SECONDA UNIVERSITA'DEGLI STUDI DI NAPOLI

FACOLTÀ DI INGEGNERIA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

GRUPPO SISTEMI DI ELABORAZIONE

TESI DI LAUREA

IN

SISTEMI WEB E BASI DI DATI

MULTIMEDIA REPOSITORY: UNA WEB APPLICATION PER LA GESTIONE DEI FILE MULTIMEDIALI

Relatore

Ch.mo Prof.

Antonio d'Acierno

Candidata

Cristina Viola

Matr. 834/496

ANNO ACCADEMICO 2006/2007

Ottobre 2007

Nullum magnum ingenium,
mixtura demientiae

Lucio Anneo Seneca

Un particolare ringraziamento alla mia famiglia, ai miei docenti e ai miei amici, per il loro supporto e per la loro costante presenza.

INDICE

1. INTRODUZIONE

1.1. Ambiti

Il mondo della comunicazione è stato ampiamente rivoluzionato dall'avvento di internet.

Milioni di persone usano quotidianamente i servizi che la rete offre, per comunicare tra loro, utilizzare la posta elettronica, trasferire dati, incontrarsi sulle chat e sfruttare il Web utilizzando i vari linguaggi: verbale, testuale, iconico, iconico-cinetico e musicale, per la pubblicazione e la consultazione di pagine web sempre più complete ed efficaci.

Oggi questo è reso sempre più possibile per l'evoluzione delle connessioni a banda larga che permettono lo scambio di innumerevoli quantità di dati in tempo reale.

Questa evoluzione tecnologica ha facilitato la condivisione e la diffusione di informazioni non solo sotto forma di testo e di immagine, ma anche e soprattutto sotto forma di informazione multimediale fatta da audio e video.

In questi ultimi anni il Web ha infatti visto la diffusione di siti legati alla divulgazione di contenuti multimediali.

In linea con le attuali forme di comunicazione si è deciso di sviluppare una applicazione web che possa permettere in modo facile ed immediato la condivisione di informazione a carattere multimediale.

1.2. Requisiti

Lo scopo principale della nostra progettazione è stato quello di realizzare una Web Application capace di immagazzinare file, di qualsiasi natura e anche di dimensioni abbastanza corpose, di permetterne l'accesso e la gestione da parte degli utenti interessati.

L'applicazione deve essere 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. All'utente sarà data la possibilità registrarsi, di modificare i propri dati, di ricercare le informazioni in essa contenute, e di interagire con le stesse. Si è poi deciso affidare la gestione dell'applicazione ad un ristretto numero di utenti a cui sono stati riservati particolari diritti e funzioni, quali la gestione degli utenti, dei loro diritti di accesso all'applicazione, e dei file.

Abbiamo successivamente diretto la nostra attenzione verso l'archiviazione e la gestione dei file multimediali, e, in particolare, nel caso di contenuti mpeg della loro esecuzione. È stato quindi coerentemente deciso di chiamare la nostra applicazione Multimedia Repository, (archivio multimediale).

1.3. Linee guida per la progettazione

La progettazione del nostro sistema è stata eseguita secondo una

sequenza di operazioni necessarie per una rapida generazione di codice con il minor numero possibile di errori.

Primo passo è stata la valutazione delle tecnologie Web già esistenti e dei vari linguaggi di programmazione. È stata operata la ricerca di soluzioni già pronte e, comunque, gratuite ed open source, nell'ambito di software di sviluppo, di librerie da utilizzare o di framework già realizzati e liberamente fruibili.

Successivamente, è stata operata la ricerca di guide e manuali che ci facilitassero la comprensione e la soluzione di problematiche che si possono riscontrare nello sviluppo, oppure, la creazione del codice per mezzo di esempi o dimostrazioni pratiche e teoriche.

Abbiamo, poi, operato la scelta della base di dati, che ci fosse più congeniale, dei modelli e delle metodologie di sviluppo che ci consentissero una buona progettazione dell'applicazione web.

La realizzazione dell'applicazione ha visto, dunque, l'utilizzo dei software precedentemente scelti per lo sviluppo, e la creazione della logica e della configurazione del progetto;
Infine è stato operato il testing dell'applicazione per mezzo di beta testers.

1.4. Architettura del sistema

Affinché l'applicazione possa girare è richiesta, sul nostro sistema, l'installazione di Java Virtual Machine, quindi possiamo fare riferimento un vasto numero di sistemi operativi (tra i quali abbiamo scelto di

Microsoft Windows XP, di larga diffusione). Funzionalità fondamentale è che si possa creare su di esso un ambiente hardware e software in grado di sorreggere lo sviluppo di 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 semiprofessionale.

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

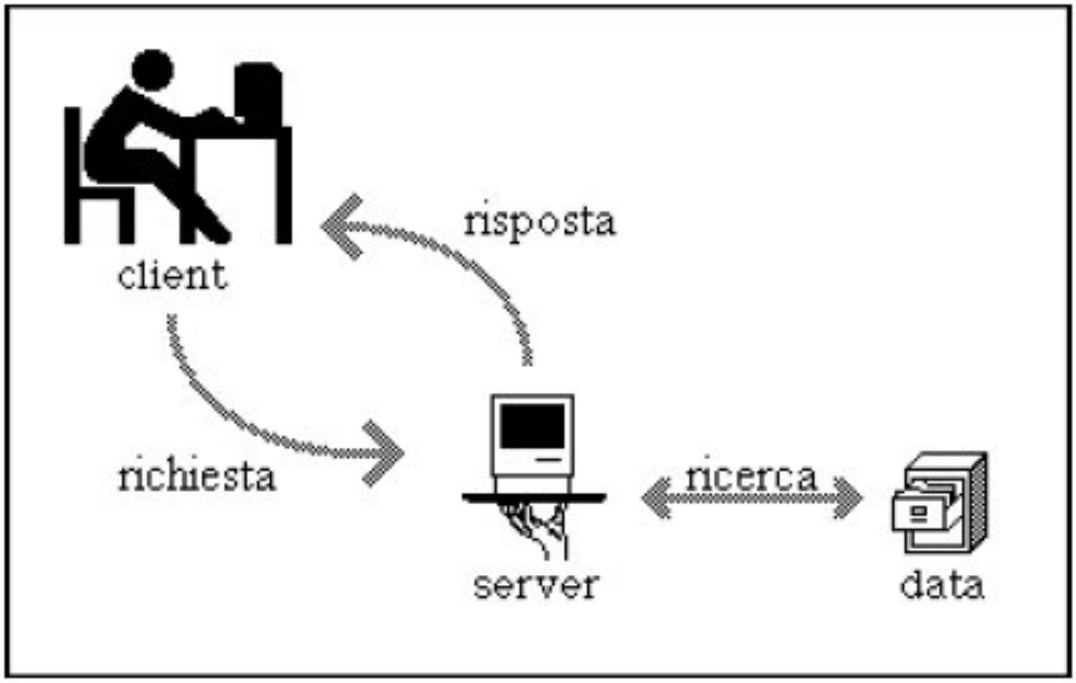


Figura 1.1. - *Architettura* Client/Server

2. MODELLO FUNZIONALE

Abbiamo decido di affrontare la presentazione della nostra applicazione protendendo per una distinzione tra le interfacce front end e back end.

Il front end è la parte di un sistema software che gestisce l'interazione con l'utente, ed è responsabile dell’acquisizione dei dati di ingresso e della loro elaborazione, tale da renderli utilizzabili dal back end.

Il back end, caratterizzata dall'insieme degli strumenti di amministrazione e gestione, è la parte del sistema che si occupa dell’elaborazione dei dati generati dal front end.

Nei sistemi più complessi non è raro che i dati subiscano elaborazioni intermedie prima di passare al back end. La distinzione di una parte di ingresso e di una parte terminale nei sistemi software è un genere di astrazione che aiuta a mantenere le diverse parti di un sistema complesso logicamente separate e quindi più semplici.

2.1. Front end

È necessario che l’interfaccia front end sia user friendly, ossia che garantisca l’utilizzo delle funzionalità della web application in modo intuitivo e semplice da parte degli utenti.

Figura 2.1. - Interfaccia grafica: *Homepage*

La realizzazione ha dunque proteso verso la scelta di un'interfaccia grafica abbastanza intuitiva. Nella parte alta della pagina web abbiamo posizionato il logo dell'applicazione, che ha anche la funzione di link alla homepage, segue una barra pensata per le operazioni di Login, o, in alternativa, per la visualizzazione del messaggio di benvenuto al momento dell'accesso. È stato poi realizzato il menu laterale contenente i link a cui si può accedere con i diritti di cui si è in possesso, e, nella parte bassa, quello che consente in cambio di lingua, costituito dall'immagine della bandiera associata alla lingua di

interesse. Le pagine vere e proprie che di volta in volta sono visitate dall'utente invece riempiono il resto della schermata.

Le funzionalità front end dell'applicazione si rivolgono innanzitutto verso il popolamento dei dati, quindi la nostra analisi ha previsto la definizione del grafico dei casi d'uso, che ci permetteva di rappresentare, in modo sintetico e completo, le funzioni richieste all'applicazione.

Gli attori che interagiscono con la nostra applicazione sono gli utenti; in particolare abbiamo pensato di rappresentare i nostri utenti mediante la definizione di quattro ruoli fondamentali:

- *Utente NonRegistrato*, cui è data la possibilità di operare la ricerca di un file e di iscriversi al sito per avere i diritti per il resto delle funzionalità;
- *Utente InRichiesta*, a cui sono concessi gli stessi diritti di quello NonRegistrato, e la possibilità di modificare i propri dati personali.
- *Utente Registrato*, possessore dei diritti finora menzionati, con, in aggiunta, la possibilità di inserire un nuovo file, eseguirlo, vederne i commenti e aggiungerne di propri.
- *Administrator*, amministratore dell'applicazione, a cui sono accordati tutti i diritti esposti in precedenza. A questo utente sono assegnate le funzionalità di gestione, rimozione e accettazione degli altri utenti, di modifica o cancellazione dei dati e commenti associati ai file, download e rimozione di un file dall'applicazione.

Procediamo ora con la presentazione delle interazioni dei vari

utenti con l’applicazione.

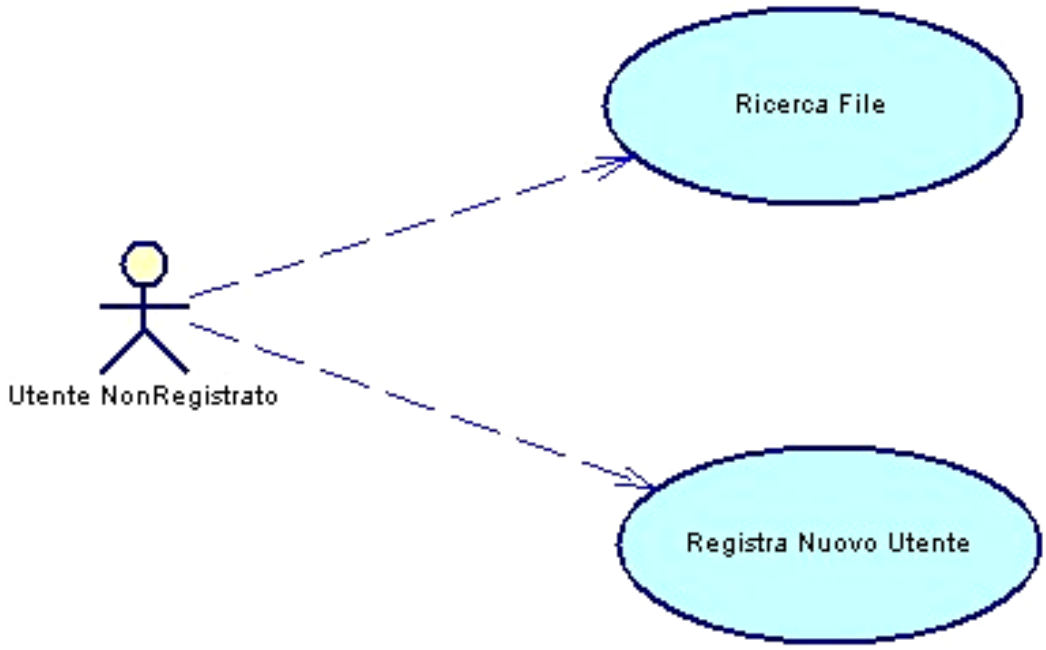


Figura 2.2. - Uses Case *Utente NonRegistrato*

L'*Utente NonRegistrato* ha con l’applicazione i seguenti tipi di interazione:

- *Registra Nuovo Utente*: consente la registrazione dei propri dati nel database dell’applicazione, in modo che la richiesta di registrazione possa essere valutata dall’amministratore;
- *Ricerca File*: permette all’utente di effettuare la ricerca di file desiderati all’interno dell’applicazione.

