Universitat Oberta de Catalunya

TFC Seguretat Informàtica

# Gestor de Contrasenyes Online

*Autor:*
Oriol Garcia Alemany
*Consultor:*
Cristina Pérez Solà

11 de gener de 2012

**Resum**

Aquest document correspon al treball de final de carrera d'Enginyeria Tècnica d'Informàtica de sistemes en el camp de la seguretat informàtica. Dels enunciats que es proposaven he escollit el de crear un gestor de contrasenyes online segur.

He desenvolupat una aplicació web en Java que funciona en un servidor Apache Tomcat 7. He volgut fer que el sevidor no tingui cap dada de l'usuari, ni tan sols el nom. Per aconseguir-ho he hagut de fer servir encriptació javascript en el client.

# Índex

# Índex de figures

# Capítol 1

# Introducció

Aquest document correspon al treball de final de carrera (TFC) de la Enginyeria Tècnica d'Informàtica de Sistemes de la Universitat Oberta de Catalunya. De les opcions que es proposaven he escollit la de desenvolupar un gestor de contrasenyes online segur.

## 1.1 Descripció del problema

Avui en dia els usuaris d'ordinadors han de recordar moltes contrasenyes. El mecanisme d'autenticació d'usuaris més utilitzat a internet és el basat en parelles de nom d'usuari i contrasenya.

Per motius de seguretat és molt recomanable no utilitzar la mateixa contrasenya en diferents llocs. Així, si un atacant aconsegueix fer-se amb la contrasenya d'un lloc les altres no queden compromeses. També per raons de seguretat es recomana que les contrasenyes siguin llargues i complexes, per evitar atacs de diccionari i, a més, el més segur és anar-les canviant sovint.

Per facilitar la feina als usuaris es van desenvolupar els gestors de contrasenyes, que són programes que s'encarreguen d'emmagatzemar tots els noms i contrasenyes que ha de fer servir l'usuari. Així només cal recordar una contrasenya, que dóna accés a totes les altres.

Els primers gestors de contrasenyes s'instal·laven en l'equip de l'usuari, però actualment, amb la proliferació de dispositius mòbils que ofereixen accés a internet, aquest tipus de gestors no cobrien totes les necessitats. Per poder accedir a totes les contrasenyes de l'usuari, des de qualsevol dispositiu, cal que el gestor estigui a internet.

La informació que es transmet per la xarxa pot ser interceptada. Per que el gestor de contrasenyes sigui segur cal que tot el trànsit vagi xifrat. A més, per garantir la privacitat de les dades, no es poden desar en clar al servidor.

## 1.2 Objectius

L'objectiu principal d'aquest TFC és implementar un gestor de contrasenyes segur. Els requisits mínims que cal tenir en compte són:

- S'ha de poder donar d'alta nous clients.

- Els clients han de poder afegir parelles d'usuari i contrasenya.
- Les contrasenyes no han de viatjar en clar per la xarxa.
- En cap moment l'administrador tindrà accés a les contrasenyes dels usuaris.

A l'enunciat es proposen altres funcionalitats que es podien afegir:

- Càlcul de la fortalesa d'una contrasenya.
- Generació de contrasenyes aleatòries.
- Compartició de contrasenyes amb altres usuaris.

Els meus objectius a l'hora d'escollir aquest projecte són:

- Veure com es desenvolupa un projecte de software.
- Aprendre i practicar Java (més alguns frameworks), javascript, Ajax i bases de dades.
- Refrescar els coneixements de les assignatures Criptografia i Seguretat en xarxes de computadors i Enginyeria del programari.
- Practicar l'edició de textos en LaTeX.

## 1.3 Metodologia

El gestor de contrasenyes que he desenvolupat és una aplicació web escrita en Java, que funciona en un servidor d'aplicacions Apache Tomcat 7 i desa les dades en una base de dades (MySQL o HSQLDB).

Al pla d'estudis s'esmenta la aplicació Clipperz (`http://clipprez.com`). Em va agradar la filosofia de maximitzar la privacitat i n'he agafat algunes idees. Per exemple, fa servir Ajax i javascript per encriptar i desencriptar al navegador del client, fent que el servidor no rebi mai dades en clar.

La idea al començar el desenvolupament era fer servir un procés en cascada, i en les primeres fases ha sigut així. A la fase de disseny no l'he pogut seguir degut a la meva falta de coneixement de les diferents tecnologies. He hagut d'aprendre sobre la marxa i fent servir tutorials i llibres Struts, Spring, Hibernate, JPA, javascript, jQuery, AJAX, crypto-js i CSS. Al final he hagut d'anar adaptant el disseny segons el que anava funcionant.

