error, encara que lleu, feia que l'execució de l'aplicació fos molt lenta. Concretament, el problema era que hi havia massa crides a Ant entre fitxers i aquesta no és una bona pràctica programant amb Ant (la bona pràctica és utilitzar el *import* i el *depends* per no haver de crear noves instàncies d'Ant). No és un problema greu perquè l'aplicació funciona bé, però sí que és un problema d'optimització. Això va comportar que moltes crides a Ant s'haguessin de canviar, bé utilitzant el *depends* (per a executar tasques sense fer una crida a Ant) o bé transformant algunes tasques en macros.

Un altre problema trobat era que la documentació de *Webtest* no explicava clarament com desactivar els reports HTML, però s'havien de desactivar perquè sinó l'aplicació no podria córrer als servidors Linux de l'empresa sense interfície gràfica. Així doncs, vaig haver de revisar el codi font de *Webtest* (és Open Source) per veure en quin punt es creava el HTML per tal de desactivar-lo. Això em va portar un parell d'hores, però em va ajudar a entendre una mica més a fons com funciona *Webtest*.

5.4 Proves

Durant el desenvolupament de l'aplicació s'anaven fent proves per veure que cada part funcionava individualment, però això no era suficient, així que es van fer més proves, com proves d'estrès, proves amb Windows, proves amb Linux, etc.

Al ser una aplicació de testing, no s'ha de provar només que aquesta funcioni, sinó que a més s'ha de comprovar que faci les comprovacions bé. Això és complicat, perquè pot ser per exemple que s'enviïn els missatges correctament però validi malament que han arribat tots els missatges, el que vol dir que no funciona correctament, encara que els missatges arribin. Amb altres paraules, s'han de fer proves de les proves.
Això de fer proves de les proves és una cosa que ens ha anat molt bé i ens ha estalviat temps, ja que una vegada comprovat que la tasca de testejar quelcom funciona correctament, ens serveix per a comprovar que la funcionalitat que testeja funciona. Per exemple, si no es creen els audit trails correctament, no ens haurem de preocupar de verificar-ho nosaltres, sinó que la tasca de verificació de creació dels audit trails ens ho verificarà.

Bàsicament, després de fer tests unitaris a cada part, vam fer uns tests generals per a comprovar la completa funcionalitat de l'aplicació.
Primer vam provar un cas bàsic, més concretament, que funcionés el *smoke*. Primerament vam fer la comprovació amb Windows (que és el sistema operatiu des del qual jo estava implementant l'aplicació). Quan vam veure que això funcionava, vam instal·lar l'aplicació en un servidor Linux i la vam provar allà.

Totes les proves fetes fins aquest moment s'havien superat (en alguns casos arreglant els errors trobats), demostrant que l'aplicació funcionava mínimament, així que la següent prova a realitzar era la d'estrès o càrrega. Per tal de fer aquesta prova, el que vam fer va ser enviar molts missatges a *kSecure* i veure que l'aplicació els enviava tots. En aquest punt el test va fallar, i d'aquí va sorgir el primer "bug"(error) de l'aplicació. Concretament, el problema és que el *kFeed* de Syslog s'atura mitjançant un timeout; si el timeout és curt no li dóna temps a enviar tots els missatges abans d'aturar-se. Així doncs, s'havia d'avisar a l'usuari d'aquest fet i de que per a evitar-ho s'havia d'augmentar aquest valor al fitxer de propietats.

Finalment i com a última prova, vam enviar l'aplicació definitiva al departament de desenvolupament per a que l'integressin dins del seu *Cruise control* per a fer un test de *smoke* automàticament cada nit després de compilar *Secure Audit Vault*. Dir que aquesta execució va funcionar perfectament, el que vol dir que les proves realitzades abans de lliurar la versió definitiva es van fer correctament i contemplant totes les funcionalitats de l'aplicació.

5.4.1 Mostra del comportament de l'aplicació

En aquesta secció es vol mostrar el comportament de l'aplicació en alguns dels casos més habituals. Així doncs, a continuació es mostren unes captures de pantalla de l'execució i resultats d'algunes comandes en un entorn real.

