Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione
Corso di Laurea Triennale in Ingegneria Informatica

Sviluppo di pagine web all'interno di un portale

Laureando
Nicola Chessa

Relatore
Michele Moro

Anno Accademico 2009/2010

INDICE

ELENCO DELLE TABELLE

ELENCO DELLE FIGURE

1 INTRODUZIONE

INDICE

1.1 L'AZIENDA

La società SMC, nata nel 1981 e presente in tutto il territorio italiano con 16 sedi, è specializzata nella realizzazione di prodotti software per le piccole e medie aziende. *www.smc.it*

Figura 1: Logo SMC

In particolare, nella sede di Lancenigo di Villorba (Tv) in cui si è svolto il periodo di tirocinio, vengono sviluppate le tecnologie inerenti al web 2.0, un insieme di applicazioni usufruibili mediante la rete internet. Il programma su cui ci si basa per realizzare tali applicazioni web è Liferay, un software open source orientato alla realizzazione di portali web che utilizza il linguaggio Java, nato dalla collaborazione della comunità open source e *www.liferay.com*

che, grazie anche al contributo di persone di diverse organizzazioni e industrie, si trova in una situazione di continuo sviluppo e perfezionamento.

1.2 CONTESTO OPERATIVO

Il tirocinio si è svolto a partire dal giorno 22/03/2010 fino al 05/06/2010 e ha riguardato prevalentemente l'inserimento della risorsa nell'area tecnica/di programmazione. L'obiettivo principale dello stage è stato quello di collaborare con il team di sviluppo software per la realizzazione di alcune applicazioni web; ciò ha richiesto inizialmente lo studio della piattaforma utilizzata, la sua struttura, il suo funzionamento e, una volta acquisite queste conoscenze, è stato necessario apprendere i diversi linguaggi di scripting o di programmazione richiesti.

1.3 DESCRIZIONE

In questa relazione saranno delineati gli strumenti e i linguaggi di programmazione utilizzati al fine di creare le portlet, dei programmi controllabili mediante l'utilizzo di qualsiasi browser presente attualmente nel mercato. Verrà inoltre descritta nel dettaglio la struttura interna sulla cui base sono stati sviluppati questi moduli web, come questa si interfaccia con il database utilizzato e i diversi vantaggi che comporta l'impiego di questo tipo di architettura software.
Come si potrà constatare da una successiva analisi, ciascun applicativo risulta composto da due parti: una chiamata "lato server" (server-side), utilizzata per poter usufruire dei servizi messi a disposizione dal server, l'altra indicata come "lato client" (client-side), impiegata per gestire le operazioni che possono essere effettuate dall'utente. Per quanto concerne questa seconda parte, è stato necessario adoperare una particolare accortezza nel rendere il software compatibile con tutti i browser più comuni, potendo garantire in questo modo un corretto e facile utilizzo da parte dei fruitori: saranno quindi descritte le problematiche incontrate in questo procedimento e le soluzioni trovate per poterle risolvere.
Infine, verrà fornito un esempio pratico dello sviluppo di una portlet, esaminando come vengono sfruttate le potenzialità messe a disposizione da questa struttura dal punto di vista della programmazione.

L'obiettivo principale di questa trattazione risulta essere perciò quello di illustrare le funzionalità e la struttura del sistema impiegato adoperando un linguaggio esauriente e dettagliato ma allo stesso tempo comprensibile alla maggior parte dei lettori, compresi coloro che hanno poca confidenza con gli argomenti in esame.

2 PORTALE WEB

In questo capitolo verrà spiegato cos'è un portale web e quali sono gli elementi principali di cui è composto.

INDICE

2.1 INTRODUZIONE

Un portale web può essere definito come un insieme di risorse web, indipendenti tra loro, messo a disposizione di ciascun utente; spesso vi è una grande confusione tra il concetto di portale web e quello di sito web, le cui differenze sostanziali possono essere riassunte nel seguente modo:

differenza tra sito-portale web

- un sito web è composto da un insieme di documenti ipertestuali (pagine web) correlati tra loro e caratterizzati da una prestabilita struttura grafica;

- un portale web è invece composto da un insieme di elementi presenti all'interno di diverse pagine web la cui disposizione può essere modificata da ciascun utente e la cui struttura grafica può essere personalizzata in modo indipendente dagli altri componenti.

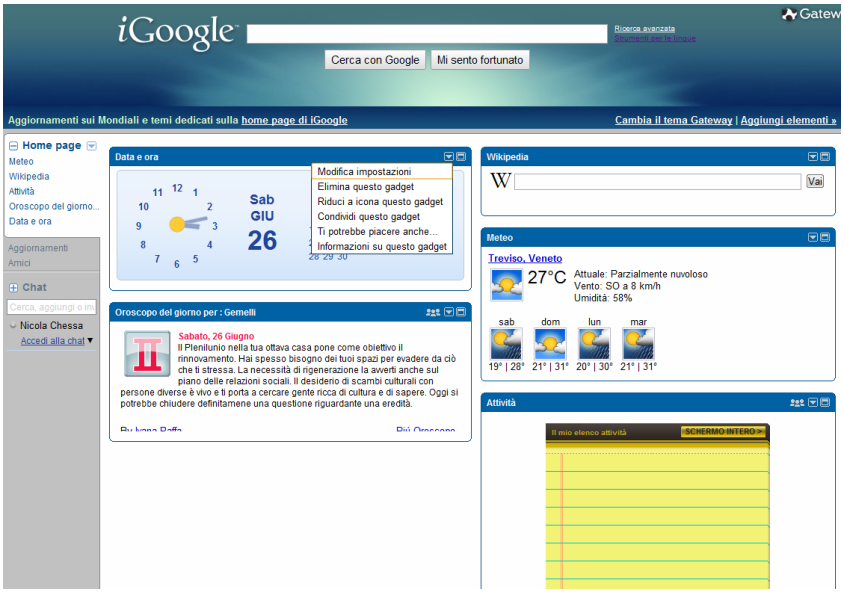

Figura 2: Esempio Portale Web

Vi è quindi la possibilità di personalizzare la propria area e adattarla alle proprie esigenze, per esempio modificando la posizione, l'aspetto grafico (colori, caratteri) o addirittura aggiungendo o rimuovendo alcuni servizi messi a disposizione del portale stesso.

2.2 SERVLET

Servlet

Definizione. *La servlet è una componente web gestita da un contenitore e basata su tecnologia Java che genera pagine web dinamiche. Come altri componenti basati sul linguaggio Java, il codice della servlet è presente all'interno di una classe e, una volta compilato, può essere caricato in qualsiasi server web che supporta tale tecnologia.*

Le pagine web generate in modo dinamico contengono al loro interno un codice sorgente che utilizza sia il linguaggio html che quello java, per esempio per effettuare i controlli per l'autenticazione degli utenti, per sottoporre richieste alla base di dati, ecc. Il difetto principale deriva quindi proprio dall'unione nella stessa struttura di due tipologie diverse di linguaggio, html, dedicato alla visualizzazione delle informazioni, e java, rendendo complicate le operazioni di manutenzione e riducendo la leggibilità del codice stesso.

Vantaggi

Per ovviare a questo problema ci si serve delle Servlet: in que-

sto modo gran parte del codice Java viene spostato in opportune classi Java (file scritti in linguaggio Java), separando così il codice dedicato alla rappresentazione dei contenuti web da quello relativo alle operazioni di gestione.

2.2.1 Servlet Container

Un Servlet Container (o Servlet Engine) è un software capace di gestire le Servlet utilizzate rispettando le specifiche necessarie per il linguaggio Java.
Questo programma si occupa della compilazione e dell'esecuzione delle classi relative alla Servlet, ossia, quando il server riceve una richiesta da parte di una pagina web, il Servlet Engine esegue un'analisi sintattica (parsing) della pagina e quando si imbatte nel codice java lo interpreta sostituendolo con un codice html concatenando e mettendo insieme i frammenti di codice per poi restituire la pagina al client.

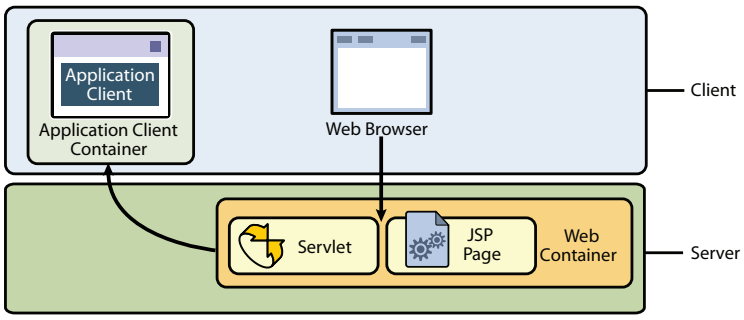

Figura 3: Servlet Container

2.3 PORTLET

Definizione. *La portlet è una componente web basata sulla tecnologia Java, gestita da un portlet container che processa richieste da parte dell'utente e genera contenuti dinamici. Le portlet sono usate dai portali come componenti d'interfaccia utente al fine di provvedere un livello di presentazione per i sistemi informativi.* *Portlet*

Le portlet sono quindi le entità principali e fondamentali che formano un portale web. Il contenuto generato da una portlet corrisponde a frammenti di codice (HTML, XHTML, WML) che soddisfano determinate regole: l'unione di più portlet forma

il contenuto del portale. Ciascuna portlet rappresenta di fatto un'applicazione web indipendente inserita all'interno di una finestra del portale: come per le applicazioni desktop, può essere chiusa, spostata o ridotta ad icona per permettere di visualizzare i contenuti a seconda delle proprie necessità.

Da questa definizione di portlet si può osservare una certa somiglianza con la descrizione fornita per la servlet: una portlet è in effetti un tipo speciale di servlet. Le principali caratteristiche comuni di questi due moduli web sono:

caratteristiche comuni servlet-portlet

- si basano entrambi su tecnologia Java;
- vengono gestiti da un contenitore specializzato;
- generano contenuti dinamici;
- permettono l'interazione con il client web mediante un paradigma di richieste/risposte.

differenze servlet-portlet

Andando a ricercare invece le differenze si può osservare che:

- le portlet generano solamente singoli frammenti di codice html, non documenti interi, poiché è il portale a occuparsi dell'aggregazione dei frammenti in una pagina web completa;
- le portlet non sono limitate ad una unica URL;
- il client web interagisce con la portlet in maniera indiretta attraverso il portale;
- le portlet hanno un sistema più raffinato per la gestione delle richieste;
- le portlet possiedono delle modalità predefinite e degli stati delle finestre che indicano la funzione che la portlet sta eseguendo;
- ciascuna portlet può essere presente più volte in una pagina del portale;
- le portlet possono essere configurate e dispongono di alcune funzionalità utili per la memorizzazione delle impostazioni;
- le portlet hanno accesso ai profili dell'utente;
- le portlet hanno una gestione delle URL tale da essere indipendente dal portale;

- le portlet non possono cambiare le specifiche di codifica utilizzate per comporre la risposta;

- le portlet non possono specificare le intestazioni HTTP delle risposte.

2.3.1 Portlet Container

Il Portlet Container è utilizzato per contenere le varie portlet e ha il compito di ricevere le richieste provenienti dal portale e di reindirizzarle alle portlet opportune; rende inoltre possibile configurare le diverse portlet facendo scegliere ad ogni utente le opzioni desiderate.

Quello che segue è un tipico esempio di eventi che vengono avviati quando si accede ad una pagina del portale:

- un client effettua una richiesta HTTP al portale;

- la richiesta viene ricevuta dal portale;

- il portale controlla se la richiesta ricevuta può essere associata a qualche portlet presente nella pagina dello stesso;

- se la richiesta può essere soddisfatta, il portale chiede al portlet container di invocare l'azione annesso alla portlet interessata;

- il portale invoca la portlet, tramite il contenitore, al fine di ottenere parti di codice che potranno essere incluse nella pagina risultante;

- il portale aggrega il risultato ottenuto dalla chiamata dell'azione richiesta alla pagina del portale e infine la rimanda al client.

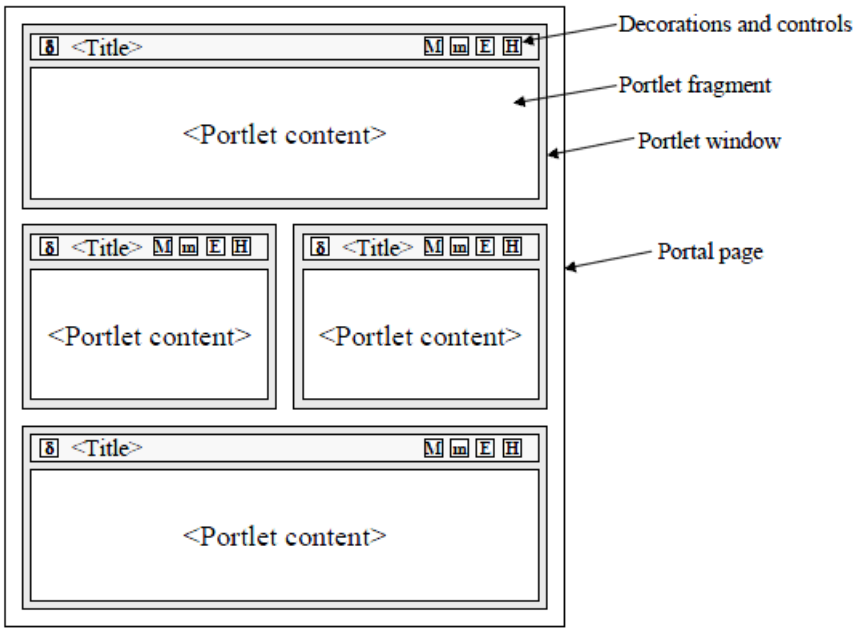

Figura 4: Modalità Portlet

2.3.2 Standard Portlet Container - Portlet

Java Portlet Specification

Nella fase di sviluppo di una portlet bisogna rispettare un insieme di regole e standard per garantire la sua portabilità, in modo tale che sia possibile aggiungere ciascuna applicazione web a qualsiasi portlet container senza dover apportare alcuna modifica al codice. Per poter garantire tutto ciò è stato sviluppato un insieme di protocolli (Java Portlet Specification) con lo scopo di definire uno standard per il dialogo fra il portale e ciascuna portlet:

- JSR 168[1] - definisce un gruppo di APIs, funzioni che servono a implementare le regole utili per l'interoperabilità tra le portlet e il portale;
- JSR 286[2] - è un insieme di specifiche per allineare lo sviluppo delle portlet con la piattaforma Java J2EE 1.4;
- JSR 127[3] - è la definizione di un'architettura standard per agevolare la creazione e la manutenzione di applicazioni Server scritte in Java;

1 http://www.jcp.org/en/jsr/detail?id=168
2 http://www.jcp.org/en/jsr/detail?id=286
3 http://www.jcp.org/en/jsr/detail?id=127

- JSR 170[4] - è un insieme di specifiche per l'accesso ai contenuti Java indipendentemente dall'implementazione.