## 1.4 Planificació

Durant el curs s'havien de presentar quatre pacs. Al principi vaig decidir repartir les fases del desenvolupament aprofitant aquestes dates. A la figura 1.1 es pot veure com estaven repartides les diferents fases.

La primera era la presentació del projecte al principi de curs durant la primera setmana. Després comptava amb dedicar un mes (del 6/10 al 2/11) a fer l'anàlisi i disseny, a més de repassar el funcionament dels frameworks. Amb l'anàlisi no hi ha hagut gaire problemes, però amb el disseny i els frameworks si.

Per la pac 3 (3/11 a 30/11) volia fer la implementació. Al final he hagut de dedicar massa temps a aprendre i practicar amb totes les tecnologies i he hagut d'anar modificant el disseny a mesura que anava programant. Si disposés de més temps hauria de tornar a començar el disseny i la implementació.

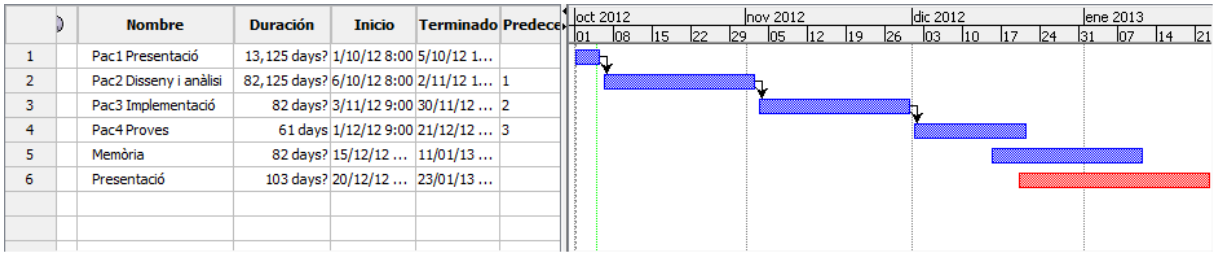


Figura 1.1: Planificació inicial

L'última pac (1/12 a 21/12) havia de ser per fer proves, retocs i provar els objectius secundaris. Els objectius secundaris els he descartat. Les úniques proves que he pogut fer han estat en els gestors d'usuaris i contrasenyes, en que he seguit un desenvolupament de tipus `Test-driven`[1].

El gener l'he dedicat sobretot a la memoria i la presentació virtual.

La planificació tal com ha anat realment és mes aviat com la de la figura 1.2.

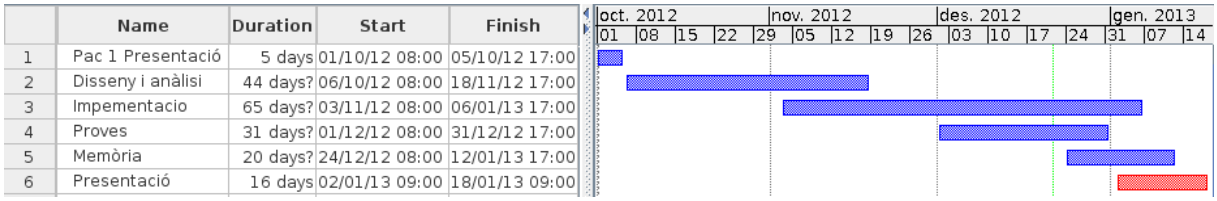


Figura 1.2: Planificació

## 1.5 Estat de l'art

Hi ha diferents aplicacions que ofereixen el servei de gestor de contrasenyes online (GCO), tant gratuïts com de pagament.

El que es menciona a l'enunciat és Clipperz[2], que és un GCO gratuït i basat en codi obert. A la web explica que intenta ser un *zero-knowledge software*. Per obrir un compte només cal nom i contrasenya i, a partir d'aquí, totes les dades s'encripten al navegador client abans d'enviar-se al servidor. Com que al servidor no es desa cap dada de contacte del client, no es pot recuperar la contrasenya. La llibreria criptogràfica de javascript que utilitza es pot baixar i utilitzar sota llicència AGPL.

Altres GCO ofereixen el servei gratuïtament, tot i que amb versions de pagament amb més funcionalitats que les gratuïtes o sense publicitat. A més no donen gaire informació del seu funcionament. Algun d'aquests GCO de amb versions de pagament són: LastPass, Roboform Everywhere o Passpack.

Per exemple, la versió gratuita de LastPass permet integració en el navegador, omplir formularis automàticament, importar/exportar contrasenyes, compartir contrasenyes i generar contrasenyes segures. La versió Premium, a més, permet l'us en telèfons mòbils.