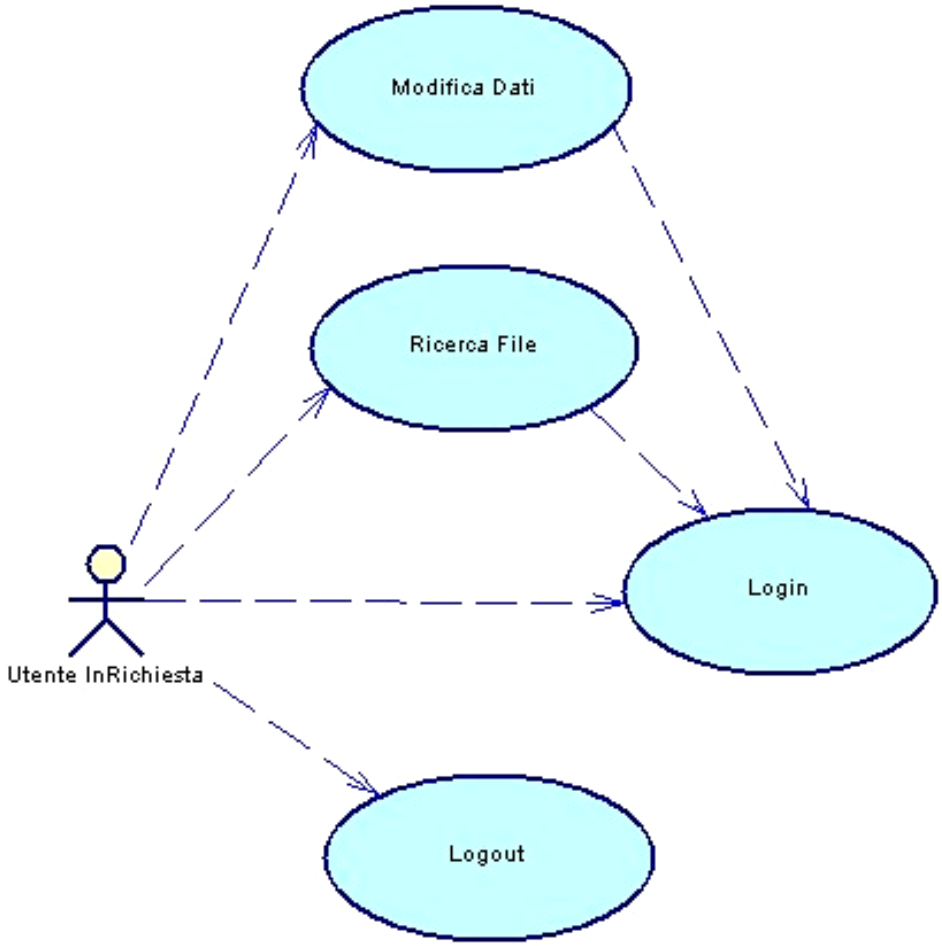


Figura 2.3. - Uses Case *Utente InRichiesta*

L'*Utente InRichiesta* ha in aggiunta i seguenti tipi di interazione con l'applicazione:

- *Login*: permette all'utente, con l'introduzione delle proprie credenziali, di entrare in possesso di determinati privilegi che gli consentono di interagire con l'applicazione;
- *Logout*: consente all'utente di informare l'applicazione che non ha più necessità dei privilegi assegnatigli precedentemente;

- *Modifica Dati*: permette ad un utente di modificare i propri dati personali, introdotti all'atto della registrazione.

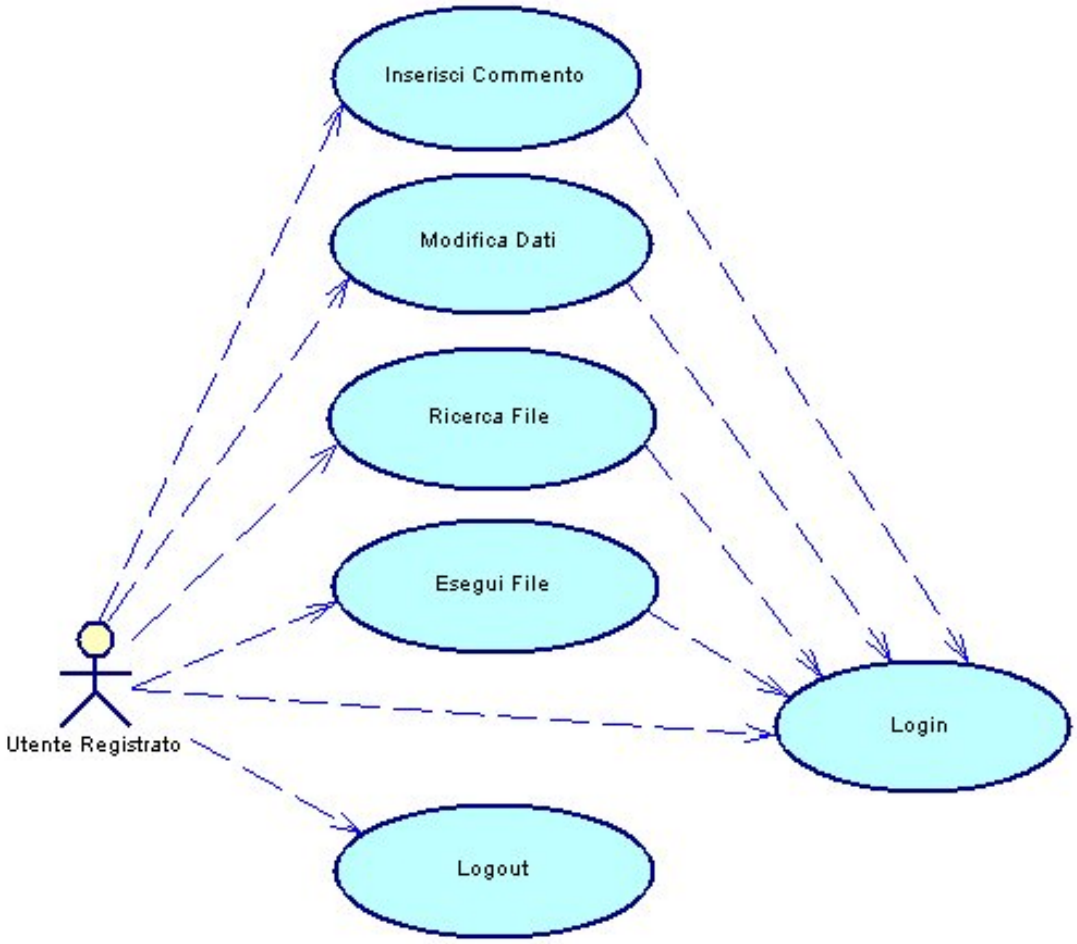


Figura 2.4. - Uses Case *Utente Registrato*

L'*Utente Registrato* ha in aggiunta i seguenti tipi di interazione con l'applicazione:

- *Esegui File*: dà la possibilità all'utente di mandare in esecuzione un file;
- *Inserisci Commento*: dà la possibilità all'utente di aggiungere un commento al file selezionato.

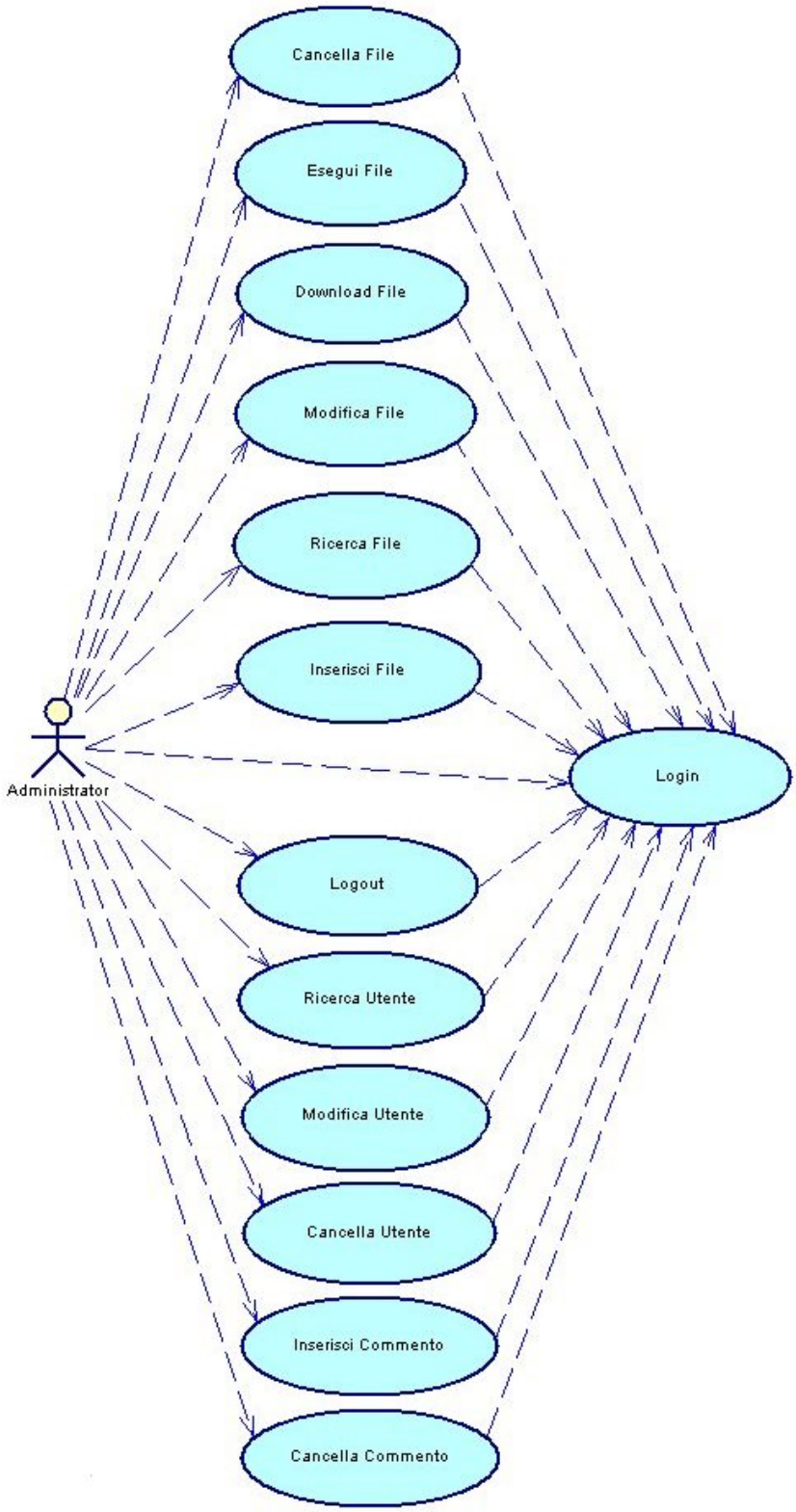


Figura 2.5. - Uses Case *Administrator*

L’*Administrator* ha in aggiunta i seguenti tipi di interazione con l’applicazione:

- *Ricerca Utente*: permette di operare la ricerca di un utente o di una categoria di utenti tra quelli 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*: consente la cancellazione di un utente dall’applicazione;
- *Inserisci File*: consente all’utente di aggiungere un nuovo file a quelli presenti all’interno dell’applicazione;
- *Modifica File*: permette la modifica dei dati associati ad un determinato file;
- *Download File*: permette lo scaricamento di un file sul client da cui proviene la richiesta;
- *Cancella File*: permette la cancellazione di un file, dei dati ad esso associati e dei commenti ad esso allegati, dall’applicazione;
- *Cancella Commento*: permette di cancella un commento dall’applicazione.

2.2. Back end

Continuando la definizione della nostra applicazione, è risultato necessario ricostruire l’insieme dei dati che l’utente è chiamato ad

inserire al momento della sua interazione con l'applicazione e dei quali la stessa ha bisogno per assolvere alle proprie funzionalità. A questo scopo ci siamo rivolti verso un'analisi più approfondita degli uses case citati precedentemente.

Rivolgiamo la nostra attenzione innanzitutto verso i casi d'uso che prevedano l'inserimento delle informazioni per opera dell'utente.

Cominciamo, dunque, dal caso *Registra Nuovo Utente*, primo tipo di interazione con l'applicazione che l'utente incontra sul suo cammino. Questa prevede l'inserimento dei dati personali da parte di un utente che voglia essere abilitato all'accesso alle funzionalità dell'applicazione. Devono essere dunque inseriti i seguenti tipi di dati: Email, sottoposto a verifica di correttezza e di unicità, Password, Cognome, Nome, Nickname, che prevede come per l'email il controllo di unicità, per i quali è operato il controllo di avvenuto inserimento, risultando questi necessari per il riconoscimento di un utente, Luogo di nascita, Data di nascita, Sesso, che risultano piuttosto opzionali.

Nel caso d'uso *Inserisci File* si presenta la necessità dell'inserimento delle informazioni associate al file che si desidera caricare nell'applicazione. In particolare saranno utili il titolo del file, del quale vene controllato l'inserimento, il nome dell'autore e dell'album, il genere di appartenenza, la lingua, l'anno di pubblicazione e naturalmente il file stesso che deve essere necessariamente inserito.

L'ultimo caso d'uso del tipo di inserimento è quello *Inserisci Commento*, nel quale troviamo come unico dato il commento.

Ci spostiamo, successivamente verso i casi d'uso che predano una risposta verso l'utente da parte dell'applicazione, quale può essere la

visualizzazione di determinati tipi di dati. In particolare la nostra attenzione va ai due casi *Esegui File* e *Download File*. Il primo pone l'attenzione sulla necessità di poter operare una selezione sui tipi di file eseguibili, avremo dunque bisogno di un dato che identifichi il formato di un file. L'analisi del secondo mette, invece, in evidenza l'importanza di tener traccia della denominazione originale del file caricato nell'applicazione in modo che possa essere facilmente ricreato sulla macchina dell'utente che ne faccia richiesta.

Una volta conclusa l'analisi, riportiamo che i casi d'uso non menzionati direttamente non prevedono l'aggiunta di nuove informazioni nel database ma solo l'utilizzo appropriato di quelle precedentemente menzionate.

2.2.1. Modello E-R

Procediamo, quindi, con la definizione del modello E-R (Entity-Relationship), conosciuto anche come Modello Concettuale, che ci consenta di organizzare nel miglior modo il database costituito dalle informazioni che dopo la nostra analisi sono risultate necessarie per adempimento delle funzionalità dell'applicazione. Riportiamo di seguito il grafico del modello E-R cui siamo giunti.

