
SECONDA UNIVERSITÀ DEGLI STUDI DI NAPOLI

FACOLTÀ DI INGEGNERIA INFORMATICA

TESI DI LAUREA
IN
SISTEMI WEB E BASI DI DATI

# DOCS STORER: UN'APPLICAZIONE WEB PER LA GESTIONE DI DOCUMENTI

Relatore
Ch.mo Prof.
Antonio d'Acierno

Candidato
Angelo Santillo
Matr. 834/485

ANNO ACCADEMICO 2005-2006

*A papà, mamma e Rosaria*
*che mi hanno*
*supportato e*
*sopportato*

# RINGRAZIAMENTI

Desidero ringraziare:

- **Il prof Antonio d'Acierno**, che ha subito accolto la mia richiesta di fare la tesi con lui e che si è sempre reso disponibile per ogni chiarimento, fornendomi di volta in volta nuovi spunti e linee guida per migliorare ciò che stavo facendo;

- **I miei familiari,** che hanno sempre creduto in me e che mi sono stati di aiuto, sia morale che materiale, nelle varie fasi di creazione e stesura della tesi;

- **I miei amici e i compagni dell'università,** che in un modo o nell'altro hanno reso meno pesanti questi anni di studio.

# INDICE

# CAPITOLO I

# INTRODUZIONE

## 1.1 Requisiti

Lo scopo della tesi era quello di realizzare una Web Application che avesse le funzionalità per la gestione di file. La flessibilità che si è deciso di avere, ha permesso all'applicazione di poter essere caratterizzata alla gestione di documenti di testo.
Le caratteristiche implementate sono le seguenti:

- Ricerca dei Documenti per tutti gli utenti;
- Registrazione di nuovi utenti;
- Scaricamento (Download) di Documenti per gli utenti che ne hanno diritto;
- Caricamento (Upload) di Documenti per gli utenti che ne hanno diritto;
- Cancellazione di Documenti per gli utenti che ne hanno diritto;
- Gestione degli utenti registrati e di quelli che hanno richiesto la registrazione; questa funzionalità è rivolta agli amministratori dell'applicazione;
- Implementazione di sistemi di sicurezza per tutelare gli utenti e per impedire l'utilizzo di alcune funzionalità, per il cui utilizzo sono richiesti determinati diritti;
- Interfaccia grafica user-friendly.

## 1.2 Linee guida per la progettazione

Il progetto è stato creato seguendo alcune direttive, che hanno permesso la generazione del codice abbastanza rapidamente e con il minor numero possibile di errori.

Si riportano di seguito le linee guida che sono state prese in considerazione (non necessariamente e rigorosamente nell'ordine presentato):

- Valutazione delle tecnologie Web già esistenti e dei vari linguaggi di programmazione: si è deciso di ricercare prima se esistessero delle soluzioni già pronte e soprattutto gratuite e open source, sia per quanto riguarda i software di sviluppo che per l'utilizzo di librerie o framework già realizzati e liberamente fruibili;
- Ricerca di guide e manuali per facilitare la comprensione e la soluzione di problematiche che si possono riscontrare nello sviluppo, oppure, per facilitare la creazione del codice per mezzo di esempi o dimostrazioni pratiche e teoriche;
- Scelta e progettazione della database: tipologia di base di dati e scelta dei vari modelli di progettazione;
- Progettazione dell'applicazione web: scelta dei modelli e delle metodologie di sviluppo;
- Realizzazione dell'applicazione mediante l'utilizzo dei software precedentemente scelti per lo sviluppo, quindi creazione della logica e della configurazione del progetto;

- Testing dell'applicazione effettuato anche da beta testers.

## 1.3 Architettura del sistema

Il sistema di riferimento è un PC su cui possa girare una Java Virtual Machine, quindi si possono prendere in considerazione un vasto numero di sistemi operativi (lo sviluppo è stato fatto su un PC dotato di Microsoft Windows XP, ma l'applicazione è stata progettata per poter essere ugualmente fatta girare senza problemi anche su una distribuzione di Linux, o qualsiasi altro Sistema Operativo su cui possa girare la Java Virtual Machine), l'importante è che si possa creare su di esso un ambiente hardware e software che possa sorreggere la scelta di sviluppare un'applicazione web, quindi principalmente client-server: è richiesta poi, una connessione ad internet abbastanza veloce, un hard-disk con buona capacità di storage e buona velocità di lettura/scrittura e di trasmissione dei dati al resto del sistema. Per il resto non c'è bisogno di un sistema eccessivamente potente, quindi si può anche scegliere di far girare l'applicazione su di un buon computer desktop per utilizzo casalingo o semi-professionale.

Si riporta di seguito un grafico che spiega come funziona un'applicazione web Client/Server.

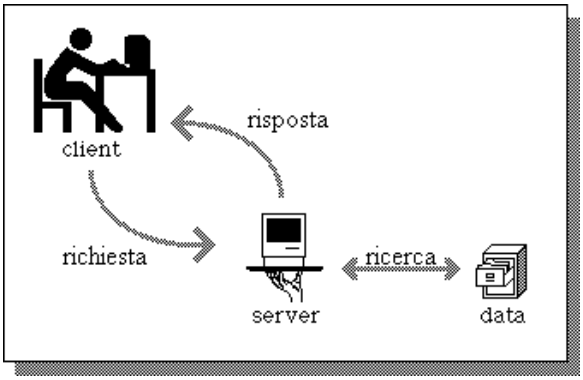



Figura 1.1 Architettura Client/Server [Morgan, 2000]

# CAPITOLO II

# METODOLOGIE E TECNOLOGIE DI SVILUPPO

## 2.1 Introduzione

Prima di esporre quali siano le metodologie e le tecnologie impiegate per progettare e realizzare la tesi, è preferibile illustrare a grandi linee cosa è e come viene sviluppata un'applicazione Web.

Per prima cosa, è bene capire a cosa ci si riferisce quando si parla di un'applicazione Web.

A differenza di un sito Web, che presenta all'utente che vi accede una serie di contenuti statici (cioè non modificabili e senza possibilità di interazione), un'applicazione Web, che è anch'essa un software, è invece in grado di presentare contenuti in modo dinamico, con la capacità di interagire con l'utente e di presentare informazioni diverse a seconda delle richieste fatte da chi accede ad essa.

## 2.2 Metodologia di sviluppo

Per lo sviluppo dell'applicazione si è fatto riferimento alla seguente metodologia:

- Analisi e specifica dei requisiti;
- Creazione del diagramma UML degli Use Cases;
- Creazione del Modello Concettuale attraverso l'utilizzo di diagrammi E-R (Entity-Relationship);
- Creazione del Modello Logico;
- Implementazione dell'applicazione;
- Testing dell'applicazione.

Al termine di queste fasi, vi è il rilascio della versione definitiva del software; si fa presente che la metodologia appena presentata non è stata seguita rigidamente, in alcuni casi si è dovuto tornare indietro di qualche fase per l'aggiunta o anche solo la correzione di alcune parti del progetto.

Alla fine dello sviluppo, dopo il rilascio dell'applicazione, il lavoro non è ancora finito, in quanto per lo sviluppatore è previsto un periodo abbastanza lungo in cui è tenuto a correggere piccoli bug e anomalie che inevitabilmente colpiscono la propria applicazione, in quanto è impossibile creare l'applicazione perfetta, cioè esente da errori, anche perché, per quanto bravo sia lo sviluppatore, ha sempre a che fare con utenti che sono in grado di creare situazioni non previste o che il progettista può aver ritenuto semplicemente poco probabili.

## 2.3 Tecnologie per la progettazione

Per progettare l'applicazione ci si è rivolti principalmente ad un solo tool di sviluppo e progettazione, in quanto si è deciso di creare un'applicazione semplice e versatile; quindi non c'è stata la necessità di effettuare una lunga ed eccessiva fase di progettazione. La scelta è ricaduta sul CASE (Computer Aided Software Engineering) PowerDesigner della Sybase, software che è stato utilizzato per la creazione dei grafici in UML e del Modello Concettuale e del Modello Logico, che rappresenta anche il punto di forza del tool stesso in quanto supporta la maggior parte di grafici UML e si interfaccia con un buon numero di DBMS (DataBase Management System).

## 2.4 Tecnologie per l'implementazione

La scelta del Sistema Operativo è stata già spiegata nel primo capitolo: visto l'utilizzo di Java come linguaggio di programmazione, è stato utilizzato Microsoft Windows XP per la progettazione; per il testing, invece, si sono utilizzati vari tipi di sistemi operativi, fra cui Linux, in modo che si potesse provare che effettivamente funzionasse tutto correttamente, anche in ambienti diversi.
La stessa filosofia con cui si è scelto il Sistema Operativo è stata anche attuata riguardo al browser (software necessario per accedere all'applicazione Web) con cui si è provata

l'applicazione: la scelta è ricaduta su Microsoft Internet Explorer e Mozilla Firefox. Il primo è stato scelto in quanto è utilizzato dalla maggior parte degli utenti, anche perché incluso nella maggior parte dei Sistemi Operativi della stessa Microsoft; l'altro browser, sviluppato da Mozilla, è invece un programma emergente e sotto molti punti di vista è innovativo, e un numero sempre crescente di utenti lo sta scegliendo come browser principale o anche come alternativa momentanea a quello della Microsoft. La scelta di questi due browser ha richiesto che il codice creato nelle pagine Web, statiche o dinamiche, fosse compatibile con entrambi i software di navigazione.

Per il lato server si è fatto affidamento su due applicazioni che sono ormai diventati stabili, potenti e di largo uso da parte degli sviluppatori:

- Apache Tomcat, realizzato dalla fondazione Apache nell'ambito del progetto Jakarta: è un Web Server che ha anche la peculiarità di essere un Container per applicazione Web. Il server in questione è diventato oggi molto stabile e, release dopo release, aderisce sempre alle nuove specifiche per quanto riguarda le Servlet, ed inoltre è molto sicuro ed è anche un applicativo open-source e gratuito, come del resto lo sono tutti gli applicativi del progetto Jakarta;
- MySQL, il più diffuso DBMS disponibile sul mercato, in quanto sicuro, potente, gratuito e disponibile per un buon numero di Sistemi Operativi.

Per la realizzazione dell'applicazione web in ambiente Java si è scelto di sviluppare il tutto mediante la creazione delle seguenti componenti:

- Servlet Java, rappresentano il cuore dell'applicazione, in quanto buona parte della logica è inclusa in esse. Il loro funzionamento è il seguente: viene creato un solo processo per ogni Servlet, e all'arrivo di una nuova richiesta all'applicazione, la JVM crea un nuovo thread, molto più leggero di un processo ma che è in grado di svolgere contemporaneamente lo stesso metodo della classe, perché ogni thread è in grado di invocare un determinato metodo separatamente. Le servlet, come tutte le classi Java, possono utilizzare le API (Application Programming Interfaces) e le classi messe a disposizione da Java o da altre società o programmatori. Le Servlet non sono eseguibili direttamente da un Web Server, ma si devono far girare all'interno di un contenitore, il cosiddetto Servlet Container;
- Java Server Pages (JSP), sono la naturale estensione delle Servlet, la loro efficacia sta nel fatto che vengono create come se fossero delle pagine HTML statiche, da esse il Servlet Container crea una determinata Servlet che svolge la logica per cui è stata creata, anche se principalmente le JSP vengono create per presentare all'utente le informazioni che ha richiesto; la logica con cui si arriva a

quel risultato può essere riposta tutta nella JSP stessa o anche altrove, ad esempio nelle Servlet stesse.

- Classi Java di Utilità e JavaBeans: nel primo caso si tratta di classi che contengono metodi principalmente statici, sono già contenute all'interno di librerie o vengono create all'occorrenza per svolgere compiti che possono essere utili in moltissime situazioni, quindi vengono riposte in quelle librerie per evitare di doverne ricreare il codice ogni qual volta ce ne sia bisogno. Riguardo ai JavaBeans, sono anch'essi delle classi Java che però vanno implementate seguendo delle direttive, di cui le principali sono che devono avere almeno un costruttore di default (cioè un costruttore senza parametri) e che tutte le proprietà devono poter essere accedute solo attraverso metodi di tipo get, set e is che devono seguire una convenzione riguardo al modo in cui vengono denominati.

Per l'utilizzo delle JSP, sono state create col tempo due tipi diversi di architetture:

- Architettura JSP Model 1, prevede che sia la sola pagina JSP a ricevere le richieste, elaborarle, ricavare le informazioni necessarie (ad es. da un database) e presentare il risultato di tutto ciò al client.
- Architetture JSP Model 2, prevede che la richiesta sia gestita da una Servlet di controllo (Controller Servlet), questa la elabora e determina quale sia la pagina JSP che deve essere visualizzata successivamente, prima di

indicare al client quale pagina visualizzare, effettua o delega ad altre classi la logica necessaria a recuperare le informazioni richieste dal client.

La differenza fra le due architetture sta nel fatto che nel primo modello tutto è gestito da un solo tipo di oggetto, cioè la JSP, il che però può essere dannoso nel caso in cui si voglia cambiare la grafica all'interno della pagina, a causa dell'inevitabile uso di scriptlet (cioè vero e proprio codice scritto in Java) all'interno di un file che è per il resto tutto HTML o custom tags.

Il punto di forza del secondo modello è nella sua divisione fra le diverse logiche, permettendo la modifica di determinati comportamenti o anche solo della grafica di presentazione, senza che si debba mettere mano a tutto il codice, con il conseguente rischio di aggiungere dei nuovi errori nella fase di modifica. Riguardo al secondo modello, si fa spesso riferimento alla separazione fra le logiche, come a un pattern *Model-View-Controller* (MVC), di cui di seguito si riporta una panoramica.