[1] `http://en.wikipedia.org/wiki/Test-driven_development`
[2] `http://www.clipperz.com/`

# Capítol 2

# Anàlisi

Els requisits de l'enunciat donen molt marge per a decidir quin tipus d'aplicació s'havia de portar a terme. A mi em va agradar la idea del *zero-knowledge* de Clipperz iaixò és el que he intentat fer. Un gestor de contrasenyes online en que el servidor té el mínim possible de coneixement de l'usuari. Com que no tinc experiència programant aplicacions web, vaig decidir deixar de banda els objectius secundaris.

El que es demana és crear un gestor de contrasenyes online segur, en que els usuaris es puguin donar d'alta i que puguin afegir parelles d'usuari i contrasenya. Suposo que també hi ha d'haver un administrador que pugui veure la llista d'usuaris i esborrar-ne algun.

## 2.1 Casos d'ús

Els casos d'us que es poden esperar són els de la figura 2.1. Trobem dos actors: l'**usuari** i l'**administrador**. Com que la idea és que sigui segur, l'administrador no cal que pugui canviar gaires coses dels usuaris. A la figura 2.1 no hi he posat el cas login, que s'inclou en la majoria de casos, per no omplir la figura amb relacions.

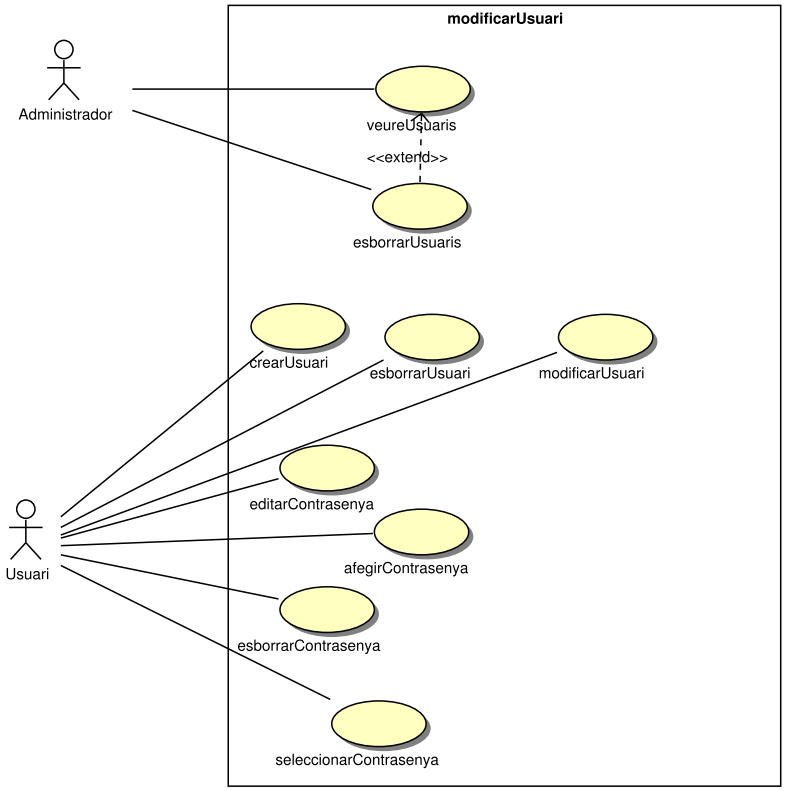


Figura 2.1: Casos d'ús

**Cas d'ús 1: "Login"**

Funció: Autenticar un usuari

Actor: **Usuari**

Casos d'us relacionats: Tots menys Crear Usuari.

Precondició: L'*usuari* existeix a la base de dades. L'usuari no està autenticat.

Postcondició: L'*usuari* s'ha autenticat.

Alternatives de proces: Si l'usuari falla l'autenticació torna a la pagina inicial.

En la majoria de casos d'us cal autenticació de l'usuari. Es demana nom i contrasenya i si son correctes es permet continuar.

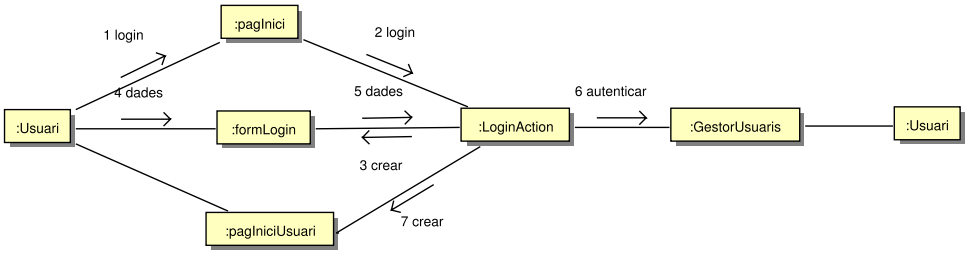


Figura 2.2: Cas 1: Login

[1]

**Cas d'ús 2: "Crear usuari"**

Funció: Afegir un usuari

Actor: **Usuari**

Casos d'us relacionats: cap

Precondició: L'*usuari* no existeix a la base de dades.