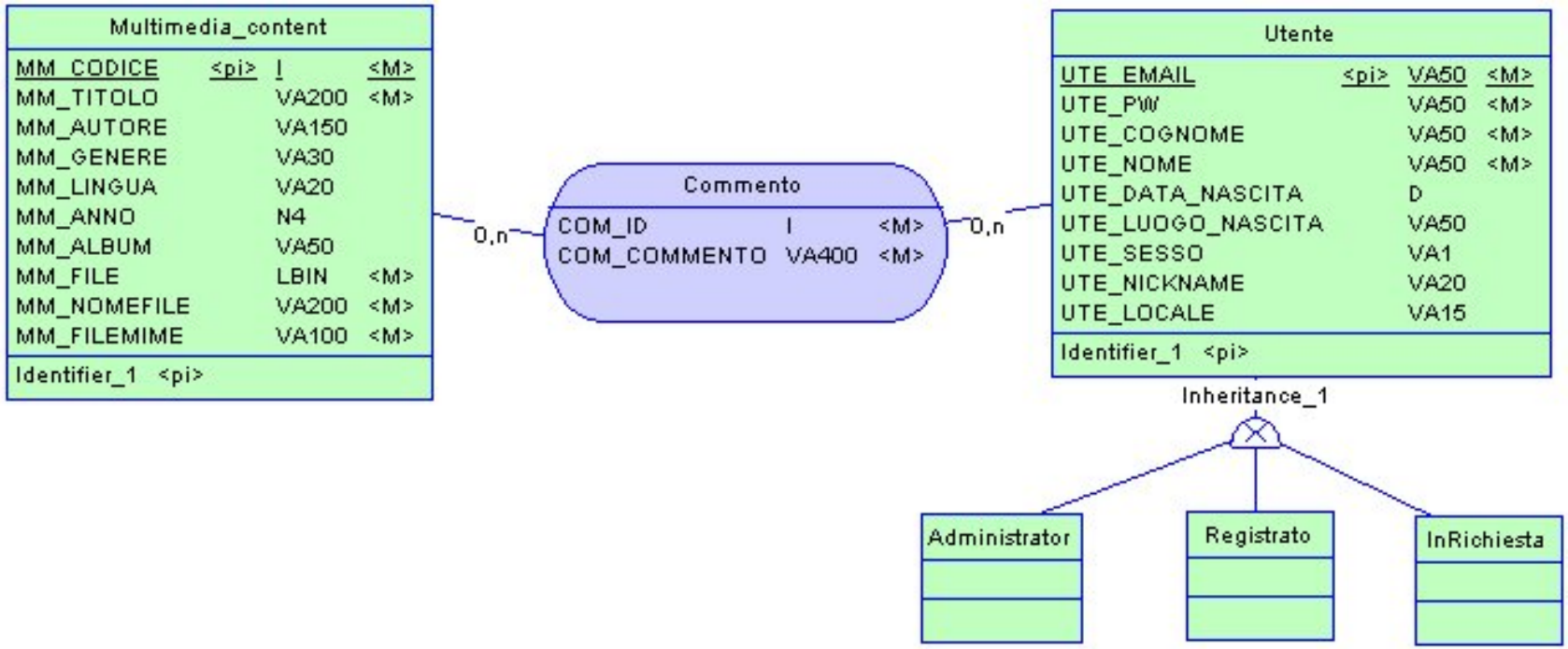


Figura 2.6. - *Modello E R*

LEGENDA DEI DATI

Procediamo ora con la presentazione delle entità e delle associazioni presenti nel grafico precedentemente esposto.

Entità

MULTIMEDIA_CONTENT

Multimedia_content			
MM_CODICE	<pi>	I	<M>
MM_TITOLO		VA200	<M>
MM_AUTORE		VA150	
MM_GENERE		VA30	
MM_LINGUA		VA20	
MM_ANNO		N4	
MM_ALBUM		VA50	
MM_FILE		VA200	<M>
MM_NOMEFILE		VA200	<M>
MM_FILEMIME		VA100	<M>
Identifier_1 <pi>			

Contiene le informazioni relative ai File inseriti nell'applicazione:

- MM_CODICE: numero che identifica il File;
- MM_TITOLO: titolo del File;
- MM_AUTORE: autore del File;
- MM_GENERE: genere del File;
- MM_LINGUA: lingua del File;
- MM_ANNO: anno di produzione del File;
- MM_ALBUM: titolo dell'eventuale album associate al File;
- MM_FILE: è il File vero e proprio;

- MM_NOMEFILE: è il nome del File;
- MM_FILEMIME: è il tipo MIME del File.

UTENTE

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

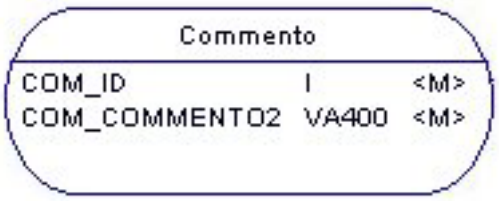
- UTE_EMAIL: email con cui l'Utente si registra, ed identificativo dello stesso in fase di Login;
- UTE_PW: Password per il Login;
- UTE_COGNOME: Cognome dell'Utente;
- UTE_NOME: Nome proprio dell'Utente;
- UTE_DATA_NASCITA: data di nascita dell'Utente;
- UTE_LUOGO_NASCITA: luogo di nascita dell'Utente;
- UTE_SESSO: sesso dell'Utente;
- UTE_NICKNAME: nickname scelto dall'Utente ed unico per ogni Utente;;
- UTE_LOCALE: indica il primo Locale (lingua) con cui l'Utente si registra;

TIPI DI UTENTE

Sono tre entità e rappresentano i tipi di Attori del grafico degli Use Cases fatta eccezione per il tipo NonRegistrato.

Associazione

COMMENTO



Mette in relazione l'entità Utente e quella Multimedia_content, e contiene i commenti inseriti dagli utenti:

- COM_ID: numero che identifica i commenti;
- COM_COMMENTO: è il commento inserito nell'applicazione.

2.2.2. Dal Modello Concettuale al Modello Logico

Ristrutturazione

Prima di eseguire l'opera di traduzione dal Modello Concettuale a quello Logico, bisogna 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. È riscontrato che in modello non presenta ridondanze, o sono già state tradotte ed è già normalizzato.

La Traduzione

L'opera di traduzione viene fatta automaticamente dal software usato, però è sempre meglio controllare ciò che viene generato, poiché potrebbero esserci incongruenze rispetto allo schema atteso.

Il principio con cui si opera la traduzione è molto semplice. Ogni entità viene espressa tramite una relazione (cioè una tabella) e i suoi attributi diventano attributo della tabella ed in particolare quelli che identificano le istanze di un'entità costituiscono la chiave primaria della

tabella. Le associazioni vengono tradotte a seconda della loro natura. Nel nostro caso l'associazione stessa viene modellata con una nuova tabella e vengono introdotte opportunamente le chiavi primarie delle tabelle legate da questa. La relazione di generalizzazione/specializzazione mutuamente esclusiva che riguarda la tabella "utente", viene tradotta in un attributo della suddetta tabella, che indica di che tipo di Utente si tratta.

2.2.3. Il Modello Logico

Riportiamo il grafico del Modello Logico precedentemente presentato:

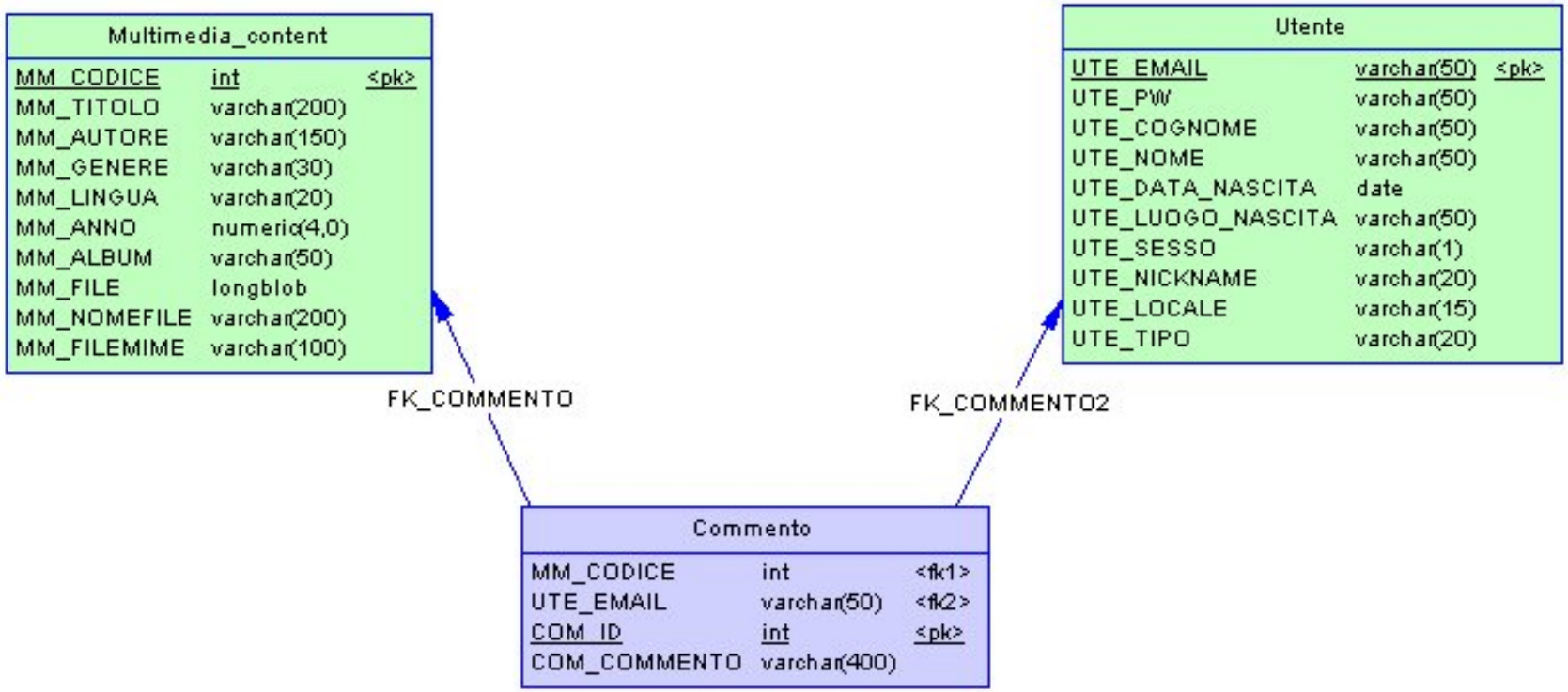


Figura 2.7. - *Modello Concettuale*

3. TECNOLOGIE E METODOLOGIE SCELTE PER LO SVILUPPO DELL'APPLICAZIONE

3.1. Tecnologie per la progettazione

La progettazione dell'applicazione è stata compiuta rivolgendosi principalmente ad un solo tool di sviluppo e progettazione, così da realizzare un'applicazione semplice e versatile. Dopo un'attenta valutazione la nostra scelta, come già accennato in precedenza, è ricaduta sul CASE (Computer Aided Software Engineering) PowerDesigner della Sybase, software utilizzato per la creazione dei grafici in UML e del Modello Concettuale e del Modello Logico, essendo questa funzionalità uno dei punti di forza del tool stesso, supportando la maggior parte di grafici UML ed interfacciandosi con un buon numero di DBMS (DataBase Management System).

3.2 Tecnologie per l'implementazione

La scelta del Sistema Operativo, come precedentemente accennato, è stata operata facendo riferimento all'utilizzo di Java come linguaggio di programmazione, e si è di conseguenza rivolta verso 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 scelta del browser (software necessario per accedere all'applicazione Web), che ha reso possibile il collaudo dell'applicazione, ha avuto due soluzioni differenti. La prima rivolta al browser Microsoft Internet Explorer è stata dettata dalla maggiore diffusione dello stesso, essendo incluso nella maggior parte dei Sistemi Operativi della stessa Microsoft; la seconda è stata operata su Mozilla Firefox essendo questo un programma emergente e sotto molti punti di vista è innovativo, scelto da un numero sempre crescente di utenti come browser principale o anche come alternativa momentanea a quello della Microsoft. Questo tipo di scelta ha dunque richiesto che il codice creato nelle pagine Web, statiche o dinamiche, fosse compatibile con entrambi i software di navigazione. A questo scopo 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 è un Database management system (*DBMS*) relazionale, composto da un client con interfaccia a caratteri e un server,

entrambi disponibili sia per i Sistemi Operativi Unix che per Windows. Supporta la maggior parte della sintassi SQL. Possiede delle interfacce per diversi linguaggi, compreso un driver ODBC, due driver Java e un driver per .NET. Inoltre è sicuro, potente, gratuito.

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

- Servlet Java: cuore dell'applicazione, racchiudono la maggior parte della logica. Il loro funzionamento si basa sulla creazone di un solo processo per ogni Servlet. Al momento della richiesta da parte dell'utente 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, essendo 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): naturale estensione delle Servlet, vengono create come pagine HTML statiche. Successivamente, il Servlet Container si occupa della una trasformazione di questo

tipo di pagina, producendo una determinata Servlet ad essa associata in grado di svolgerne la logica. Scopo principale delle JSP è presentare le informazioni all'utente; la logica con cui si arriva a quel risultato ha due possibili implementazioni: JSP Model 1 e JSP Model 2. La prima 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 delle varie operazioni all'utente; la seconda 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. Nel primo caso, essendo tutto gestito da un solo tipo di oggetto, potrebbero verificarsi problemi nel caso di cambiamenti di grafica legati all'introduzione di scriptlet Java all'interno di codice HTML; nel secondo modello la divisione fra le diverse logiche permette 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.

- 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.

Legato alla scelta della programmazione in Java è l'utilizzo di Eclipse, in particolare della sua versione 3.2.1, 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 di utilizzo. E'stato scelto, non a caso, uno dei suoi plug-in, Exadel Studio, nella versione 4.0.2 che permette una più facile realizzazione di applicazioni web sviluppate utilizzando Struts, e fornisce 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.

Per l'implementazione della funzionalità di esecuzione del file da parte dell'applicazone web, abbiamo scelto di utilizzare:

- JMF (*Java Media Framework*): API create per permettere di incorporare tipi di dati Multimediali in applicazioni o applet Java e sviluppate per supportare i più comuni standard, quali: MPEG-1, MPEG-2, Quick Time, AVI, WAV, AU e MIDI. Usando JMF si ha la possibilità di riprodurre gli standard sopracitati e di gestirli su base temporale, ossia di lavorare sulle sottosequenze di un filmato o sincronizzare la riproduzione di più stream di dati.

- Protocollo RTSP (Real Time Streaming Protocol): protocollo di livello applicativo che si appoggia al Real Time Protocol (RTP) per fornire servizi di streaming multimediale. Permette la trasmissione di un flusso di dati (stream) video e/o audio da una sorgente senza che sia necessario il preventivo scaricamento sul computer dell'intero file contenente il video o il suono.