En primer lloc, a la figura 5.2 es mostra l'execució de la tasca *smoke*, el resultat de la qual es satisfactori (figura 5.3). Concretament, podem veure com després d'enviar un missatge a cada audit trail, el nombre de missatges enviats i els que hi ha a la base de dades concorda (es fa la comprovació per a cada un dels audit trails). Si ara intentem tornar a fer un *smoke*, aquest fallarà perquè quan vagi a

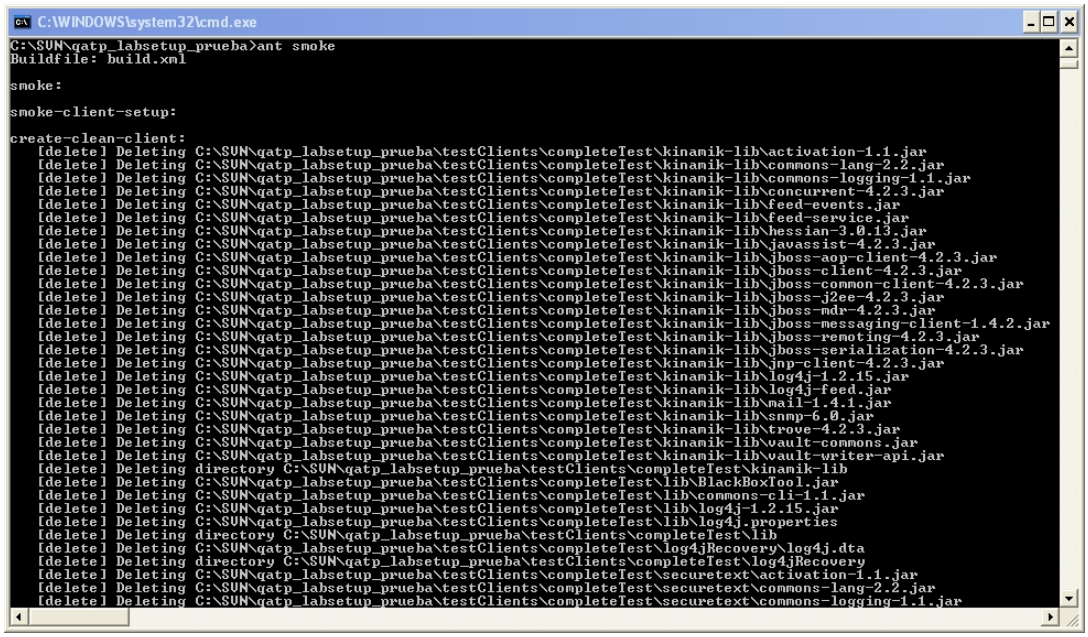


Figura 5.2: *Execució de la tasca smoke*

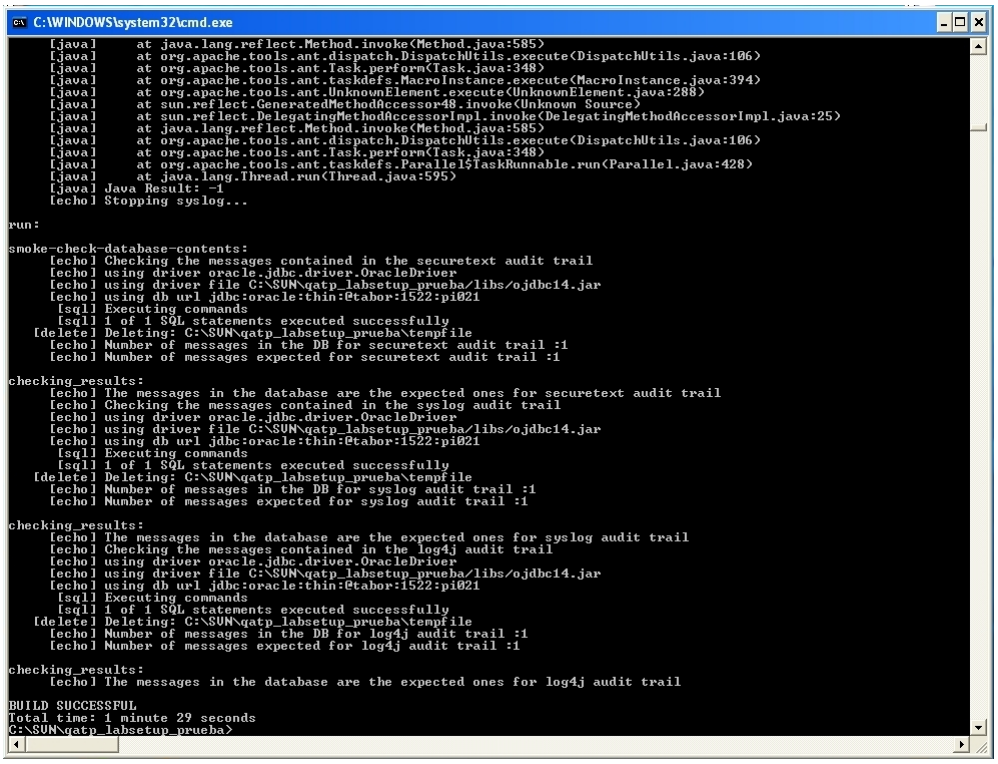


Figura 5.3: *Resultat de l'execució de la tasca smoke*

crear els audit trails necessaris, veurà que ja han estat creats anteriorment i per tant ja s'ha executat un test de *smoke*. A la figura 5.4 es pot veure com, efectivament, això succeeix i sabem que ha fallat en aquest punt gràcies als "echo", que ens diuen què estava fent l'aplicació quan ha fallat l'execució.

Ara executem la tasca *run* (figura 5.5), amb la intenció d'enviar un missatge més a cada fill, però ara sense fer tot el *smoke*. Només farem l'enviament de missatges. En aquest punt, després d'haver fet un *smoke* i un enviament extra de missatges, volem veure que si fem una comprovació de la base de dades per saber si aquesta conté exclusivament els missatges enviats per el *smoke*, l'aplicació ens avisa de que hi ha més missatges dels esperats. Concretament, a la figura 5.6 podem veure com l'execució s'avorta perquè validant l'audit trail "securetext" troba que aquest conté dos missatges, quan només hauria de tenir-ne un.