### 2.4.1 Il Pattern Model-View-Controller (MVC)

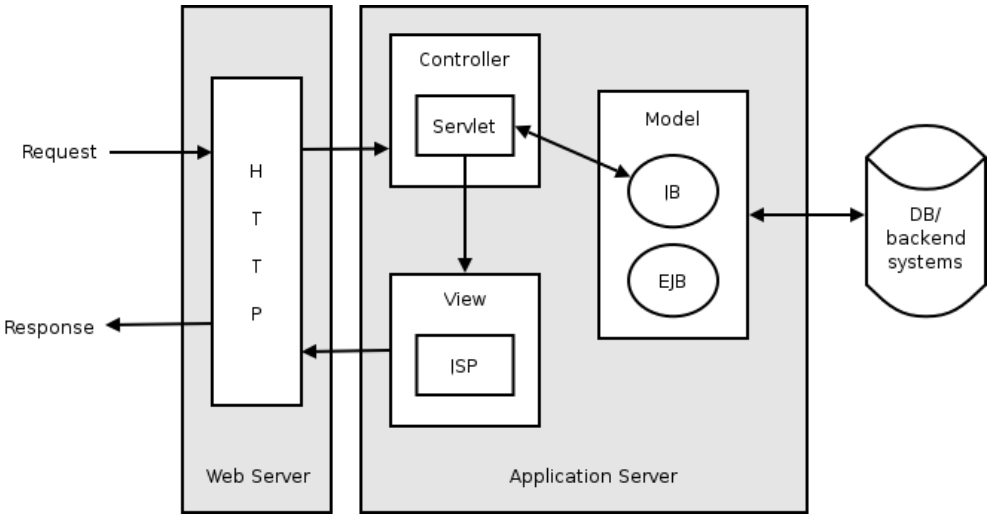



Figura 2.1 Pattern Model-View-Controller (MVC) [Dongilli, 2004]

Come appena detto, il pattern MCV è un'implementazione dell'architettura JSP Model 2, quindi possiamo trovare in esso tre componenti fondamentali, ognuno dei quali è asservito al compito di gestire una determinata logica. Abbiamo rispettivamente un tipo di logica per ogni componente del pattern, quindi abbiamo una business logic, una presentation logic e una control logic. Il vantaggio di questa divisione in più logiche, permette la creazione di un codice di più facile realizzazione, manutenzione e riutilizzazione.

Di seguito si riportano i tre componenti del pattern MVC con relativa descrizione:

- Controller, si occupa di intercettare tutte le richieste che pervengono alla web application; da esse ricava i dati e i parametri che deve tradurre in specifiche operazioni della business logic da effettuare, ed infine deve decidere la

successiva vista da presentare all'utente al termine dell'elaborazione della richiesta;

- Model, può essere rappresentato da un database o da un JavaBeans; rappresenta ciò che viene elaborato e successivamente presentato all'utente, si occupa della conoscenza del dominio dell'applicazione;
- View, è la parte dell'applicazione addetta alla visualizzazione delle informazioni di cui l'utente aveva fatto richiesta. In un'applicazione web è semplicemente la pagina a cui l'utente fa accesso al seguito di una richiesta.

### 2.4.2 Selezione delle metodologie realizzative

Dopo aver vagliato le varie possibilità e le varie architetture, studiandone pregi e difetti, si è scelto di realizzare la web application seguendo l'architettura JSP Model 2 aderendo al pattern Model-View-Controller, e si è scelto di farlo utilizzando un framework già esistente: il framework Jakarta Struts.
Per lo sviluppo vero e proprio del codice si è scelto un ambiente di sviluppo per la programmazione in Java e per la realizzazione di applicazione web, la scelta è quindi caduta su Eclipse, in particolare la sua versione 3.2, un IDE (Integrated Development Environment) Java sviluppato dall'IBM, un ambiente di sviluppo molto diffuso, gratuito e distribuito sotto licenza open-source. Una delle sue particolarità è anche quella di essere espandibile attraverso un'architettura a plug-in, garantendo così la necessaria

flessibilità e versatilità, per poter essere utilizzato anche in situazioni non previste dagli sviluppatori. E' stato scelto, non a caso, uno dei suoi plug-in, Exadel Studio, nella versione 4.0. Questo plug-in, oltre a permettere una più facile realizzazione di applicazioni web sviluppate utilizzando Struts, fornisce anche al suo interno una versione del Server Apache Tomcat, o in alternativa, permette un facile deploy (caricamento) su un Servlet Container a scelta del programmatore.

# CAPITOLO III

# IL FRAMEWORK JAKARTA STRUTS

## 3.1 Definizione di Framework

Prima di addentrarci nella descrizione di Struts, si ritiene opportuno dare una definizione a grandi linee di cos'è un framework.

Nella sua accezione più semplice, un framework è una serie di classi e interfacce che cooperano per risolvere un determinato tipo di problema software. Un framework ha le seguenti caratteristiche:

- Comprende molte classi e componenti, ciascuna delle quali può fornire un'astrazione di un particolare concetto;
- Definisce il modo in cui queste astrazioni collaborano per risolvere un problema;
- I comportamenti di un framework sono riutilizzabili;
- Organizza i pattern a un livello più elevato.

Un buon framework dovrebbe fornire un comportamento generico di cui possano far uso molte diversi tipi di applicazioni.

Esistono molte interpretazioni di ciò che costituisce un framework. Alcuni potrebbero considerare come framework le classi e le interfacce fornite dal linguaggio Java, ma in realtà si tratta piuttosto di una libreria. Esiste una differenza sottile, ma molto importante fra una libreria e un framework. Una libreria

software contiene funzioni o routine a cui la nostra applicazione può fare delle invocazioni. Un framework, invece, fornisce componenti generici che collaborano tra loro, i quali possono essere estesi dalla nostra applicazione per fornire una serie particolare di funzioni. I punti in cui il framework può essere esteso sono detti *punti di estensione*. Ci si riferisce spesso ai framework come a delle "librerie alla rovescia" per il modo alternativo in cui essi operano. Di seguito si riporta un'immagine che illustra le sottili differenze tra framework e libreria [Cavaness, 2003].

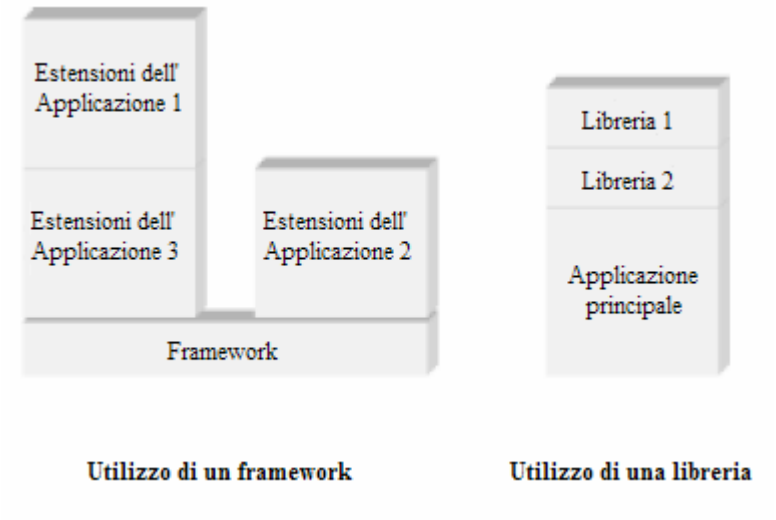



Figura 3.1 Confronto fra framework e libreria [Cavaness, 2003]

## 3.2 Il Framework Jakarta Struts

Struts è un framework open-source per lo sviluppo di applicazioni web, facente parte del progetto Jakarta della fondazione Apache, per la realizzazione di applicazioni Web sulla piattaforma J2EE della SUN, sviluppato per incoraggiare l'utilizzo di un tipo di architettura che si basa sul pattern MVC (model-view-control).

In realtà tale framework solo inizialmente è stato inquadrato come un sottoprogetto di Jakarta, ma in breve, dato il suo sviluppo, è diventato un progetto a sé stante.

Esso viene definito "*MVC web application framework*", in quanto è composto da un insieme di classi ed interfacce, che costituiscono l'infrastruttura per costruire web application pienamente aderenti alla piattaforma J2EE, conformi al design pattern MVC.

I componenti fondamentali di Struts sono:

- *ActionServlet*: è la servlet di controllo centralizzata che gestisce tutte le richieste dell'applicazione;
- *struts-config.xml*: è il file XML di configurazione di tutta l'applicazione. In questo file vengono definiti gli elementi dell'applicazione e le loro associazioni;
- *Action*: le Action sono le classi alle quali la ActionServlet delega l'elaborazione della richiesta;

- *ActionMapping*: contiene gli oggetti associati ad una Action nello struts-config.xml, come ad esempio gli ActionForward, le proprietà, ecc.;
- *ActionForm*: gli ActionForm sono classi contenitori di dati. Vengono popolati automaticamente dal framework con i dati contenuti nelle request http;
- *ActionForward*: contengono i path ai quali la servlet di Struts inoltra il flusso elaborativo in base alla logica dell'applicazione;
- *Custom-tags*: Struts fornisce una serie di librerie di tag per assolvere a molti dei più comuni compiti delle pagine JSP, per cercare di evitare o quanto meno di ridurre il più possibile l'utilizzo di scriptlet.

La ActionServlet è la servlet di controllo di Struts. Essa gestisce tutte le richieste client e smista il flusso applicativo in base alla logica configurata. Si potrebbe definire come la "spina dorsale" di una applicazione costruita su Struts.

Tutta la configurazione dell'applicazione è contenuta nel file *struts-config.xml*.

Questo file XML viene letto in fase di start-up dell'applicazione dalla ActionServlet e definisce le associazioni tra i vari elementi di Struts. Nello struts-config.xml, come vedremo più dettagliatamente in seguito, sono ad esempio definite le associazioni tra i path delle richieste http e le classi Action associate alla richieste stesse, le associazioni tra le Action e gli ActionForm, che vengono automaticamente popolati dal

framework con i dati della richiesta ad essi associata e passati in input alla Action. Contiene inoltre l'associazione tra la Action e le ActionForward , ovvero i path configurati nello strutsconfig.xml ai quali la ActionServlet redirigerà il flusso applicativo al termine della elaborazione della Action.

A tutto ciò va, ovviamente, aggiunto il file *web-xml*, che non è specifico di Struts, ma è utilizzato in tutte le applicazioni Web che fanno uso delle Servlet. In esso viene specificato al server qual è la servlet controller dell'applicazione. [Festa, 2005]

## 3.3 L'implementazione del pattern Model-View-Controller in Struts

Nella figura 3.2 è raffigurato il Flow Diagram di Struts:

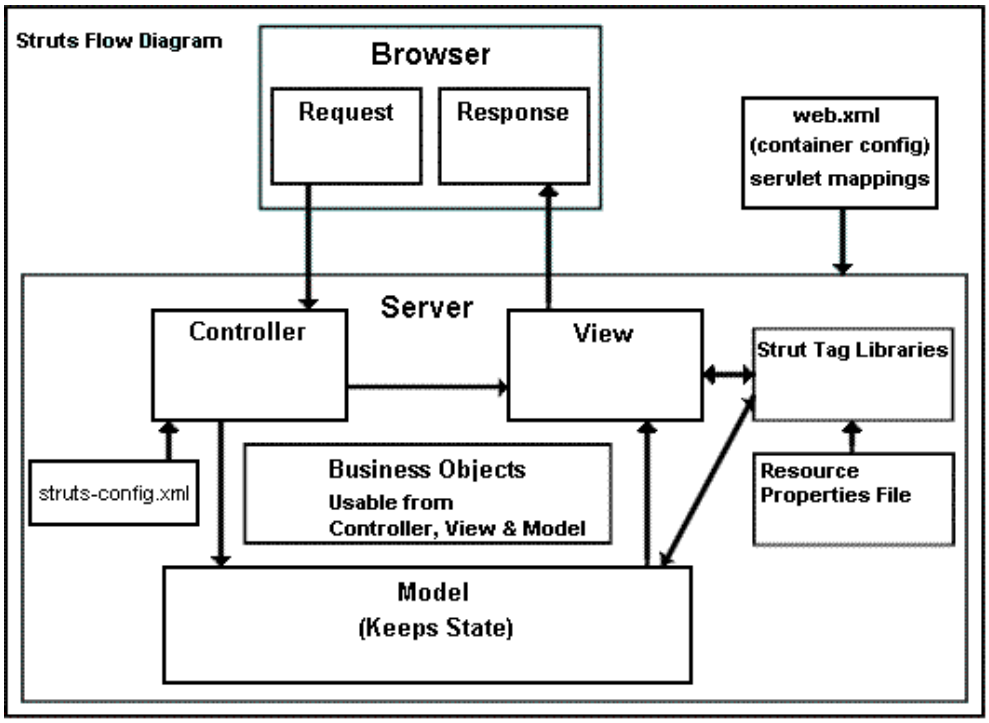



Figura 3.2 Flow Diagram di Struts [JSP Insider, 2002]

Analizziamo più nel dettaglio la sequenza di operazioni scatenate da una richiesta HTTP:

1. Il client invia una richiesta http tramite il proprio browser;
2. La richiesta viene ricevuta dalla servlet di controllo di Struts che provvede a popolare l'ActionForm, se presente, associato alla richiesta con i dati della request e l'ActionMapping con gli oggetti relativi alla Action associata alla richiesta. Tutti i dati di configurazione sono stati letti in fase di start-up dell'applicazione dal file XML struts-config.xml;
3. La ActionServlet delega l'elaborazione della richiesta alla Action associata al path della richiesta passandole in input la request http, l'ActionForm e l'ActionMapping precedentemente valorizzati;
4. La Action si interfaccia con lo strato di business che implementa la logica applicativa. Al termine dell'elaborazione restituisce alla ActionServlet un ActionForward contenente l'informazione del path della vista da fornire all'utente;
5. La ActionServlet esegue il forward alla vista specificata nell'ActionForward.