3.3. Selezione delle metodologie realizzative

Prese in considerazione 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 di Model-View-Controller (MVC). Questo è caratterizzato da tre componenti fondamentali, ognuno dei quali è asservito al compito di gestire una determinata logica: business logic, presentation logic e control logic. Il vantaggio di questa divisione in più logiche, permette la creazione di un codice di più facile realizzazione, manutenzione e riutilizzazione. I tre

componenti del pattern MVC sono:

- Controller: intercetta tutte le richieste che pervengono alla web application, ricava i dati e i parametri che traduce in specifiche operazioni della business logic da effettuare, ed infine decide la successiva vista da presentare all'utente al termine dell'elaborazione della richiesta;
- Model: 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: parte dell'applicazione addetta alla visualizzazione delle informazioni richieste dall'utente, pagina a cui l'utente fa accesso al seguito di una richiesta.

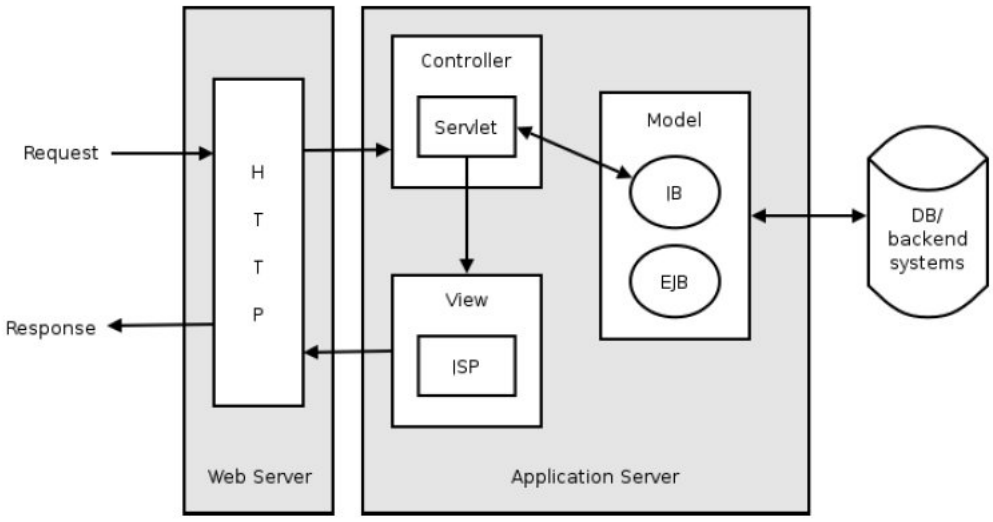


Figura 3.1. - Pattern Model - View - Controller (MVC)

Questo pattern può essere implementato utilizzando il framework Jakarta Struts.

3.4. Il Framework Jakarta Struts

Si definisce framework una serie di classi e interfacce che cooperano per risolvere un determinato tipo di problema software. Definisce il modo in cui le astrazioni fornite dalle classi e dai componenti che lo compongono collaborano per risolvere un problema. I comportamenti di un framework sono riutilizzabili.

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), ed è per questa ragione che è stato definito "*MVC web application framework*".

I componenti fondamentali di Struts sono:

- *ActionServlet*: è la servlet di controllo centralizzata che gestisce tutte le richieste client e smista il flusso applicativo in base alla logica configurata;

- *struts-config.xml*: è il file di configurazione dell'applicazione che letto in fase di start-up permette la definizione delle associazioni tra i vari elementi di Struts. In esso sono, ad esempio, definite le associazioni tra i path delle richieste http e le classi Action ad esse associate, le associazioni tra le Action e gli ActionForm e quelle tra la Action e le ActionForward.;

- *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 automaticamente popolati dal framework con i dati contenuti nelle request http e passati in input alla Action;

- *ActionForward*: contengono i path ai quali la servlet di Struts inoltra il flusso elaborativo in base alla logica dell'applicazione al termine della elaborazione della Action;

- *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.

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.

Poniamo dunque l’attenzione sulla effettiva implementazione secondo la logica Struts del modello MVC, precisando, innanzitutto, che questo è un framework model-neutral, ovvero implementa esclusivamente i livelli di controller e view.

Decidiamo dunque di descriverne i livelli implementati.

3.4.1 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 in base a quali delle due riceva, invoca rispettivamente i metodi doGet() e doPost(), e di conseguenza il metodo process() della ActionServlet. In quest'ultimo metodo ottenuta l'istanza della classe org.apache.struts.Action.RequestProcessor se ne invoca il metodo process() compiendo 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 legge il file struts-config.xml per trovare un elemento <action> corrispondente al path della richiesta ed una volta trovatolo, 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. 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. Se il controllo è ok è invocato il metodo execute() dell'Action cui si delega l'elaborazione della richiesta. Al termine dell'elaborazione, si ottiene 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. La Action si preoccupa di acquisire i dati della request dal form, delegare l'elaborazione della business logic alle classi del Model, acquisire i risultati dell'elaborazione e prepararli per la vista da inviare all'utente mettendoli nello scope opportuno (se necessario), inoltrare il flusso elaborativo in base alla logica applicativa. È il "ponte applicativo" tra il Controller ed il Model di un'applicazione Struts.

Un'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.

3.4.2 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, come ad esempio 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.

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.

3.4.3 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.

4. MODELLO IMPLEMENTATIVO

Anche per il modello implementativo si può affrontare da discussione operando la distinzione tra interfaccia front end ed interfaccia back end.

4.1 Front end

Intraprendiamo la presentazione dell'implementazione dell'interfaccia front end illustrando la scelta di realizzazione dei casi d'uso, ponendo maggiore attenzione a quelli che riguardano la visualizzazione dei dati richiesti dall'utente da parte dell'applicazione.

Registra Nuovo Utente

Si accede a questa form di registrazione cliccando sul tasto apposito posto in fondo alla barra del Login di cui abbiamo parlato in precedenza.

Al momento della registrazione all'utente sono richiesti dati fondamentali ai fini della registrazione quali email e password, e dati che ne aiutino l'identificazione da parte dell'amministratore che dovrà valutare l'inscrizione. Per questa ragione alcuni di questi dati sono ritenuti obbligatori e ne sarà segnalata la necessità nel caso di mancata immissione.

Figura 4.1. - Interfaccia grafica - *Registrazione Utente*

Login

La form di Login si presenta nella pagina iniziale nella barra sottostante il logo dell'applicazione. Sono richiesti come dati di validazione l'email e la password, inseriti al momento della registrazione. (Fig. 4.2a) Nel caso che si verifichi un errore nell'inserimento dei dati, o che ci si fosse dimenticati la password, viene visualizzata una nuova pagina contenente la form utile per il recupero della password nella quale si deve inserire l'email (Fig. 4.2b). Se la richiesta di login va a buon fine, verrà visualizzato un messaggio di benvenuto (Fig. 4.2c)

Logout

Questa funzionalità è rappresentata da un pulsante posizionato in una cella posizionata sopra il menù laterale. (Fig. 4.2c)

Figura 4.2a. - Login**:** *Form di inserimento*

Figura 4.2b. - Login**:** *Messaggio di errore e recupero Password*

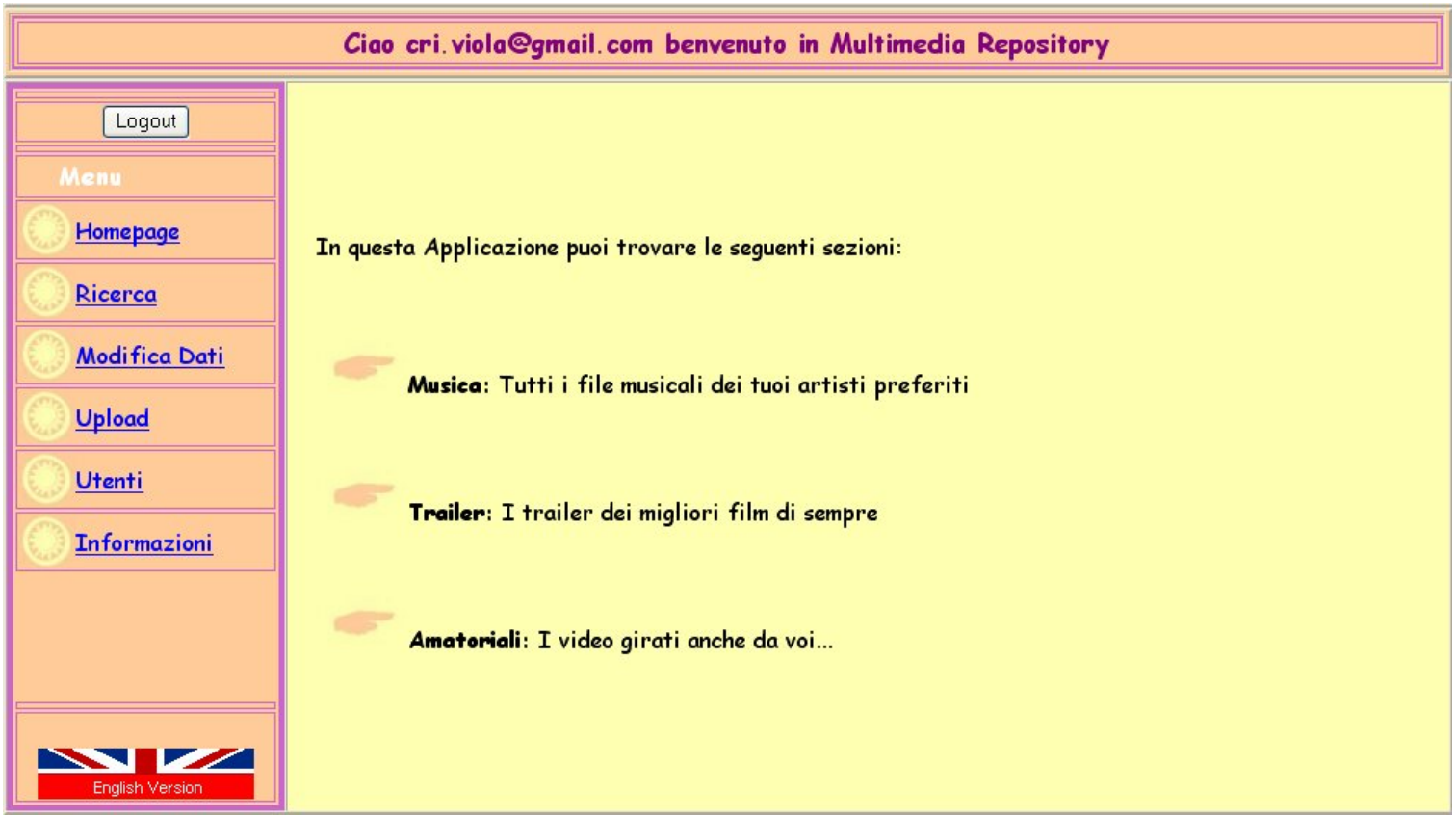


Figura 4.2c. - Login**:** *Interfaccia di benvenuto*

Modifica Dati

L'interfaccia per la modifica dei dati è costituita da due parti: la prima è una tabella nella quale sono visualizzati i dati inseriti fino a quel momento; la seconda è costituita dalla sequenza di form di inserimento. In particolare l'operazione di modifica deve essere eseguita selezionando attraverso un check i dati che si vogliono modificare.

Figura 4.3. - *Interfaccia modifica dati utente*

Inserisci File

L'interfaccia per l'inserimento dei dati è costituita dalla sequenza di form di inserimento per i dati associati ai file, quali *Titolo*, *Autore*, *Album*, *Genere*, *Anno*, *Lingua*.

Figura 4.4. - *Interfaccia inserimento dati*

Ricerca File

Questa funzionalità prevede l'utilizzo di due interfacce separate, l'una destinata alla ricerca vera e propria; l'altra alla visualizzazione dei dati ricercati.

L'interfaccia di ricerca prevede la possibilità di recuperare tutti i file contenuti nell'applicazione, oppure una selezione degli stessi in funzione dei dati associati (Fig. 4.5a).

La visualizzazione avviene attraverso una sequenza numerata di tabelle, una per ognuno dei file, ordinate per Titolo (Fig. 4.5b). Ogni singola tabella è organizzata nel seguente modo:

- nella prima riga è visualizzato il titolo del file;
- nella seconda sono riportati nelle diverse celle le denominazioni dei dati associati al file, quali *Autore*, *Album*, *Genere*, *Anno*, *Lingua*;
- nella terza riga i veri e propri dati (nel caso di mancato inserimento comparirà la scritta *Non Inserito*).

Per la gestione dei file ricercati è disponibile una serie di tasti che nell'ordine sono:

- *Esegui File* (che compare nel caso di file mpeg e solo per gli Utenti Registrato ed Administrator);
- *Modifica* (che compare solo per gli Administrator);
- *Cancella* (che compare solo per gli Administrator);
- *Download* (che compare solo per gli Administrator);
- *Vedi Commenti* (che compare solo per gli Utenti Registrato ed Administrator).