Tra i protocolli sopra elencati quello più importante per il programmatore è il primo in quanto specifica un modello adeguato per la creazione delle portlet: vengono infatti descritti i modi di operare delle portlet e gli stati di ciascuna finestra, i quali vengono a loro volta utilizzati per indicare quanto spazio deve essere riservato alla specifica applicazione all'interno della pagina.

Modalità portlet

Lo stato di funzionamento attivo per le portlet viene indicato dal modo di operare che può essere definito in tre modi:

- VIEW = rappresenta lo stato in cui l'utente può interagire con le funzionalità messe a disposizione dall'applicazione web;
- EDIT = fornisce le operazioni di configurazione delle portlet per personalizzare il suo comportamento nella modalità VIEW;
- HELP = mette a disposizione informazioni di assistenza per la portlet specifica.

Oltre a queste tre modalità, il programmatore ha in ogni caso la possibilità di crearne altre e personalizzarle per eventuali funzionalità speciali.

Stati della finestra portlet

Per quanto riguarda gli stati di ciascuna finestra, invece, si possono individuare i seguenti:

- NORMAL = la portlet condivide la pagina con altre: viene quindi riservato uno spazio limitato per la visualizzazione del contenuto;
- MAXIMIZED = la portlet possiede tutta la pagina per visualizzare il proprio contenuto e non condivide altro spazio con altre portlet;
- MINIMIZED = in questo stato l'applicazione viene ridotta ad icona, in modo simile a quanto accade con le normali applicazioni desktop.

Per ciascuna applicazione web è inoltre possibile eseguire operazioni di drag&drop per lo spostamento della finestra all'interno della pagina (possibile quando si è in modalità NORMAL o MINIMIZED).

4 http://www.jcp.org/en/jsr/detail?id=170

Figura 5: Modalità di operare

Figura 6: Drag&Drop

2.3.3 Ciclo di vita

Il ciclo di vita di una portlet è gestito dal portlet container che si occupa di caricarla, istanziarla e inizializzarla utilizzando i quattro metodi messi a disposizione dalla portlet stessa:

- *init()* = chiamato per inizializzare e istanziare le portlet;
- *processAction()* = chiamato quando viene generata un'azione dall'utente;
- *render()* = chiamato ogni volta che deve essere visualizzato il contenuto in una finestra;
- *destroy()* = chiamato al momento della chiusura per deallocare (liberare la memoria) la portlet.

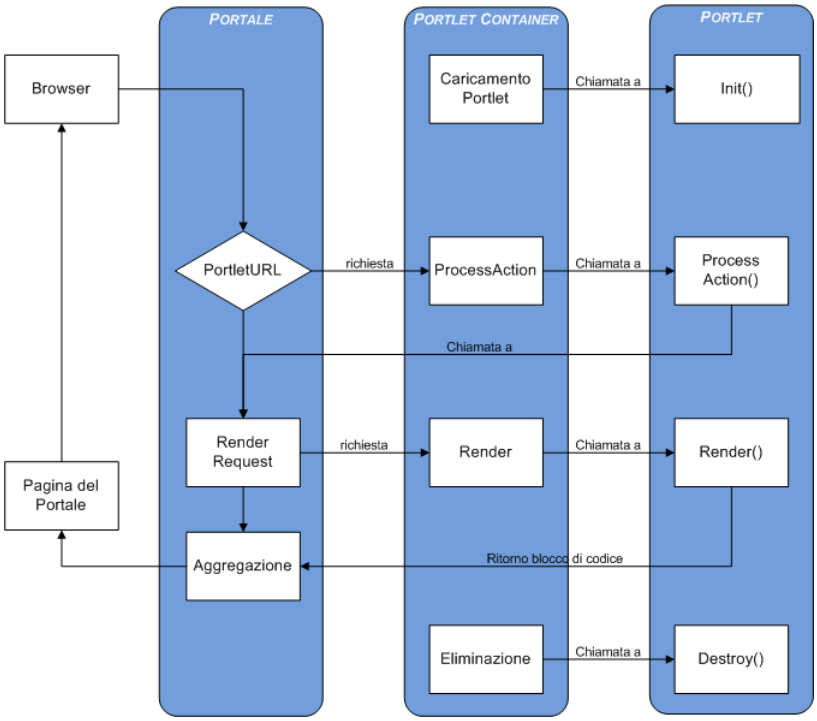

Figura 7: Ciclo di vita di una portlet

Un'implementazione in java delle portlet è data dalla classe GENERICPORTLET che realizza i metodi sopra menzionati aggiungendone altri tre (specializzando il metodo render che si occupa della raffigurazione del contenuto), che hanno lo scopo di visualizzare la portlet a seconda del modo in cui sta operando:

- *doView()* = visualizza la portlet quando si trova in modalità View;
- *doEdit()* = visualizza la portlet quando si trova in modalità Edit;
- *doHelp()* = visualizza la portlet quando si trova in modalità Help.

2.3.4 Portlet URL

Molto spesso all'interno di una portlet si ha la necessità di creare dei collegamenti ipertestuali che facciano riferimento ad altre portlet, come per esempio all'invio di un modulo dati. Per garantire la completa portabilità delle portlet, come accennato precedentemente, non è possibile creare dei link a specifici URL (Uniform Resource Locator): per risolvere questo problema si ricorre perciò alle *PortletURL*, oggetti generati dal portale al momento di una richiesta diretta verso la portlet desiderata da parte dell'utente. Nel dettaglio, la specifica JSR-168 (2.3.2) permette la creazione di due tipi di PortletUrl:

- *ActionURL*
- *RenderURL*.

La RenderUrl è un tipo speciale di ActionURL: viene utilizzata con lo scopo di impostare correttamente i parametri di configurazione per il render della portlet nel momento in cui si presenta una richiesta di renderizzazione; non è perciò possibile adoperarla per il passaggio di parametri proveniente da un modulo dati.

La creazione dei due tipi di PortletUrl avviene mediante la chiamata dei metodi createActionURL e createRenderURL a seconda che si tratti del primo o del secondo tipo. Per il passaggio di parametri si può utilizzare il metodo setParameter applicabile all'oggetto PortletURL creato in precedenza.

2.4 PATTERN MVC

Durante lo sviluppo delle portlet possono sorgere molti problemi quando le applicazioni web contengono all'interno di uno stesso file codice sorgente inerente alla gestione dei dati, alla logica di controllo e alla visualizzazione; mantenere il codice strutturato in questo modo non agevola la manutenzione per la correzione degli errori, per l'implementazione di nuove funzionalità o per il riutilizzo di metodi già realizzati.
Per risolvere questo problema è stato introdotto il pattern MVC (Model-View-Controller): attraverso l'utilizzo di questo modello le porzioni di codice sorgente riguardanti l'accesso ai dati, la logica di controllo e la presentazione vengono tenute disgiunte. In questo modo si cerca di rendere le tre componenti disaccoppiate in generale facendo si che una modifica ad una di esse non provochi cambiamenti anche nelle due rimanenti.
Lo schema MVC è formato da tre componenti (figura 8):

- *Model* = fornisce le funzionalità per la gestione dei dati quali il controllo degli accessi, dei privilegi, dei meccanismi di aggiornamento e per eseguire delle interrogazioni al database per la raccolta delle informazioni presenti;

- *View* = visualizza il contenuto del modello, i dati grazie al *Model* e interagisce con gli utenti;

- *Controller* = riceve da *View* le richieste provenienti dall'utente e successivamente le esegue modificando lo stato di *View* e *Model*.

Questo paradigma viene applicato allo sviluppo delle portlet implementando la parte View con pagine JSP, la parte Model dell'interfaccia con il database e la parte Controller con la portlet stessa.

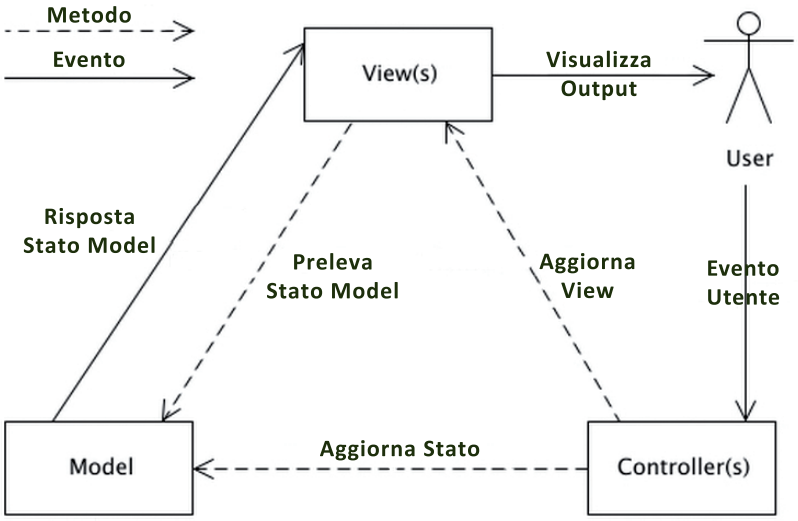

Figura 8: Schema MVC

2.5 SVILUPPO PORTLET

Di seguito sono riportati i linguaggi di programmazione utilizzati per la creazione di applicazioni web; sono stati separati in due macro categorie, lato Server e lato Client, per definire con maggiore chiarezza il modo con il quale operano e per esporre gli scopi del loro impiego.

2.5.1 Linguaggi Lato Client

I linguaggi lato client sono quei linguaggi la cui elaborazione e interpretazione vengono affidate alla macchina dell'utente (client). Di conseguenza, una pagina web può essere visualizzata correttamente solo se il browser, con cui la si sta visitando, supporta tutte le tecnologie utilizzate.
I linguaggi appartenenti a questa categoria sono:

- *Html*
- *Css*
- *Javascript*

Html

Html (HyperText Markup Language) è un linguaggio di markup (non di programmazione) utilizzato per la definizione della struttura di una pagina web: viene interpretato dal browser web il quale lo elabora e genera la visualizzazione della pagina richiesta. Per la definizione della pagina vengono utilizzati i MARKUP TAG, parole chiave usate per rappresentare gli elementi della pagina.

Css

I CSS (Cascading Style Sheet - Fogli di stile a cascata), introdotti a partire dall'HTML 4.0, sono utilizzati per definire come una pagina deve essere visualizzata; se l'html descrive il contenuto della pagina, i fogli di stile a cascata si occupano della sua formattazione (colori, caratteri, bordi, ecc..).
Esistono diversi vantaggi dovuti al loro utilizzo che sono principalmente:

- riutilizzo dello stesso tema per più pagine web senza bisogno di aggiunta di codice;
- separazione del codice relativo al contenuto da quello relativo alla rappresentazione grafica, facilitando quindi la manutenzione.

L'aggregazione degli elementi della pagina web con gli stili avviene con l'utilizzo dei selettori, quali i markup tag (si associa lo stile a tutti i componenti di uno stesso tipo), di un identificativo (si associa lo stile a un determinato elemento) oppure di una classe (si associa lo stile agli elementi per i quali è stata definita quella medesima classe).

NOME	SINTASSI	DESCRIZIONE
markup tag	`nomeMarkup`	Associa uno stile agli elementi corrispondenti al tag html specificato (<p>,<a>)
Id	`#nomeIdentificatore`	Associa uno stile a tutti gli elementi con questo identificativo

Tabella 1: continua nella prossima pagina

Tabella 1: continua dalla pagina precedente

Nome	Sintassi	Descrizione
Class	`.nomeClasse`	Associa uno stile ad un gruppo di elementi che utilizzano la classe specificata

Tabella 1: Selettori

Per ciascuno di questi selettori può essere annesso un insieme di proprietà con lo scopo di definire un particolare stile. Qui di seguito si possono trovare elencate le più comuni:

Proprietà	Descrizione
`background-color`	sfondo di un elemento
`text-color`	colore del testo
`text-align`	allineamento orizzontale del testo
`letter-spacing`	spazio tra i caratteri
`text-decoration`	aggiunge una decorazione al testo, come per esempio la sottolineatura
`font-family`	tipo di font
`font-size`	dimensione del font
`font-style`	forma del font (normale, italica, obliqua)
`font-weight`	spessore del font
`list-style`	tipo di elenco (puntato, numerato)
`border-color`	colore del bordo
`border-style`	forma del bordo (linea continua, tratteggiata)
`border-width`	dimensione del bordo
`margin`	margine
`padding`	padding
`height`	altezza
`width`	larghezza
`position`	posizione rispetto alla pagina o all'elemento
`overflow`	visualizzazione se il contenuto presente all'interno di un elemento eccede la sua dimensione
`left`	posizione dal margine sinistro
`top`	posizione dal margine superiore

Tabella 2: continua nella prossima pagina

Tabella 2: continua dalla pagina precedente

PROPRIETÀ	DESCRIZIONE
`cursor`	tipo di cursore del mouse

Tabella 2: Proprietà CSS

Javascript

Javascript è un linguaggio di scripting interpretato (non è perciò necessaria alcuna operazione di compilazione), sviluppato per aumentare l'interattività con l'utente nelle pagine web.
L'attività d'interpretazione del codice Javascript non richiede elevate risorse hardware: questa caratteristica è di rilevante importanza in quanto, essendo un linguaggio lato client, l'elaborazione deve essere fatta nel minor tempo possibile per non appesantire la pagina e, di conseguenza, renderla poco usufruibile da parte dell'utilizzatore.
Le principali funzionalità di Javascript sono:

- creazione di componenti dinamici all'interno della pagina web;
- controllo degli eventi generati dall'utente, quali il clic del mouse, la pressione di un pulsante, ecc.;
- interazione con i moduli web compilati, per esempio per il controllo dei valori inseriti;
- lettura e scrittura di cookie per il salvataggio di informazioni nel browser.