Ovviamente il flusso di operazioni elencato non è completo ma fornisce una indicazione di base su come viene gestito il flusso elaborativo in un'applicazione sviluppata con Struts.

In realtà Struts non implementa interamente il pattern Model-View-Controller, esso è un *framework model-neutral*, ovvero

implementa esclusivamente i livelli di controller e view. Utilizzando Struts è possibile realizzare il livello di business logic in base alle proprie scelte.
Descriviamo più dettagliatamente i due livelli implementati da Struts, mettendo in evidenza tutti i componenti e le operazioni da questi effettuate. [Festa, 2005]

### 3.3.1 Componenti del Controller

Per un'applicazione costruita secondo l'architettura Model 2 delle JSP il controller è implementato per mezzo di una Servlet Java. In Struts, la servlet di controllo è la classe *org.apache.struts.Action.ActionServlet*, la quale estende la classe javax.servlet.http.HttpServlet. Tale Servlet riceve sia richieste di tipo Get che di tipo Post ed effettua i seguenti step:

1. In base alla richiesta invoca i metodi *doGet()* e *doPost()*, i quali, a loro volta, invocano il metodo *process()* della ActionServlet;
2. Nel metodo *process()* si ottiene l'istanza della classe *org.apache.struts.Action.RequestProcessor*, di cui si esegue il metodo *process()*;
3. Nel metodo *process()* del Request Processor viene eseguita l'elaborazione vera e propria.

Il RequestProcessor viene configurato all'interno del tag <controller> del file struts-config.xml ed è possibile usarne uno

proprio estendendolo e facendo l'override del metodo *processPreprocess()*; nel caso in cui non venga definito,
viene naturalmente usato quello di default.
Il RequestProcessor esegue i seguenti step:

1. Legge il file struts-config.xml per trovare un elemento <action> corrispondente al path della richiesta;
2. Una volta trovato l'elemento <action> corrispondente, verifica se è presente l'attributo name che corrisponde al nome dell'ActionForm configurato per la richiesta in elaborazione. In tal caso provvede a reperire una istanza dell'ActionForm e a popolarne gli attributi con i valori presenti nella request http, facendo una corrispondenza tra nome parametro e nome attributo;
3. Se nell'elemento <action> è presente l'attributo validate al valore true chiama il metodo *validate()* dell'ActionForm per il controllo dei dati forniti dalla request;
4. Se il controllo è ok a questo punto il RequestProcessor chiama il metodo *execute()* dell'Action delegandole l'elaborazione della richiesta.

Il metodo *execute()* dell'Action, al termine dell'elaborazione, restituisce un oggetto ActionForward che consente al RequestProcessor di inoltrare il flusso elaborativo all'oggetto successivo, sia essa una pagina JSP o un'altra Action.
La classe *org.apache.struts.Action* è l'elemento fondamentale del controller di Struts, essa contiene al proprio interno il metodo

*execute()*, all'interno del quale lo sviluppatore inserisce il proprio codice di elaborazione della richiesta per la funzione specifica.
Gli step principali eseguiti da una Action sono:

1. Acquisire i dati della request dal form;
2. Delegare l'elaborazione della business logic alle classi del Model;
3. Acquisire i risultati dell'elaborazione e prepararli per la vista da inviare all'utente mettendoli nello scope opportuno (se necessario);
4. Inoltrare il flusso elaborativo in base alla logica applicativa.

In pratica, la Action è il "ponte applicativo" tra il Controller ed il Model di un'applicazione Struts.
Altra classe molto importante è la *org.apache.struts.action.ActionForm*. I form, che estendono tale classe, sono, sostanzialmente, dei bean contenenti dati, che il framework popola con i dati della request, liberando di tale compito lo sviluppatore. Associando il form ad una Action, viene data la possibilità al metodo *execute()* di tale Action di avere a disposizione tutti i dati inseriti in un Form Html, con la possibilità di validarli prima che gli stessi giungano alla Action stessa tramite il metodo *validate()*; in alternativa, ad esempio nei casi in cui non si voglia validare il contenuto dei campi, la ActionForm può essere sostituita da un form dinamico, la *DynaActionForm*, tramite la dichiarazione di un form particolare all'interno del file struts-config.xml. Quando si configura questo

form bisogna dichiararlo di tipo *org.apache.struts.action.DynaActionForm* e bisogna inoltre aggiungere un campo di tipo property per ogni campo del Form HTML corrispondente. [Festa, 2005]

### 3.3.2 Componenti della View

La View ha due compiti fondamentali: presentare all'utente i risultati di una elaborazione eseguita e consentire all'utente l'immissione di dati da elaborare.

Per quanto riguarda la presentazione di dati all'utente si utilizzano i tag JSP per visualizzare i contenuti di JavaBeans oppure altri oggetti. Invece, per consentire l'immissione di dati all'utente si realizza un apposito Form HTML nella pagina JSP e si estende la classe ActionForm per validarli ed acquisirli.

Per la costruzione delle view delle applicazioni, Struts mette a disposizione alcune librerie di tag, che svolgono i compiti più frequenti. Tra queste le più utilizzate sono:

- Html tag, per la generazione degli elementi html;
- Bean tag, per accedere a proprietà di JavaBeans e per creare istanze di bean;
- Logic tag, per implementare logica condizionale, iterazioni e controllo di flusso.

In aggiunta a queste librerie specifiche di Struts è possibile utilizzare la JSP Standard Tag Library (JSTL), nata per

permettere agli sviluppatori di trasferire la proprie pagine JSP da un container ad un altro senza modificare le pagine stesse.
Tale libreria, allo stato attuale, non permette di eliminare le tre librerie viste prima, ma solo alcuni tag.
L'utilizzo dei custom-tag è altamente consigliato, in quanto essi non prevedono l'introduzione di logica all'interno delle pagine, a differenza degli scriptlet. Gli sviluppatori suggeriscono di utilizzare esclusivamente tag, al fine di rispettare il più possibile il design pattern MVC. Ovviamente si è seguita tale direttiva nella realizzazione dell'applicazione oggetto della tesi.
Altro aspetto importante della View è rappresentato dalla possibilità di raggruppare tutti i messaggi in un resource bundle, con estensione *.properties*, ottenuto associando il messaggio ad un identificativo unico. Così facendo all'interno delle pagine JSP si troveranno solo gli identificativi dei messaggi. Questa possibilità comporta notevoli vantaggi in termini di manutenzione dell'applicazione, in quanto, se dovesse sorgere in futuro la necessità di modificare un messaggio presente in più pagine, esso dovrà essere cambiato solo nel file delle proprietà e non in ogni pagina, ed in termini di internazionalizzazione, come si vedrà in seguito. [Festa, 2005]

## 3.4 Principali vantaggi nell'uso di Struts

Da quanto appena esposto è già possibile evidenziare alcune delle caratteristiche di un'applicazione sviluppata con Struts e alcuni vantaggi conseguenti al suo utilizzo:

- *Modularità e Riusabilità* : i diversi ruoli dell'applicazione sono affidati a diversi componenti. Ciò consente di sviluppare codice modulare e più facilmente riutilizzabile;
- *Manutenibilità*: L'applicazione è costituita da livelli logici ben distinti. Una modifica in uno dei livelli non comporta modifiche negli altri. Ad esempio una modifica ad una pagina JSP non ha impatto sulla logica di controllo o sulla logica di business, cosa che avviene nel JSP Model 1;
- *Rapidità di sviluppo*: A differenza di quanto avviene utilizzando il JSP Model 1, è possibile sviluppare in parallelo le varie parti dell'applicazione, View (JSP/HTML) e logica applicativa (Java), sfruttando al meglio le conoscenze dei componenti del team di sviluppo. Si possono utilizzare sviluppatori meno esperti e anche con poche conoscenze di Java per la realizzazione delle View, permettendo agli sviluppatori Java più esperti di concentrarsi sulla realizzazione della logica applicativa [Festa, 2005].

# CAPITOLO IV

# PROGETTAZIONE

## 4.1 Introduzione

Prima di addentraci nella vera e propria progettazione dell'applicazione, mediante la creazione di grafici e modelli appositi, è conveniente dapprima illustrare l'applicazione, il suo contesto e i requisiti che sono richiesti ad essa.

Lo scopo che ci si è prefissati è stato quello di realizzare un'applicazione web che fosse capace di immagazzinare file, di qualsiasi natura e anche di dimensioni abbastanza corpose, in modo che fosse flessibile e fosse utilizzabile in molti campi di applicazione, non solo in quello su cui ci si è focalizzati, come vedremo in seguito.

L'applicazione deve essere in grado, a seconda di chi la utilizzi, di dare all'utente la possibilità di ricercare cosa è stato caricato in essa, e volendo, anche di poterla scaricare; in più, naturalmente, bisogna dare possibilità, a chi ne ha i diritti, di poter caricare nuovo materiale nell'applicazione o di eliminarlo. Il resto dell'applicazione è naturalmente rivolto agli utenti, il cui compito è quello di amministrare l'applicazione; per loro sono previste delle funzioni particolari, quali la gestione dell'utente, e più in generale gli amministratori hanno il potere di fare quello che fanno gli altri tipi di utenti.

Infine, si è deciso di specializzare l'applicazione dandole la connotazione di un "deposito di documenti", da qui il nome Docs Storer (Memorizzatore di Documenti).

## 4.2 Use Cases

Il grafico degli Use Cases (o Casi d'Uso) è il primo grafico che viene creato in fase di progettazione di qualsiasi applicazione.
La sua utilità è nel fatto che rappresenta, in modo rapido, sintetico e completo, quali siano le funzioni richieste all'applicazione, così che sia chiaro cosa l'applicazione debba fare, sia al progettista sia a chi ne ha richiesto la creazione.
Il grafico è davvero molto semplice, e consta di due tipi di oggetti:

- L'Attore;
- Il Caso d'Uso.

L'Attore è colui che fa uso di una determinata funzione; vanno specificati tutti i tipi di Attori, perché successivamente questi ruoli verranno implementati ed è quindi necessario specificarli fin dall'inizio.
Il Caso d'Uso rappresenta invece la funzionalità che un Attore può o meno fare. In questo caso si possono o meno riportare tutte le funzionalità sin dall'inizio, in quanto è possibile fare altri Use Cases in cui, di volta in volta, si può entrare nello specifico di alcune funzioni per spiegare come vengono realizzate e quindi anche da quali altre funzioni sono formate.

Fra di essi ci possono essere due tipi di relazione:

- Relazione d'Uso;
- Relazione d'Estensione.

Nella nostra applicazione abbiamo fatto uso di un solo tipo di relazione, quella d'Uso, in quanto nessun Caso era estensione di un altro, ma si verifica solo che alcuni Casi fanno uso di altri.

Si riporta di seguito il grafico degli Use Cases:

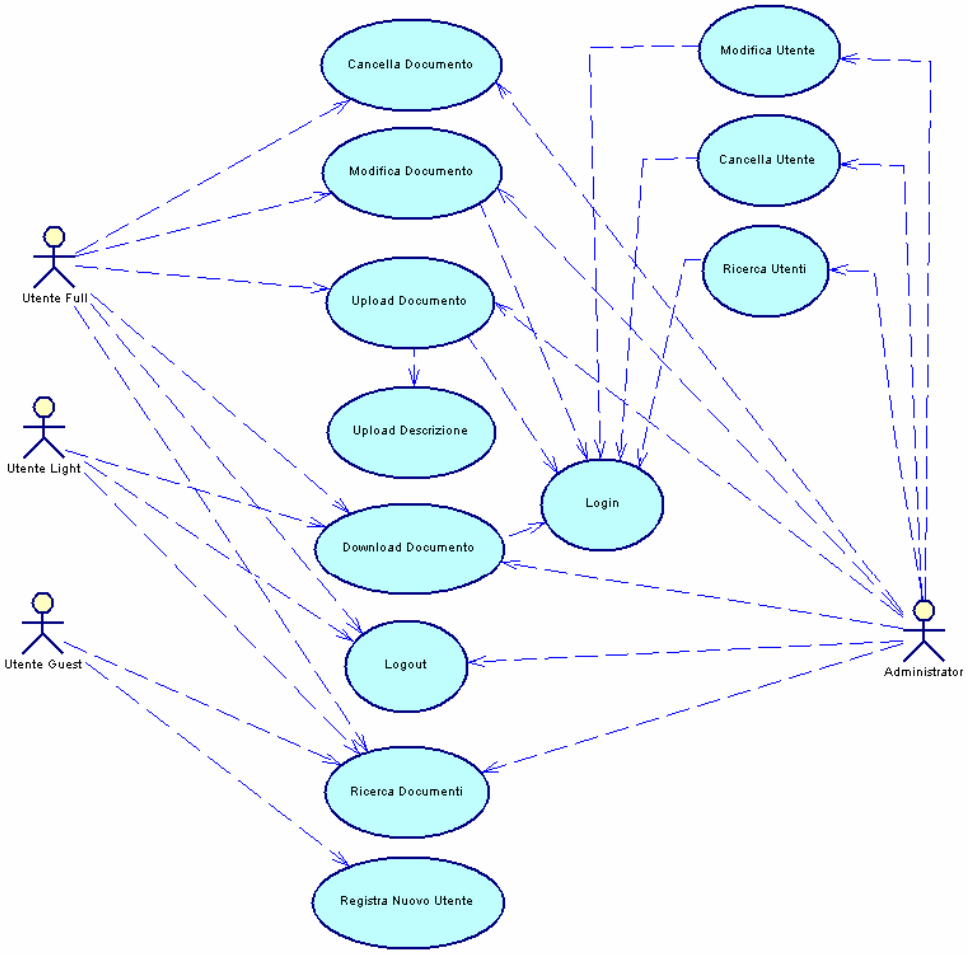



Figura 4.1 Grafico degli Use Cases

Prendiamo ora in esame quali sono gli Attori presenti nella nostra applicazione:

- Utente Guest;
- Utente Light;
- Utente Full;
- Administrator.