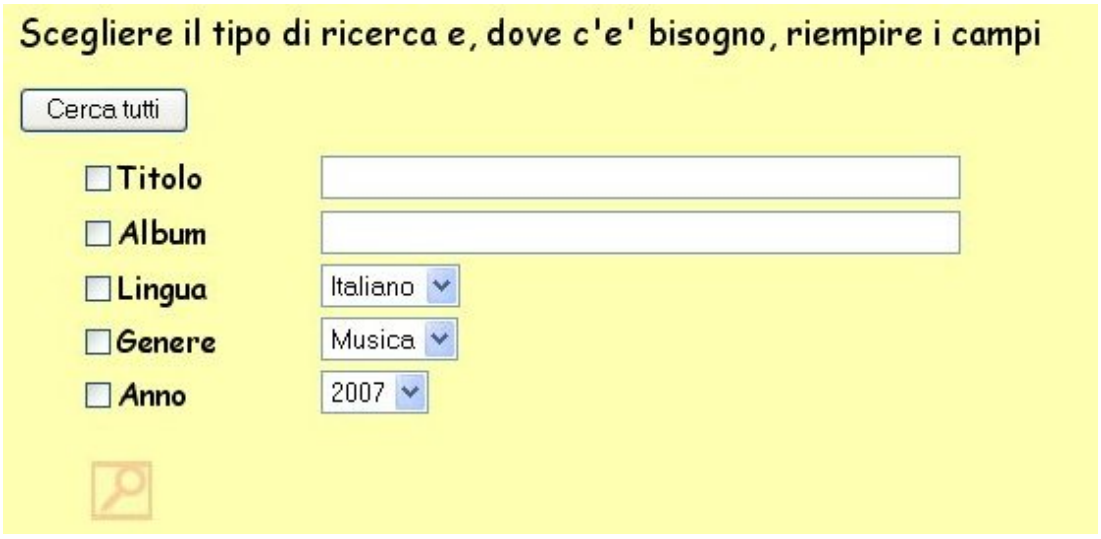


Figura 4.5a. - *Interfaccia di ricerca*

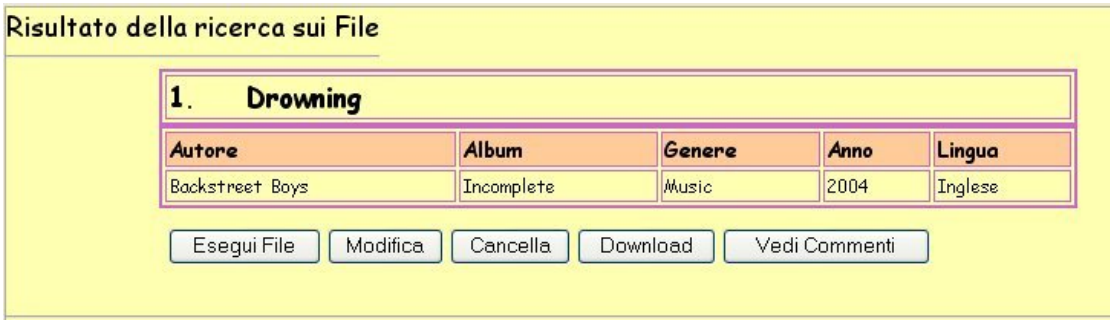


Figura 4.5b. - *Interfaccia per la visualizzazione dei dati ricercati*

Esegui File

Questa funzionalità prevede la redirezione alla pagina predisposta per la visualizzazione del file, di cui parleremo più avanti.

Figura 4.6. - *Interfaccia per la visualizzazione del file*

Modifica File

Questa funzionalità prevede la redirezione alla interfaccia predisposta per la modifica dei dati, che contiene la sequenza delle form di inserimento dei dati associati ai file. In particolare l'operazione di modifica deve essere eseguita selezionando attraverso un check i dati che si vogliono modificare.

Figura 4.7. - *Interfaccia per la modifica dei dati*

Download File

Questa funzionalità prevede la visualizzazione di una finestra di dialogo che consente la scelta del percorso in cui salvare il file.

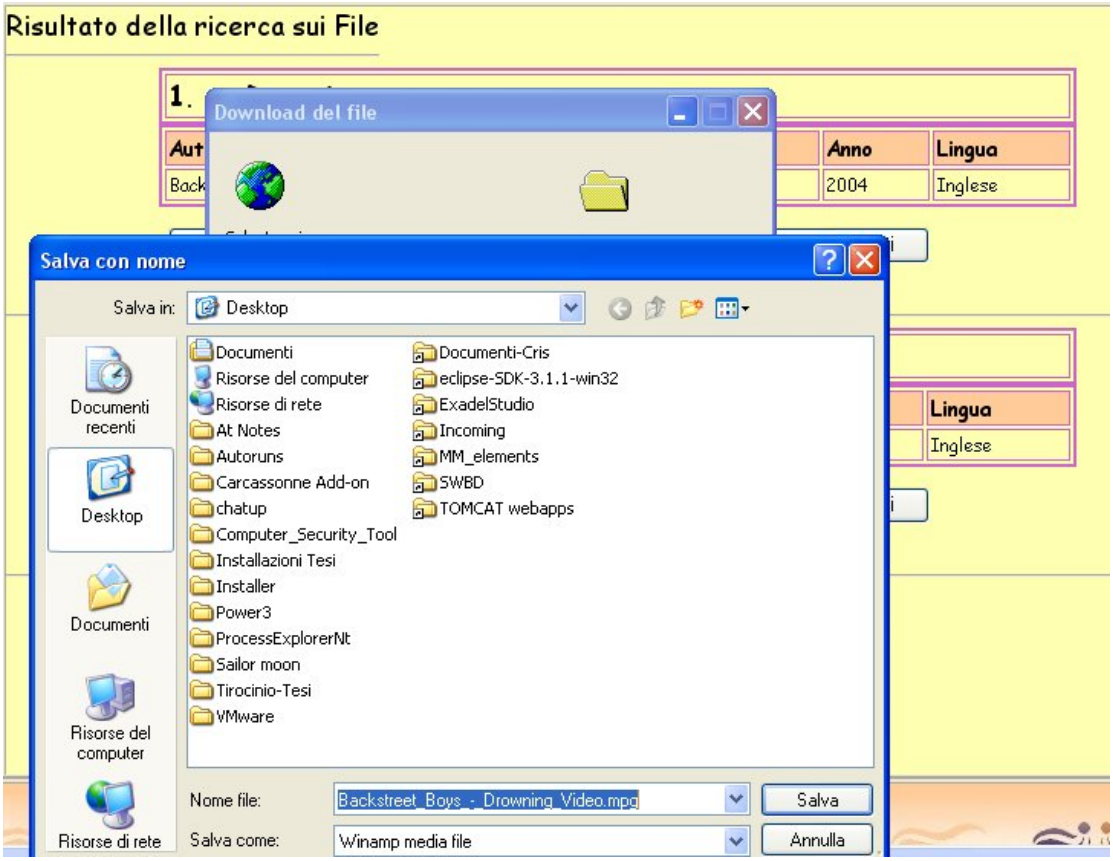


Figura 4.8. - *Interfaccia di Download*

Cancella File

Questa funzionalità consente la cancellazione del file selezionato.

Inserisci Commento - Cancella Commento

Le due funzionalità legate all'intervento sui commenti sono inserite nella stessa interfaccia, raggiungibile grazie al tasto *Vedi Commenti*. La prima è costituita da una semplice form che ha come unico campo *Commento*. La seconda è invece associata al tasto *Cancella*.

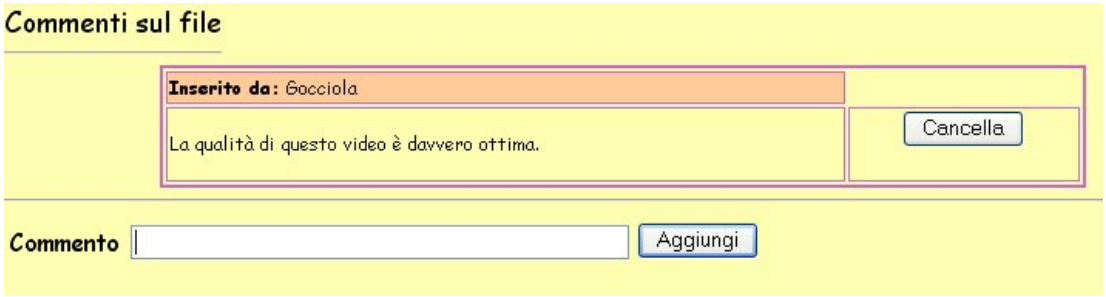


Figura 4.9. - *Interfaccia per la gestione dei commenti*

Ricerca Utente

Questa funzionalità prevede l'utilizzo di due interfacce separate, l'una destinata alla ricerca vera e propria; l'altra alla visualizzazione dell'elenco degli utenti registrati o in fase di registrazione.

L'interfaccia di ricerca prevede la possibilità di recuperare le informazioni associate a tutti gli utenti, oppure ad una selezione degli stessi in funzione dei dati associati (Fig. 4.10a).

La visualizzazione avviene attraverso una sequenza numerata di tabelle, una per utente, ordinate per email (Fig. 4.10b). Ogni singola tabella è organizzata nel seguente modo:

- nella prima riga è visualizzata l'email dell'utente;
- nella seconda sono riportati nelle diverse celle le

denominazioni dei dati associati all'utente, quali *Nickname*, *Cognome*, *Nome*, *Data di nascita*, *Luogo di nascita*, *Sesso*, *Tipo*;

- nella terza riga i veri e propri dati (nel caso di mancato inserimento comparirà la scritta *Non Inserito*).

La gestione degli utenti ricercati è differenziata a seconda che gli utenti siano registrati (compare in coda alla tabella la scritta *registrato*) , oppure in attesa di registrazione (compare in coda alla tabella una form di selezione che consente la registrazione degli utenti in attesa) sono disponibili una serie di tasti che nell'ordine sono:

- *Registra*;
- *Modifica*;
- *Cancella*.

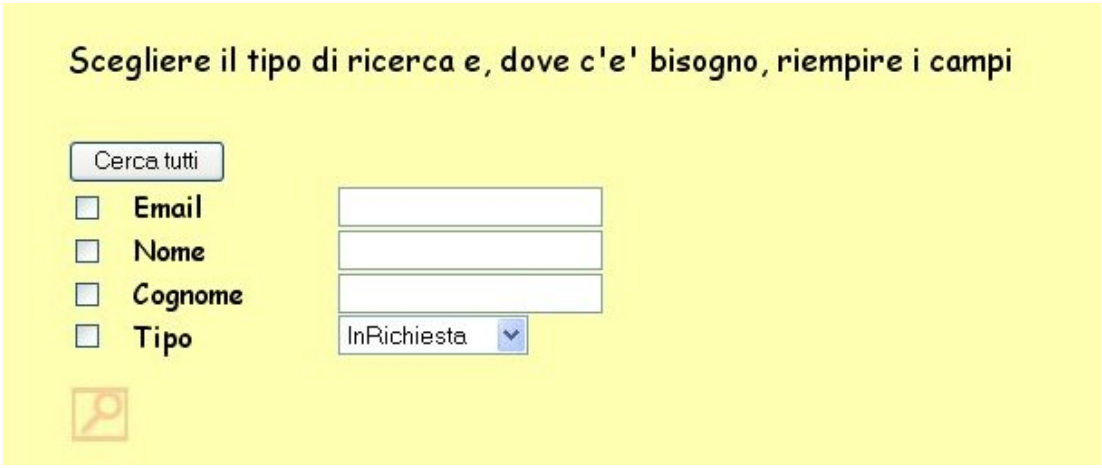


Figura 4.10a. - *Interfaccia per la ricerca*

Figura 4.10a. - *Interfaccia per la gestione*

Modifica Utente

Questa interfaccia prevede la visualizzazione di una tabella, organizzata nel seguente modo:

- nella prima riga è visualizzata l'email dell'utente;
- nella seconda sono riportati i seguenti campi: *Nome*, *Cognome*, *Tipo*, *Nuovo Tipo*;
- nella terza riga sono visualizzati i dati utenti (nel campo *Nuovo Tipo* è possibile la modifica).

Figura 4.11. - *Interfaccia per la modifica del Tipo*

Cancella Utente

Questa funzionalità consente la cancellazione dell'utente selezionato.

4.2 Back end

Continuando la definizione dell'implementazione della nostra applicazione, volgiamo la nostra attenzione verso l'interfaccia back end.

4.2.1 Configurazione dell'applicazione

Il framework Struts prevede, per la configurazione di una Web Application, l'utilizzo di due file: web.xml e struts-config.xml.

Web.xml

Il file web.xml, denominato anche descrittore di deploy, definisce ogni servlet e pagina JSP all'interno di una Web Application, va inserito nella directory WEB-INF nella Web Application ed è utilizzato dal web container, quando viene avviato, per configurare e caricare l'applicazione cui si riferisce. Il tipo dei tag utilizzabili in questo tipo di file sono definiti per mezzo di un Document Type Definition(DTD). Per maggiori informazioni riguardo le specifiche delle Servlet Java è possibile consultare il seguente link: http://java.sun.com/dtd/index.html.

Riportiamo di seguito il file web.xml rispetto alle possibili

modalità di interazione che il plug-in di Eclipse, Exadel Studio, ci mette a disposizione: modalità ad albero e versione testuale.