Compatibilità Cross Browser

Come affermato precedentemente, i linguaggi client-side devono essere interpretati dal browser; purtroppo, non tutti i tipi di browser riescono a comprendere correttamente il codice scritto, rendendo nella peggiore delle ipotesi la pagina internet totalmente illeggibile e di conseguenza inutilizzabile. Durante la scrittura di applicazioni web risulta pertanto doveroso rispettare alcuni standard imposti dal W3C (World Wide Web Consortium) ed è altamente sconsigliato utilizzare un codice appartenente solamente ad alcuni browser.

www.w3.org

Ecco una lista dei principali browser con cui la compatibilità è attualmente richiesta:

- Mozilla Firefox 3.6
- Google Chrome 5.0
- Opera 10
- Apple Safari 5
- Internet Explorer 8
- Internet Explorer 7

Tra i linguaggi elencati precedentemente, quello che ha creato più problematiche sotto questo punto di vista è stato senz'altro Javascript: ciò è dovuto alle varie attuazioni di Javascript sviluppate all'interno dei diversi browser.

http://jquery.com/

Per rimediare a questo problema si utilizza la libreria jQuery: è una libreria di funzioni javascript che ha come obiettivo principale quello di rendere le pagine web cross browser, cioè visualizzabili in modo corretto dalla maggior parte dei browser presenti. In più, jQuery offre ulteriori servizi atti a diminuire la scrittura di codice per il raggiungimento di prefissati obiettivi quali la gestione degli eventi, la modifica del foglio di stile in modo dinamico, la creazione di effetti o animazioni, ecc.
Questa libreria offre al programmatore anche una raccolta di componenti chiamati widget, impiegati per facilitare all'utente l'interazione con la pagina web. JQuery, inoltre, mette a disposizione un insieme di funzionalità molto utili e veloci per l'utilizzo della tecnologia Ajax.

http://api.jquery.com/category/ajax/

Ajax

Ajax (Asynchronous JavaScript and XML) è una tecnologia scritta in Javascript per lo sviluppo di applicazioni web dinamiche dedicate allo scambio asincrono di informazioni tra il client e il server.
La novità principale introdotta da questa tecnologia è proprio l'asincronia: ciò significa che, quando si verifica una richiesta da parte del client verso il server, non è più necessario attendere che venga ultimata per effettuare altre operazioni.
Come si può vedere dalla Figura 9, l'impiego di questa tecnologia provoca la modifica del normale flusso di scambio dei dati previsto dal protocollo HTTP. Nel modello classico, infatti, per

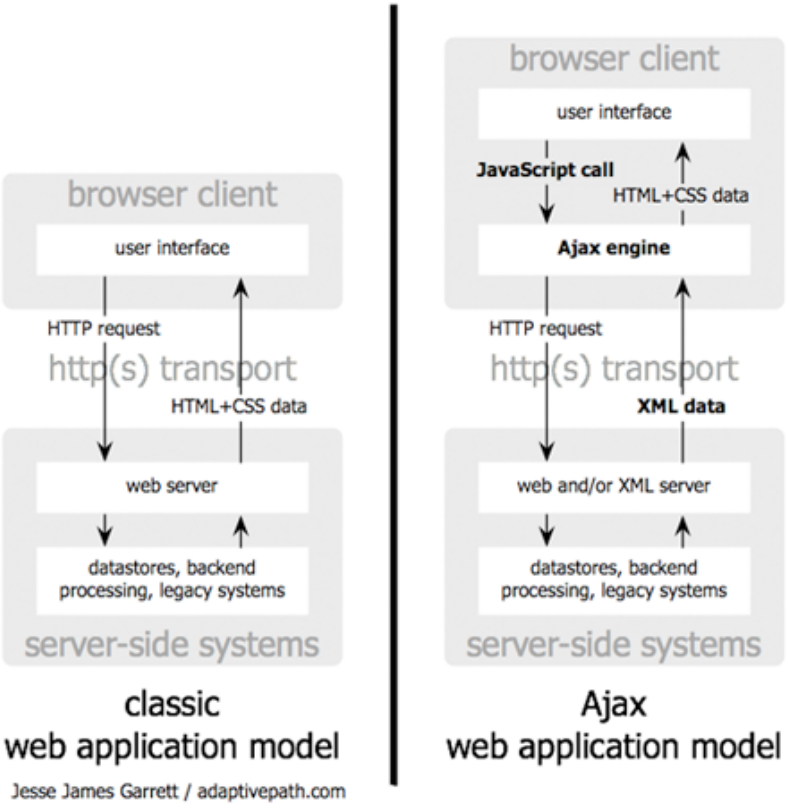

Figura 9: Differenza flusso dati Ajax - standard HTTP

ogni richiesta effettuata dal client si attende la risposta dal server (la risposta consiste nella pagina web aggiornata), mentre nel modello con metodo Ajax, il client richiede delle informazioni al server e quest'ultimo risponde con dei dati scritti in formato XML o HTML, che verranno poi utilizzati per l'aggiornamento della pagina web.

I vantaggi derivanti dall'utilizzo di Ajax sono:

- minor quantità di traffico scambiato tra client e server: non è necessario attendere il ricaricamento di tutta la pagina ma solamente una sua porzione;
- maggiore velocità di interazione con l'utente;
- possibilità di effettuare richieste simultanee.

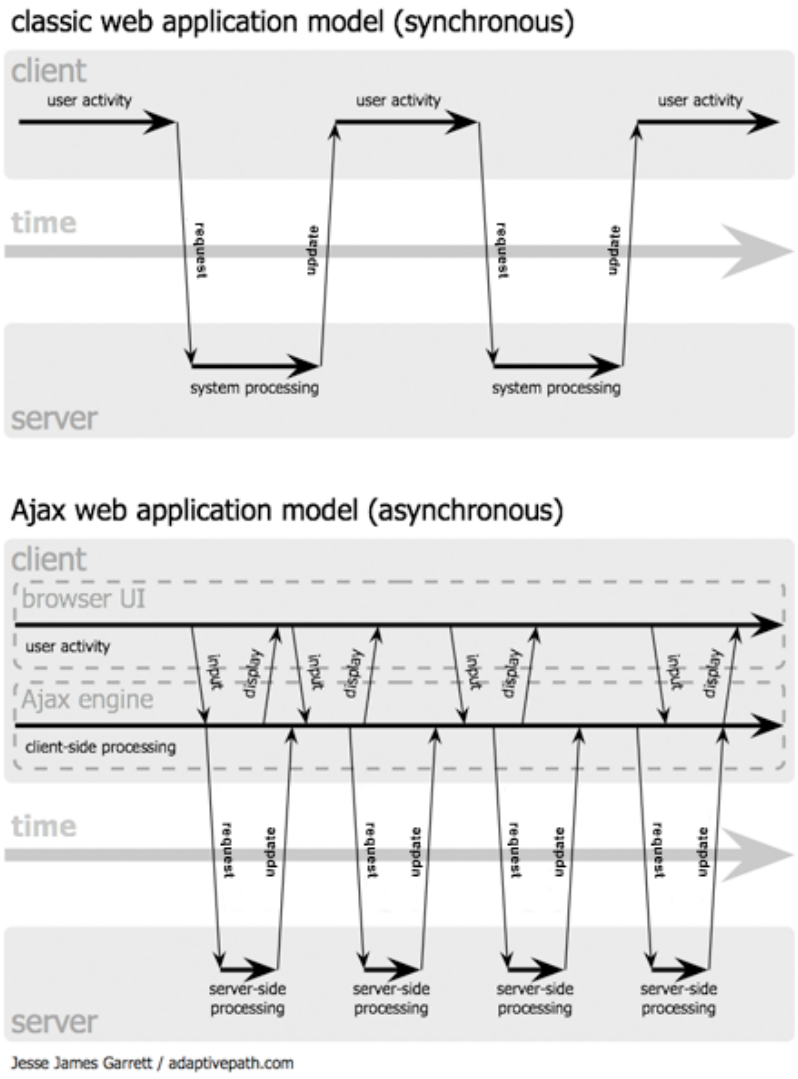

Figura 10: Ajax - diagramma temporale

La possibilità di effettuare delle richieste simultanee costituisce un vantaggio ottenuto proprio grazie al cambiamento del flusso di dati. Questa funzionalità non è usufruibile nel modello classico in quanto, durante il tempo d'attesa per l'aggiornamento della pagina, non sono concesse ulteriori interazioni dell'utente con la stessa (nella Figura 10 si può facilmente osservare la maggiore efficienza temporale di Ajax; attraverso questa tecnologia è possibile continuare a utilizzare i servizi messi a disposizione per l'utente anche nell'intervallo tra una richiesta e l'aggiornamento della pagina, cosa non possibile, invece, con il modello classico).

2.5.2 Linguaggi Lato Server

I linguaggi lato server sono quei linguaggi che vengono elaborati dal server, il quale mette a disposizione un insieme di servizi utili per il reperimento di informazioni o funzionalità non disponibili nella macchina dell'utente. Ad ogni richiesta effettuata dall'utilizzatore verrà poi spedito il risultato prodotto dal server.

Java EE

Java EE (Enterprise Edition) è una piattaforma costituita da un insieme di specifiche per lo sviluppo di applicazioni per il lato server utilizzando il linguaggio Java. Rispetto alla versione Java SE (Standard Edition), vengono aggiunte delle librerie per applicare alcune nuove caratteristiche, quali:

Caratteristiche Java EE

- *Java Servlet* = insieme di API (Application Programming Interface) per lo sviluppo di Servlet;
- *JavaServer Pages (JSP)* = tecnologia per la creazione di pagine web dinamiche;
- *Java Persistance* = framework utilizzato per il controllo dei dati mediante l'utilizzo delle classi java.

JavaServer Pages - JSP

JavaServer Pages è una tecnologia sviluppata in Java per la creazione di pagine web dinamiche. Dalla tecnologia Java eredita i principali vantaggi quali la compatibilità con più piattaforme e la tecnica di sviluppo Object Oriented.
Le pagine JSP (.jsp) sono delle pagine web composte da due tipi di codice: uno statico (html) e uno dinamico, formato quest'ultimo da elementi jsp che hanno il compito di generare il contenuto in modo dinamico.
Questo linguaggio di programmazione è strettamente legato alla tecnologia delle Servlet in quanto le pagine JSP vengono tradotte automaticamente da un compilatore JSP in servlet. Tutto ciò è reso possibile dalla presenza nel server web di un motore JSP che, quindi, risulta essere indispensabile.

Gli elementi JSP possono essere di due tipi: *direttive* e *script*. Le direttive sono un insieme di comandi necessari per la corretta interpretazione della pagina da parte del motore JSP.

Direttive JSP

Le direttive principali sono:

- *page* = specifica alcune impostazioni riguardanti la pagina, come le librerie da caricare per la sua compilazione oppure quale pagina visualizzare in caso di errore;
- *include* = aggiunge dei file sorgenti da compilare insieme alla pagina;
- *taglib* = specifica un insieme di librerie per i tag personalizzati.

Tag	JSP
page	`<%@page ... %>`
include	`<%@include file="nomefile" %>`
taglib	`<%@taglib uri="nomeURI" prefix="nomePrefix" %>`

Tabella 3: Tag JSP per le direttive

Gli Script sono elementi che contengono al proprio interno codice sorgente Java e sono utilizzati principalmente per generare il contenuto dinamico oppure per effettuare dei controlli.
Per facilitare la creazione di pagine dinamiche, la gestione delle funzionalità usate più frequentemente è stata semplificata grazie all'introduzione della libreria JSTL (JavaServer Pages Standard Tag Library). Questa libreria apporta il vantaggio di scrivere meno codice sorgente potendolo riutilizzare più volte. In questo modo le operazioni possono essere richiamate grazie all'uso di speciali tag invece che essere scritte all'interno degli script stessi. Qualora fosse necessario utilizzare spesso delle funzionalità personali, è inoltre possibile creare una libreria JSTL personale contenente un insieme di tag (custom tag).

Libreria JSTL

I tag JSP più comuni sono:

Tag	JSP	Descrizione
Dichiarazione	`<%! ... %>`	Utilizzati per dichiarare le variabili e i metodi; possono essere chiamati in qualsiasi punto della pagina jsp

Tabella 4: continua nella prossima pagina

Tabella 4: continua dalla pagina precedente

TAG	JSP	DESCRIZIONE
Espressione	`<%= ... %>`	Utilizzati per la conversione di un risultato di un'espressione in Stringa
Script	`<% ... %>`	Utilizzati per inserire un frammento di codice Java per poter attivare logiche di controllo del flusso
Direttiva	`<%@ ... %>`	Utilizzati per definire le direttive page, include, taglib

Tabella 4: Principali Tag JSP

Si può trovare una lista di tutti i tag che è possibile impiegare con le varie opzioni disponibili per ciascuno al seguente link: Lista Tag JSP[5]

Java Persistence

Java Persistence API è lo standard del linguaggio Java utilizzato per fornire al programmatore uno strumento utile per la creazione di una relazione tra una classe Java con un'entità del database relazionale utilizzato.
Java Persistence è strutturato in 4 aree: *Settori Java Persistence*

1. *Java Persistence API* = pacchetto contenente l'insieme di metodi e classi utilizzabili;

2. *Java Persistence Query Language (JPQL)* = linguaggio simile a SQL adoperato per realizzare interrogazioni o aggiornamenti delle entità del database;

3. *Java Persistence Criteria API* = insieme di operazioni impiegate per l'interrogazione o l'aggiornamento di dati; la differenza con JPQL è che queste operazioni vengono eseguite sugli oggetti e non sul database;

4. *Metamodel API* = area usata per creare la relazione tra la classe ed una entità del database.

5 http://java.sun.com/products/jsp/syntax/2.0/card20.pdf

Con questa libreria viene quindi creata e messa a disposizione un'interfaccia tra il database e le classi applicative, il che comporta una serie di vantaggi in quanto il programmatore non ha più, di fatto, il problema di collegare il database utilizzato con il linguaggio.

Vantaggi

Le agevolazioni più importanti che si possono riscontrare sono:

1. elevata portabilità del software utilizzato rispetto alla tecnologia del DBMS utilizzato. Nel caso in cui si cambiasse database, non sarà necessario apportare alcuna modifica alle procedure scritte per la gestione/interrogazione delle entità; sarà necessario modificare solamente qualche riga in un file di configurazione per indicare al software di utilizzare il nuovo DBMS;

2. meno codice da scrivere; non è più richiesta quella parte di software per creare la connessione al database e per verificare le eventuali eccezioni che si potrebbero generare;

3. la mappatura tra gli oggetti e le entità viene effettuata grazie a JPQL, per cui vengono semplificate le operazioni per l'associazione dei dati alle classi;

4. l'utilizzo di queste API impone l'isolamento della parte di codice relativa alla logica per la persistenza dei dati da quella della logica per il controllo dell'applicazione.

I punti 1 e 3 agevolano il controllo per l'accesso ai dati mentre i punti 2 e 4 semplificano la manutenzione del programma e la risoluzione di eventuali bug che possono sorgere durante il suo sviluppo.

3 LIFERAY PORTAL SERVER

In questo capitolo verrà chiarito il concetto di portale e si descriveranno alcune sue caratteristiche.

INDICE

3.1 INTRODUZIONE

Liferay Portal Server è un'applicazione open source basata sul linguaggio Java, indirizzata alla creazione e alla gestione di portali web e, di conseguenza, al controllo delle portlet.

Figura 11: Logo Liferay

Le caratteristiche di questo prodotto sono: *Proprietà Liferay*

- supporto alla maggior parte delle applicazione server esistenti;
- supporto a più Servlet Container (Apache TomCat, JBoss, GlassFish);
- supporto a gran parte degli odierni database (Mysql, PostgresSQL, Oracle, SQL Server);
- compatibilità con Java Portlet Specification;

- supporto a Javascript, JQuery e Ajax.

3.2 FUNZIONALITÀ

Liferay si può definire anche come un Content Management System (CMS) - Sistema di gestione dei contenuti: raggruppa infatti un insieme di procedure sviluppate per semplificare il controllo dei contenuti web. Vengono così agevolate le operazioni riguardanti l'autenticazione degli utenti per l'accesso ai dati, lo scambio di informazioni tra i vari utilizzatori e la pubblicazione di documenti. Liferay viene sviluppato seguendo i processi di adattamento dell'internazionalizzazione e include, di conseguenza, il supporto per più lingue.