Postcondició: L'*usuari* existeix a la base de dades.

El primer cop que un usuari arriba al sistema cal que es doni d'alta. Escull un nom d'usuari i una contrasenya.

**Cas d'ús 3: "Modificar usuari"**

Funció: Canviar les dades d'un usuari

Actor: **Usuari**

Casos d'us relacionats: Login

Precondició: L'*usuari* existeix a la base de dades

Postcondició: Les dades de l'*usuari* s'han actualitzat.

Un usuari vol modificar el seu compte.

**Cas d'ús 4: "Esborrar usuari"**

Funció: Donar de baixa un usuari

Actor: **Usuari**

Casos d'us relacionats: Login

Precondició: L'*usuari* existeix a la base de dades

[1] A la figura 2.2 he representat com ho faria si fes servir la solució més clàssica. He d'acabar de mirar com ho represento si faig servir javascript per modificar la pàgina enlloc de demanar-la al servidor. Deixo per mes endavant representar el altres casos d'ús.

Postcondició: L'*usuari* no existeix a la base de dades

Un usuari vol donar-se de baixa del sistema i eliminar totes les dades.

**Cas d'ús 5: "Afegir contrasenya"**

Funció: Afegir una parella usuari/contrasenya per un usuari.

Actor: **Usuari**

Casos d'us relacionats: Login

Precondició: La contrasenya no existeix a la base de dades

Postcondició: La contrasenya existeix a la base de dades

Un usuari afegeix les dades d'una nova contrasenya. Pot afegir el nom, l'adreça d'una pagina web, el nom d'usuari i la contrasenya.

**Cas d'ús 6: "Editar contrasenya"**

Funció: Modificar una parella usuari/contrasenya per un usuari.

Actor: **Usuari**

Casos d'us relacionats: Login

Precondició: La base de dades conté la parella usuari/contrasenya antigues.

Postcondició: La base de dades conté la parella usuari/contrasenya noves.

Un usuari modifica qualsevol de les dades d'un element contrasenya.

**Cas d'ús 7: "Esborrar contrasenya"**

Funció: Esborrar una parella usuari/contrasenya.

Actor: **Usuari**

Casos d'us relacionats: Login

Precondició: La base de dades conté la parella usuari/contrasenya.

Postcondició: La base de dades no conté la parella usuari/contrasenya.

Un usuari elimina les dades d'un element contrasenya.

**Cas d'ús 8: "Seleccionar contrasenya"**

Funció: Veure una contrasenya.

Actor: **Usuari**

Casos d'us relacionats: Login

Precondició: L'usuari existeix. La contrasenya existeix.

Postcondició: cap

Un usuari selecciona pel nom una contrasenya de la llista de les seves contrasenyes. Se li mostren totes les dades.

**Cas d'ús 9: "Veure usuaris"**

Funció: Mostrar una llista dels usuaris de l'aplicació

Actor: **administrador**

Casos d'us relacionats: Esborrar usuaris, Login

Precondició: cap

Postcondició: cap

L'administrador entra per veure la llista d'usuaris de l'aplicació i els recursos que utilitzen.

Alternatives de procés: Si l'usuari vol esborrar algun usuari va a Esborrar usuaris

**Cas d'ús 10: "Esborrar usuaris"**

Funció: Eliminar usuaris de l'aplicació

Actor: **administrador**

Casos d'us relacionats: Veure usuaris, Login

Precondició: Un o més usuaris existeixen a la base de dades.

Postcondició: Aquests usuaris no existeixen a la base de dades.

L'administrador selecciona una serie d'usuaris i els elimina del sistema.

## 2.2 Classes d'anàlisi

En aquest cas el model del domini és bastant senzill, només calen dues classes: Usuari i Contrasenya.