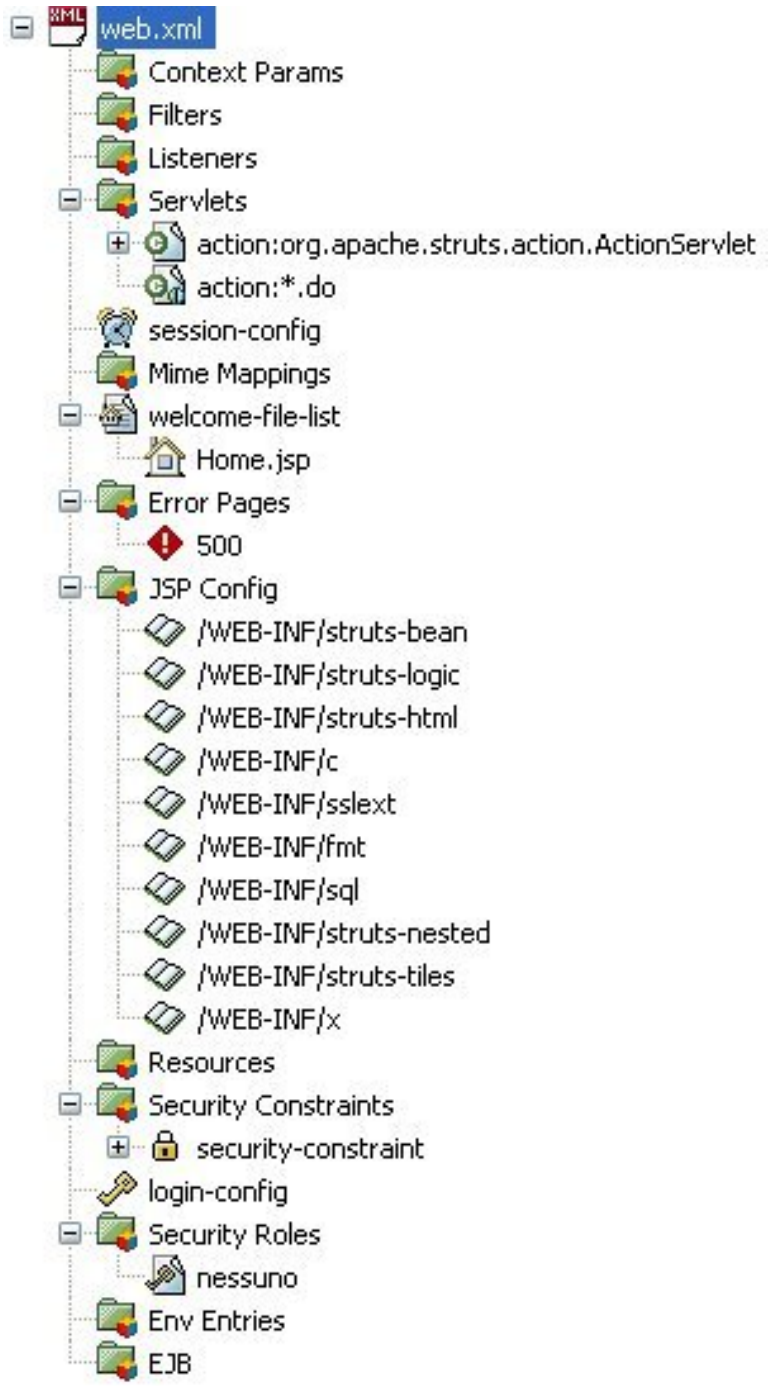


Figura 4.12. - Web.xml *: Visualizzazione ad albero*

```
<?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>MultimediaRepository</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>
 <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/c</taglib-uri>
 <taglib-location>/WEB-INF/c.tld</taglib-location>
</taglib>
<taglib>
 <taglib-uri>/WEB-INF/sslext</taglib-uri>
 <taglib-location>/WEB-INF/sslext.tld</taglib-location>
</taglib>
<taglib>
 <taglib-uri>/WEB-INF/fmt</taglib-uri>
 <taglib-location>/WEB-INF/fmt.tld</taglib-location>
</taglib>
<taglib>
 <taglib-uri>/WEB-INF/sql</taglib-uri>
 <taglib-location>/WEB-INF/sql.tld</taglib-location>
</taglib>
<taglib>
 <taglib-uri>/WEB-INF/struts-nested</taglib-uri>
 <taglib-location>/WEB-INF/struts-nested.tld</taglib-location>
</taglib>
<taglib>
 <taglib-uri>/WEB-INF/struts-tiles</taglib-uri>
 <taglib-location>/WEB-INF/struts-tiles.tld</taglib-location>
</taglib>
<taglib>
 <taglib-uri>/WEB-INF/x</taglib-uri>
 <taglib-location>/WEB-INF/x.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 4.13. - Web.xml *: Versione testuale*

Procediamo ora, facendo riferimento alle modalità testuale, con la descrizione del file e delle varie voci che lo compongono e che ci hanno permesso la configurazione della nostra applicazione.

Le prime righe presentano informazioni sulla versione, sulla codifica e sul tipo di documento DTD utilizzato.

Nel blocco <servlet> di configurazione standard della ActionServlet, settato ad 1, figura il parametro <load-on-startup> in base al quale il container istanzia la ActionServlet allo start-up della web-application e ne invoca il metodo init(). Inoltre mediante il parametro config è specificata la posizione del file XML di configurazione dell'applicazione (struts-config.xml). Il parametro debug abilita il debugging dell'applicazione. Mediante il blocco <servlet-mapping> si specifica che tutte le richieste con path terminante in .do vengono mappate sulla ActionServlet, servlet di controllo di Struts. Nel tag <welcome-file-list></welcome-file-list> viene indicata la pagina iniziale

dell’applicazione (Home.jsp). Nel tag <taglib></taglib> vengono specificate le librerie di tag JSP utilizzate nel progetto. Nel tag <error-page> sono indicati la pagina da inviare al client quando il server da un determinato codice di errore; noi abbiamo indicato 500 come codice di errore, ciò determina che all’accesso alle pagine protette all’interno di /protected il server redireziona il client sulla home page. Nel tag <security-constraint></security-constraint> si specifica 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.

Struts-config.xml

Nel file struts-config.xml è contenuta la configurazione dell’intera applicazione. È letto in fase di start-up dalla ActionServlet e definisce le associazioni tra i vari elementi di Struts. Definisce le associazioni tra le Action e gli ActionForm, che vengono automaticamente popolate dal framework con i dati della richiesta ad essi associata e passati in input alla Action e quelle tra le Action e le ActionForward, ovvero i path ai quali la ActionServlet redirigerà il flusso applicativo al termine della

elaborazione della Action.

L'utilizzo di Exadel Studio ci fornisce la possibilità di creare e modificare il file in tre modalità distinte: modalità ad albero, modalità grafica e modalità testuale.

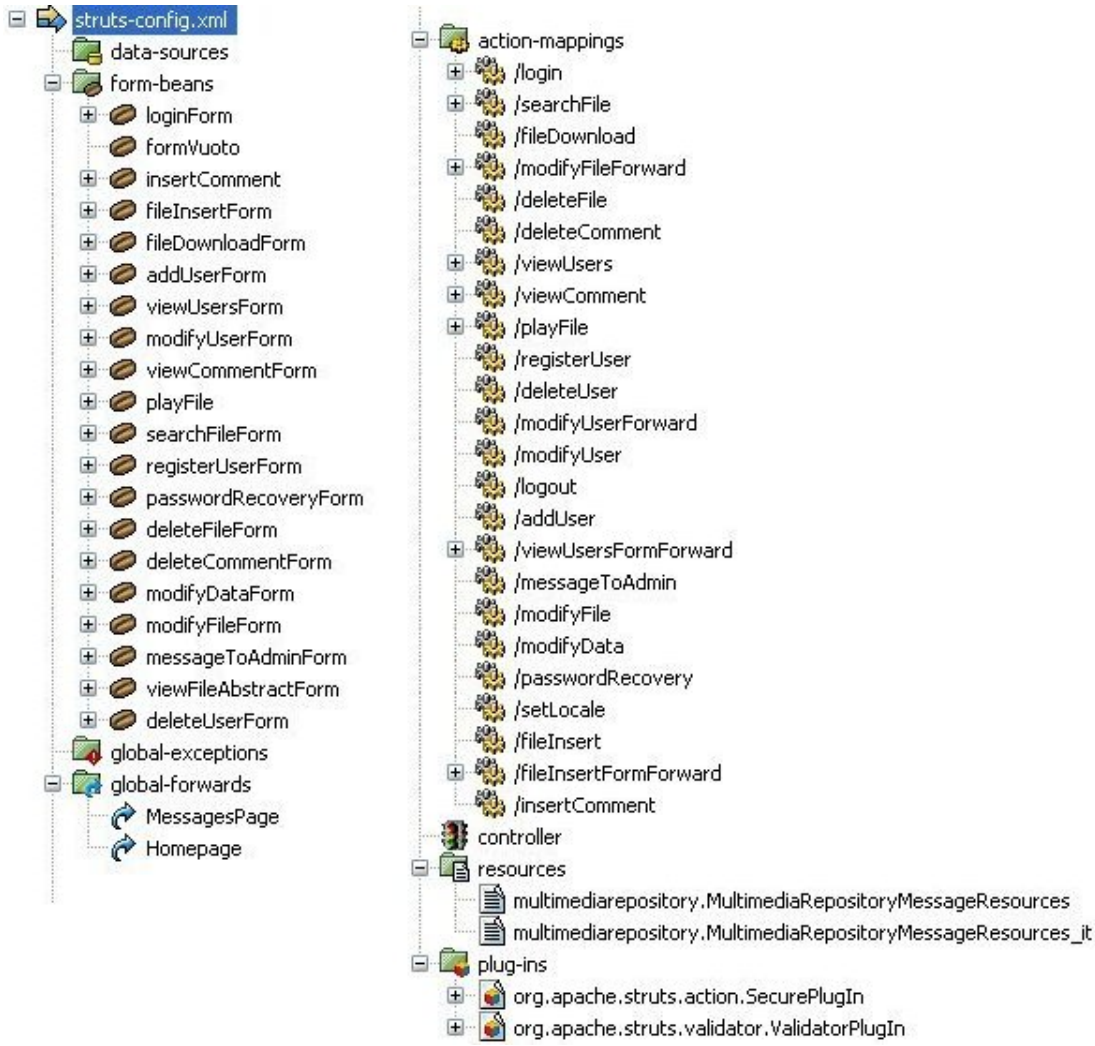
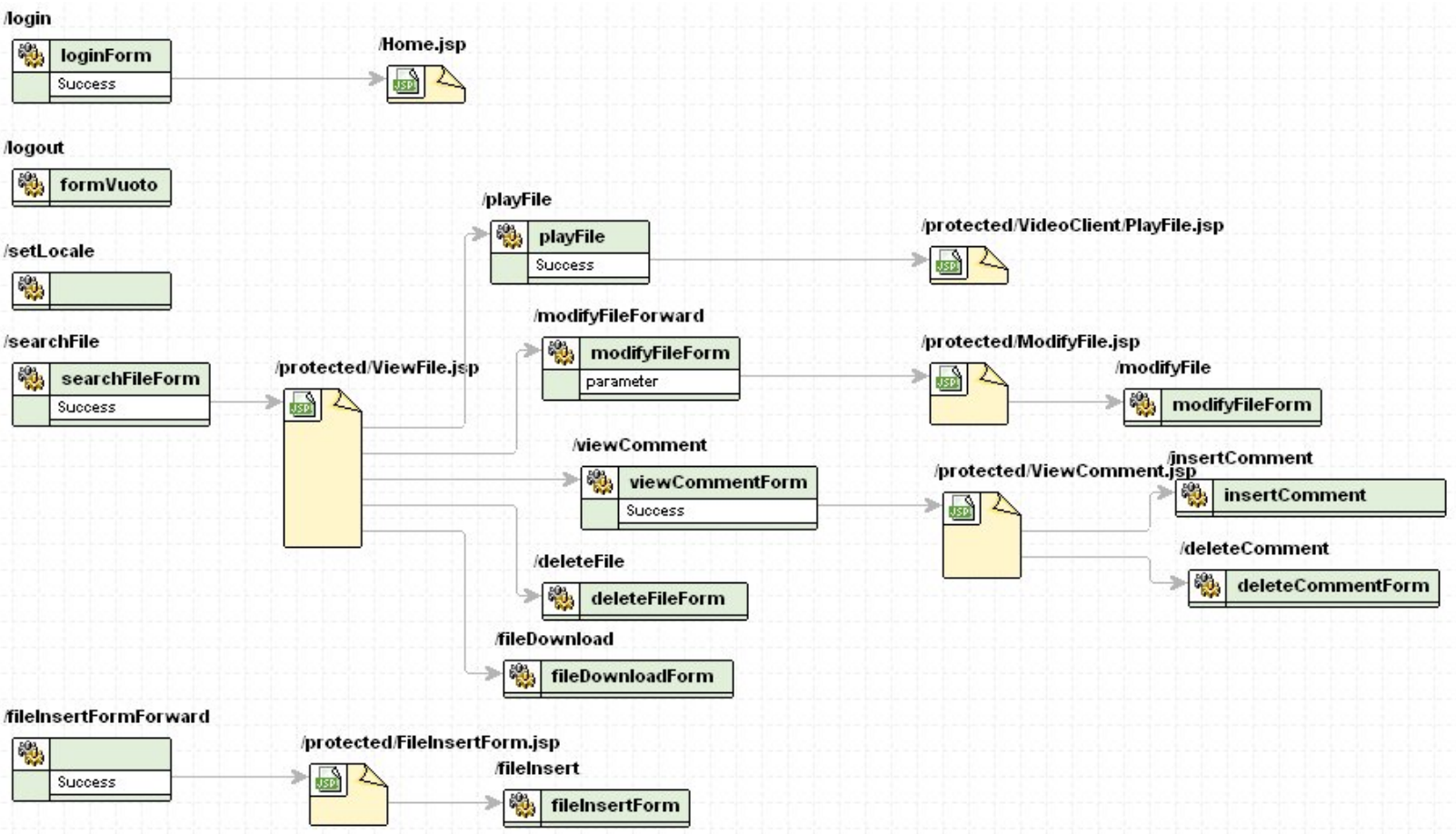


Figura 4.14. - Struts-config.xml *: Modalità ad albero*



↓ continua a pagina successiva ↓

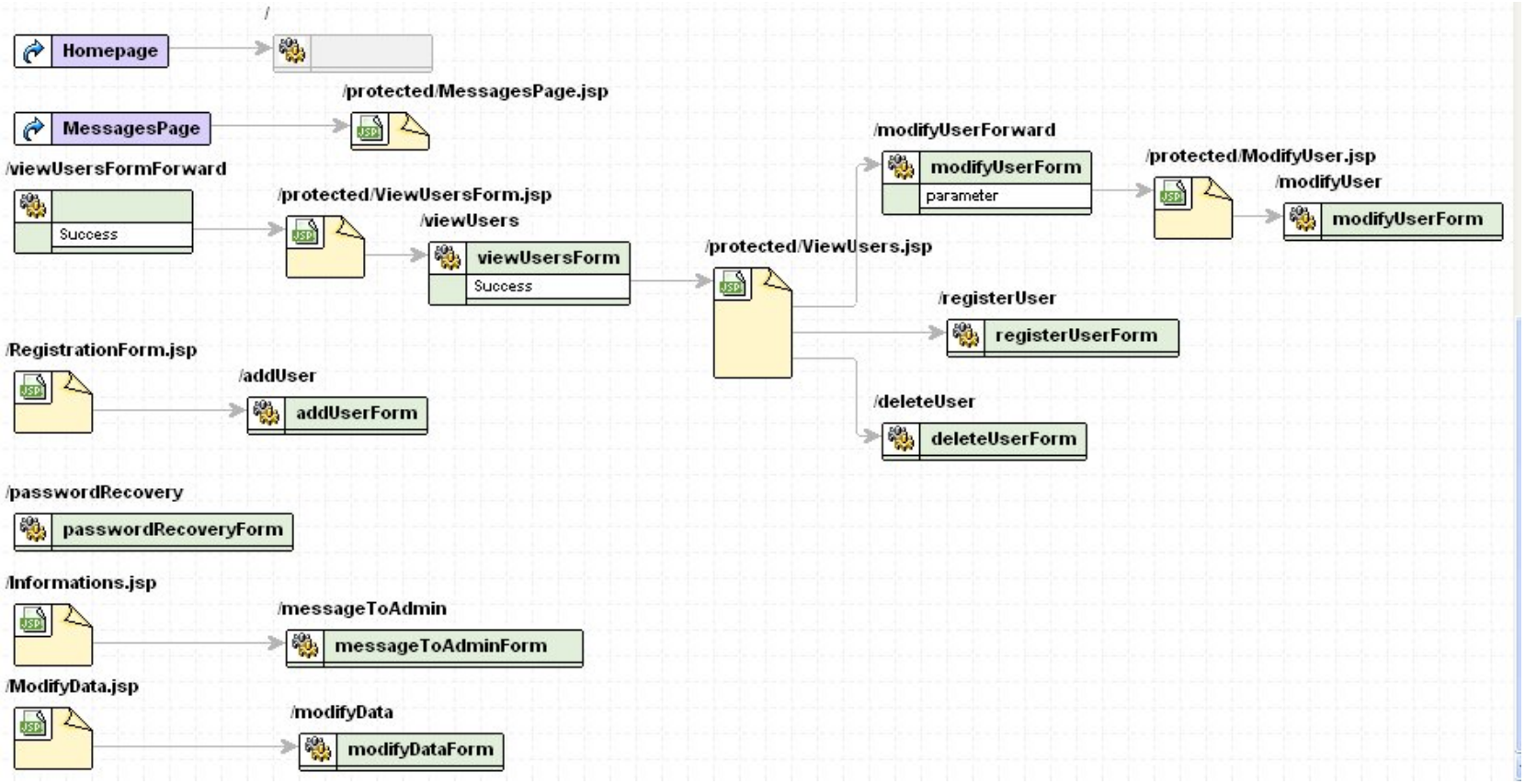


Figura 4.15. - Struts-config.xml *: Modalità grafica*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD
Struts Configuration 1.1//EN"

"http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">
<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="insertComment"
type="org.apache.struts.action.DynaActionForm">
   <form-property name="id_file" type="java.lang.Integer"/>
   <form-property name="commento" type="java.lang.String"/>
   <form-property name="ute_email" type="java.lang.String"/>
  </form-bean>
  <form-bean name="fileInsertForm"
type="org.apache.struts.validator.DynaValidatorForm">
   <form-property name="genere" type="java.lang.String"/>
   <form-property name="titolo" type="java.lang.String"/>
   <form-property name="autore" type="java.lang.String"/>
   <form-property name="anno" type="java.lang.Integer"/>
   <form-property name="album" type="java.lang.String"/>
   <form-property name="lingua" type="java.lang.String"/>
   <form-property name="file"
type="org.apache.struts.upload.FormFile"/>
  </form-bean>
  <form-bean name="fileDownloadForm"
type="org.apache.struts.action.DynaActionForm">
   <form-property name="codice" type="java.lang.Integer"/>
  </form-bean>
```

```
  <form-bean name="addUserForm"
type="multimediarepository.AddUserForm">
    <form-property name="password" type="java.lang.String"/>
    <form-property name="email" type="java.lang.String"/>
    <form-property name="cognome" type="java.lang.String"/>
    <form-property name="password2" type="java.lang.String"/>
    <form-property name="foto"
type="org.apache.struts.upload.FormFile"/>
    <form-property name="nome" type="java.lang.String"/>
    <form-property name="luogonascita" type="java.lang.String"/>
  </form-bean>
  <form-bean name="viewUsersForm"
type="multimediarepository.ViewUsersForm">
    <form-property name="luogonascita" type="java.lang.String"/>
    <form-property name="email" type="java.lang.String"/>
    <form-property initial="false" name="ricercapernome"
type="java.lang.Boolean"/>
    <form-property initial="true" name="registered"
type="java.lang.Boolean"/>
    <form-property name="datanascita" type="java.lang.Integer"/>
    <form-property name="nome" type="java.lang.String"/>
    <form-property name="tipo" type="java.lang.String"/>
    <form-property name="ricercaperemail" type="java.lang.Boolean"/>
    <form-property initial="false" name="ricercaperregistered"
type="java.lang.Boolean"/>
    <form-property initial="false" name="ricercapertipo"
type="java.lang.Boolean"/>
    <form-property name="cognome" type="java.lang.String"/>
    <form-property name="ricercapercognome"
type="java.lang.Boolean"/>
    <form-property name="ricercatutto" type="java.lang.Boolean"/>
  </form-bean>
  <form-bean name="modifyUserForm"
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="viewCommentForm"
type="org.apache.struts.action.DynaActionForm">
    <form-property name="id_file" type="java.lang.Integer"/>
  </form-bean>
  <form-bean name="playFile"
type="org.apache.struts.action.DynaActionForm">
    <form-property name="codice" type="java.lang.Integer"/>
  </form-bean>
  <form-bean name="searchFileForm"
type="multimediarepository.SearchFileForm">
    <form-property name="titolo" type="java.lang.String"/>
    <form-property name="codice" type="java.lang.Integer"/>
    <form-property initial="null" name="anno"
type="java.lang.Integer"/>
    <form-property name="lingua" type="java.lang.String"/>
    <form-property name="ricercaperlingua" type="java.lang.Boolean"/>
    <form-property name="ricercatutto" type="java.lang.Boolean"/>
    <form-property name="ricercaperanno" type="java.lang.Boolean"/>
    <form-property name="autore" type="java.lang.String"/>
    <form-property initial="false" name="ricercapertitolo"
type="java.lang.Boolean"/>
    <form-property name="fileabstract" type="java.lang.String"/>
  </form-bean>
  <form-bean name="registerUserForm"
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="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="codice" type="java.lang.Integer"/>
    <form-property name="tipo" type="java.lang.String"/>
  </form-bean>
```

```
  <form-bean name="deleteCommentForm"
type="org.apache.struts.action.DynaActionForm">
    <form-property name="tipo" type="java.lang.String"/>
    <form-property name="id" type="java.lang.Integer"/>
  </form-bean>
  <form-bean name="modifyDataForm"
type="org.apache.struts.validator.DynaValidatorForm">
    <form-property name="email" type="java.lang.String"/>
    <form-property name="cognome" type="java.lang.String"/>
    <form-property name="nome" type="java.lang.String"/>
    <form-property name="luogonascita" type="java.lang.String"/>
    <form-property name="anno" type="java.lang.Integer"/>
    <form-property name="mese" type="java.lang.Integer"/>
    <form-property name="giorno" type="java.lang.Integer"/>
    <form-property name="modificacognome" type="java.lang.Boolean"/>
    <form-property name="modificanome" type="java.lang.Boolean"/>
    <form-property name="modificaluogonascita"
type="java.lang.Boolean"/>
    <form-property name="modificadatanascita"
type="java.lang.Boolean"/>
  </form-bean>
  <form-bean name="modifyFileForm"
type="org.apache.struts.validator.DynaValidatorForm">
    <form-property name="titolo" type="java.lang.String"/>
    <form-property initial="false" name="modificaanno"
type="java.lang.Boolean"/>
    <form-property initial="false" name="modificafile"
type="java.lang.Boolean"/>
    <form-property initial="false" name="modificaalbum"
type="java.lang.Boolean"/>
    <form-property name="genere" type="java.lang.String"/>
    <form-property name="codice" type="java.lang.Integer"/>
    <form-property name="file"
type="org.apache.struts.upload.FormFile"/>
    <form-property initial="false" name="modificaautore"
type="java.lang.Boolean"/>
    <form-property name="lingua" type="java.lang.String"/>
```

```
    <form-property name="anno" type="java.lang.Integer"/>
    <form-property initial="false" name="modificatitolo"
type="java.lang.Boolean"/>
    <form-property name="album" type="java.lang.String"/>
    <form-property initial="false" name="modificalingua"
type="java.lang.Boolean"/>
    <form-property initial="false" name="modificagenere"
type="java.lang.Boolean"/>
    <form-property name="autore" type="java.lang.String"/>
   </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="viewFileAbstractForm"
type="org.apache.struts.action.DynaActionForm">
    <form-property name="codice" type="java.lang.String"/>
    <form-property name="fileabstract" 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/>
  <global-forwards>
   <forward name="MessagesPage" path="/protected/MessagesPage.jsp"/>
   <forward name="Homepage" path="/" redirect="true"/>
  </global-forwards>
  <action-mappings
type="org.apache.struts.config.SecureActionConfig">
   <action input="/LoginError.jsp" name="loginForm" path="/login"
    scope="request" type="multimediarepository.LoginAction"
validate="true">
    <set-property property="secure" value="true"/>
    <forward name="Success" path="/Home.jsp" redirect="true"/>
   </action>
```

```
  <action input="/SearchFileForm.jsp" name="searchFileForm"
   path="/searchFile" type="multimediarepository.SearchFileAction"
validate="true">
   <forward name="Success" path="/protected/ViewFile.jsp"/>
  </action>
  <action input="/protected/ViewFile.jsp" name="fileDownloadForm"
   path="/fileDownload"
type="multimediarepository.FileDownloadAction"/>
  <action name="modifyFileForm"
parameter="/protected/ModifyFile.jsp"
   path="/modifyFileForward"
type="multimediarepository.AdministratorForwardAction">
   <forward name="parameter" path="/protected/ModifyFile.jsp"/>
  </action>
  <action name="deleteFileForm" path="/deleteFile"
type="multimediarepository.DeleteFileAction"/>
  <action input="/protected/ViewComment.jsp"
name="deleteCommentForm"
   path="/deleteComment"
type="multimediarepository.DeleteCommentAction"/>
  <action input="/protected/ViewUsersForm.jsp" name="viewUsersForm"
   path="/viewUsers" scope="request"
type="multimediarepository.ViewUsersAction">
   <forward name="Success" path="/protected/ViewUsers.jsp"/>
  </action>
  <action input="/protected/ViewFile.jsp" name="viewCommentForm"
   path="/viewComment" scope="request"
type="multimediarepository.ViewCommentAction">
   <forward name="Success" path="/protected/ViewComment.jsp"/>
  </action>
  <action input="/protected/ViewFile.jsp" name="playFile"
   path="/playFile" scope="request"
type="multimediarepository.PlayFile">
   <forward name="Success"
path="/protected/VideoClient/PlayFile.jsp"/>
  </action>
  <action input="/protected/ViewUsers.jsp" name="registerUserForm"
```

```
    path="/registerUser" scope="request"
type="multimediarepository.RegisterUserAction"/>
  <action name="deleteUserForm" path="/deleteUser"
type="multimediarepository.DeleteUserAction"/>
  <action name="modifyUserForm"
parameter="/protected/ModifyUser.jsp"
    path="/modifyUserForward"
type="org.apache.struts.actions.ForwardAction"/>
  <action name="modifyUserForm" path="/modifyUser"
type="multimediarepository.ModifyUserAction"/>
  <action name="formVuoto" path="/logout"
type="multimediarepository.LogoutAction"/>
  <action input="/RegistrationForm.jsp" name="addUserForm"
    path="/addUser" type="multimediarepository.AddUserAction"
validate="true"/>
  <action parameter="/protected/ViewUsersForm.jsp"
    path="/viewUsersFormForward"
type="multimediarepository.AdministratorForwardAction">