In seguito, dopo aver enunciato quali sono le funzionalità disponibili, si darà spiegazione dei privilegi e delle funzioni messe a disposizione per ogni tipo di Attore.

Per quanto riguarda invece i Casi d'Uso, essi sono:

- Login, serve a fornire le proprie credenziali all'applicazione per poter avere determinati privilegi;
- Logout, serve ad informare l'applicazione che da quel momento in poi non si vuole più far uso di determinati privilegi fino al successivo Login;
- Ricerca Documenti, serve ad effettuare delle ricerche all'interno dell'applicazione per trovare determinati Documenti;
- Download Documento, permette lo scaricamento di un Documento (ovvero del file associato ad esso) sul client da cui proviene la richiesta;
- Upload Documento, permette ad un utente di aggiungere un nuovo Documento all'applicazione;
- Upload Descrizione, permette di aggiungere la Descrizione al Documento che si sta inviando;

- Modifica Documento, permette la modifica di un Documento, sia che si tratti della descrizione o del file ad esso associato;
- Cancella Documento, cancellazione del Documento dall'applicazione, semplicemente del file ad esso associato o anche della descrizione del documento stesso;
- Registra Nuovo Utente, permette ad un utente non registrato di fare una richiesta di Registrazione all'Amministratore (o Amministratori, nel caso ce ne fosse più di uno);
- Ricerca Utenti, permette di fare una ricerca sugli Utenti registrati o in fase di Registrazione;
- Modifica Utente, permette di modificare i privilegi di un Utente registrato o di accettare la richiesta di Registrazione di un nuovo Utente;
- Cancella Utente, cancellazione di un Utente dall'applicazione, previa cancellazione dei Documenti caricati da lui nell'applicazione.

Di seguito si riportano le associazioni fra Casi d'Uso e Attori:

- Agli Utenti Guest è data solo la possibilità di fare una ricerca sui Documenti e di iscriversi al sito per avere i diritti per il resto delle funzionalità;
- Agli Utenti Light sono concessi gli stessi diritti degli Utenti Guest, in più loro oltre a poter ricercare i Documenti, hanno anche attiva la funzionalità di scaricarne il File ad essi associato;

- Agli Utenti Full sono concessi i diritti di cui sopra, in più loro hanno i privilegi di inserimento di Nuovi Documenti all'interno dell'applicazione ed inoltre possono cancellare da essa solo i Documenti da loro inseriti;
- Agli Administrator, cioè gli amministratori dell'applicazione, sono concessi tutti i diritti finora esposti; a loro sono anche rivolte le funzionalità di gestione, rimozione e accettazione di Utenti; a differenza degli Utenti Full, gli Administrator possono cancellare i Documenti di qualsiasi Utente.

## 4.3 Progettazione del Modello Concettuale

### 4.3.1 Il Modello Concettuale

Il Modello Concettuale, conosciuto anche come modello E-R (Entity-Relationship), viene creato in fase di progettazione e serve ad indicare quali sono i dati o le informazioni che cn amicil'applicazione deve in ogni caso mantenere, anche in caso di riavvio del server o semplicemente dell'applicazione stessa.
Di seguito se ne riporta il grafico:

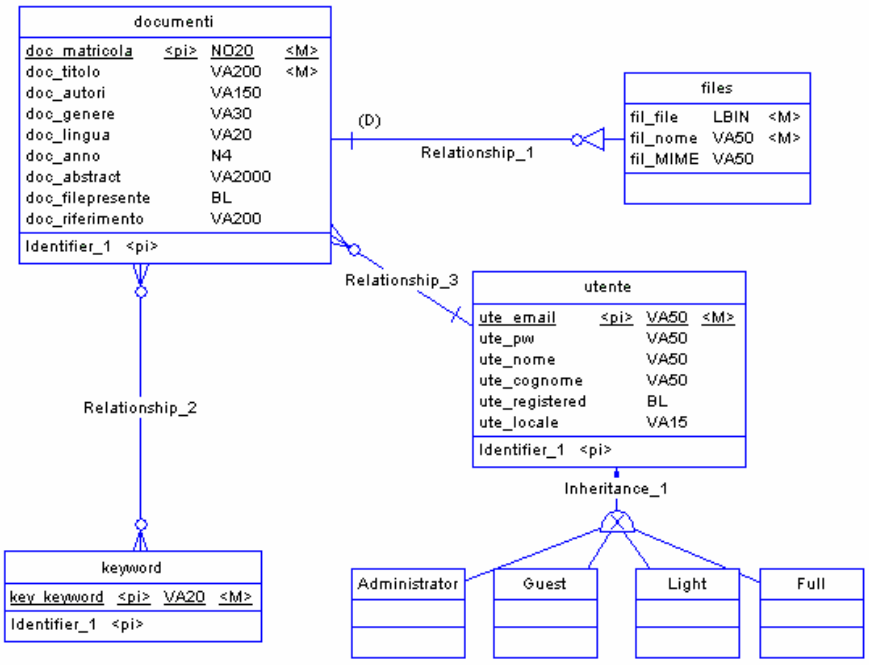
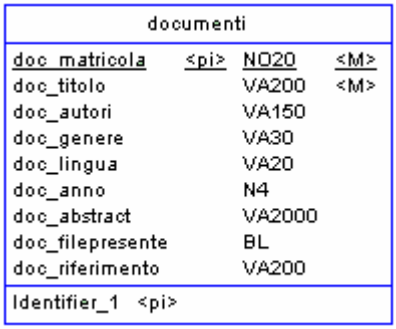



Figura 4.2 Modello Concettuale

### 4.3.2 Dizionario dei Dati

Di seguito si riporta la descrizione di ogni entità presente nel grafico precedentemente esposto:

- documenti

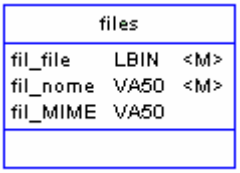



Contiene tutte le informazioni relative ad un Documento:

- doc_matricola: numero che identifica il Documento;
- doc_titolo: titolo del Documento;
- doc_autori: autore/i del Documento;
- doc_genere: genere del Documento (per ora non è usato nell'applicazione);
- doc_lingua: lingua con cui è stato scritto il Documento;
- doc_anno: anno di pubblicazione del Documento (dal 1980 in poi);
- doc_abstract: l'abstract, cioè una piccola descrizione del Documento;
- doc_filepresente: indica se è associato un file ad un determinato Documento;

- doc_riferimento: indica i riferimenti di un determinato Documento.

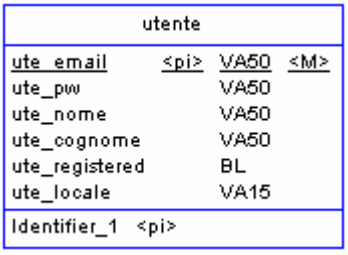
- files




Contiene le informazioni relative ai file inseriti nell'applicazione:

- fil_file: è il file vero e proprio;
- fil_nome: è il nome del file;
- fil_MIME: è il tipo MIME del file (per ora non è usato nell'applicazione).

- utente

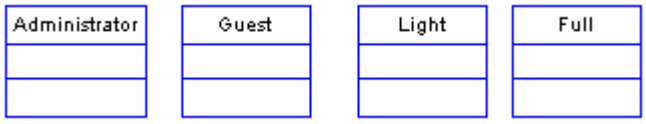



Contiene le informazioni relative agli utenti iscritti e in fase di iscrizione:

- ute_email: email con cui l'Utente si registra, è anche il suo identificativo da utilizzare nel Login;
- ute_pw: Password per il Login;
- ute_nome: Nome proprio dell'Utente;
- ute_cognome: Cognome dell'Utente;

- ute_registered: indica se l'account è stato attivato;
- ute_locale: indica il primo Locale (lingua) con cui l'Utente si registra;

➢ tipi di Utente




Sono le quattro entità che rappresentano i 4 tipi di Attori del grafico degli Use Cases.

Non viene presa in considerazione la tabella "keyword" perché non è stata più utilizzata in seguito.

## 4.4 Dal Modello Concettuale al Modello Logico

### 4.4.1 Ristrutturazione

Prima di eseguire l'opera di traduzione dal Modello Concettuale a quello Logico, bisogna prima effettuare la Ristrutturazione; essa consta di varie operazioni:

- Analisi di minimalità;
- Analisi di leggibilità;
- Analisi di normalità.

Avendo progettato il Modello Concettuale con PowerDesigner, si ottiene che, senza ulteriori operazioni, le tre analisi danno risultato positivo e quindi si può proseguire con la traduzione da un modello all'altro; nello specifico il Modello Concettuale non presenta ridondanze, o sono già state tradotte perché non è

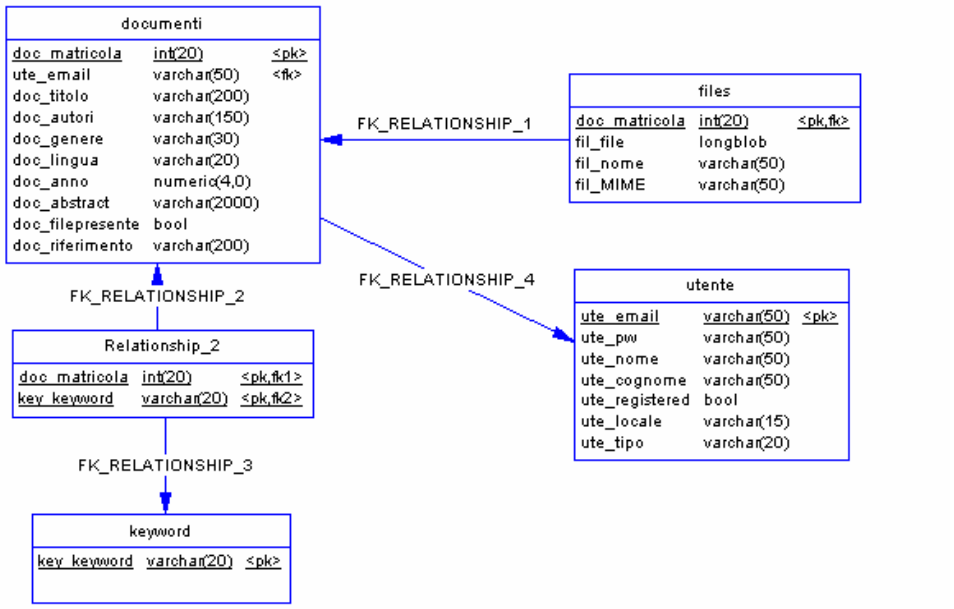
possibile averle nel software di CASE utilizzato, ed è già normalizzato.

### 4.4.2 La Traduzione

L'opera di traduzione viene fatta automaticamente dal software usato, però è sempre meglio controllare ciò che viene generato, in quanto possono sempre esserci incongruenze con quello che ci si aspettava di avere, un po' per qualche errore che può capitare durante la progettazione del Modello Concettuale e un po' per errori di utilizzo del programma, che esegue solo i nostri comandi, corretti o sbagliati che siano.
Come ci si aspettava, le associazioni (relationship) sono state correttamente tradotte nelle Integrità Referenziali aspettate e la relazione di generalizzazione/specializzazione mutuamente esclusiva che riguarda la tabella "utente", viene correttamente tradotta in un attributo della suddetta tabella, che indica di che tipo di Utente si tratta.

### 4.4.3 Il Modello Logico

Si riporta di seguito il grafico del Modello Logico di cui si è discusso finora:



# CAPITOLO V

# IMPLEMENTAZIONE DELL'APPLICAZIONE WEB

## 5.1 Introduzione

Questo capitolo è dedicato all'esposizione di come sono state implementate le idee e i progetti prima esposti; per quanto riguarda invece le procedure di configurazione e inizializzazione dell'applicazione sull'host su cui essa girerà, si rimanda al capitolo VI.

Tenendo conto della semplicità con cui si è voluta creare l'applicazione, il progetto è formato da un solo modulo software progettato per far fronte a tutte le funzionalità richieste in fase di progettazione.

Si riporta inoltre che tutti i grafici e i file di configurazione sono stati creati mediante Eclipse ed Exadel Studio, con cui è stata sviluppata l'applicazione.

## 5.2 Configurazione dell'Applicazione

Il framework Struts utilizza due diversi tipi di file di configurazione, tra loro correlati, entrambi in formato XML:

1. il file **web.xml**, detto anche *descrittore di deploy*, che non è specifico di Struts, ma è necessario per tutte le web application che utilizzano servlet. Nonostante ciò, sono previste in tale file delle informazioni specifiche per Struts. Esso viene utilizzato dal web container, quando viene avviato, per configurare e caricare le web application. Il formato di tale file si basa su un *Document Type Definition(DTD)*, che definisce i tag che possono essere lecitamente utilizzati. Si procederà alla descrizione solo dei tag utilizzati nell'applicazione realizzata, rimandando per ulteriori informazioni alle specifiche delle Servlet Java (http://java.sun.com/dtd/index.html).