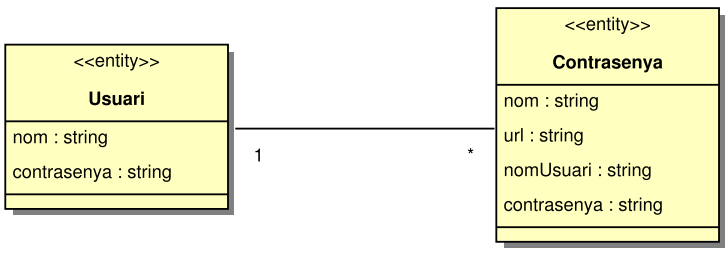


Figura 2.3: Classes d'entitats

Per gestionar aquestes entitats faré servir dues classes de control: GestorUsuaris i GestorContrasenyes

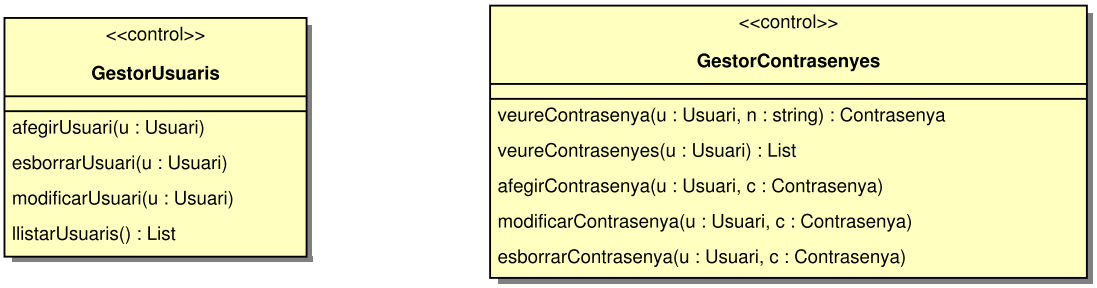


Figura 2.4: Classes de control

A més d'aquestes classes, hi ha les classes de frontera, que formen la interfície d'usuari. Per cada un dels casos d'ús hi ha una classe.

## 2.3 Bases de dades

La informació que cal desar en aquesta aplicació bàsicament són usuaris i contrasenyes.

USUARIS(*nom*,*contrasenya*)

CONTRASENYES(*usuari*,*nom*,*url*,*contrasenya*) on *usuari* és clau forana que referencia USUARIS

# Capítol 3

# Disseny

Com que no domino cap dels llenguatges ni frameworks, he intentat que la solució no fos massa complicada.

## 3.1 Introducció

És demana que l'aplicació sigui segura i que només el client tingui accés a la seva informació. Per tant tot el que s'enviï ha d'estar encriptat.

Per encriptar faré servir una clau que es genera al moment de crear un usuari. Al servidor es desa aquesta clau encriptada amb AES fent servir com a clau la contrasenya de login actual. Tot el que l'usuari desi al servidor anirà encriptat en AES. Per tant abans d'enviar una nova contrasenya nova al servidor, s'ha de fer servir javascript per encriptar-la. Igualment, s'ha de fer servir javascript per desencriptar la llista de contrasenyes de l'usuari que retorna el servidor.

Per fer que la clau estigui sempre disponible al client faré servir AJAX per anar actualitzant parts de la pàgina. Si no ho fes així hauria de demanar la clau a cada pagina o desar-la al servidor.

Per autenticació i identificació dels usuaris faré servir un hash del nom d'usuari i el hash de la contrasenya. Amb això aconsegueixo que no viatgi res en clar per la xarxa i que el servidor no conegui ni el nom de l'usuari.

La part del servidor faré servir una aplicació web en Java. Per gestionar la persistència faré servir JPA i Hibernate. Això permet no preocupar-se massa del gestor de bases de dades que s'utilitza. Jo faig servir HSQLDB i MySQL, perquè m'és còmode per fer proves, però si es posés en producció es podria canviar pel que fos només modificant un petit fitxer de text amb les dades de connexió.

Per la part de la interfície d'usuari faré servir el framework Struts. El desenvolupament és més fàcil que treballar directament amb pàgines jsp.

A la part del client faré servir javascript per encriptar/desencriptar i per actualitzar les parts de la pàgina.

## 3.2 Tria de tecnologies

La base de l'aplicació són les entitats Usuari i Contrasenya, que representen les dades que es desen a la base de dades. Són classes Java amb anotacions JPA (Java Persistence Api). Per accedir a aquestes entitats faig servir dues interfícies que defineixen els DAOs (Data Acces Object) GestorUsuaris i GestorContrasenyes. Com que havia de fer proves per aprendre i posar a punt JPA, Hibernate i les bases de dades, també vaig crear una implementació dels DAOs basat en ArrayLists, que dona menys problemes.

Per assegurar-me del funcionament dels gestors d'usuaris i contrasenyes vaig crear uns jocs de proves amb JUnit. Així si he de modificar algun punt dels gestors puc comprovar que tot segueix funcionant. Els tests es troben al paquet `tfc.tests`.

Per crear una aplicació web, Java fa servir Servlets i pàgines jsp. El framework Struts és una implementació del patró MVC (Model Vista Controlador). Internament fa servir Servlets però ofereix una API més còmoda i implementacions de les funcions comuns de la majoria d'aplicacions.