Oltre all'opportunità di controllare i privilegi che un utente possiede per l'accesso a determinate risorse, un altro beneficio che si ha con l'utilizzo dei CMS è la creazione dei portali, in quanto è possibile creare un portale web utilizzando le portlet standard già disponibili con il pacchetto software senza che sia necessario scrivere alcuna riga di codice. Alcune delle portlet preimpostate riguardano la gestione di blog, di RSS (formati per la distribuzione di contenuti Web), visualizzazione di un calendario, di una galleria di immagini o anche il controllo delle Email.

È possibile personalizzare il portale organizzando le proprie pagine secondo una struttura grafica ben definita (page layout): per esempio, si può suddividere una pagina in una, due, tre colonne oppure si può ricorrere alla creazione di propri schemi di disposizione personalizzati. Un altro aspetto grafico che è possibile caratterizzare è il tema: anche in questo caso si possono utilizzare alcuni temi predefiniti oppure crearne di personalizzati a seconda delle proprie necessità.

Funzionalità Liferay

Per quanto riguarda le principali funzionalità di questo prodotto, se ne possono riassumere alcune come elencato qui di seguito:

- *Portlet integrate* = fornisce diverse portlet predefinite per la gestione dei contenuti digitali e la loro pubblicazione;
- *Configurazione semplificata* = semplifica le operazioni di configurazione del portale grazie ad un'interfaccia ben sviluppata;
- *Secure Single Sign On (SSO)* = permette di accedere a tutte le applicazioni effettuando l'accesso una volta sola;

- *Gestione Autorizzazioni* = garantisce un servizio ottimale per l'assegnazione dei permessi ai vari utenti del portale, al fine di consentire a ciascuno di essi la visualizzazione o la modifica di determinate informazioni;
- *Supporto Multilingua* = come già accennato, supporta e gestisce più lingue grazie all'internazionalizzazione;
- *Social Network* = mette a disposizione un insieme di applicazioni disponibili per la comunicazione fra persone (Email, Chat, Forum, Newsletter);
- *Gestione documenti* = procura un insieme di strumenti per il controllo delle informazioni (libreria, Wiki, Blog);
- *Integrazione con Microsoft Office* = rende compatibile il salvataggio di documenti con i formati Office.

3.3 ARCHITETTURA UTENTI

Il vantaggio principale derivante dall'utilizzo dei CMS è la funzionalità tramite la quale si può procedere all'amministrazione degli Utenti, cosa che Liferay rende possibile grazie alla seguente architettura:

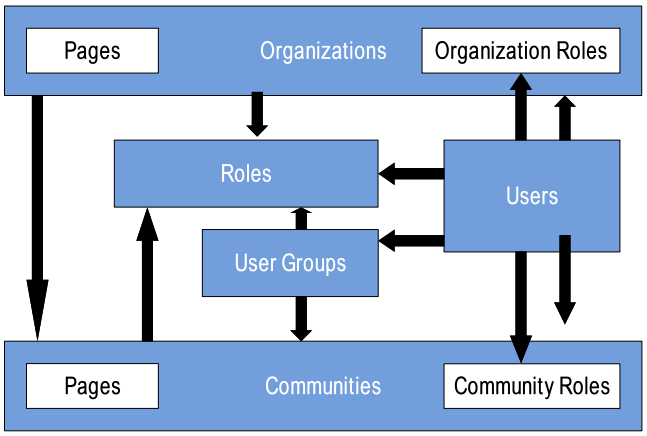

Figura 12: Architettura Liferay

Analizzando nel dettaglio la struttura delle Figura 12 si possono descrivere le seguenti proprietà:

- *Users* = rappresenta gli utenti che accedono al portale; prima di autenticarsi, ciascun utente è definito come guest ed

ha un accesso limitato alle funzionalità implementate. Una volta autenticato, il client può avere determinati privilegi a seconda della comunità a cui appartiene oppure in base all'organizzazione di cui fa parte;

- *User Groups* = indica un insieme di utenti che condividono una serie di permessi e ruoli, gestiti dall'amministratore stesso;

- *Roles* = individua i ruoli, ossia tutti quei permessi relativi all'intero portale;

- *Organizations* = raggruppa alcuni utenti strutturando il tutto in modo gerarchico;

- *Organization Roles* = specifica i ruoli relativi ad una determinata Organizzazione;

- *Communities* = riunisce in gruppi gli utenti che hanno in comune un determinato interesse; questi gruppi possono essere di tre tipi: *a*) *open*= permette ad un utente di entrare o uscire dalla comunità in qualsiasi momento autonomamente; *b*) *Restricted*= subordina l'inserimento di un utente in una comunità al controllo e alla valutazione dell'amministratore della stessa; *c*) *Hidden*= rappresenta una tipologia di comunità simile alla Restricted, con l'aggiunta del fatto che non è possibile visualizzarla attraverso i classici strumenti messi a disposizione dal portale;

- *Community Roles* = determina i ruoli relativi ad una particolare comunità.

Diversità tra organizzazioni e comunità

La differenza strutturale tra le organizzazioni e le comunità è che le prime hanno una struttura gerarchica e un utente, una volta registrato, può far parte di molte comunità ma può essere inserito solamente in una organizzazione. Il motivo principale per il quale sono state distinte queste due categorie è che con le organizzazioni si vuole rappresentare nel miglior modo possibile il modello realmente esistente riferito all'associazione o all'ente per cui si vuole sviluppare il portale, mentre con le comunità si vuole raffigurare un insieme di attività comuni a più utenti, che possono appartenere a diverse organizzazioni o addirittura a nessuna di queste.

3.4 GESTIONE DATI

Durante lo sviluppo delle portlet, le informazioni sono state strutturate secondo uno schema relazionale: il database utilizzato è MySQL, usufruendo della piattaforma Hibernate per l'amministrazione dei dati.

www.mysql.it

3.4.1 Database MySQL

MySQL è un sistema per la gestione di basi di dati relazionale (RDBMS - Relational database management system) disponibile sia per sistemi operativi Windows che Linux.
Questo database supporta gran parte della sintassi standard SQL e fruisce di varie interfacce per diversi linguaggi di programmazione attualmente esistenti. Questo è possibile mediante l'uso dei driver proprietari: ciascuna applicazione web Java si interfaccia al database per mezzo del JDBC driver, chiamato anche MySQL Connector/J.

MySQL è stato scelto in quanto è un prodotto molto affidabile, è open source, il portale Liferay ne garantisce il supporto ed è inoltre data per certa la compatibilità con la piattaforma Hibernate.

3.4.2 Piattaforma Hibernate

La piattaforma Hibernate è una struttura di supporto basata sulle librerie JPA costruita per arricchirne le funzionalità messe a disposizione.

www.hibernate.org

Hibernate è stato sviluppato prima dell'implementazione dello standard JPA, il quale ha infatti incorporato gran parte delle funzionalità proprie di Hibernate. L'utilizzo di quest'ultimo e non dello standard JPA che è quindi in tutto e per tutto un derivato, è stata una scelta motivata dal fatto che Hibernate mette a disposizione un insieme di funzioni aggiuntive per lo sviluppo dei servizi ORM (Object-relational mapping).

Il principale obiettivo di Hibernate è quindi quello di fornire la mappatura tra le tabelle contenute nel database relazionale e le classi Java, ossia viene creato un collegamento logico tra una classe e l'entità relazionale (ad ogni classe viene specificato a quale tabella si riferisce, ad ogni colonna presente nella tabella viene associato un attributo della classe, ecc..). Le classi utilizzate per effettuare questo collegamento logico sono più precisamente delle JavaBean, le quali si differenziano dalle altre in

JavaBean

quanto devono rispettare delle specifiche ben definite (reperibili alla seguente pagina web[1]).

caratteristiche JavaBean

Le note più importanti da rispettare sono:

- al costruttore (metodo usato per creare le istanze e inizializzarle) non deve essere passato alcun parametro;
- gli attributi devono essere tutti privati;
- l'accesso agli attributi privati deve avvenire tramite i metodi get, set (il nome del metodo deve utilizzare la convenzione standard per i nomi - getNomeAttributo, setNomeAttributo);
- la classe deve supportare la serializzazione (per la persistenza dei dati nel database).

Una volta creata la suddetta mappatura, l'accesso e l'aggiornamento delle informazioni relative alla base dati avviene attraverso l'utilizzo di metodi descritti nella classe mappata; in questa classe, inoltre, devono essere applicate tutte le query tramite le quali si vogliono reperire determinate informazioni: le query sono scritte utilizzando il linguaggio HQL (Hibernate Query Language), molto simile al linguaggio SQL standard e al JPQL (quest'ultimo derivato proprio da HQL).

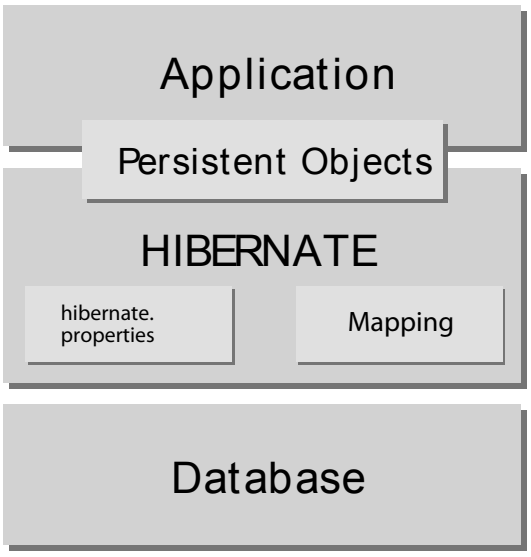

Figura 13: Architettura Hibernate

1 http://www.oracle.com/technetwork/java/javase/documentation/spec-136004.html

Come si può osservare dalla figura 13, di fatto viene introdotto un nuovo strato tra l'applicazione che si vuole sviluppare e il database; in questo nuovo livello vengono aggiunti tutti i componenti necessari per la comunicazione tra il database e le entità create.
Nella parte identificata come HIBERNATE.PROPERTIES vengono descritte tutte le opzioni indispensabili per la creazione della connessione con il database, quali il nome del database, il driver JDBC utilizzato, username e password per l'accesso, ecc. Nel caso in cui si migrasse la base dati corrente in una nuova, sarà sufficiente modificare tale descrittore.
La parte identificata come MAPPING è quella che si occupa del collegamento tra le classi e le tabelle.
Infine, il blocco PERSISTENT OBJECT è essenziale per mantenere gli oggetti creati in uno stato persistente; con questo tipo di stato si vuole indicare la possibilità di scrivere i dati sul database in modo definitivo, evitando così il problema della perdita dei dati qualora si riscontrasse qualche problema al software creato.

L'utilizzo di questo framework apporta i seguente miglioramenti rispetto allo standard JPA:

- supporta le annotazioni;
- ottimizza le query;
- facilita la creazione di relazioni tra entità e classi;
- migliora il supporto per le API Java inerenti alle transazioni;
- migliora il supporto per le query inerenti alle operazioni di aggiornamento o di eliminazione.

Di seguito sono riportate le annotazioni utilizzate più frequentemente per la mappatura delle classi:

ANNOTAZIONE	DESCRIZIONE
`@Entity`	dichiara, in base al nome dato alla classe, l'entità a cui si riferisce
`@Table {name="nome"}`	dichiara, in base al nome specificato, l'entità a cui si riferisce; viene utilizzato quando la denominazione della classe è diversa da quella della tabella alla quale ci si vuole collegare

Tabella 5: continua nella prossima pagina

Tabella 5: continua dalla pagina precedente

Annotazione	Descrizione
`@Id`	associa la variabile della classe Java alla chiave primaria della tabella se quest'ultima è composta solamente da un campo
`@IdClass`	utilizzata quando la chiave primaria è composta da più attributi poiché è richiesta la creazione di una classe contenente tutti gli attributi della chiave
`@Column {name="nome"}`	effettua l'associazione tra la variabile utilizzata nella classe e il rispettivo attributo presente nell'entità
`@OneToOne`	specifica che la classe è associata ad un'altra mediante una relazione di tipo uno-a-uno
`@OneToMany`	specifica che la classe è associata ad un'altra mediante una relazione di tipo uno-a-molti
`@ManyToOne`	specifica che la classe è associata ad un'altra mediante una relazione di tipo molti-a-uno
`@JoinTable`	nelle relazioni uno-a-uno, uno-a-molti, molti-a-uno, indica a quale entità viene annessa la classe corrente
`@JoinColumns`	nelle relazioni uno-a-uno, uno-a-molti, molti-a-uno, indica le chiavi esterne utilizzate

Tabella 5: Annotazioni Hibernate

3.4.3 Framework: OpenXava

www.openxava.org

OpenXava è un framework per le applicazioni web impiegato per lo sviluppo di software basato sulle librerie JavaEE. Il principale obiettivo di questo framework è la semplificazione di tutte le attività inerenti alla manutenzione e alla persistenza dei dati presenti all'interno del database utilizzato. Infatti, è possibile sviluppare alcuni moduli chiamati CRUD (Create Read Update Delete) tramite i quali si eseguono operazioni di creazione, lettura, aggiornamento ed eliminazione delle informazioni presenti nelle tabelle del database.

Per la creazione di questi componenti non è richiesta la scrittura di codice Java: OpenXava di fatto si occupa della realizzazione dei modelli interpretando le annotazioni JPA scritte all'interno delle classi e alcune annotazioni proprietarie di OpenXava per la generazione dell'interfaccia grafica. Una volta definite tutte le mappature classi/entità, il framework genera per ciascuna un'interfaccia grafica per l'accesso e la modifica dei dati. È inoltre disponibile un insieme di funzionalità quali la creazione di report in PDF, tabelle in Excel, operazioni per la ricerca.
Tutte queste operazioni risultano pertanto automatizzate: l'unico intervento da parte del programmatore riguarda la definizione della struttura della base dati e qualche operazione per la configurazione del framework.

Grazie alla flessibilità di questo prodotto, il programmatore comunque ha sempre la possibilità di sviluppare particolari funzioni dell'applicazione gestita da OpenXava; questa caratteristica è di primaria importanza, per esempio, nella risoluzione di particolari problemi.
I moduli che verranno generati saranno in questo caso delle portlet web che dovranno essere inserite in Liferay: queste portlet rispettano il pattern MVC e utilizzano la tecnologia Ajax per evitare che ad ogni modifica fatta sui dati venga ricaricata l'intera pagina web.

segue il pattern MVC

Per quanto riguarda l'aspetto grafico (View), OpenXava mette a disposizione l'annotazione @View, utilizzabile all'interno della classe java per definire una propria interfaccia utente diversa da quella predefinita.
La sintassi di @View è la seguente:

```
@View(
   name="name",
   members="members",
   extendsView="view"
)
```

dove i tre parametri (facoltativi) hanno il seguente significato:

1. *name* = NAME: assegna un nome alla visualizzazione utilizzata; impiegato per poter adoperare lo stesso modello in altri frammenti di codice;

2. *members* = MEMBERS: specifica quali sono gli attributi della classe da rendere visibili e come devono essere disposti graficamente; se non viene specificato questo campo, vengono visualizzati tutti gli attributi presenti;

3. *extendsView* = VIEW: include tutti gli attributi specificati in altre visualizzazioni indicandone specificamente il nome.