    <set-property property="secure" value="false"/>
    <forward name="Success" path="/protected/ViewUsersForm.jsp"/>
  </action>
  <action input="/Informations.jsp" name="messageToAdminForm"
    path="/messageToAdmin"
    type="multimediarepository.MessageToAdminAction"
validate="true"/>
  <action input="/protected/ModifyFile.jsp" name="modifyFileForm"
    path="/modifyFile" type="multimediarepository.ModifyFileAction"
validate="true"/>
  <action input="/ModifyData.jsp" name="modifyDataForm"
  path="/modifyData" type="multimediarepository.ModifyDataAction"
validate="true"></action>
  <action input="/LoginError.jsp" name="passwordRecoveryForm"
    path="/passwordRecovery" scope="request"
    type="multimediarepository.PasswordRecoveryAction"
validate="true"/>
  <action path="/setLocale"
type="multimediarepository.SetLocaleAction"/>
```

```
  <action input="/protected/FileInsertForm.jsp"
name="fileInsertForm"
   path="/fileInsert" scope="request"
   type="multimediarepository.FileInsertAction" validate="true"/>
  <action parameter="/protected/FileInsertForm.jsp"
   path="/fileInsertFormForward"
type="multimediarepository.AdministratorForwardAction">
   <forward name="Success" path="/protected/FileInsertForm.jsp"/>
  </action>
  <action input="/protected/ViewComment.jsp" name="insertComment"
   path="/insertComment" scope="request"
   type="multimediarepository.InsertCommentAction" validate="true"/>
 </action-mappings>
 <controller locale="true"/>
 <message-resources null="false"
parameter="multimediarepository.MultimediaRepositoryMessageResources
"/>
 <message-resources key="it" null="false"
parameter="multimediarepository.MultimediaRepositoryMessageResources
_it"/>
 <plug-in className="org.apache.struts.action.SecurePlugIn">
  <set-property property="httpsPort" value="8443"/>
  <set-property property="addSession" value="true"/>
  <set-property property="enable" value="true"/>
  <set-property property="httpPort" value="8080"/>
 </plug-in>
 <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 4.16. - Struts-config.xml *: Versione testuale*

La modalità grafica risulta molto utile per capire i collegamenti tra i vari oggetti, mentre la definizione di sorgenti dati, plug-in, form, viene

operata servendosi delle altre due modalità di visualizzazione.
Anche in questo caso i tag e i parametri inseribili all'interno del file sono specificati mediante un DTD di riferimento, per maggiori chiarimenti rimandiamo al sito della fondazione Apache (http://jakarta.apache.org/commons/dtds/struts-config_1_1.dtd).

Come già detto possiamo renderci conto delle pagine jsp, delle action e dei link, che le mettono in relazione, utilizzate nell'applicazione attraverso la modifica grafica.
Facendo invece riferimento alla modalità testuale procediamo con la presentazione dei vari elementi. I tag <form-bean></form-bean> ci permettono di definire gli ActionForm, specificando per ognuno il nome e il tipo, corrispondente alla classe Java che estende la ActionForm. Il tag <action-mappings></action-mappings> permette di descrivere per ogni action 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; questo facilita le modifiche dell'applicazione infatti se in un secondo momento si volesse cambiare un link, sarebbe sufficiente cambiare il nome della pagina in tale file, lasciando inalterato il nome simbolico. Deve, inoltre, 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 il tag controller, definito quando viene creato un proprio RequestProcessor, o comunque quando ne viene utilizzato uno diverso da quello di default, e quello plug-in, utilizzato per aggiungere le funzionalità che vengono caricate all'avvio dell'applicazione e distrutte al termine, in modo da non doverle aggiungere in maniera permanente nel codice.

4.2.2 Implementazione della funzionalità di esecuzione del file

L'esecuzione del file si basa sulla comunicazione tra un client RTSP ed un server RTSP.

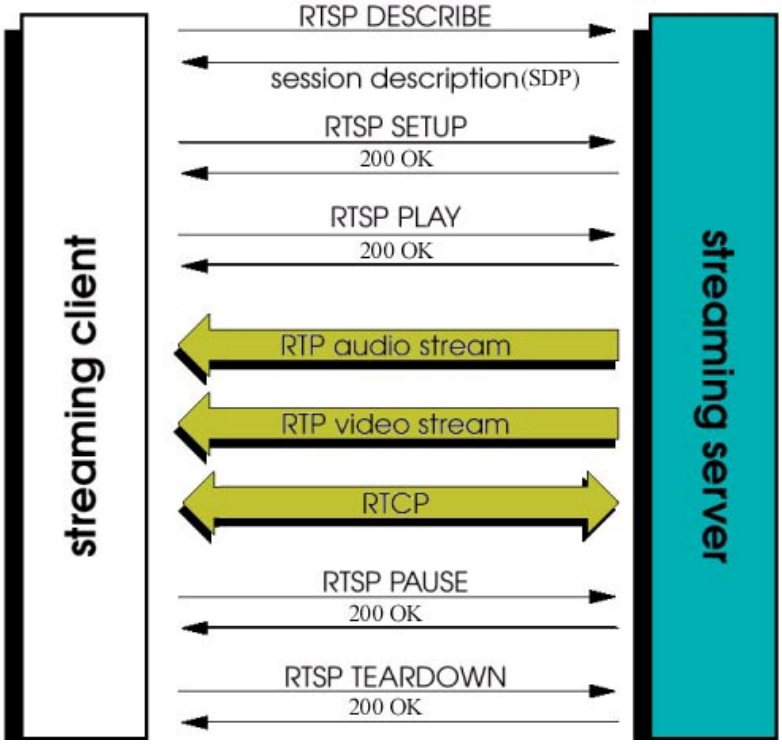


Figura 4.17. - Protocollo RTSP*: Comunicazione client/server*

Come già detto in precedenza, questa funzionalità prevede la pressione del tasto “Esegui File” visualizzato nella pagina dei risultati della ricerca. Poiché la comunicazione da noi implementata permetteva la riproduzione da parte di un Player dei soli file mpeg, abbiamo imposto un controllo sul tipo del file, in modo che il pulsate venisse visualizzato solo nel caso che il file fosse effettivamente riproducibile.

Procediamo ora con la presentazione della funzionalità, eseguendo una distinzione tra le operazioni rimandate al Server e quella rimandate al Client.

Lato Server

Le operazione lato Server vedono l’intervento di tre classi interagenti tra loro. Innanzitutto l’attenzione si sposta una action Play File, cui viene passato come parametro di ingresso il codice del file da riprodurre. In essa vengono inoltre definiti l’indirizzo IP Multicast scelto per la comunicazione, e il numero di porta attraverso la quale saranno inviati gli stream. Vista la possibilità di molteplici richieste è stato, dunque, implementato un algoritmo per il calcolo di un nuovo numero di porta per le diverse comunicazioni. Vista la necessità di agganciare il client sulla stessa porta, ossia metterlo in ascolto su di essa, questo parametro sarà settato come attributo della request e successivamente recuperato.

Recuperato il valore del codice dalla form viene compiuta l’interrogazione al database, per il recupero del file di interesse, e generato il file che dovrà essere inviato.

Segue la dichiarazione di un oggetto di tipo Server, al costruttore

del quale sono passati i seguenti parametri: HttpServletRequest request, HttpServletResponse response, String porta, che individua il numero di porta per la comunicazione, String IndIP, indirizzo multicast, precedentemente definito, e String link, che individua il link associato al file precedentemente creato.
È operato, dunque, il forward verso la JSP preposta alla visualizzazione del file.

La classe Server è un thread. Questo ci consente di istanzianla senza doverne attendere la terminazione e quindi poter proseguire le operazioni d interesse. La stessa istanzia, a sua volta, un oggetto di tipo AVTrasmit, vero responsabile della comunicazione, al costruttore del quale passa come parametri un oggetto di tipo MediaLocator, che individua il percorso per raggiungere il file, l'IndIP, la porta e un Format, a cui sarà successivamente associato in format del file per valutarne la sopportabilità. Su questa istanza viene successivamente invocato il metodo .start(), che consente l'avvio della trasmissione dei dati.

Lato Client

Le operazioni lato Client hanno inizio al momento della redirezione verso la pagina PlayFile.jsp. In questa è previsto, inizialmente, il recupero della porta sulla quale avverrà la comunicazione. L'indIP è preimpostato quindi non ci sarà bisogno di scambiare questo tipo di dato. Passaggio successivo sarà l'invocazione di una applet cui vengono passati come parametri due sessioni individuate la indIP e numero di porta. Il loro numero è dovuto al fatto che la

trasmissione sarà operata su due porte distinte, una per l'audio una per il video.

Affinché l'applet funzionasse è stato necessario posizionare sia le classi Java che le librerie di interesse, nella stessa cartella che conteneva il file PlayFile.jsp.

La classe applet vede l'istanziamento di sessioni, che saranno passate come parametro al costruttore di un oggetto della classe AVReceive, sulla quale è invocato il metodo initialize(). Sono stati successivamente implementati alcuni metodi tra i quali *paint* che permette la visualizzazione di messaggi all'interno della applet, o *destroy*, ultimo ad essere eseguito, del quale di serviamo per annullato l'istanza della classe AVReceive.

Il metodo inizialize sarà responsabile della creazione delle sessioni. Il client sarà messo in ascolto sulle due porte indicate. Riscontrata la ricezione dei dati viene inizializzato il Player che ne consente la visualizzazione.

4.2.3 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 avviene il cambio di lingua, e quindi di locale.

4.2.4 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.

4.2.5 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.

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.

4.2.6 Validator

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

4.2.7 Log4j

È 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.

4.2.8 Facilità di configurazione

L'ultima caratteristica di cui facciamo riferimento è quella della facilità di configurazione dell'applicazione.

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.

5. CONFIGURAZIONE ED INIZIALIZZAZIONE DELL'APPLICAZIONE WEB

5.1 Introduzione

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

5.2 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:     02/10/2007 20.39.12                          */
/*==============================================================*/


drop index COMMENTO2_FK on COMMENTO;
drop index COMMENTO_FK on COMMENTO;
```

```
drop table if exists COMMENTO;
drop table if exists MULTIMEDIA_CONTENT;
drop table if exists UTENTE;

/*==============================================================*/
/* Table: COMMENTO                                              */
/*==============================================================*/
create table COMMENTO
(
   COM_ID                         int                            not
null AUTO_INCREMENT,
   COM_COMMENTO                   varchar(400)                   not
null,
   MM_CODICE                      int                            not
null,
   UTE_EMAIL                      varchar(50)                    not
null,
   primary key (COM_ID)
)
type = InnoDB;

/*==============================================================*/
/* Index: COMMENTO_FK                                           */
/*==============================================================*/
create index COMMENTO_FK on COMMENTO
(
   MM_CODICE
);

/*==============================================================*/
/* Index: COMMENTO2_FK                                          */
/*==============================================================*/
create index COMMENTO2_FK on COMMENTO
(
   UTE_EMAIL
);
```

```
/*==============================================================*/
/* Table: MULTIMEDIA_CONTENT                                    */
/*==============================================================*/
create table MULTIMEDIA_CONTENT
(
   MM_CODICE                      int                            not
null AUTO_INCREMENT,
   MM_TITOLO                      varchar(200)                   not
null,
   MM_AUTORE                      varchar(150),
   MM_GENERE                      varchar(30),
   MM_LINGUA                      varchar(20),
   MM_ANNO                        numeric(4,0),
   MM_ALBUM                       varchar(50),
   MM_FILE                        longblob                      not
null,
   MM_NOMEFILE                    varchar(200)                   not
null,
   MM_FILEMIME                    varchar(100)                   not
null,
   primary key (MM_CODICE)
)
type = InnoDB;

/*==============================================================*/
/* Table: UTENTE                                                */
/*==============================================================*/
create table UTENTE
(
   UTE_EMAIL                      varchar(50)                    not
null,
   UTE_PW                         varchar(50)                    not
null,
   UTE_COGNOME                    varchar(50)                    not
null,
   UTE_NOME                       varchar(50)                    not
null,
```

```
   UTE_DATA_NASCITA              date,
   UTE_LUOGO_NASCITA             varchar(50),
   UTE_SESSO                     varchar(1),
   UTE_NICKNAME                  varchar(20),
   UTE_LOCALE                    varchar(15),
   UTE_TIPO                      varchar(20)                    not
null,
   primary key (UTE_EMAIL)
)
type = InnoDB;

alter table COMMENTO add constraint FK_COMMENTO foreign key
(MM_CODICE)
      references MULTIMEDIA_CONTENT (MM_CODICE) on delete restrict
on update restrict;

alter table COMMENTO add constraint FK_COMMENTO2 foreign key
(UTE_EMAIL)
      references UTENTE (UTE_EMAIL) on delete restrict on update
restrict;

/*==============================================================*/
/* Aggiungi Admin                                               */
/*==============================================================*/
insert                                                          into
UTENTE(UTE_EMAIL,UTE_PW,UTE_COGNOME,UTE_NOME,UTE_SESSO,UTE_NICKNAME,
UTE_TIPO,UTE_LOCALE)
values("cri.viola@gmail.com","ciaociao","Viola","Cristina","F","Gocc
ia","Administrator","it");
insert                                                          into
UTENTE(UTE_EMAIL,UTE_PW,UTE_COGNOME,UTE_NOME,UTE_SESSO,UTE_NICKNAME,
UTE_TIPO,UTE_LOCALE)
values("dacierno.a@isa.cnr.it","antonio","d'Acierno","Antonio","M","
Prof","Administrator","it");
```