Les encarregades de rebre les peticions dels clients són les accions, que representen el controlador de MVC. Una acció rep la petició http, la processa i torna un resultat. El resultat es fa servir per decidir què es retorna al client. Normalment és una pàgina jsp, però també es pot tornar un resultat JSON.

En aquest projecte he volgut que el servidor no tingui cap informació sobre els clients, ni adreça de correu, ni informació de contacte ni claus. Per tant cal fer la feina al client. El servidor treballa igual si les cadenes de text estan encriptades o no. En una aplicació normal l'usuari clica en un enllaç i el servidor li envia una pàgina html. Com que el servidor no té la contrasenya i totes les dades estan encriptades, això no es pot fer.

L'opció que he triat és obtenir una clau a partir del login que permet desencriptar la clau que hi ha al servidor. Per això necessito javascript. A més, per no haver de generar la clau contínuament, faig servir AJAX i objectes JSON per actualitzar només parts de la pàgina. Per facilitar la programació de les pagines dinàmiques faig servir jQuery.

### 3.2.1 Patró MVC

En les aplicacions web és habitual fer servir el patró de disseny Model-Vista-Controlador (MVC). Aquest model proposa separar la presentació de la informació, la lògica del negoci i les dades, permetent un desenvolupament per separat de cada part.

**Model:** És la representació de les dades del problema. Inclou les entitats i les dades desades a la base de dades.

**Vista:** La representació visual que rep l'usuari i li permet interactuar amb el sistema. En aplicacions d'internet sol ser una pàgina HTML.

**Controlador:** És l'encarregat de rebre i processar les peticions de l'usuari i modificar el model.

El flux habitual d'una interacció amb MVC seria així:

- L'usuari fa una petició mitjançant la interfície d'usuari (vista).
- El controlador rep la petició i la processa.
- Si cal, el controlador accedeix a la base de dades, per consultar o modificar algun valor segons la lògica de negoci.
- La interfície d'usuari s'actualitza i espera interacció de l'usuari.

### 3.2.2 Struts

Apache Struts és un framework opensource que implementa el patró Model-Vista-Controlador per aplicacions JavaEE.

Una aplicació web en Java fa servir la Servlet API per respondre a peticions i formularis d'HTML. Un Servlet és una classe de Java que rep un objecte HttpRequest el processa i retorna un objecte HttpResponse a l'usuari. És molt flexible, però porta massa feina programar-hi directament. Les pàgines JSP són una manera de facilitar-ne l'ús. Una pàgina JSP té l'aspecte d'una pàgina html, però pot incloure codi Java o tags especials. Abans de ser utilitzada, una pàgina JSP és processada i convertida internament en un servlet.

L'objectiu de Struts és facilitar la programació d'aplicacions web. A més d'implementar el model MVC, oculta la Servlet API i implementa moltes de les funcionalitats que es solen trobar en la majoria d'aplicacions, com per exemple, validació de formularis, pas de paràmetres a variables, internacionalització o gestió d'errors.

Una aplicació Struts fa servir tres components: un arxiu de configuració, pàgines JSP i les accions. Una acció (Action) és una classe que té un mètode `execute()` que retorna una cadena de text. Aquesta cadena es fa servir per decidir, segons l'arxiu de configuració, quin resultat es mostra (normalment una pàgina jsp).

L'arxiu de configuració `struts.xml` té una estructura com aquesta:

```
<struts>
    <package name="default" extends="struts-default">
        <action name="login" class="tfc.Login">
            <result name="success">/success.jsp</result>
            <result name="error">/error.jsp</result>
        </action>
    </package>
</struts>
```

En aquest cas, quan l'usuari fa una consulta a `http://exemple.com/login` el framework busca a l'arxiu de configuració, troba l'acció `login` i executa el metode `execute()` de la classe `tfc.Login`. Si el resultat és `"success"` es processa la pàgina `"/success.jsp"`. Si el resultat és `"error"` es processa la pàgina `"/error.jsp"`.

En aquest projecte, en alguns punts, enlloc de retornar els resultats com a pàgines JSP els he retornat com a objectes JSON (JavaScript Object Notation), fent servir un plugin de Struts.

JSON és un estàndard d'intercanvi de dades basat en text. Com que utilitza la representació d'objectes de javascript, la utilització en scripts és senzilla.

### 3.2.3 JPA i Hibernate

Hibernate[1] és un framework de mapeig d'objectes relacionals (ORM en anglès) per aplicacions Java que facilita la persistència d'objectes i la portabilitat entre SGBDs. La funció principal d'Hibernate és el mapeig de classes Java en taules de la base de dades. Fa servir un llenguatge de consultes que es diu HQL (*Hibernate Query Language*) que després tradueix a SQL. Amb això aconsegueix que el codi sigui independent del SGBD. Hibernate permet treballar amb la majoria de SGBD que fan servir SQL.