Caratteristiche

Alcune caratteristiche di OpenXava sono:

- supporto del portale Liferay;
- rispetto dello standard JSR 168;
- supporto delle annotazioni di Hibernate;
- supporto della tecnologia Ajax;
- possibilità di creare interfacce grafiche multilingua;
- possibilità di inserire codice personale;
- aumento della produttività per lo sviluppo di applicazioni web;
- facilità di utilizzo.

Esempio di applicazione OpenXava

Si riporta di seguito un semplice esempio per dimostrare le potenzialità di questo framework.
Viene costruita un'entità PERSONA all'interno del database, contenente i seguenti attributi:

- CF (Codice Fiscale)
- nome
- cognome

La chiave primaria per l'identificazione delle tuple (entità base memorizzate nella base di dati) è ovviamente il Codice Fiscale.
Una volta creata la tabella, si può iniziare a scrivere la classe Java utilizzata per effettuare il mapping: in questo caso si tratta della classe PERSONA.JAVA

Persona.java

```
package Tesi.model;

import javax.annotation.*;
import javax.persistence.*;

import org.openxava.*;
import org.hibernate.*;

@Entity
@Table(name="Persona", schema="Tesi")
public class Persona {
```

```
        @Id @Column(name="CF") @Required
        private String CF;

        @Column(name="nome")
        private String nome;
        @Column(name="cognome")
        private String cognome;

        public void setCF(String a) {
                this.CF = a;
        }

        public String getCF() {
                return this.CF;
        }

        public void setNome(String a) {
                this.nome = a;
        }

        public String getNome() {
                return this.nome;
        }

        public void setCognome(String a) {
                this.cognome = a;
        }

        public String getCognome() {
                return this.cognome;
        }
}
```

La classe contiene una serie di istruzioni per importare le librerie necessarie per la compilazione, la generazione della portlet del framework e per poter inserire all'interno del file le annotazioni jpa, hibernate, openxava. All'interno della classe è necessario creare tutti gli attributi che devono essere associati a ciascuna colonna presente nella tabella. Per ciascun attributo è inoltre obbligatorio implementare due metodi (setNomeAttributo, getNomeAttributo), utilizzati per accedere alle variabili private. Quando la classe è stata completata, è possibile generare la corrispettiva portlet utilizzando gli strumenti messi a disposizione. Il risultato che si ottiene è il seguente:

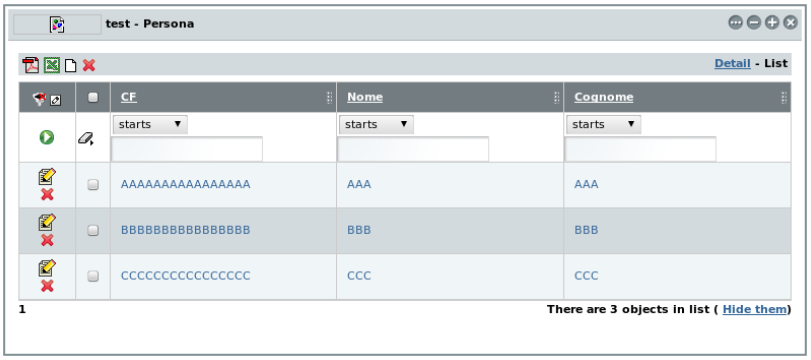

Figura 14: Portlet Persona - List

Come si può notare dalla figura 14, all'interno della portlet vengono visualizzate tutte le tuple presenti attualmente nella tabella del database. L'applicazione creata mette a disposizione dell'utente una serie di funzionalità, tra le quali:

- filtro per la ricerca dei dati;
- cancellazione tuple;
- inserimento tuple;
- modifica tuple;
- generazione tabella in formato pdf;
- generazione tabella in formato excel.

Per testare il corretto funzionamento dell'applicazione creata, è stata fatta una serie di inserimenti e cancellazioni di tuple, utilizzando la seguente schermata:

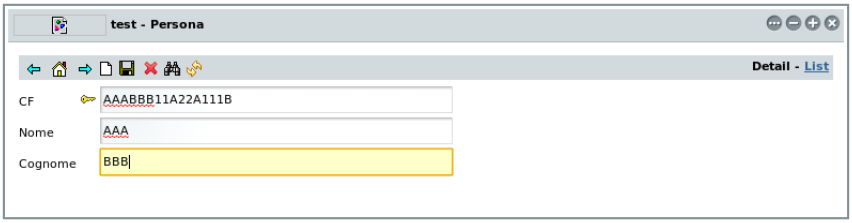

Figura 15: Portlet Persona - Detail

La visualizzazione utilizzata per l'inserimento dei dati può essere modificata tramite l'annotazione @View messa a disposizione da OpenXava. Inserendo nel codice precedente la seguente annotazione,

```
@View(
   members="Persona [CF; nome, cognome]";
)
```

si ottiene:

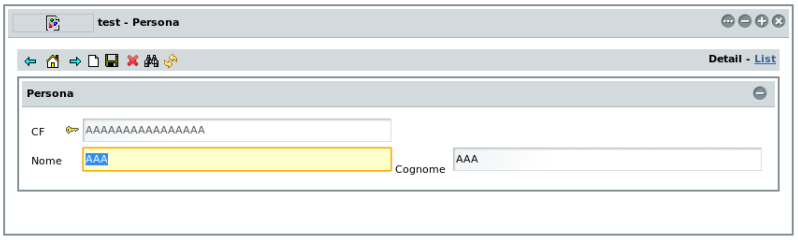

Figura 16: Portlet Persona - Detail @View

A differenza della figura 15, nella figura 16 gli elementi sono stati raggruppati in una nuova finestra, posizionando gli attributi utilizzati su due righe invece che su tre, ponendo l'informazione relativa alla chiave primaria nella prima riga separata da quelle relative al nome e cognome sistemate entrambe in successione nella seconda.

4 SVILUPPO PORTLET

In questo capitolo verranno descritte tutte le operazioni necessarie per la corretta installazione e configurazione del software utilizzato e verranno presentate le portlet Web sviluppate analizzandone le funzionalità tramite i file sorgente creati opportunamente commentati.

INDICE

4.1 CONFIGURAZIONE AMBIENTE DI SVILUPPO

Per installare il software necessario alla creazione delle applicazioni web si è optato per il sistema operativo (SO) Ubuntu 10.04 LTS (64 bit), basato su kernel Linux: alcuni comandi e file di configurazione saranno perciò strettamente legati a questo SO. I programmi che dovranno essere installati sono:

Programmi Utilizzati

- Java SDK
- MySQL
- Eclipse
- Apache Ant
- Liferay Portal

Per semplificare la gestione dei diversi file di configurazione presenti all'interno di ciascuna applicazione, i programmi saranno installati nella seguente cartella di sistema: /opt

4.1.1 Java SDK

Java SDK (Software Development Kit) è il pacchetto java utilizzato per la compilazione dei file sorgente utilizzati poiché non è sufficiente la JRE (Java Runtime Environment, motore per eseguire programmi java) per poter espletare questa funzione; per installarlo è sufficiente scaricarlo dal seguente sito web (link[1] : la versione scaricata è nel formato .bin), una volta scaricata la si può installare rispettando i seguenti passaggi:

```
chmod +x jdk-6u21-linux-x64.bin
./jdk-6u21-linux-x64.bin
sudo mv jdk1.6.0_20 /opt
```

Con queste istruzioni si installa la sdk all'interno del sistema e si sposta la cartella creata nella directory /opt. Per eseguire il comando `javac` (necessario per la compilazione dei sorgenti) dalla cartella del progetto bisogna impostare due variabili d'ambiente: per settarle è necessario aprire il terminale linux e digitare i comandi seguenti:

```
export JAVA_HOME=/opt/jdk1.6.0_20
export PATH=$PATH:/opt/jdk1.6.0_20/bin
```

Per poter mantenere in memoria le variabili anche successivamente al riavvio della macchina, è necessario inserire le due istruzioni sopra descritte all'interno del file: /home/nomeutente/.bashrc

4.1.2 MySQL

www.mysql.it

E' possibile scaricare il database MySQL dal seguente sito web (link[2] : la versione scaricata è nel formato .tar). Una volta scaricato il pacchetto è necessario scompattarlo per poter poi installare il database; prima di avviare l'installazione bisogna però creare il nome dell'utente che sarà amministratore del database stesso e il gruppo a cui l'utente dovrà appartenere (in questo caso si è scelto di impostare il nome dell'utente uguale al nome del gruppo, cioè "mysql").
Attraverso l'utilizzo del terminale di linux, l'insieme dei comandi necessari per l'installazione del prodotto è:

1 http://java.sun.com/javase/downloads/widget/jdk6.jsp
2 http://www.mysql.it/downloads/mysql/

```
tar -xzvf mysql-5.1.48-linux-ia64-glibc23.tar.gz
sudo mv mysql-5.1.48-linux-ia64-glibc23.tar.gz /opt
ln -s mysql-5.1.48-linux-ia64-glibc23.tar.gz mysql
cd /opt/mysql

sudo -s
groupadd mysql
useradd -g mysql mysql
cd mysql
chown -R mysql .
chgrp -R mysql .
scripts/mysql_install_db --user=mysql
chown -R root .
chown -R mysql data
bin/mysqld_safe --user=mysql &
```

Terminata questa procedura, anche in questo caso occorre impostare una variabile d'ambiente: come precedentemente descritto per l'installazione della Java SDK, si aggiunge al file:
/home/nomeutente/.bashrc
la seguente riga:

File del sistema operativo Linux utilizzato per le variabili d'ambiente

```
export MYSQL_HOME=/etc/mysql
```

Una volta completato il processo, è possibile avviare o fermare il database mysql invocando rispettivamente i seguenti script:

```
/opt/mysql/support-files/mysql.server start
/opt/mysql/support-files/mysql.server stop
```

Sebbene sia possibile utilizzare l'interfaccia testuale per la creazione o la modifica del proprio database e delle diverse entità richieste dal progetto, è disponibile un'ulteriore interfaccia grafica che agevola queste operazioni. Questa utility, chiamata MySQL Workbench e disponibile per il sistema operativo utilizzato, la si può scaricare utilizzando questo link[3]; per la sua installazione è necessario digitare sul terminale il seguente comando:

http://wb.mysql.com/

```
sudo dpkg -i
mysql-workbench-gpl-5.2.25-1ubu1004-amd64.deb
```

3 http://dev.mysql.com/downloads/workbench/#downloads

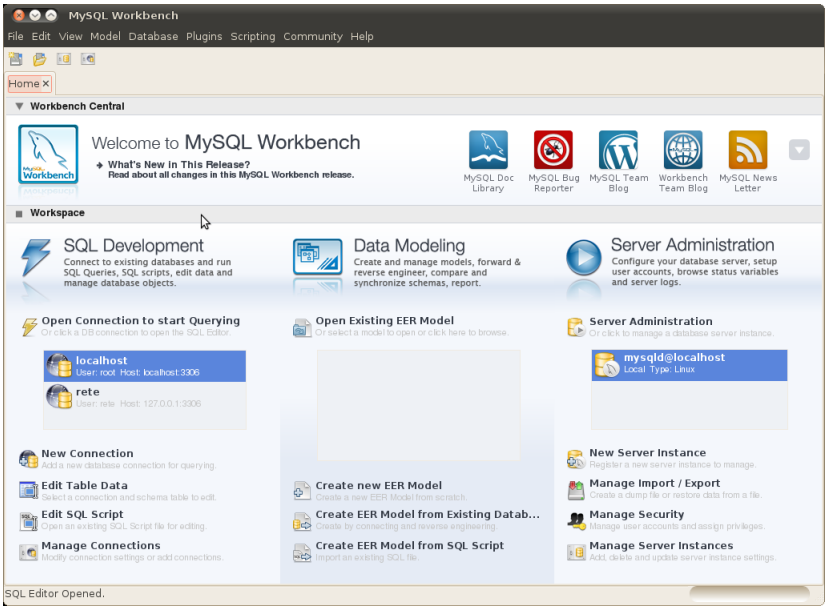

Figura 17: MySQL Workbench

4.1.3 Eclipse

www.eclipse.org/galileo

Per la compilazione delle proprie applicazioni web è consigliato l'impiego di un IDE (Integrated development environment - Ambiente di sviluppo integrato): tra quelli disponibili la scelta è ricaduta su Eclipse Galileo, in quanto mette a disposizione un insieme di strumenti già integrati sia per la creazione delle portlet web sia per la redazione dei propri codici sorgente scritti, inoltre, ha il vantaggio di supportare l'evidenziazione della sintassi di tutti i linguaggi utilizzati.
I file per l'installazione di questo IDE si possono trovare al seguente link[4] (la versione scaricata è nel formato .tar.gz). Per installare Eclipse, è sufficiente estrarre il contenuto del pacchetto scaricato; la cartella così creata verrà spostata in /opt. I comandi sono:

```
tar -xvzf
eclipse-jee-galileo-SR2-linux-gtk-x86_64.tar.gz
sudo mv eclipse /opt
```

Essendo un IDE compilato in java, al momento dell'avvio del programma viene effettuato un controllo al fine di determinare

4 http://www.eclipse.org/downloads/

se nel sistema operativo in cui si sta lavorando è installata la JVM (Java Virtual Machine). Quest'ultima viene rilevata da eclipse grazie alla precedente impostazione delle variabili d'ambiente per la Java SDK, la quale al suo interno contiene anche la JVM.

Plugin Hibernate

Per questo IDE è disponibile il plugin per la piattaforma Hibernate, il quale mette a disposizione una serie di operazione atte a semplificare la creazione della mappatura tra ciascuna classe e l'entità.
Per installare questo plugin, una volta avviato Eclipse è sufficiente inserire il seguente link[5] nella finestra Install, raggiungibile tramite la voce

$$\text{Help} \rightarrow \text{Install New Software.}$$

All'interno di Eclipse è necessario configurare anche l'ambiente per l'utilizzo del framework OpenXava: prima di tutto è necessario scaricare il framework, reperibile al seguente indirizzo[6] e scompattarlo in /opt:

```
cd openxava-4m4.zip /opt
unzip openxava-4m4.zip
```

Per poter integrare l'ambiente all'interno del programma è necessario creare un nuovo progetto utilizzando i file scaricati; per poterlo fare è sufficiente accedere alla voce

$$\text{File} \rightarrow \text{New} \rightarrow \text{JavaProject} \rightarrow$$

$$\rightarrow \text{Create project from existing source}$$

ed inserire nella directory il percorso
/opt/openxava-4m4/workspace/OpenXavaTemplate

4.1.4 Apache Ant

http://ant.apache.org/

Apache Ant è un programma il cui scopo principale è quello di automatizzare le operazioni di compilazione dei progetti Java; si occupa di attività quali: ricerca delle dipendenze tra i file sorgente, creazione delle proprie librerie, creazione della struttura delle cartelle finale del progetto finale.
L'utilizzo di questo programma facilita inoltre la portabilità del

5 http://download.jboss.org/jbosstools/updates/stable/galileo/
6 http://sourceforge.net/projects/openxava/files/openxava/4m4/openxava-4m4.zip/download

progetto in diverse piattaforme (Windows, Unix, ecc.): infatti, l'insieme delle istruzioni che deve eseguire per la regolare generazione del progetto sono definite all'interno di un file in formato xml, modificabile a seconda delle proprie necessità.