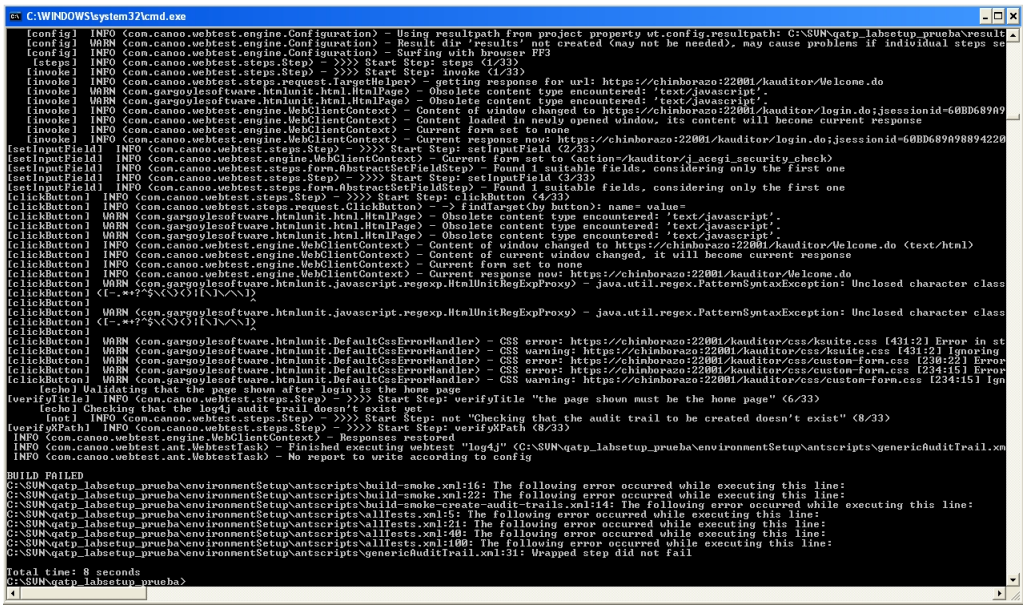


Figura 5.4: *Resultat de la segona execució de la tasca smoke*

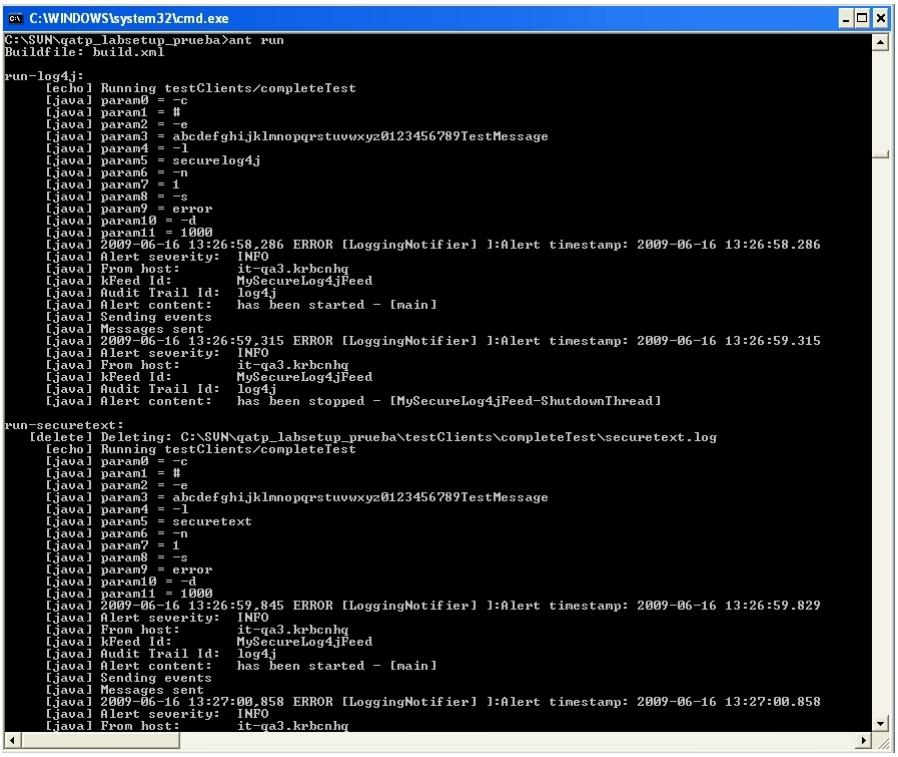


Figura 5.5: *Execució de la tasca run*

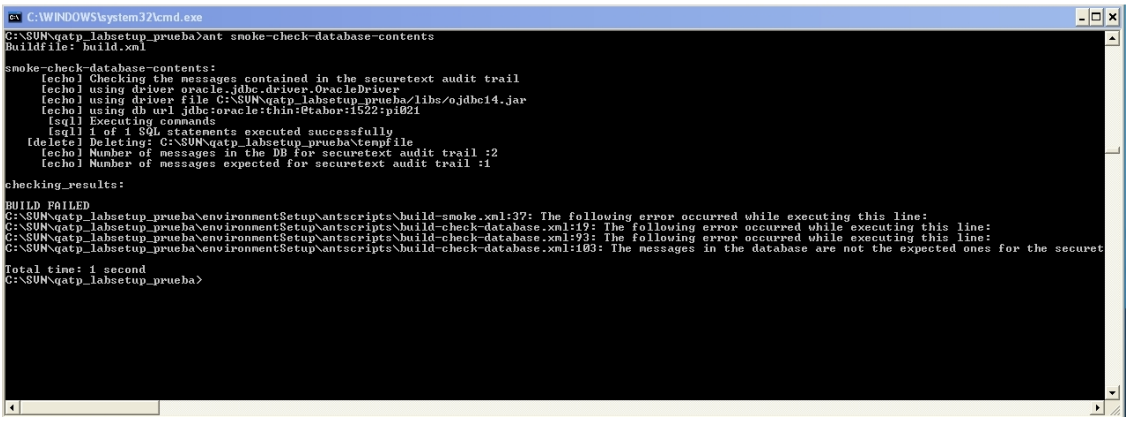


Figura 5.6: *Resultat de la comprovació de la base de dades*

Capítol 6

Conclusions

6.1 Revisió dels objectius

Després d'haver realitzat aquest projecte i veure els resultats obtinguts, crec certament que s'han acomplert tots els objectius plantejats al començament d'aquest, cobrint tots els requisits establerts per l'empresa. S'ha aconseguit crear una eina que facilita molt el treball al departament de qualitat de l'empresa, automatitzant tasques com les de configurar l'entorn per a enviar missatges, creació d'audit trails, enviament de missatges i consulta de base de dades. També ens permet fer un test de totes aquestes funcionalitats en conjunt (test de *smoke*), el qual és molt costós en quant a temps si s'ha de fer manualment.