[1] `http://www.hibernate.org/`

JPA (*Java Persistence Api*) [2] apareix per estandarditzar la persistència en Java. Abans de JPA calia treballar directament amb un framework de persistència ORM (Hibernate, TopLink, iBatis...). JPA defineix una API i un llenguatge de consultes propi (JQPL), però delega la implementació als frameworks ORM. Aquest projecte esta fa servir JPA i Hibernate com a proveïdor de la implementació.

Per fer el mapeig entre objectes i taules JPA fa servir anotacions en les entitats. Una entitat és un objecte java normal o POJO (*Plain Old Java Object*), amb l'anotació `@Entity`. Per exemple la classe Usuari es fan servir les anotacions per definir la estructura:

```
@Entity
@NamedQuery(name="Usuari.perNom", query = "SELECT u FROM Usuari u WHERE u.nom = :nom")
public class Usuari {

@Id
@GeneratedValue(strategy=GenerationType.AUTO)
@Column(name="USUARI_ID")
private long id;

private String nom;

...
}
```

JPA fa servir un arxiu de configuració en el que es pot escollir, entre altres, el proveïdor de la implementació, la base de dades que es fa servir, les dades de connexió a la base de dades. En aquest cas, com que faig servir Spring, això es troba en l'arxiu de configuració de Spring `WEB-INF\aplicationContext.xml`. He creat un arxiu `WEB-INF\classes\database.properties` que facilita el canvi de configuracions de la base de dades.

### 3.2.4 Spring

La classe `EntityManagerFactory` de JPA és la encarregada de llegir els fitxers de configuració i establir connexions amb la base de dades. És un objecte pesat que no convé anar creant contínuament. Però en una aplicació web s'hauria d'instanciar cada cop que un servlet ha de tractar una petició. Per evitar això, enlloc de crear-la cada cop, se n'instancia una i s'injecta a cada objecte que en necessita. D'això se'n diu inversió de control (Inversion of Control o IoC). En aplicacions JavaEE, els servidors d'aplicacions com JBoss o GlasFish permeten controlar el cicle de vida dels objectes i la injecció de dependències. Si no es fa servir un servidor d'aplicacions, una altra manera d'aconseguir el mateix és fer servir Spring.

Spring[3] és un framework de Java molt complet, que entre altres coses permet fer injecció de dependències. En l'arxiu de configuració (`WEB-INF\aplicationContext.xml`) es defineixen els objectes que controlarà el framework, anomenats `beans`. Spring controla la instanciació d'objectes i en el procés mira si té alguna dependència que es pugui injectar. En aquest cas controla les connexions a la base de dades i la injecció d'`EntityManager`. Una altra cosa que permet Spring és el control de les transaccions. En els gestors d'usuaris i contrasenyes es fan servir anotacions de Spring per fer que cada mètode estigui dins d'una transacció. Així si falla alguna cosa dins d'un mètode mentre es feien canvis a la base de dades es desfan tots els canvis.

[2] `http://www.oracle.com/technetwork/articles/javaee/jpa-137156.html`
[3] `http://www.springsource.org/`

### 3.2.5 jQuery

jQuery [4] és un framework de javascript que facilita la programació, simplificant processos com la manipulació del DOM, events o AJAX.

### 3.2.6 CryptoJS

Molts llenguatges de programació disposen d'una llibreria criptogràfica més o menys oficial. A javascript no hi és i cal buscar-ne alguna que ofereixi el que es necessita. En aquest cas he fet servir la `CryptoJS` perquè m'ha semblat prou complerta i documentada. Les funcions que necessitava per aquest projecte eren el hash SHA2, un xifrat de bloc AES-256 i una funció per obtenir una clau forta a partir d'una contrasenya (PBKDF).

La documentació no és prou completa com jo hauria necessitat i he tingut problemes per entendre quin tipus d'objecte retornava una funció i quin prenia com a paràmetre.

## 3.3 Criptografia

Com que volia que el servidor no tingués cap informació del client he hagut de fer servir criptografia a la part del client, en javascript. Hi ha molts llocs a internet on es desaconsella fer servir javascript per criptografia, pels diferents problemes que comporta. Per exemple, el client pot deshabilitar javascript o modificar funcions sense massa problemes. En tot cas, jo he intentat que fos segur en condicions normals.

Les llibreries criptogràfiques de javascript que he trobat difícilment inclouen res de criptografia asimètrica, per tant no n'he fet servir. M'hauria anat be la criptografia asimètrica per compartició de claus entre usuaris, per exemple.