   Si riporta il file web.xml realizzato per l'applicazione in esame. Il plug-in di Eclipse, Exadel Studio, fornisce la possibilità di operare sul file in una modalità ad albero(Fig. 5.1), tramite un'interfaccia amichevole oltre che nella classica versione testuale (Fig. 5.2).
   Si procede alla descrizione delle varie voci presenti nella modalità testuale:

   - le prime righe presentano informazioni sulla versione, sulla codifica e sul tipo di documento DTD utilizzato;

- all'interno dei tag <servlet></servlet> viene dapprima specificata la ActionServlet, che riceverà tutte le request in entrata all'applicazione, poi vengono specificati i file di configurazione specifici dell'applicazione realizzata. E' inoltre presente il tag <load-on-startup></load-on-startup>, in base al quale il container istanzia la ActionServlet allo startup dell'applicazione;
- mediante il tag <servlet-mapping></servlet-mapping> si specifica che tutte le richieste con path terminante in .do vengono mappate sulla ActionServlet, ovvero quando c'è un URL terminante in .do viene riconosciuto dal container e la richiesta viene inviata alla Servlet di controllo;
- nel tag <welcome-file-list></welcome-file-list> viene indicata la pagina iniziale dell'applicazione;
- all'interno dei <taglib></taglib> vengono specificate le librerie di tag JSP che verranno utilizzate nel progetto [Festa, 2005];
- nel tag <error-page> si indica al server quale pagina deve inviare al client quando il server da un determinato codice di errore; in questo caso il codice di errore è 500, cioè quando si prova ad accedere alle pagine protette all'interno di */protected* il server redireziona il client sulla home page;

- nel tag <security-constraint></security-constraint> viene specificato che qualunque richiesta di tipo Get o Post sui file all'interno della cartella */protected*, dove sono contenute alcune JSP, non deve essere soddisfatta. In tal modo non sarà possibile accedere direttamente alle pagine presenti in tale cartella, ma solo utilizzando i link presenti nel menu oppure quando è l'applicazione a fare un forward della pagina al client;
- infine per evitare che il server generi errori in fase di lettura del file di configurazione, va anche riempito il campo <security-role> in cui vengono indicati i tipi di utenti che possono accedere alle risorse protette, nel nostro caso c'è un fittizio tipo di utente (*nessuno*) in quanto a nessuno è permesso l'accesso diretto a quelle risorse, quelle contenute nella cartella protected.

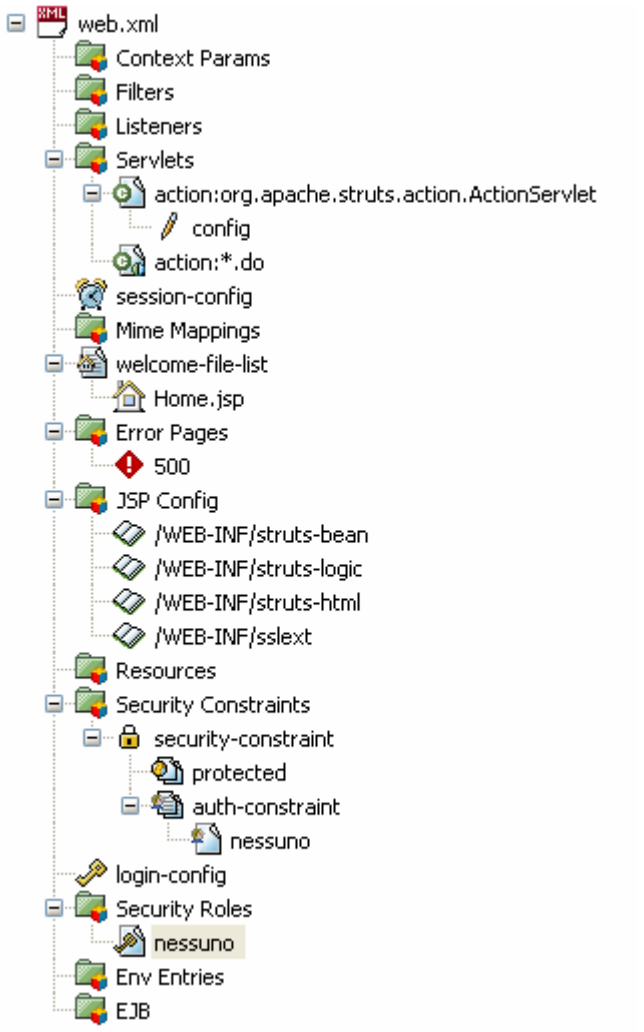



Figura 5.1 - File di configurazione *web.xml* in modalità ad albero

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN"
                        "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
 <display-name>DocsStorer</display-name>
 <servlet>
  <servlet-name>action</servlet-name>
  <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
  <init-param>
   <param-name>config</param-name>
   <param-value>/WEB-INF/struts-config.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
 </servlet>
```

Figura 5.2 - File di configurazione *web.xml* in modalità testuale

```
<servlet-mapping>
  <servlet-name>action</servlet-name>
  <url-pattern>*.do</url-pattern>
 </servlet-mapping>
 <welcome-file-list>
  <welcome-file>Home.jsp</welcome-file>
 </welcome-file-list>
 <error-page>
  <error-code>500</error-code>
  <location>/Home.jsp</location>
 </error-page>
 <taglib>
  <taglib-uri>/WEB-INF/struts-bean</taglib-uri>
  <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
 </taglib>
 <taglib>
  <taglib-uri>/WEB-INF/struts-logic</taglib-uri>
  <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
 </taglib>
 <taglib>
  <taglib-uri>/WEB-INF/struts-html</taglib-uri>
  <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
 </taglib>
 <taglib>
  <taglib-uri>/WEB-INF/sslext</taglib-uri>
  <taglib-location>/WEB-INF/sslext.tld</taglib-location>
 </taglib>
 <security-constraint>
  <web-resource-collection>
   <web-resource-name>protected</web-resource-name>
   <url-pattern>/protected/*</url-pattern>
   <http-method>GET</http-method>
   <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint>
   <role-name>nessuno</role-name>
  </auth-constraint>
 </security-constraint>
 <login-config>
  <auth-method>BASIC</auth-method>
 </login-config>
 <security-role>
  <role-name>nessuno</role-name>
 </security-role>
</web-app>
```

Figura 5.2 - File di configurazione *web.xml* in modalità testuale (continua)

2. il file **struts-config.xml**, che permette di specificare in maniera dichiarativa il comportamento dei componenti del framework, evitando che tale comportamento debba essere incluso nel codice dell'applicazione. L'utilizzo di Exadel Studio ci fornisce la possibilità di creare e modificare il file in tre modalità distinte: modalità ad albero (Fig. 5.3), modalità grafica (Fig. 5.4) e modalità testuale (Fig. 5.5) classica di Struts [Festa, 2005].

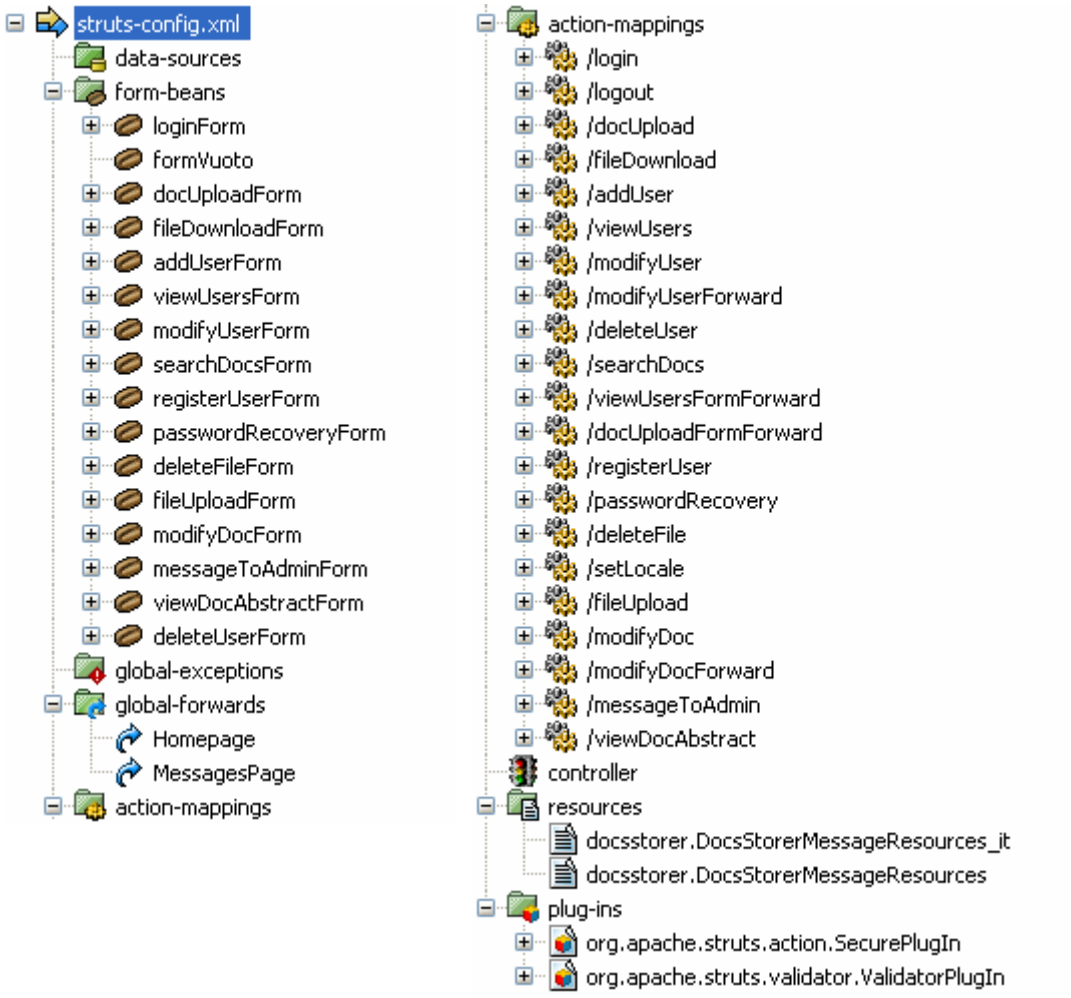



Figura 5.3 - struts-config.xml in modalità ad albero

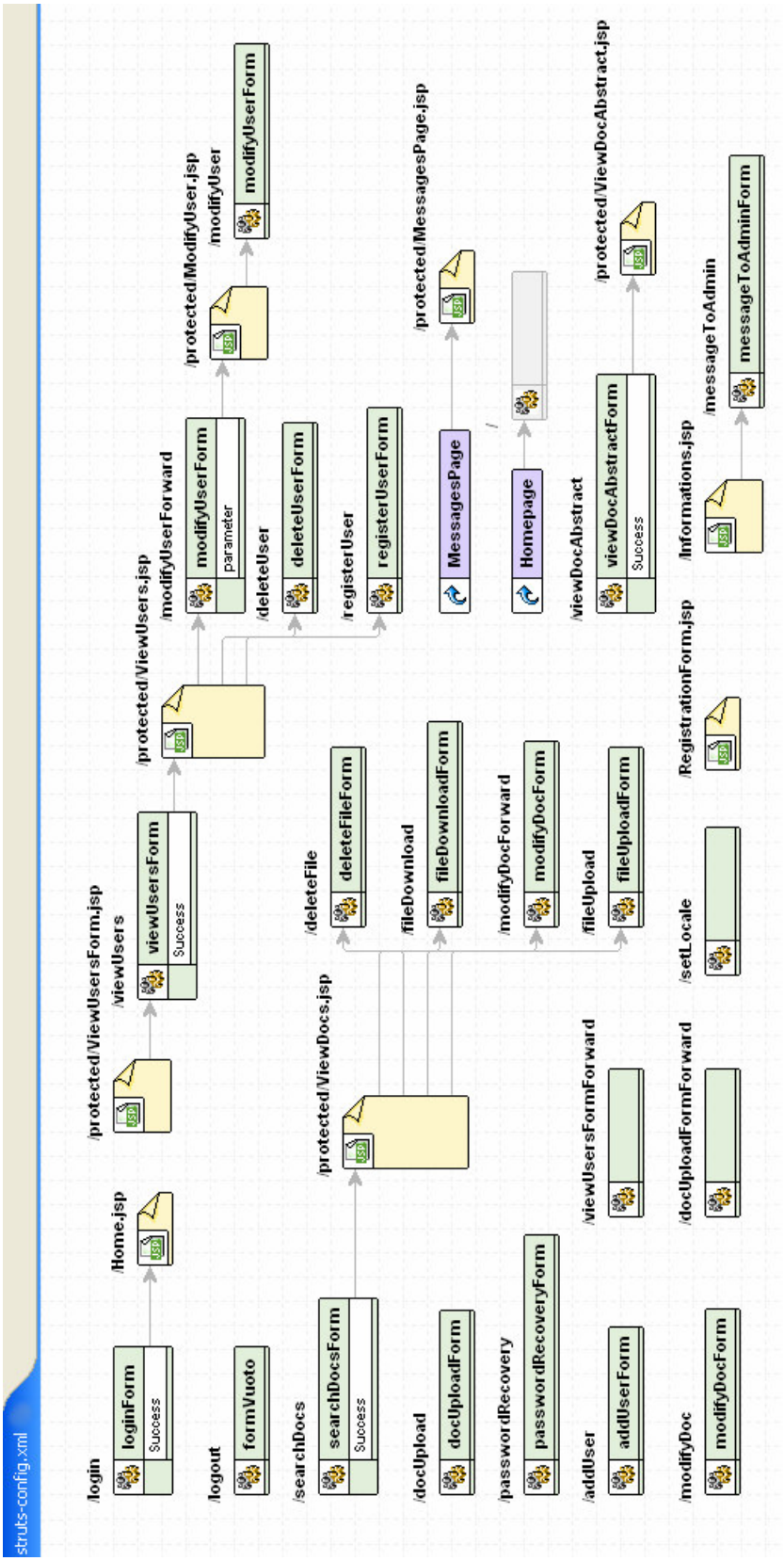



Figura 5.4 - struts-config.xml in modalità grafica