L'aplicació, tal i com s'especifica als requisits, és compatible tant amb Linux com amb Windows i a la vegada és compatible amb el Cruise Control. Això vol dir que aquesta aplicació és útil tant per al departament de qualitat de l'empresa com per al departament de desenvolupament, ja que es poden executar tests automàticament, programant la seva execució amb el Cruise Control.
Un altre dels objectius importants era que aquesta eina fos amigable i fàcilment actualitzable, cosa que crec que s'ha aconseguit sobradament, gràcies a la modularitat de l'aplicació i la seva estructura construïda per capes.

Per últim, notar que tot això s'ha aconseguit sense haver d'invertir capital en software ni hardware, un altre dels principals requisits preestablerts.

6.2 Revisió de la planificació temporal

La planificació temporal del desenvolupament del projecte ha resultat ser bastant encertada en quant a la seva duració, encara que hi ha hagut alguna variació amb les dates. Aquestes variacions han estat degudes, sobretot, a que hi va haver fases en què em vaig haver de sincronitzar amb un company de feina, encarregat de crear els clients que jo havia d'utilitzar a la meva aplicació i el qual es va demorar un parell de dies en la seva entrega. Aquesta demora es va veure compensada pel fet de que jo tenia planificats 15 dies per a implementar el tema d'enviament de missatges, el qual, gràcies a aquests clients creats pel meu company, només em va portar dos dies de feina. Així doncs, encara que el temps d'implementació es va reduir uns 13 dies, aquests es van haver d'utilitzar a la fase de proves, a la qual van sorgir errors que es van haver de solucionar.
M'agradaria remarcar que quan parlo de dies no em refereixo literalment a dies de treball, sinó a un interval de temps, és a dir, per exemple, si una tasca dura quinze dies, no vol dir que li dediqui el treball de 15 dies, sinó que aquesta tasca termina quinze dies després de començar. Dit d'una altra manera, aquests quinze dies segurament seran menys en quant a treball, ja que no he dedicat totes aquestes hores ni tots aquests dies al projecte, però és útil per a posar un termini.

6.3 Futures vies de continuació

Encara que s'ha aconseguit crear una eina útil i que estalvia molt de temps, crec que encara es podria millor bastant afegint noves funcionalitats a testejar. De fet, al departament de qualitat de l'empresa ja estem treballant per ampliar aquestes funcionalitats.
Si bé podem parlar de millores, no podrem dir mai que l'aplicació ja està totalment acabada, perquè els tests a executar poden ser infinits, tant com donin de sí

les funcionalitats de l'aplicació a testejar i/o la imaginació de la persona encarregada de fer els tests. El més important és que l'aplicació creixi, però que ho faci bé, estructurant molt bé el codi, de manera que sigui entenedor i fàcilment modificable, per a que així pugui ser actualitzada per qualsevol persona sense molt d'esforç.
Ara per ara, crec que el camí que hem de seguir és el d'afegir algunes funcionalitats bàsiques que encara no estan incloses i trobar algun sistema per a documentar totes les tasques i macros que conté l'aplicació (a l'estil dels *javadocs* d'aplicacions java), que potser ara mateix no és necessari, però a mida que aquesta vagi creixent ho serà.

6.4 Valoració i conclusió final

Crec que el projecte ha estat un èxit, ja que era una eina necessària per a l'empresa, sobretot per al departament de qualitat i hem aconseguit una aplicació que funciona i que estableix les bases per a seguir millorant. De fet, l'aplicació ja s'està utilitzant a l'empresa per a testejar el nostre producte i, de moment, ja ha trobat dos bugs importants.

Sovint no se li dóna gaire importància a l'automatització, ja que es tendeix a pensar que s'ha de dedicar molt de temps, un temps que es podria dedicar a testejar manualment, però ara, des de la meva experiència personal tant com a tester com a automatitzador, puc afirmar que l'automatització és molt útil i necessària, ja que el temps que s'ha de dedicar és molt menor que el que estalviarà en un futur. De totes formes, s'ha de definir un límit: no tot és automatitzable, hi ha coses que no val la pena automatitzar, ja que el cost de la seva automatització es més gran que els beneficis que aportarà, per això és molt important definir els requisits i fer un bon disseny.