Figura 5.1. - File di creazione del database

5.3 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=multimediarepository
db.user=root
db.pw=goccia

#EmailSender
emailsender.adminEmail=cri.viola@gmail.com
emailsender.smtpServer=smtp.gmail.com
emailsender.port=25
```

Figura 5.2. - 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;

5.4 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:

- Andare alla homepage del server Tomcat;
- Accedere a Tomcat Manager;
- Selezionare il file .WAR di cui fare il deploy nell'area chiamata WAR file to deploy
- Ciccare sul tasto Deploy
- Provare l'applicazione cliccando sul link a suo nome che è apparso nell'area Applications.
- 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/sslhowto.html

6. CONCLUSIONI

Giunti alla fine del progetto di tesi, operata una attenta analisi della progettazione e dell'implementazione concludiamo che tutti i requisiti iniziali sono stati soddisfatti.

Abbiamo cercato di rendere la nostra applicazione fruibile dalle varie tipologie di utenti. Per questa ragione si è rivelata necessaria in qualche occasione l'aggiunta di nuove funzionalità. A tale scopo abbiamo utilizzato framework potenti, approfondito lo studio delle varie tecnologie utilizzate ed affrontato problematiche sempre nuove.

Come per ogni applicazione, andrebbe ora eseguita la fase di correzione dei bug e di raffinazione.

Per quanto riguarda gli sviluppi futuri sono rimaste aperte varie opportunità. Si potrebbe pensare di rendere l'applicazione universalmente valida per ciò che riguarda l'esecuzione dei file, operando ad esempio l'implementazione dello streaming per i contenuti multimediali che abbiano formato diverso dal quello mpeg.

7. REFERENCES

7.1 Bibliografia

- **Apache Jakarta Project**

  The Apache Jakarta Tomcat 5 Servlet/JSP Container Documentation - http://tomcat.apache.org/tomcat-5.0-doc/index.html

- **Phil Hanna**

  La guida completa JSP – Mc Graw Hill 2004

- **Cavaness,**

  CProgrammare con Jakarta Struts, O'Reilly, Ulrico Hoepli Editore. **.** [2003],

- **Festa M.**

  Progetto e realizzazione di una applicazione web-based sulle proteine. . [2005]

7.2 Fonti di riferimento sul Web

- **Dongilli, P.**

  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,**

  http://www.jspinsider.com/content/jsp/struts/struts.jsp, Amberjack. Software LLC

- **Log4j,**

  http://logging.apache.org/log4j, Apache Sofware Foundation

- **Morgan, E.**

  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

- **Java Media Framework**

  http://java.html.it/articoli/leggi/1945/il-java-media-framework/

http://telemat.die.unifi.it/book/1998/JMF/JMF.html

http://www.cdt.luth.se/~johank/smd151/jmf/jmf2_0-guide.pdf

- **Real Time Striming Protocol**

  http://www.live555.com/mplayer/