Les funcions d'encriptació es troben a l'arxiu `js/funcions.js` i la llibreria que he fet servir és CryptoJS[5]. El funcionament de l'aplicació és el següent:

Quan es crea un usuari només es demana un nom i una contrasenya. El hash d'aquestes cadenes de text serà el que s'enviarà al servidor. A més es fa servir la funció PBKDF2 per generar una clau més segura a partir de la contrasenya. Aquesta serà la clau que es farà servir en tota la aplicació pel xifrat i desxifrat. Per desar-la al servidor l'encripto en AES fent servir el hash de la concatenació del nom d'usuari i contrasenya. Així quan l'usuari fa login rep la clau encriptada i la desencripta abans de fer-la servir. Quan l'usuari vol canviar la seva contrasenya d'usuari cal tornar a enviar al servidor la clau encriptada amb les noves dades.

Amb aquest procediment al servidor només es desen cadenes encriptades i no hi ha manera de conèixer res sobre l'usuari.

## 3.4 Interfície

#### Cas d'ús 1: "Login"

El primer que es troba un usuari és la pàgina d'inici(fig. 3.1), des d'on pot fer login o crear un usuari.

[4] `http://jquery.com/`
[5] `http://code.google.com/p/crypto-js/`


Figura 3.1: Pagina d'inici

La pàgina de login (fig. 3.2) permet introduir el nom i la contrasenya per autenticar l'usuari. Un script pot comprovar que s'han omplert els dos camps.

Per crear un usuari es té la pagina de la figura 3.3. Com que no cal saber res de l'usuari només calen dos camps. Un script comprova que es compleixen les restriccions en la mida i complexitat de nom i contrasenya.


Figura 3.2: Pagina de login


Figura 3.3: Crear usuari

# Capítol 4

# Producte final

L'aplicació tal com la tinc ara està disponible als enllaços següents:

- Arxiu WAR
- Màquina virtual (800MB)

El primer és l'arxiu WAR per instal·lar a un servidor Tomcat 7 o per importar a un IDE com l'Eclipse. El segon eś una màquina virtual per VirtualBox preparada amb el Tomcat. No cal entrar per res, però si es vol, la contrasenya de l'usuari root és toor. Fa servir DHCP per configurar la xarxa. Cal buscar quina ip fa servir i anar a:

`http://ipMaquinaVirtual:8080/TFC/`

o

`https://ipMaquinaVirtual:8443/TFC/`

El certificat l'he creat jo, per tant el navegador es queixarà. L'aplicació de configuració del Tomcat està a `http://ipMaquinaVirtual:8080/`.

Per entrar al Manager App cal fer servir com a nom i contrasenya tomcat:tomcat.

## 4.1 Introducció

Com que es demanava una aplicació segura he fet que tot el que s'envia al servidor estigui encriptat. Per tant, la feina d'encriptació es fa al client mitjançant javascript. Cal que el client tingui un navegador raonablement actual i que tingui habilitat javascript.

Des de `https://localhost:8443/TFC/` es pot accedir a la zona d'usuaris i a la d'administració. La contrasenya d'administració és admin:admin.

Tal com està l'aplicació fa servir com a base de dades HyperSQL i desa les dades en un arxiu. Es pot canviar als arxius de configuració.

## 4.2 Arxius de configuració

En principi l'aplicació funciona i no cal tocar res. Tot i així, si es vol canviar la base de dades només cal modificar l'arxiu `WEB-INF\classes\database.properties`. Està preparat per funcionar amb HyperSQL, però també pot funcionar amb MySQL només comentant unes línies i descomentant-ne unes altres. En aquest arxiu també es pot canviar si volem que la base de dades es crei a cada execució o no.

A més de la base de dades també hi ha els arxius de configuració dels frameworks Struts, Spring i log4j. Els de Struts i log4j es troben a `WEB-INF\classes`. L'ApplicationContext de Spring a `WEB-INF`.

He fet servir log4j per recollir els missatges de debug. Jo he fet servir el Apache Chainsaw[1]. Tal com està ara, la majoria de missatges s'envien a un SocketAppender al port 4560.

[1] `http://logging.apache.org/chainsaw/`

# Conclusió

Al final l'aplicació funciona i compleix els requeriments mínims de l'enunciat. Possiblement hauria d'haver escollit una solució més senzilla. L'haver decidit fer que res passés al servidor sense encriptar ha dificultat bastant el disseny i tots els frameworks que he fet servir me'ls podia haver estalviat. M'ha portat massa temps fer que tot funcionés.

Tenint en compte que l'única programació de xarxa que he fet a la carrera és obrir un socket java i intercanviar cadenes de text encara ha quedat prou bé.