```
<struts-config>
 <data-sources/>
 <form-beans>
  <form-bean name="loginForm" type="org.apache.struts.validator.DynaValidatorForm">
   <form-property name="password" type="java.lang.String"/>
   <form-property name="email" type="java.lang.String"/>
  </form-bean>
  <form-bean name="formVuoto" type="org.apache.struts.action.DynaActionForm"/>
  <form-bean name="docUploadForm"
type="org.apache.struts.validator.DynaValidatorForm">
   <form-property name="file" type="org.apache.struts.upload.FormFile"/>
   <form-property name="lingua" type="java.lang.String"/>
   <form-property name="titolo" type="java.lang.String"/>
   <form-property name="anno" type="java.lang.Integer"/>
   <form-property name="docabstract" type="java.lang.String"/>
   <form-property name="autori" type="java.lang.String"/>
   <form-property name="owner" type="java.lang.String"/>
   <form-property name="riferimento" type="java.lang.String"/>
  </form-bean>
  <form-bean name="fileDownloadForm"
type="org.apache.struts.action.DynaActionForm">
   <form-property name="matricola" type="java.lang.Integer"/>
  </form-bean>
  <form-bean name="addUserForm" type="docsstorer.AddUserForm">
   <form-property name="email" type="java.lang.String"/>
   <form-property name="password2" type="java.lang.String"/>
   <form-property name="password" type="java.lang.String"/>
   <form-property name="nome" type="java.lang.String"/>
   <form-property name="cognome" type="java.lang.String"/>
  </form-bean>
  <form-bean name="viewUsersForm" type="docsstorer.ViewUsersForm">
   <form-property name="email" type="java.lang.String"/>
   <form-property initial="true" name="registered" type="java.lang.Boolean"/>
   <form-property name="ricercatutto" type="java.lang.Boolean"/>
   <form-property initial="false" name="ricercaperregistered" type="java.lang.Boolean"/>
   <form-property name="ricercapercognome" type="java.lang.Boolean"/>
   <form-property name="cognome" type="java.lang.String"/>
   <form-property initial="false" name="ricercapernome" type="java.lang.Boolean"/>
   <form-property name="nome" type="java.lang.String"/>
   <form-property name="ricercaperemail" type="java.lang.Boolean"/>
   <form-property initial="false" name="ricercapertipo" type="java.lang.Boolean"/>
   <form-property name="tipo" type="java.lang.String"/>
  </form-bean>
  <form-bean name="modifyUserForm" type="org.apache.struts.action.DynaActionForm">
   <form-property name="email" type="java.lang.String"/>
   <form-property name="tipo" type="java.lang.String"/>
  </form-bean>
  <form-bean name="searchDocsForm" type="docsstorer.SearchDocsForm">
   <form-property name="ricercatutto" type="java.lang.Boolean"/>
   <form-property name="matricola" type="java.lang.Integer"/>
   <form-property name="docabstract" type="java.lang.String"/>
   <form-property initial="null" name="anno" type="java.lang.Integer"/>
   <form-property name="ricercaperriferimento" type="java.lang.Boolean"/>
```

Figura 5.5 - struts-config.xml in modalità testuale

```
   <form-property name="ricercaperowner" type="java.lang.String"/>
   <form-property initial="false" name="ricercaperautori" type="java.lang.Boolean"/>
   <form-property name="ricercaperlingua" type="java.lang.Boolean"/>
   <form-property name="titolo" type="java.lang.String"/>
   <form-property name="owner" type="java.lang.String"/>
   <form-property name="ricercaperanno" type="java.lang.Boolean"/>
   <form-property initial="false" name="ricercaperdocabstract" type="java.lang.Boolean"/>
   <form-property name="lingua" type="java.lang.String"/>
   <form-property name="riferimento" type="java.lang.String"/>
   <form-property name="autori" type="java.lang.String"/>
   <form-property initial="false" name="ricercapertitolo" type="java.lang.Boolean"/>
  </form-bean>
  <form-bean name="registerUserForm" type="org.apache.struts.action.DynaActionForm">
   <form-property name="tipo" type="java.lang.String"/>
   <form-property name="email" type="java.lang.String"/>
  </form-bean>
  <form-bean name="passwordRecoveryForm"
type="org.apache.struts.validator.DynaValidatorForm">
   <form-property name="email" type="java.lang.String"/>
  </form-bean>
  <form-bean name="deleteFileForm" type="org.apache.struts.action.DynaActionForm">
   <form-property name="matricola" type="java.lang.Integer"/>
   <form-property name="tipo" type="java.lang.String"/>
  </form-bean>
  <form-bean name="fileUploadForm"
type="org.apache.struts.validator.DynaValidatorForm">
   <form-property name="matricola" type="java.lang.Integer"/>
   <form-property name="file" type="org.apache.struts.upload.FormFile"/>
  </form-bean>
  <form-bean name="modifyDocForm"
type="org.apache.struts.validator.DynaValidatorForm">
   <form-property name="titolo" type="java.lang.String"/>
   <form-property name="docabstract" type="java.lang.String"/>
   <form-property name="riferimento" type="java.lang.String"/>
   <form-property name="autori" type="java.lang.String"/>
   <form-property name="anno" type="java.lang.Integer"/>
   <form-property name="lingua" type="java.lang.String"/>
   <form-property name="matricola" type="java.lang.Integer"/>
  </form-bean>
  <form-bean name="messageToAdminForm"
type="org.apache.struts.validator.DynaValidatorForm">
   <form-property name="msg" type="java.lang.String"/>
  </form-bean>
  <form-bean name="viewDocAbstractForm"
type="org.apache.struts.action.DynaActionForm">
   <form-property name="matricola" type="java.lang.String"/>
   <form-property name="docabstract" type="java.lang.String"/>
  </form-bean>
  <form-bean name="deleteUserForm" type="org.apache.struts.action.DynaActionForm">
   <form-property name="email" type="java.lang.String"/>
  </form-bean>
 </form-beans>
 <global-exceptions/>
```

Figura 5.5 - struts-config.xml in modalità testuale (continua)

```
<global-forwards>
  <forward name="Homepage" path="/" redirect="true"/>
  <forward name="MessagesPage" path="/protected/MessagesPage.jsp"/>
 </global-forwards>
 <action-mappings type="org.apache.struts.config.SecureActionConfig">
  <action input="/LoginError.jsp" name="loginForm" path="/login"
   scope="request" type="docsstorer.LoginAction" validate="true">
   <set-property property="secure" value="true"/>
   <forward name="Success" path="/Home.jsp" redirect="true"/>
  </action>
  <action name="formVuoto" path="/logout" type="docsstorer.LogoutAction">
   <set-property property="secure" value="false"/>
  </action>
  <action input="/protected/DocUploadForm.jsp" name="docUploadForm"
   path="/docUpload" scope="request" type="docsstorer.DocUploadAction"
validate="true">
   <set-property property="secure" value="false"/>
  </action>
  <action name="fileDownloadForm" path="/fileDownload"
type="docsstorer.FileDownloadAction">
   <set-property property="secure" value="false"/>
  </action>
  <action input="/RegistrationForm.jsp" name="addUserForm"
   path="/addUser" type="docsstorer.AddUserAction" validate="true">
   <set-property property="secure" value="true"/>
  </action>
  <action input="/protected/ViewUsersForm.jsp" name="viewUsersForm"
   path="/viewUsers" scope="request" type="docsstorer.ViewUsersAction"
validate="true">
   <set-property property="secure" value="false"/>
   <forward name="Success" path="/protected/ViewUsers.jsp"/>
  </action>
  <action name="modifyUserForm" path="/modifyUser"
type="docsstorer.ModifyUserAction">
   <set-property property="secure" value="false"/>
  </action>
  <action name="modifyUserForm" parameter="/protected/ModifyUser.jsp"
   path="/modifyUserForward" type="org.apache.struts.actions.ForwardAction">
   <set-property property="secure" value="false"/>
  </action>
  <action name="deleteUserForm" path="/deleteUser"
type="docsstorer.DeleteUserAction">
   <set-property property="secure" value="false"/>
  </action>
  <action input="/SearchDocsForm.jsp" name="searchDocsForm"
   path="/searchDocs" type="docsstorer.SearchDocsAction" validate="true">
   <forward name="Success" path="/protected/ViewDocs.jsp"/>
  </action>
  <action parameter="/protected/ViewUsersForm.jsp"
   path="/viewUsersFormForward" type="docsstorer.AdministratorForwardAction">
   <set-property property="secure" value="false"/>
  </action>
  <action parameter="/protected/DocUploadForm.jsp"
```

Figura 5.5 - struts-config.xml in modalità testuale (continua)

```
  path="/docUploadFormForward" type="docsstorer.FullAdministratorForwardAction">
   <set-property property="secure" value="false"/>
  </action>
  <action input="/protected/ViewUsers.jsp" name="registerUserForm"
   path="/registerUser" scope="request" type="docsstorer.RegisterUserAction">
   <set-property property="secure" value="false"/>
  </action>
  <action input="/LoginError.jsp" name="passwordRecoveryForm"
   path="/passwordRecovery" scope="request"
   type="docsstorer.PasswordRecoveryAction" validate="true">
   <set-property property="secure" value="false"/>
  </action>
  <action name="deleteFileForm" path="/deleteFile" type="docsstorer.DeleteFileAction">
   <set-property property="secure" value="false"/>
  </action>
  <action path="/setLocale" type="docsstorer.SetLocaleAction" unknown="true">
   <set-property property="secure" value="false"/>
  </action>
  <action input="/protected/MessagesPage.jsp" name="fileUploadForm"
   path="/fileUpload" scope="request" type="docsstorer.FileUploadAction"
validate="true">
   <set-property property="secure" value="false"/>
  </action>
  <action input="/protected/ModifyDoc.jsp" name="modifyDocForm"
   path="/modifyDoc" type="docsstorer.ModifyDocAction" validate="true">
   <set-property property="secure" value="false"/>
  </action>
  <action name="modifyDocForm" parameter="/protected/ModifyDoc.jsp"
   path="/modifyDocForward" type="docsstorer.FullAdministratorForwardAction">
   <set-property property="secure" value="false"/>
  </action>
  <action input="/Informations.jsp" name="messageToAdminForm"
   path="/messageToAdmin" type="docsstorer.MessageToAdminAction" validate="true">
   <set-property property="secure" value="false"/>
  </action>
  <action name="viewDocAbstractForm" path="/viewDocAbstract"
   type="docsstorer.ViewDocAbstractAction" validate="false">
   <set-property property="secure" value="false"/>
   <forward name="Success" path="/protected/ViewDocAbstract.jsp"/>
  </action>
 </action-mappings>
 <controller locale="true"/>
 <message-resources key="it" null="false"
parameter="docsstorer.DocsStorerMessageResources_it"/>
 <message-resources null="false" parameter="docsstorer.DocsStorerMessageResources"/>
 <plug-in className="org.apache.struts.action.SecurePlugIn">
  <set-property property="httpPort" value="8080"/>
  <set-property property="httpsPort" value="8443"/>
  <set-property property="enabled" value="true"/>
  <set-property property="addSession" value="true"/>
 </plug-in>
```

Figura 5.5 - struts-config.xml in modalità testuale (continua)