Personalment, aquest projecte m'ha aportat moltes coses positives. D'una banda, m'ha permès conèixer millor el producte de la meva empresa, cosa que em beneficia de cara al treball diari a l'empresa. D'altra banda, m'ha servit per a portar un projecte real, amb unes fites i uns requisits marcats, el qual és la primera

vegada que faig, amb la satisfacció afegida de què ha sortit com s'havia previst. A més a més, m'ha servit per a conèixer una mica més a fons l'interessant món de l'automatització, el qual és molt aplicable dins de l'àmbit de la qualitat.
Sense dubte, crec que ha estat un projecte molt interessant i que beneficiarà a la meva carrera personal.

Bibliografia

[1] Steve Loughran, Erik Hatcher, *"Ant in Action, Second Edition of Java Development with Ant"*, Manning Publications, 2007

[2] *"Apache Ant 1.7.1 Manual"*
<http://ant.apache.org/manual/index.html>

[3] Entrada Wikipedia: *"Apache Ant"*
<http://es.wikipedia.org/wiki/Apache_Ant>
Última modificació: 11 de juny de 2009

[4] Rick Hower, *"Web Site Test Tools and Site Management Tools"*
<http://www.softwareqatest.com/qatweb1.html>
Última revisió: 17 April de 2009

[5] *"opensourcetesting.org, open source software testing tools, news and discussion"*
<http://www.opensourcetesting.org>

[6] *"Testing Experience, the magazine for professional testers"*, número 4 (desembre 2008)

[7] Entrada Wikipedia: *"Pruebas de software"*
<http://es.wikipedia.org/wiki/Beta_testing>
Última modificació: 13 de març de 2009

[8] *"One Stop Testing, Your One Stop Guide To Testing"*
<http://www.onestoptesting.com/>

[9] *"Software Testing Life Cycle"*
<http://www.slideshare.net/sunny.deb/software-testing-life-cycle-164292>

[10] Rajesh Updhyay, *"Software Testing Concept and Methodologies"*
<http://www.slideshare.net/srinatha/testing-sw-concept>

[11] Belal Raslan, *"Software Quality Engineering & Testing Basics"*, octubre de 2007
<http://www.slideshare.net/belalraslan/software-testing-basics>

[12] Antoni Aloy López, "*LaTeX para usuarios de procesadores de texto"*, 2003

[13] Paco Aldarias Raya, *"LaTeX con Linux Debian"*
<http://pacolatex.iespana.es>

[14] Walter Mora F., Alex Borbón A., *"Edición de textos científicos con LaTeX, LaTeX2Html y presentaciones con Beamer"*, Escuela de Matemática Instituto Tecnológico de Costa Rica, 2009

Signat: Iván Vargas Checa

Sant Andreu de la Barca, juny de 2009

Resum

Aquest projecte consisteix en la creació d'una eina que permeti automatitzar els tests que s'han de fer a una aplicació comercial j2ee, amb el propòsit de facilitar i estalviar feina a les persones encarregades de testejar aquesta aplicació, ajudant-les així en la seva tasca de cerca d'errors. Concretament, s'ha creat una aplicació construïda per capes, modulable i fàcilment ampliable, la qual arriba més enllà de l'automatització dels tests més habituals, permetent executar un conjunt de tests per tal de validar si la versió de l'aplicació a testejar és vàlida o no.

Resumen

Este proyecto consiste en la creación de una herramienta que permita automatizar los tests que se tienen que realizar a una aplicación comercial j2ee, con el propósito de facilitar y ahorrar trabajo a las personas encargadas de testear esta aplicación, ayudándolas así en su tarea de encontrar errores. Concretamente, se ha desarrollado una aplicación construida por capas, fácilmente ampliable, la cual va más allá de la automatización de los tests más habituales, permitiendo ejecutar un conjunto de tests para validar si la versión de la aplicación a testear es válida o no.

Abstract

This project consists of creating a tool to automate the testing of a j2ee commercial application, with the aim of facilitating and saving work to the testers in their task of finding bugs. So, a modular and very easily expandable application has been developed wich goes beyond the automation of the most common tests, providing the funcionality of executing a set of tests to validate if the current version of the application to be tested is acceptable or not.