Per il suo corretto funzionamento, come per l'IDE Eclipse, necessita della presenza della Java SDK all'interno del pc in cui si sta lavorando e delle apposite variabili d'ambiente di Java. Nonostante questo strumento sia già presente all'interno di Eclipse, per non essere completamente vincolati all'utilizzo di quest'ultimo per la compilazione del progetto, risulta più comodo utilizzare le sue funzionalità digitandole dalla riga di comando della shell di linux: per poterlo fare, una volta scaricato il programma, è necessario impostare un'altra variabile d'ambiente.

E' possibile scaricare Apache ANT dal seguente link[7] (la versione scaricata è nel formato zip). Di seguito vengono riportate le operazioni per l'installazione di questo software:

```
unzip apache-ant-1.8.1-bin.zip
sudo move apache-ant-1.8.1-bin.zip /opt
ln -s apache-ant-1.8.1-bin.zip ant
```

e la definizione delle sue variabili d'ambiente:

```
export ANT_HOME=/opt/ant
export PATH=$PATH:$ANT_HOME/bin
```

4.1.5 Liferay Portal

www.liferay.com

La versione utilizzata del portale Liferay è la 5.2.3 equipaggiata e già configurata con il servlet container TomCat 6.0. Per l'installazione è necessario scaricare i seguenti file dal sito web di Liferay, raggiungibile dal seguente link[8]:

- liferay-portal-tomcat-6.0-5.2.3.zip
- liferay-portal-src-5.2.3.zip
- liferay-plugins-sdk-5.2.3.zip

Il primo pacchetto contiene l'intero portale (file di configurazione, esempi, file per l'interfaccia grafica), alcuni script di servizio

7 http://mirror.nohup.it/apache/ant/binaries/apache-ant-1.8.1-bin.zip
8 http://sourceforge.net/projects/lportal/files/

per l'avvio e l'arresto del server web tomcat e un insieme di librerie necessario per la creazione di portlet web.
Il secondo contiene i sorgenti e le librerie Java del portale: sono necessarie qualora si dovessero utilizzare delle funzioni o delle strutture strettamente legate a Liferay (per esempio per vedere se un utente è autenticato).
Infine, l'ultimo file contiene un insieme di utility e librerie per la creazione di particolari tools aggiuntivi da integrare a quelli già presenti nel portale, come per esempio la creazione di temi e layout di pagina. Inoltre, all'interno del pacchetto è presente anche un file script (sia per il sistema operativo windows sia per quello linux) per la generazione della struttura delle cartelle relative ad una semplice portlet.

Una volta scaricati i file, è necessario estrarli e spostarli nella cartella di sistema /opt utilizzando i seguenti comandi:

```
unzip -d liferay liferay-portal-tomcat-6.0-5.2.3.zip
cd liferay
unzip -d sdk ../liferay-plugins-sdk-5.2.3.zip
unzip -d src ../liferay-portal-src-5.2.3.zip
mv liferay /opt
```

Completata l'estrazione, è possibile avviare e arrestare il server web del portale grazie a questi comandi:

```
./liferay-portal-5.2.3/tomcat-6.0.18/bin/startup.sh
./liferay-portal-5.2.3/tomcat-6.0.18/bin/shutdown.sh
```

Anche questi due script richiedono, per il corretto funzionamento, la presenza nel sistema utilizzato delle variabili d'ambiente precedentemente impostate.

All'interno di Liferay è necessario compiere una serie di operazioni di configurazione al fine di poter eseguire le applicazioni web create con OpenXava.
Per prima cosa, è necessario eseguire la copia dei file EJB.JAR e JTA.JAR contenuti all'interno del file zip scaricabile dal seguente link[9] all'interno della cartella comprendente le librerie del portale.

```
unzip openxava-portal-3.1.zip
cp openxava-portal-3.1/common/lib/ejb.jar
```

9 http://sourceforge.net/projects/openxava/files/openxava-portal/3.1/openxava-portal-3.1.zip/download

```
    /opt/liferay-portal-5.2.3/tomcat-6.0.18/lib/
cp openxava-portal-3.1/common/lib/ext/jta.jar
    /opt/liferay-portal-5.2.3/tomcat-6.0.18/lib/ext/
```

Successivamente è necessario modificare il file CONTEXT.XML (utilizzato per inserire alcuni parametri opzionali di configurazione utilizzabili da tutte le applicazioni web disponibili all'interno del portale) aggiungendo le seguenti direttive

```
<Resource name="jdbc/nomeSchemaDatabase"
 auth="Container" type="javax.sql.DataSource"
 maxActive="100" maxIdle="30" maxWait="10000"
 username="username" password="password"
 driverClassName="com.mysql.jdbc.Driver"
 url="jdbc:mysql://localhost:3306/nomeSchemaDatabase"
 />
```

nelle quali vengono elencate le impostazioni utilizzate per permettere alle applicazioni web sviluppate di servirsi del database installato dall'utente.

4.2 DESCRIZIONE APPLICAZIONE WEB

Obiettivo

Di seguito si descrive l'applicazione web creata: l'obiettivo prefissato prima di iniziare a creare questo software è stato quello di capire il funzionamento del portale Liferay e, in seguito, quello di assimilare le conoscenze necessarie per la creazione e gestione di una portlet web.

Introduzione

L'applicazione realizzata si occupa della visualizzazione di una lista di alcuni veicoli presenti attualmente sul mercato. Ciascuno di questi è suddiviso prima di tutto in categorie, trattandosi principalmente di autoveicoli oppure di motoveicoli; un'ulteriore suddivisione si basa poi sull'azienda produttrice dei veicoli e, infine, per ciascuna di queste sarà possibile visualizzare i mezzi prodotti.
Per lo sviluppo di questa applicazione web sono state utilizzate tutte le tecnologie menzionate nei capitoli precedenti e ci si è serviti del pattern MVC descritto nel capitolo 2.4.
Per la realizzazione del software sono state create diverse portlet web: quella principale si occupa della visualizzazione del catalogo dei veicoli presenti nel database mentre le altre portlet sono state sviluppate utilizzando il framework Openxava (3.4.3) e sono state perciò usate per eseguire operazioni di inserimento,

modifica o cancellazione di informazioni all'interno del database.

4.2.1 Base Dati

La base dati consiste in tre entità: *Entità Relazionali*

1. CATEGORIA: questa tabella contiene le informazioni quali l'id per l'identificazione, il nome della categoria e una breve descrizione della tipologia di veicoli che potranno essere visualizzati;

2. MARCA: in questo schema sono inseriti alcuni dati appartenenti a ciascuna casa automobilista/motociclistica, quali il nome, il logo dell'azienda e la descrizione. Per l'identificazione è stato utilizzato un numero intero auto-incrementale;

3. MODELLO: questa entità contiene le informazioni di ciascun veicolo come nome, foto, descrizione; anche in questo caso è stato utilizzato un id per l'identificazione di ciascun modello.

Lo schema relazionale utilizzato è il seguente:

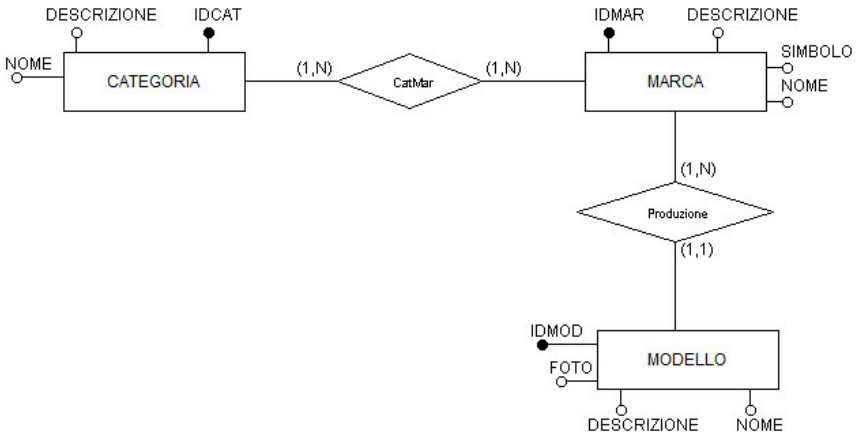

Figura 18: Schema ER

Come si può osservare è presente una relazione di tipo molti-a-molti tra categoria e marca e una relazione di tipo uno-a-molti tra modello e marca: la prima relazione è stata necessaria in quanto ci sono alcune aziende che producono sia autoveicoli che motoveicoli, mentre la seconda relazione dipende dal fatto che

ciascun modello può appartenere esclusivamente ad una azienda.

Associazioni

Per la risoluzione dell'associazione molti-a-molti, è necessaria la creazione di un'altra entità, denominata CatMar, avente come identificativo le chiavi primarie della tabella Cateogoria e Marca.
Per creare il collegamento logico previsto dalla relazione uno-a-molti, invece, è indispensabile aggiungere all'entità Modello un ulteriore attributo utilizzato per associare il modello alla marca alla quale appartiene: questo attributo è la chiava esterna dell'entità Marca, denominato appunto IDMAR.
Pertanto, lo schema logico finale è:

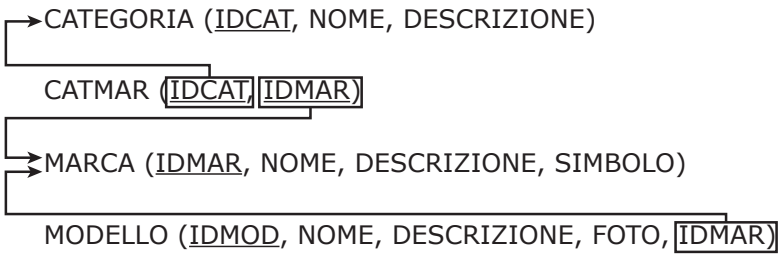

Figura 19: Schema Logico

4.2.2 Portlet OpenXava

Una volta create le varie entità è possibile creare le portlet per mezzo del framework OpenXava.
Utilizzando eclipse, è necessario generare un nuovo progetto java tramite il comando `ant build` raggiungibile all'interno della cartella

$$OpenXavaTemplate \rightarrow CreateNewProject.xml \rightarrow Run\ As$$

precedentemente creata.
Tutte le librerie necessarie per lo sviluppo delle portlet web, nonché l'utilizzo delle librerie di Hibernate, sono automaticamente incorporate nel progetto e, di conseguenza, non è richiesta nessun'altra operazione per la corretta compilazione dei file sorgente che verranno elaborati.

Configurazione Hibernate

L'unica azione da effettuare per la configurazione del progetto prima di iniziare a scrivere codice Java, interessa l'interfacciamento del progetto con il database utilizzato. É possibile far questo inserendo delle direttive nei due file hibernate.cfg.xml

e PERSISTENCE.XML, creati in precedenza con il progetto.
I file risultanti sono i seguenti:

File: hibernate.cfg.xml

```
1  <!DOCTYPE hibernate-configuration PUBLIC
2  "-//Hibernate/Hibernate Configuration DTD//EN"
3  "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
4
5  <hibernate-configuration>
6
7    <session-factory>
8
9      <!-- Tomcat + Hypersonic -->
10     <property name="hibernate.connection.datasource">
11       java:comp/env/jdbc/Laurea
12     </property>
13     <property name="hibernate.dialect">
14       org.hibernate.dialect.MySQLInnoDBDialect
15     </property>
16     <property name="hibernate.jdbc.use_get_generated_keys">
17       false</property>
18     <property name="hibernate.show_sql">true</property>
19
20     <property name="hibernate.query.factory_class">
21       org.hibernate.hql.classic.ClassicQueryTranslatorFactory
22     </property>
23     <property name="hibernate.cache.provider_class">
24       org.hibernate.cache.EHCacheProvider
25     </property>
26
27     <!-- GalleryImage is needed only if you uses
28          IMAGES_GALLERY/GALERIA_IMAGENES stereotype -->
29     <mapping resource="GalleryImage.hbm.xml"/>
30
31   </session-factory>
32
33 </hibernate-configuration>
```

Le righe 10 e 13 presenti all'interno del file sono di fondamentale importanza in quanto specificano il nome (indicato nel file di configurazione del portale Liferay: 4.1.5) e il tipo di database utilizzato. La riga 18 è invece opzionale: è stata inserita successivamente in quanto permette di visualizzare le query effettuate durante l'utilizzo delle portlet create, facilitando così le operazioni di manutenzione svolte durante lo sviluppo dell'applicazione web.