```
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
  <set-property property="pathnames" value="/WEB-INF/validator-rules.xml,/WEB-
INF/validation.xml"/>
 </plug-in>
</struts-config>
```

Figura 5.5 - struts-config.xml in modalità testuale (continua)

La modalità grafica risulta molto utile per capire i collegamenti tra i vari oggetti, ma risulta impossibile definire sorgenti dati, plug-in, form, per cui diventa necessario utilizzare almeno un'altra delle tre modalità di visualizzazione.

Anche per questo file esiste un DTD di riferimento, ma anche in questo caso non verranno specificati tutti i tag possibili, per tutti gli altri possibili parametri si rimanda al sito della fondazione Apache (http://jakarta.apache.org/commons/dtds/struts-config_1_1.dtd).

Si effettua un'analisi dei vari elementi:

- è possibile individuare in modalità grafica le pagine jsp, le action e i link tra di essi;
- all'interno dei tag <form-bean></form-bean> sono presenti le definizioni degli ActionForm, specificando per ognuno il nome e il tipo, dove il tipo corrisponde alla classe Java che estende la ActionForm;
- nei tag <action-mappings></action-mappings> per ogni action viene descritto il mapping tra il path di una specifica request e la corrispondente classe Java che estende la classe Action; viene definito il path, il tipo, l'eventuale form associato, l'attributo validate per indicare se bisogna richiamare il metodo *validate()* del

form associato, i forward, che rappresentano i path di una pagina jsp o di una Action verso cui verrà effettuato il forward, ecc. Per quanto riguarda i forward, in tale file viene effettuata un'associazione tra il nome reale della pagina o della Action e un nome simbolico che verrà utilizzato in tutta l'applicazione. Se in un secondo momento si volesse cambiare un link, bisognerebbe esclusivamente cambiare il nome della pagina in tale file, lasciando inalterato il nome simbolico;

- deve, ovviamente, essere definito il resource bundle, ovvero il file *.properties*, di default, in cui sono presenti i messaggi e i testi presenti all'interno dell'applicazione, per evitare di inserirli all'interno delle pagine jsp;
- nella parte finale del file si trovano i tag controller e plug-in. Il tag controller viene definito quando viene creato un proprio RequestProcessor, o comunque quando ne viene utilizzato uno diverso da quello di default. Il tag plug-in viene utilizzato per aggiungere funzionalità, le quali vengono caricate all'avvio dell'applicazione e distrutte al termine. Ciò per evitare di aggiungere tali funzionalità in maniera permanente nel codice [Festa, 2005].

Dopo aver esposto come viene configurata l'applicazione, si preferisce non riportare il codice di tutta l'implementazione, che comunque resta disponibile per chi ne fosse interessato.

## 5.3 Caratteristiche di rilievo

Durante la fase realizzativa della web application, ci si è soffermati parecchio sullo sviluppo e l'aggiunta di alcune caratteristiche; di seguito se ne riporta un'esposizione.

### 5.3.1 L'internazionalizzazione (I18N)

Al giorno d'oggi ci si trova a creare applicazioni non più rivolte al solo utente italiano, quindi è nata l'esigenza di rendere l'applicazione "internazionale"; per fare ciò è stata presa la decisione di implementare l'internazionalizzazione (da ora la chiameremo I18N).
Con I18N si intende quel processo di progettazione del proprio software in previsione del supporto linguistico per più aree onde evitare di tornare a riscrivere l'applicazione tutte le volte che devono essere supportate nazioni e/o lingue nuove [Cavaness, 2003].
Java e Struts rendono molto facile l'aggiunta dell'I18N all'interno delle proprie web application. Per renderle multilingua è richiesta la creazione di un *resource-boundle* per ogni lingua implementata, al cui interno contenga il testo in una sola lingua; il nome da dare ai vari resource-boundle deve seguire queste direttive:

- ogni resource-boundle deve contenere all'interno del suo nome l'identificativo della lingua e/o regionale a cui si riferisce quel file;
- per il file contenente invece la lingua di default, non bisogna inserire all'interno del proprio nome alcun riferimento regionale o della lingua.

Dopo aver creato i due file e averne aggiunto le righe di configurazione all'interno del file struts-config.xml, il passo successivo è quello di utilizzare all'interno delle JSP un tag particolare, <bean:message/>, quindi sarà la JSP a selezionare la stringa giusta da copiare, in base alla chiave e al locale (lingua) del client che si collega all'applicazione web.

Per effettuare invece il cambio di lingua (quella di default è l'inglese) è stata creata una classe apposita che effettua la conversione italiano/inglese e viceversa.

Per quanto riguarda l'interfaccia grafica è stata semplicemente creata un'immagine per ogni lingua, cliccando su di essa avviena il cambio di lingua, e quindi di locale.

### 5.3.2 Automatizzazione

Proseguendo nell'implementazione, si è visto che buona parte delle operazioni inizialmente svolte dagli amministratori potevano essere automatizzate, così da sgravare di una buona misura il lavoro di gestione dell'applicazione oppure per poter dare loro determinati avvisi.

Le funzioni che sono state rese automatiche sono le seguenti:

- Invio di un'email all'amministratore principale ogni qual volta vi sia una nuova richiesta di registrazione;
- Invio di un'email agli utenti a cui è stata accettata la richiesta di registrazione, includendovi anche quali erano i dati necessari per il login;
- Recupero della password in caso di smarrimento attraverso l'invio di un'email all'indirizzo identificativo dell'utente;
- Invio di messaggi all'amministratore principale, contenenti ad esempio segnalazioni di bug, reclami o richieste.

Per rendere possibile tutto questo, è stata creata anche un'apposita classe delegata all'invio delle email ad un server di posta SMTP; si ricorda che la configurazione di questi parametri è oggetto del prossimo capitolo.

### 5.3.3 Sicurezza

Quasi tutte le applicazioni, oggigiorno, devono prevedere una qualche forma di protezione. Una buona parte del lavoro di implementazione è stata dedicata anche a questo aspetto.
Si riporta di seguito in che modo è stata implementata la sicurezza:

- Link, pagine ed Action sono accessibili solo a chi ne ha i privilegi, in caso contrario l'applicazione redirezionerà il client verso la homepage;

- Protezione dei dati sensibili e personali (quali password, nome, cognome) mediante l'adozione di trasmissione sicura dei dati, aggiungendo l'SSL all'applicazione.

Per quanto riguarda il primo punto, i link alle pagine protette sono visibili solo agli utenti che possono accedere a quelle funzionalità. In più, nel caso in cui qualcuno accedesse direttamente a quelle risorse inserendo l'indirizzo di quei link, essi sono stati resi sicuri attraverso la creazione di classi di forward apposite che estendono *ForwardAction* e aggiungono la logica necessaria per la sicurezza.
Alle pagine protette, quelle contenute nella cartella */protected*, si può accedere solo mediante uno dei forward protetti o nel caso in cui sia l'applicazione a fare un forward della pagina all'utente.
Un'operazione simile è stata fatta per le classi Action; quelle da proteggere estendono la classe *ProtectedAction* appositamente creata, la quale estende la classe *Action* di Struts; questa classe mette a disposizione un metodo a cui passare il tipo di utente che vuole utilizzare quella Action; questo metodo resistuisce una variabile booleana che indica se l'utente ha i diritti o meno per effettuare quella determinata funzionalità.

Per quanto riguarda l'SSL, esso è stato implementato attraverso l'utilizzo di un plugin di Struts, **SSLext**.
SSLext è un piccolo progetto che è nato per facilitare l'implementazione del Secure Socket Layer (SSL) all'interno di web application create con l'utilizzo del Framework Struts.

L'SSL, come si intuisce già leggendo il suo nome per esteso, è un protocollo che è stato creato per rendere più sicura la trasmissione di dati fra client e server soprattutto quando vi è uno scambio di dati sensibili e importanti (numero di carta di credito o password) o anche quando si vuole preservare la propria privacy. Il meccanismo di protezione è abbastanza semplice e si basa principalmente sulla criptazione a chiave pubblica, anche tramite l'utilizzo di certificati.

Il Plug-in in questione è davvero semplicissimo da configurare e da usare: per prima cosa bisogna controllare che il proprio server abbia abilitato l'SSL e abbia un certificato, poi si aggiunge un campo plug-in al file di configurazione di Struts, cioè lo struts-config.xml, dove si specificano il nome della libreria, le porte HTTP e HTTPS ed eventuali altri parametri configurativi. Per attivare invece l'SSL si può proseguire in due modi, il primo è dichiarare sempre in struts-config.xml quali sono le Action che si vuole rendere più sicure, oppure aggiungendo alcuni tag proprietari all'interno delle JSP della web application per indicare al framework che quella pagina deve eseguire trasmissioni di dati sicure. Si riporta inoltre che in tutta la web application vanno sostituiti tutti i link **(<a>)** con il tag **<sslext:link>** che serve a tener conto della sessione corrente anche nel caso si passi da HTTP e HTTPS e viceversa, caso in cui altrimenti si perderebbe la sessione.

### 5.3.4 Validator

Per effettuare una più pulita e meno ridondante validazione, è stato deciso di implementare un altro plug-in di Struts, **Validator**.

Validator è uno dei tantissimi progetti varati da Jakarta e rivolti a semplificare lo sviluppo di web application. Questo plug-in è rivolto essenzialmente alla validazione dei campi delle form all'interno delle JSP. Il progetto è molto flessibile e, oltre a mettere a disposizione un corposo numero di regole di validazione, permette quindi di poter aggiungere delle nuove regole in modo semplice creandosi una propria classe con all'interno la logica per effettuare un determinato tipo di validazione e configurando un file (**validator-rules.xml**) all'interno del quale si può anche aggiungere il corrispettivo in Javascript da importare all'interno delle JSP per la validazione lato client. Per legare un campo ad una certa regola la configurazione è semplice, basterà creare una classe form (sia dinamicamente che staticamente) espandendone una facente parte del package di Validator. Fatto ciò, si settano nel file **validation.xml** quali sono tutti i form e i campi su cui Validator deve agire, e per ogni campo, quali sono le regole di validazione da applicare; nel caso si volessero applicare sia regole di Validator che altra logica di validazione, si possono semplicemente modificare le classi di form affinché siano loro ad

accedere con un metodo appropriato alla validazione di Validator per poi continuare con la propria logica.

### 5.3.5 Log4j

E' sempre più importante, nelle applicazioni odierne, tenere traccia di tutto ciò che succede nell'applicazione, sia in automatico sia su richiesta dell'utente. Tutto questo, nel mondo dell'informatica, prende il nome di **logging**. Jakarta mette a disposizione una delle migliori librerie software adatte a fare questo. Come il resto dei plug-in sviluppati da Jakarta, questo è anch'esso facile da installare, configurare ed usare. Per prima cosa va importata la sua libreria, va creato un suo file di configurazione e poi, per effettuare il logging, basta importare due classi nella classe che voglia utilizzarlo oppure si può usare una libreria di tag proprietari per le JSP che rendono le operazioni di questo tipo utilizzabili anche da coloro i quali non conosco minimamente i linguaggi di programmazione. Per effettuare invece l'operazione vera e propria di salvare messaggi di logging, basta usare il metodo o il tag apposito inclusi entrambi in Log4j.

### 5.3.6 Interfaccia grafica user-friendly

Lo studio dell'interfaccia grafica è uno dei punti su cui ci si dovrebbe soffermare di più, in quanto un'interfaccia ben progettata può rendere di successo anche un'applicazione

mediocre, viceversa un'interfaccia macchinosa può rendere fallimentare anche l'applicazione più brillante.
La scelta è stata quella di realizzare un'interfaccia graficamente sufficientemente curata ma, soprattutto, semplice e facilmente accessibile.
Per realizzare tutto ciò, in alto, è stato posizionato il logo dell'applicazione che, come spesso accade anche in altre applicazioni, è anche un link alla homepage; inoltre è stato creato il menu laterale: in esso è contenuto il form con cui si può effettuare il Login, o in alternativa le informazioni con cui ci si è loggati; al di sotto invece troviamo i link a cui si può accedere con i diritti di cui si è in possesso, nella parte più bassa del menu è invece presente l'immagine con cui si cambia lingua, come già spiegato in precedenza.
Le pagine vere e proprie che di volta in volta sono visitate dall'utente invece riempiono il resto della schermata.

### 5.3.7 Facilità di configurazione

L'ultima caratteristica di cui si fa cenno, ultima non in ordine di importanza, è quella della facilità di configurazione dell'applicazione, di cui presenteremo tutti i dettagli nel prossimo capitolo.
La scelta in fase di progettazione è stata quella di portare all'esterno delle classi tutti i dati che possono variare a seconda

di chi utilizzi l'applicazione o meglio a seconda di chi faccia il deploy di essa su un server.

La configurazione dei parametri che riguardano il server SMTP di riferimento, l'email dell'amministratore principale e i dati per il collegamento al DBMS MySQL è stata tutta inserita all'interno di un file di tipo *.properties*, da cui tutta l'applicazione può attingere i dati necessari alle determinate funzioni che si devono svolgere.

# CAPITOLO VI

# CONFIGURAZIONE ED INIZIALIZZAZIONE DELL'APPLICAZIONE WEB

## 6.1 Introduzione

In questo capitolo vedremo tutti i passi necessari a configurare e caricare l'applicazione sul server che abbiamo a disposizione.

## 6.2 Creazione del Database

La prima operazione da fare è creare il database su DBMS MySQL.
Di seguito portiamo il contenuto del file contente la query di creazione del database, il quale crea un utente di tipo Administrator con cui effettuare il primo accesso; si consiglia di sostituirlo con un altro utente Administrator registrandone uno nuovo e cancellando quello creato all'inizio, in quanto poco sicuro.

```
/*==============================================================
===*/
/* DBMS name:      MySQL 4.0
*/
/* Created on:     11/09/2006 17.22.15
*/
/*==============================================================
===*/
create database if not exists docsstorer;

use docsstorer;
```

Figura 6.1 File di creazione del database

```
drop index RELATIONSHIP_4_FK on DOCUMENTI;

drop index RELATIONSHIP_2_FK on RELATIONSHIP_2;

drop index RELATIONSHIP_3_FK on RELATIONSHIP_2;

drop table if exists DOCUMENTI;

drop table if exists FILES;

drop table if exists KEYWORD;

drop table if exists RELATIONSHIP_2;

drop table if exists UTENTE;

/*==============================================================
===*/
/* Table: DOCUMENTI
*/
/*==============================================================
===*/
create table DOCUMENTI
(
   DOC_MATRICOLA                  int(20)
not null AUTO_INCREMENT,
   UTE_EMAIL                      varchar(50)
not null,
   DOC_TITOLO                     varchar(200)
not null,
   DOC_AUTORI                     varchar(150),
   DOC_GENERE                     varchar(30),
   DOC_LINGUA                     varchar(20),
   DOC_ANNO                       numeric(4,0),
   DOC_ABSTRACT                   varchar(2000),
   DOC_FILEPRESENTE               bool,
   DOC_RIFERIMENTO                varchar(200),
   primary key (DOC_MATRICOLA)
);

/*==============================================================
===*/
/* Index: RELATIONSHIP_4_FK
*/
/*==============================================================
===*/
create index RELATIONSHIP_4_FK on DOCUMENTI
(
   UTE_EMAIL
);
```

Figura 6.1 File di creazione del database (continua)

```
/*==============================================================
===*/
/* Table: FILES
*/
/*==============================================================
===*/
create table FILES
(
   DOC_MATRICOLA                  int(20)
not null,
   FIL_FILE                       longblob
not null,
   FIL_NOME                       varchar(50)
not null,
   FIL_MIME                       varchar(50),
   primary key (DOC_MATRICOLA)
);

/*==============================================================
===*/
/* Table: KEYWORD
*/
/*==============================================================
===*/
create table KEYWORD
(
   KEY_KEYWORD                    varchar(20)
not null,
   primary key (KEY_KEYWORD)
);

/*==============================================================
===*/
/* Table: RELATIONSHIP_2
*/
/*==============================================================
===*/
create table RELATIONSHIP_2
(
   DOC_MATRICOLA                  int(20)
not null,
   KEY_KEYWORD                    varchar(20)
not null,
   primary key (DOC_MATRICOLA, KEY_KEYWORD)
);
/*==============================================================
===*/
/* Index: RELATIONSHIP_2_FK
*/
/*==============================================================
===*/
create index RELATIONSHIP_2_FK on RELATIONSHIP_2
(
   DOC_MATRICOLA
);
```

Figura 6.1 File di creazione del database (continua)

```
/*==============================================================
===*/
/* Index: RELATIONSHIP_3_FK
*/
/*==============================================================
===*/
create index RELATIONSHIP_3_FK on RELATIONSHIP_2
(
   KEY_KEYWORD
);