File: persistence.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3
4  <persistence xmlns="http://java.sun.com/xml/ns/persistence"
5     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6     xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
7   http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
8                version="1.0">
9
10   <!-- Tomcat + Hypersonic -->
11   <persistence-unit name="default">
12     <non-jta-data-source>
13       java:comp/env/jdbc/Laurea
14     </non-jta-data-source>
15     <class>org.openxava.session.GalleryImage</class>
16     <class>model.Categoria</class>
17     <class>model.Marca</class>
18     <class>model.Modello</class>
19     <properties>
20       <property name="hibernate.dialect"
21                 value="org.hibernate.dialect.MySQLInnoDBDialect"/>
22     </properties>
23   </persistence-unit>
24
25
26   <!-- JUnit Hypersonic -->
27   <persistence-unit name="junit">
28     <properties>
29       <property name="hibernate.connection.driver_class"
30                 value="com.mysql.jdbc.Driver"/>
31       <property name="hibernate.dialect"
32                 value="org.hibernate.dialect.MySQLInnoDBDialect"/>
33       <property name="hibernate.connection.url"
34                 value="jdbc:hsqldb:hsql://localhost:3306/Laurea"/>
35       <property name="hibernate.connection.username"
36                 value="username" />
37       <property name="hibernate.connection.password"
38                 value="password" />
39     </properties>
40   </persistence-unit>
41
42
43 </persistence>
```

Le righe 29, 31 e 33 specificano al server web in quale database effettuare le operazioni di interrogazione, salvataggio o modifica; le righe 35 e 37 sono invece utilizzate per accedere con le corrette credenziali al database SQL (credenziali create durante l'installazione del server MySQL, 4.1.2).

Le portlet sviluppate con il framework OpenXava vengono generate automaticamente a partire dalle classi JavaBean, utilizzate

per la mappatura tra gli oggetti che verranno istanziati e le entità presenti nel database.
Di seguito si riportano le tre classi Java utilizzate per mappare le tabelle CATEGORIA, MARCA e MODELLO:

Classi Java utilizzate per mappare le entità

File: Categoria.java

```
1  package model;
2
3  import javax.annotation.*;
4  import javax.persistence.*;
5
6
7  import java.util.HashSet;
8  import java.util.Set;
9
10 import model.Marca;
11
12 @Entity
13 @Table(name = "Categoria", schema = "Laurea")
14 public class Categoria {
15
16         @Id
17         @Column(name = "IDCAT")
18         private int IDCAT;
19         @Column(name = "NOME")
20         private String NOME;
21         @Column(name = "DESCRIZIONE")
22         private String DESCRIZIONE;
23
24         @ManyToMany(cascade=CascadeType.ALL)
25         @JoinTable(name="CATMAR",
26                 joinColumns = {
27                         @JoinColumn(name="IDCAT")},
28                 inverseJoinColumns = {
29                         @JoinColumn(name="IDMAR")}
30         )
31         private Set<Marca> MARCA;
32
33         public Categoria() {
34                 this.MARCA = new HashSet<Marca>();
35         }
36
37         public void setIDCAT(int newID) {
38                 this.IDCAT = newID;
39         }
40
41         public int getIDCAT() {
42                 return this.IDCAT;
43         }
44
45         public void setNOME(String newNOME) {
46                 this.NOME = newNOME;
```

```
47          }
48
49          public String getNOME() {
50                  return this.NOME;
51          }
52
53          public void setDESCRIZIONE(String newDescr) {
54                  this.DESCRIZIONE = newDescr;
55          }
56
57          public String getDESCRIZIONE() {
58                  return this.DESCRIZIONE;
59          }
60
61          public void setMARCA(Set<Marca> newMARCA) {
62                  this.MARCA = newMARCA;
63          }
64
65          public Set<Marca> getMARCA() {
66                  return this.MARCA;
67          }
68
69  }
```

Questa classe definisce l'entità CATEGORIA; oltre ai vari metodi get e set utilizzati per salvare i dati prelevati dalla corrispondente tabella, è presente l'attributo

- `private Set<Marca> MARCA`

indispensabile per creare l'associazione molti-a-molti con l'entità MARCA ed effettuare le query SQL contenenti la clausola JOIN tra queste due tabelle. Le annotazioni presenti dalla riga 24 alla riga 31 indicano la presenza di questa relazione, specificando la tabella di join e le due chiavi tramite le quali è possibile effettuare la combinazione delle due tabelle.

File:Marca.java

```
1   package model;
2
3   import javax.annotation.*;
4   import javax.persistence.*;
5
6   import java.util.HashSet;
7   import java.util.Set;
8
9   import model.Modello;
10  import model.Categoria;
11
12  @Entity
13  @Table(name = "Marca", schema = "Laurea")
```

```
14 public class Marca {
15
16         @Id
17         @Column(name = "IDMAR")
18         private int IDMAR;
19         @Column(name = "NOME")
20         private String NOME;
21         @Column(name = "DESCRIZIONE")
22         private String DESCRIZIONE;
23         @Column(name = "SIMBOLO")
24         private String SIMBOLO;
25
26         @ManyToMany(mappedBy="MARCA")
27         private Set<Categoria> CATEGORIA;
28
29         @OneToMany(mappedBy="IDMARCA")
30         private Set<Modello> MODELLO;
31
32         public Marca() {
33                 this.MODELLO = new HashSet<Modello>();
34                 this.CATEGORIA = new HashSet<Categoria>();
35         }
36
37         public void setNOME(String newNOME) {
38                 this.NOME = newNOME;
39         }
40
41         public String getNOME() {
42                 return this.NOME;
43         }
44
45         public void setIDMAR(int newIDMAR) {
46                 this.IDMAR = newIDMAR;
47         }
48
49         public int getIDMAR() {
50                 return this.IDMAR;
51         }
52
53         public void setDESCRIZIONE(String newDESCRIZIONE) {
54                 this.DESCRIZIONE = newDESCRIZIONE;
55         }
56
57         public String getDESCRIZIONE() {
58                 return this.DESCRIZIONE;
59         }
60
61         public void setSIMBOLO(String newSIMBOLO) {
62                 this.SIMBOLO = newSIMBOLO;
63         }
64
65         public String getSIMBOLO() {
66                 return this.SIMBOLO;
67         }
```

```
68
69          public void setMODELLO(Set<Modello> mod) {
70                  this.MODELLO = mod;
71          }
72
73          public Set<Modello> getMODELLO() {
74                  return this.MODELLO;
75          }
76
77          public void setCATEGORIA(Set<Categoria> mod) {
78                  this.CATEGORIA = mod;
79          }
80
81          public Set<Categoria> getCATEGORIA() {
82                  return this.CATEGORIA;
83          }
84
85  }
```

Questa classe definisce l'entità MARCA; anche in questo caso sono presenti tutti i metodi get e set usati per estrarre i dati dal database e sono inoltre presenti i due attributi

- `private Set<Categoria> CATEGORIA`
- `private Set<Modello> MODELLO`

Il primo si riferisce alla relazione molti-a-molti con la tabella Categoria e, infatti, attraverso l'annotazione
`@ManyToMany(mappedBy=MARCA)`
viene specificato l'attributo della classe Categoria contenente i dati della tabella Marca.
In modo pressoché uguale viene specificata la relazione con l'entità MODELLO, modificando solo il tipo di associazione, che in questo caso è uno-a-molti, e l'attributo utilizzato per la JOIN, che in questo caso corrisponde a `IDMARCA`, specificato dall'annotazione
`@OneToMany(mappedBy = IDMARCA).`

File:Modello.java

```
1   package model;
2
3   import javax.annotation.*;
4   import javax.persistence.*;
5
6   import org.hibernate.*;
7
8   import model.Marca;
9
10  @Entity
```

```
11 @Table(name = "Modello", schema = "Laurea")
12 public class Modello {
13
14         @Id
15         @Column(name = "IDMOD")
16         private int IDMOD;
17         @Column(name = "NOME")
18         private String NOME;
19         @Column(name = "DESCRIZIONE")
20         private String DESCRIZIONE;
21         @Column(name = "FOTO")
22         private String FOTO;
23
24         @ManyToOne
25         @JoinColumn(name="IDMAR")
26         private Marca IDMARCA;
27
28         public void setIDMOD(int n) {
29                 this.IDMOD = n;
30         }
31
32         public int getIDMOD() {
33                 return this.IDMOD;
34         }
35
36         public void setIDMARCA(Marca n) {
37                 this.IDMARCA = n;
38         }
39
40         public Marca getIDMARCA() {
41                 return this.IDMARCA;
42         }
43
44         public void setNOME(String n) {
45                 this.NOME = n;
46         }
47
48         public String getNOME() {
49                 return this.NOME;
50         }
51
52         public void setDESCRIZIONE(String n) {
53                 this.DESCRIZIONE = n;
54         }
55
56         public String getDESCRIZIONE() {
57                 return this.DESCRIZIONE;
58         }
59
60         public void setFOTO(String n) {
61                 this.FOTO = n;
62         }
63
64         public String getFOTO() {
```

```
65                  return this.FOTO;
66          }
67  }
```

Quest'ultima classe specifica infine l'entità MODELLO; vengono indicati tutti gli attributi presenti nella tabella del database, i relativi metodi per accedere alle variabili private, l'associazione uno-a-molti e la colonna con la quale deve essere effettuata la JOIN, qualora fosse richiesta da una query SQL (righe 24 e 25). A differenza delle altre due classi, si può osservare come non sia necessario definire un attributo di tipo `Set` per effettuare l'associazione: il tutto dipende dal fatto che a ciascun modello è associata solamente una marca.
L'annotazione utilizzata pertanto è:

- ```
  @ManyToOne
  @JoinColumn(name="IDMAR")
  private Marca IDMARCA;
  ```

Completata la creazione delle classi, è possibile generare le portlet OpenXava attraverso il comando `ant build`, eseguibile con gli strumenti messi a disposizione dall'IDE Eclipse. Attraverso questo comando vengono generate tante portlet quante sono le classi descritte: in questo caso vengono quindi create tre portlet web che permettono di manipolare le informazioni presenti in tutte le entità mappate dalle rispettive classi.
Le portlet generate rispettano il pattern MVC: la parte Model è rappresentata dalle classi sopra descritte, mentre le parti View (utilizzata per l'interfaccia con l'utente) e Controller (utilizzata per eseguire le query sul database) rientrano tra i compiti del framework, il quale provvede a generarle.

Di seguito si riportano alcuni screenshot dell'interfaccia con la quale vengono visualizzate queste applicazioni web.
Nella figura 20 si possono osservare i dettagli associati alla categoria MACCHINA: in questo caso, viene illustrata una tabella contenente tutte le varie marche delle case automobilistiche associate a questa categoria.
Nelle figure 21 e 22 vengono illustrate le finestre che compaiono quando si vuole inserire una nuova marca: nella prima figura si possono notare le varie caselle di testo associate a ciascun attributo presente nell'entità MARCA e, una volta creato il nuovo elemento, è possibile associargli la categoria a cui appartiene ed i modelli di veicoli che vengono prodotti.
Infine, la figura 23 visualizza la portlet Modello nella modalità dettagli: anche in questo caso, per ciascun elemento presente nella tabella MODELLO vengono illustrate la marca e la categoria
  ```

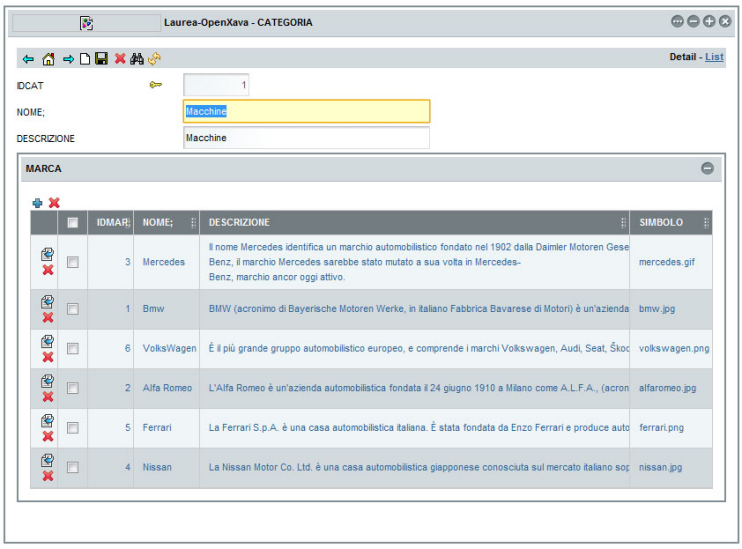

Figura 20: Portlet Categoria: Visualizzazione dettagli

a cui l'oggetto appartiene.
Per ciascuna portlet generata vengono visualizzate tutte le tuple presenti in quel momento all'interno del database: questa funzionalità è molto utile per la gestione dei dati, in quanto permette di modificarli attraverso l'interfaccia grafica creata senza nessuna scrittura di codice SQL.

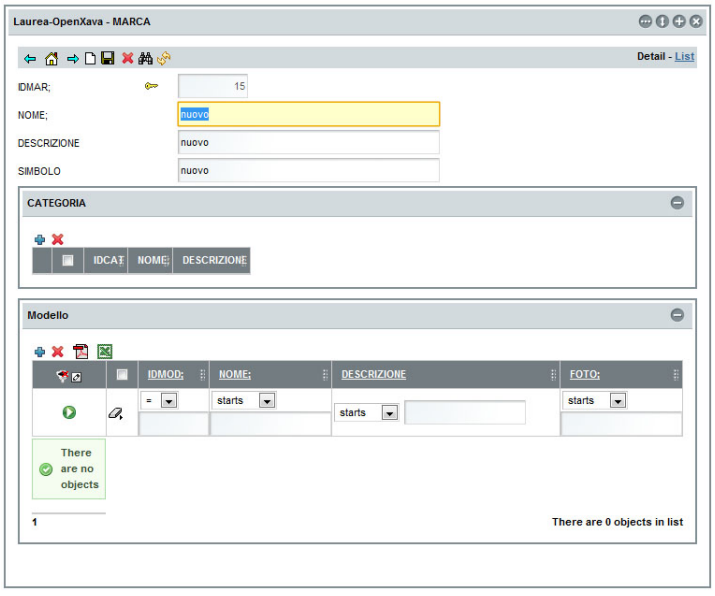

Figura 21: Portlet Marca: Inserimento di una nuova marca

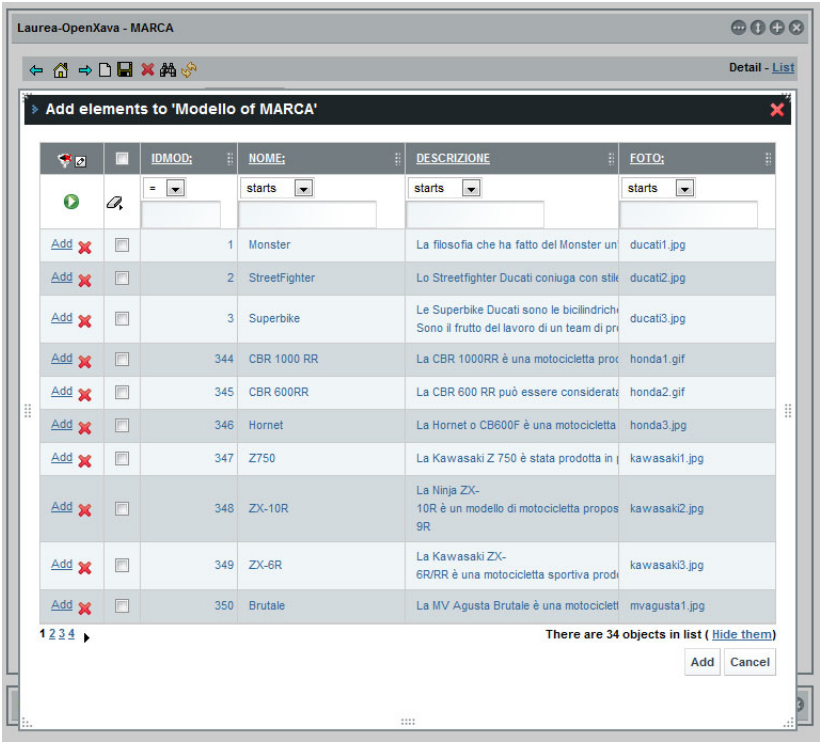

Figura 22: Portlet Marca: Associazione di un modello alla nuova marca creata

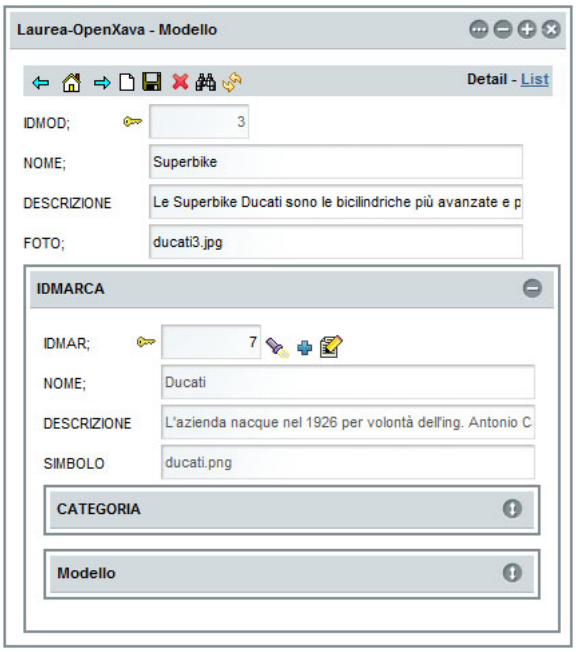

Figura 23: Portlet Modello: Visualizzazione dettagli

4.2.3 Portlet Catalogo

Descrizione

La portlet Catalogo è l'applicazione web che si occupa di illustrare i diversi veicoli presenti all'interno del database. Questo applicativo suddivide la ricerca dei mezzi presenti nella tabella MODELLO a seconda della categoria a cui essi appartengono; una volta scelta la categoria, operazione che può essere effettuata quando la portlet è in modalità EDIT (2.3.2), vengono visualizzati tutti i marchi associati e, selezionando uno di questi, vengono elencati tutti i modelli collegati alla la casa produttrice.