/*==============================================================
===*/
/* Table: UTENTE
*/
/*==============================================================
===*/
create table UTENTE
(
   UTE_EMAIL                      varchar(50)
not null,
   UTE_PW                         varchar(50),
   UTE_NOME                       varchar(50),
   UTE_COGNOME                    varchar(50),
   UTE_REGISTERED                 bool,
   UTE_LOCALE                     varchar(15),
   UTE_TIPO                       varchar(20)
not null,
   primary key (UTE_EMAIL)
);

alter table DOCUMENTI add constraint FK_RELATIONSHIP_4
foreign key (UTE_EMAIL)
      references UTENTE (UTE_EMAIL) on delete restrict on
update restrict;

alter table FILES add constraint FK_RELATIONSHIP_1 foreign
key (DOC_MATRICOLA)
      references DOCUMENTI (DOC_MATRICOLA) on delete restrict
on update restrict;

alter table RELATIONSHIP_2 add constraint FK_RELATIONSHIP_2
foreign key (DOC_MATRICOLA)
      references DOCUMENTI (DOC_MATRICOLA) on delete restrict
on update restrict;

alter table RELATIONSHIP_2 add constraint FK_RELATIONSHIP_3
foreign key (KEY_KEYWORD)
      references KEYWORD (KEY_KEYWORD) on delete restrict on
update restrict;
```

Figura 6.1 File di creazione del database (continua)

```
/*==============================================================
===*/
/* Aggiungi Admin
*/
/*==============================================================
===*/
insert into
UTENTE(UTE_EMAIL,UTE_PW,UTE_NOME,UTE_COGNOME,UTE_REGISTERED,U
TE_TIPO,UTE_LOCALE)
values("a@a.com","a","a","a",true,"Administrator","it");
```

Figura 6.1 File di creazione del database (continua)

## 6.3 Configurazione del file Settings.properties

Dopo aver creato il database, è arrivato il momento di immettere i propri dati nell'unico file dell'applicazione che deve essere modificato in base alle esigenze: *Settings.properties***.**
Il file in questione è presente all'interno del file .WAR (un file .ZIP a cui è stata rinominata l'estensione) dell'applicazione, nella cartella *WEB-INF\classes\docsstorer* .
Se ne riporta di seguito un file di esempio e una piccola guida alla configurazione di esso:

```
#DB
db.host=localhost
db.nomedb=docsstorer
db.user=root
db.pw=password

#EmailSender
emailsender.adminEmail=nome@dominio.it
emailsender.smtpServer=smtp.dominio.it
emailsender.port=25
```

Figura 6.2 Esempio di un file Settings.properties

- db.host: indica l'indirizzo IP di dov'è situato il database;

- db.nomedb: indica il nome del database creato in MySQL, o quello messo a disposizione da MySQL del server;
- db.user: nome utente con cui si accede a MySQL;
- db.pw: password con cui si accede a MySQL;
- emailsender.adminEmail: indirizzo email dell'amministratore principale;
- emailsender.smtpServer: server SMTP a cui mandare le email;
- emailsender.port: porta del server SMTP a cui mandare l'email;

## 6.4 Deploy su Tomcat

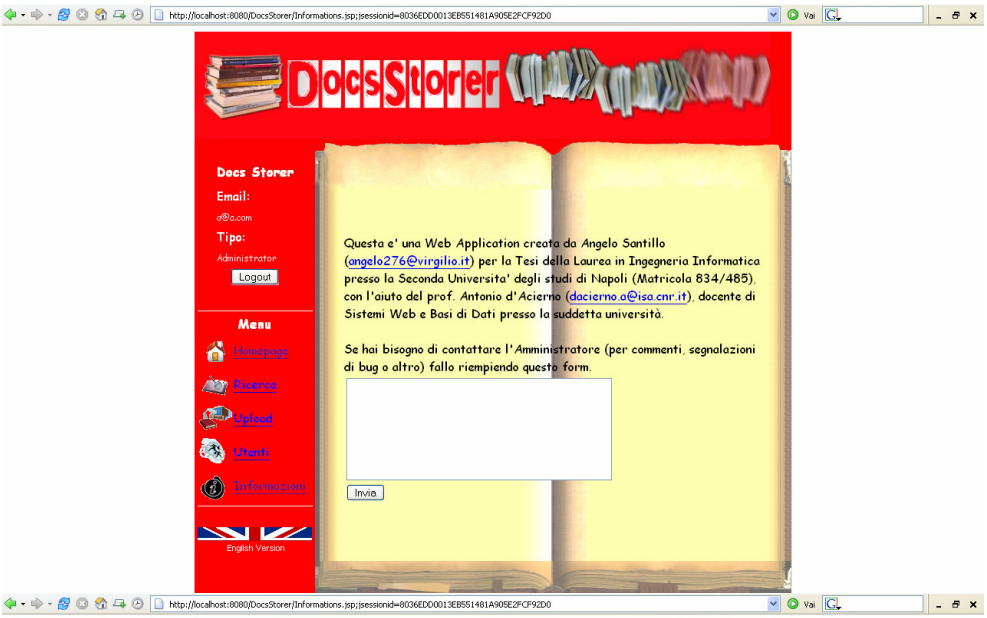
Dopo aver creato il database e aggiornato il file contenuto nel WAR come appena esposto, l'ultima operazione da fare è quella di fare il deploy dell'applicazione sul Tomcat che abbiamo a disposizione; nel caso voi siate amministratori di quel server:

1. Andare alla homepage del server Tomcat;
2. Accedere a *Tomcat Manager*;
3. Selezionare il file .WAR di cui fare il deploy nell'area chiamata *WAR file to deploy*
4. Ciccare sul tasto *Deploy*
5. Provare l'applicazione cliccando sul link a suo nome che è apparso nell'area *Applications*.

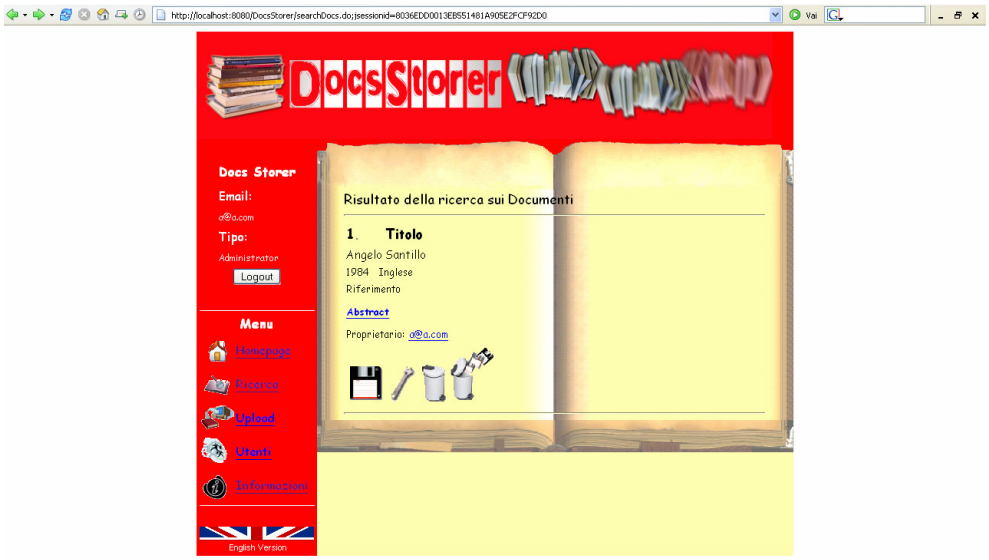
6. Se l'applicazione non parte riavviare Tomcat, cliccare sull'*Undeploy* della riga dell'applicazione e ripartire dal punto 3.

Si ricorda che inoltre è indispensabile che su Tomcat sia attiva l'SSL e l'HTTPS e che si disponga di un certificato; per maggiori informazioni su questi argomenti si rimanda il lettore alla guida presente sul sito http://tomcat.apache.org/tomcat-5.5-doc/ssl-howto.html

# GALLERIA DI SCREENSHOTS



http://localhost:8080/DocsStorer/Informations.jsp;jsessionid=8036EDD0013EB551481A905E2FCF92D0
Vai
DocsStorer
Docs Storer
Email:
a@a.com
Tipo:
Administrator
Logout
Menu
Homepage
Ricerca
Upload
Utenti
Informazioni
English Version
Questa e' una Web Application creata da Angelo Santillo (angelo276@virgilio.it) per la Tesi della Laurea in Ingegneria Informatica presso la Seconda Universita' degli studi di Napoli (Matricola 834/485), con l'aiuto del prof. Antonio d'Acierno (dacierno.a@isa.cnr.it), docente di Sistemi Web e Basi di Dati presso la suddetta università.
Se hai bisogno di contattare l'Amministratore (per commenti, segnalazioni di bug o altro) fallo riempiendo questo form.
Invia


http://localhost:8080/DocsStorer/Informations.jsp;jsessionid=8036EDD0013EB551481A905E2FCF92D0
Vai
DocsStorer
Docs Storer
Email:
a@a.com
Tipo:
Administrator
Logout
Menu
Homepage
Search
Upload
Utenti
Informations
Versione Italiana
This is a Web Application created by Angelo Santillo (angelo276@virgilio.it) for his Thesis for the Laurea in Informatic Engineering at Seconda Universita' degli studi di Napoli (Registration Number 834/485), with the help of prof. Antonio d'Acierno (dacierno.a@isa.cnr.it), teacher of Sistemi Web e Basi di Dati at the aforesaid university.
If you need to contact the Administrator (for comments, bug segnalation or other things) do it filling this form.
Send


http://localhost:8080/DocsStorer/searchDocs.do;jsessionid=8036EDD0013EB551481A905E2FCF92D0
Vai
DocsStorer
Docs Storer
Email:
a@a.com
Tipo:
Administrator
Logout
Menu
Homepage
Ricerca
Upload
Utenti
Informazioni
English Version
Risultato della ricerca sui Documenti
1. Titolo
Angelo Santillo
1984 Inglese
Riferimento
Abstract
Proprietario: a@a.com


http://localhost:8080/DocsStorer/docUploadFormForward.do;jsessionid=8036EDD0013EB551481A905E2FCF92D0
Vai
DocsStorer
Docs Storer
Email:
a@a.com
Tipo:
Administrator
Logout
Menu
Homepage
Ricerca
Upload
Utenti
Informazioni
English Version
Inserire i dati del Documento
* =Campi obbligatori
Titolo*
Autori
Lingua
Italiano
Anno
2006
Riferimento
Abstract
MAX 2000
caratteri
File
Sfoglia...
Invia

# CONCLUSIONI

Siamo finalmente arrivati alla fine, dopo settimane di studio, progettazione, implementazione e stesura, si può finalmente dire che la tesi è giunta al suo termine; tutti i requisiti iniziali sono stati soddisfatti e tutte le varie funzionalità di cui si è sentita la necessità di essere aggiunte sono state implementate con successo.

Naturalmente il lavoro non è finito qui, come ogni applicazione, ora resta la fase di correzione dei bug e di raffinazione. L'applicazione è attualmente in funzione su un server dove può essere continuata la fase di testing.

La tesi mi ha permesso di approfondire una materia che ho conosciuto per la prima volta all'università e che fin da subito mi ha incuriosito. L'utilizzo di applicazioni di sviluppo professionali e di framework potenti mi ha permesso di accrescere le mie conoscenze lanciandomi ogni volta il guanto di sfida riguardo all'aggiunta di nuove funzionalità o semplicemente per cercare di farne funzionare qualcuna standard, alcune delle quali hanno dato problemi di implementazione.

Riguardo agli sviluppi futuri sono rimaste aperte varie opportunità.

Una di queste è l'aggiunta di un vero e proprio motore di indicizzazione per dare l'opportunità agli utenti di fare ricerche veloci e mirate.

Infine, nonostante Struts sia un framework completo e potente, in questi ultimi anni il mondo dell'informatica è in agitazione per l'imminente avvento del Web 2.0, con l'utilizzo massivo di AJAX (Asynchronous JavaScript & XML) per la realizzazione di applicazioni che non risiedono più solo sui server ma che coinvolgono soprattutto i client, spostando molte delle funzionalità su di essi. Proprio a questo scopo è nato il framework JSF (Java Server Faces) che, volendo, può anche essere usato in combinazione con Struts.

# BIBLIOGRAFIA


| |
|---|
| Cavaness, C. [2003], *Programmare con Jakarta Struts*, O'Reilly, Ulrico Hoepli Editore. |
| Festa, M. [2005], *Progetto e realizzazione di una applicazione web-based sulle proteine*. |
| Gigliotti, G [2005], *HTML 4.01*, Apogeo s.r.l. |

# SITOGRAFIA


| |
|---|
| Dongilli, P. [2004], *http://sewasie.ing.unimo.it:8080/sewasie/doc/integration/index.jsp* |
| Eclipse, *http://www.eclipse.org*, Eclipse Foundation |
| Exadel Studio, *http://www.exadel.com* |
| HTML.it, *http://www.html.it* |
| Jakarta, *http://Jakarta.apache.org*, Apache Sofware Foundation |
| JSP Insider [2002], *http://www.jspinsider.com/content/jsp/struts/struts.jsp*, Amberjack Software LLC |
| Log4j, *http://logging.apache.org/log4j*, Apache Sofware Foundation |

| |
|---|
| Morgan, E. [2000], *http://www.infomotions.com* |
| MySQL, *http://www.mysql.com*, MySQL AB |
| PowerDesigner, *http://www.sybase.com/products/developmentintegration/powerdesigner*, Sybase Inc. |
| SSLext, *http://sslext.sourceforge.net* |
| Struts, *http://struts.apache.org*, Apache Software Foundation |
| Tomcat, *http://tomcat.apache.org*, Apache Software Foundation |
| Validator, *http://Jakarta.apache,org/commons/validator*, Apache Software Foundation |