Creazione Portlet

A differenza delle portlet generate in precedenza, dopo aver definito la struttura delle cartelle all'interno delle quali andranno creati i diversi file sorgente, per la creazione di questa applicazione è necessario innanzitutto compiere un insieme di operazioni utili per la definizione della portlet stessa, informazioni indispensabili per l'inserimento dell'applicazione all'interno del portale Liferay (azioni svolte in modo completamente automatico per le portlet OpenXava grazie all'utilizzo del framework).
É possibile attuare la definizione delle cartelle eseguendo gli script messi a disposizione dal portale Liferay:

```
cd /opt/liferay/sdk/portlets
./create.sh catalogo
cd catalogo-portlet
ant setup-eclipse
```

Con queste istruzioni, oltre alla creazione delle varie cartelle, viene generato anche il progetto da importare successivamente con Eclipse, contenente già le librerie standard per la creazione delle portlet.

Prima di definire i file per la configurazione della portlet, è doveroso descrivere la struttura appena creata, al fine di rendere più comprensibili in seguito le impostazioni che si inseriranno nei vari file.
Come si può osservare dalla figura 24, il progetto generato è ripartito in due cartelle principali: SRC, all'interno della quale saranno contenute tutte le classi java create per implementare la parte Controller e Model della portlet, mentre la cartella DOCROOT dovrà racchiudere la parte View e la configurazione della portlet stessa. Per la configurazione dell'applicazione web è necessario modificare i tre file di configurazione:

File di configurazione

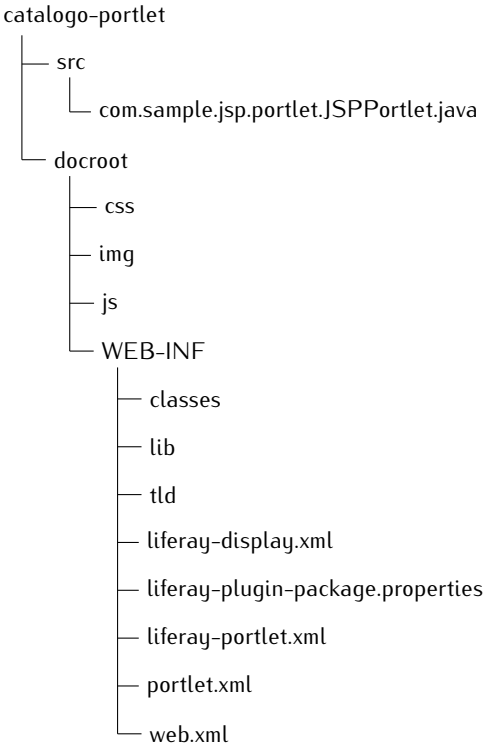

Figura 24: Struttura Cartelle

- portlet.xml
- liferay-portlet.xml
- liferay-plugin-package.properties

Il file PORTLET.XML contiene le direttive per la descrizione della portlet: al suo interno sono presenti dei tag xml che indicano il nome, la descrizione, i file che si occupano della visualizzazione delle pagine, il file che gestisce le azioni generate dall'utente, le modalità implementate e ulteriori informazioni per la ricerca della portlet all'interno del portale. Rispetto al file originario, è stata inserita la modalità EDIT e il rispettivo file jsp, utilizzato per settare il tipo di categoria scelto.

portlet.xml

```
<?xml version="1.0"?>

<portlet-app
   xmlns="http://java.sun.com/xml/ns/
portlet/portlet-app_2_0.xsd"
   version="2.0"
```

```
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/
portlet-app_2_0.xsd http://java.sun.com/xml/
ns/portlet/portlet-app_2_0.xsd"
>

  <portlet>
    <portlet-name>Veicoli</portlet-name>
    <display-name>Veicoli</display-name>
    <portlet-class>
      com.sample.jsp.portlet.JSPPortlet
    </portlet-class>
    <init-param>
      <name>view-jsp</name>
      <value>/view.jsp</value>
    </init-param>
    <init-param>
      <name>edit-jsp</name>
      <value>/edit.jsp</value>
    </init-param>
    <expiration-cache>0</expiration-cache>
    <supports>
      <mime-type>text/html</mime-type>
      <portlet-mode>VIEW</portlet-mode>
      <portlet-mode>EDIT</portlet-mode>
    </supports>
    <portlet-info>
      <title>Veicoli</title>
      <short-title>Laurea</short-title>
      <keywords>Veicoli</keywords>
    </portlet-info>
    <security-role-ref>
      <role-name>administrator</role-name>
    </security-role-ref>
    <security-role-ref>
      <role-name>guest</role-name>
    </security-role-ref>
    <security-role-ref>
      <role-name>power-user</role-name>
    </security-role-ref>
    <security-role-ref>
      <role-name>user</role-name>
    </security-role-ref>
  </portlet>
</portlet-app>
```

Il file LIFERAY-PORTLET.XML è un'estensione del precedente file richiesto dal portale Liferay per impostare ulteriori parametri, tra i quali si possono individuare il file che si occupa dello stile

della portlet, il file in cui sono contenuti gli script javascript, il nome della classe da utilizzare all'interno del foglio di stile per impostare la grafica della portlet. La presenza delle impostazioni appena descritte determina, quindi, la differenza sostanziale rispetto al file originario.

liferay-portlet.xml

```
<?xml version="1.0"?>
<!DOCTYPE liferay-portlet-app PUBLIC
"-//Liferay//DTD Portlet Application 5.2.0//EN"
"http://www.liferay.com/dtd/liferay-portlet-app_5_2_0.dtd"
>

<liferay-portlet-app>
  <portlet>
    <portlet-name>Veicoli</portlet-name>
    <icon>/icon.png</icon>
    <instanceable>true</instanceable>
    <header-portlet-css>
      /css/test.css
    </header-portlet-css>
    <footer-portlet-javascript>
      /js/test.js
    </footer-portlet-javascript>
    <css-class-wrapper>
      portlet-Laurea
    </css-class-wrapper>
  </portlet>
  <role-mapper>
    <role-name>administrator</role-name>
    <role-link>Administrator</role-link>
  </role-mapper>
  <role-mapper>
    <role-name>guest</role-name>
    <role-link>Guest</role-link>
  </role-mapper>
  <role-mapper>
    <role-name>power-user</role-name>
    <role-link>Power User</role-link>
  </role-mapper>
  <role-mapper>
    <role-name>user</role-name>
    <role-link>User</role-link>
  </role-mapper>
</liferay-portlet-app>
```

Il file LIFERAY-PLUGIN-PACKAGE.PROPERTIES infine, contiene alcune proprietà relative alla portlet e le librerie non standard utiliz-

zate per la compilazione dei sorgenti e richieste durante l'esecuzione dell'applicazione (librerie prelevate dalla cartella tomcat-6.0.18/webapps/ROOT/WEB-INF/lib/ contenuta all'interno del portale liferay). Rispetto al file originario, sono state inserite le librerie necessarie per l'utilizzo della piattaforma Hibernate.

liferay-plugin-package.properties

```
name=Laurea
module-group-id=liferay
module-incremental-version=1
tags=
short-description=
change-log=
page-url=http://www.liferay.com
author=Nicola
licenses=MIT

portal.dependency.jars=\
        hibernate-jpa-2.0-api-1.0.0.Final.jar\
        antlr-2.7.6.jar\
        commons-collections-3.1.jar\
        dom4j-1.6.1.jar\
        javassist-3.9.0.GA.jar\
        jta-1.1.jar\
        slf4j-api-1.5.8.jar\
        cglib-2.2.jar\
        javassist-3.9.0.GA.jar\
        c3p0-0.9.1.jar\
        ehcache-1.5.0.jar\
        infinispan-core-4.0.0.FINAL.jar\
        jbosscache-core-3.2.1.GA.jar\
        oscache-2.1.jar\
        proxool-0.8.3.jar\
        swarmcache-1.0RC2.jar
```

File di configurazione con il database e mappatura delle entità

Per quanto concerne le impostazioni relative all'interfacciamento tra l'applicazione e il database, è sufficiente inserire i file di configurazione e le classi java creati in precedenza per le portlet OpenXava all'interno del progetto attuale.
Per mantenere ordinata la struttura dei file all'interno della cartella SRC è stata creata un'ulteriore sottocartella denominata Model al cui interno sono stati copiati i seguenti file:

- HIBERNATE.CFG.XML
- PERSISTENCE.XML
- CATEGORIA.JAVA
- MARCA.JAVA

- MODELLO.JAVA

Logica e Presentazione della portlet

La sezione Controller della portlet, prevista dal pattern MVC, comprende tutte le funzioni impiegate per descrivere la logica dell'applicazione. Nell'esempio qui sviluppato, le attività create interessano essenzialmente la gestione degli eventi generati dall'utente durante l'utilizzo dell'applicazione: questi eventi sono gestiti dal file JSPPORTLET.JAVA.

JSPPortlet.java

```
1  package com.sample.jsp.portlet;
2
3  import com.liferay.portal.SystemException;
4  import com.liferay.portal.kernel.log.Log;
5  import com.liferay.portal.kernel.log.LogFactoryUtil;
6  import com.liferay.portal.kernel.util.ParamUtil;
7  import com.liferay.portlet.PortletPreferencesFactoryUtil;
8
9  import java.io.IOException;
10
11 import javax.portlet.ActionRequest;
12 import javax.portlet.ActionResponse;
13 import javax.portlet.GenericPortlet;
14 import javax.portlet.PortletException;
15 import javax.portlet.PortletPreferences;
16 import javax.portlet.PortletRequestDispatcher;
17 import javax.portlet.RenderRequest;
18 import javax.portlet.RenderResponse;
19
20 public class JSPPortlet extends GenericPortlet {
21
22     protected String editJSP;
23     protected String helpJSP;
24     protected String viewJSP;
25
26     private static Log _log =
27         LogFactoryUtil.getLog(JSPPortlet.class);
28
29
30     public void init() throws PortletException {
31         editJSP = getInitParameter("edit-jsp");
32         helpJSP = getInitParameter("help-jsp");
33         viewJSP = getInitParameter("view-jsp");
34     }
35
36     public void doDispatch(RenderRequest renderRequest,
37                            RenderResponse renderResponse)
38         throws IOException, PortletException {
39
```

```
40          String jspPage =
41              renderRequest.getParameter("jspPage");
42
43          if (jspPage != null) {
44              include(jspPage, renderRequest, renderResponse);
45          } else {
46              super.doDispatch(renderRequest, renderResponse);
47          }
48      }
49
50      public void doEdit(RenderRequest renderRequest,
51                         RenderResponse renderResponse)
52          throws IOException, PortletException {
53
54          if (renderRequest.getPreferences() == null) {
55              super.doEdit(renderRequest, renderResponse);
56          } else {
57              include(editJSP, renderRequest, renderResponse);
58          }
59      }
60
61      public void doHelp(RenderRequest renderRequest,
62                         RenderResponse renderResponse)
63          throws IOException, PortletException {
64
65          include(helpJSP, renderRequest, renderResponse);
66      }
67
68      public void doView(RenderRequest renderRequest,
69                         RenderResponse renderResponse)
70          throws IOException, PortletException {
71
72          include(viewJSP, renderRequest, renderResponse);
73      }
74
75     protected void include(String path,
76                            RenderRequest renderRequest,
77                            RenderResponse renderResponse)
78          throws IOException, PortletException {
79
80          PortletRequestDispatcher portletRequestDispatcher =
81              getPortletContext().getRequestDispatcher(path);
82
83          if (portletRequestDispatcher == null) {
84              _log.error(path + " is not a valid include");
85          } else {
86              portletRequestDispatcher.include(renderRequest,
87                                               renderResponse);
88          }
89      }
```

I metodi `init`, `doDispatch`, `doEdit`, `doHelp`, `doView` e `include` sono utilizzati per gestire il ciclo di vita della portlet (figura 7). All'interno del metodo `init`, utilizzato per l'inizializzazione del-

la portlet, vengono caricati i parametri che servono per indicare all'applicazione quali file jsp sono impiegati per l'interfaccia grafica; sono presenti 3 assegnazioni, una per ciascuna modalità di operazione della portlet (2.3.2): i parametri sono prelevati dal file di configurazione PORTLET.XML (4.2.3) mostrato in precedenza.

```
90      public void processAction(ActionRequest actionRequest,
91                                ActionResponse actionResponse)
92          throws IOException, PortletException {
93
94          String formAction =
95              ParamUtil.getString(actionRequest,
96                                  ActionRequest.ACTION_NAME);
97
98          if (formAction.equalsIgnoreCase("SALVACATEGORIA")) {
99
100             PortletPreferences preferences = null;
101             try {
102                 preferences =
103                     PortletPreferencesFactoryUtil.getPortletSetup(
104                         actionRequest);
105             } catch (SystemException e) {
106                 e.printStackTrace();
107             }
108
109             String categoriaScelta =
110                 actionRequest.getParameter("categoria");
111             preferences.setValue("categoria", categoriaScelta);
112             preferences.store();
113
114         } else {
115             actionResponse.setRenderParameter
116                 ("marcaId", actionRequest.getParameter("marcaId"));
117         }
118     }
119 }
```

Gestione eventi

Il metodo `processAction` serve per controllare gli eventi generati. In questa applicazione l'utente può effettuare due tipi di interazione: il primo interessa la modifica dell'impostazione per la configurazione della portlet, mentre il secondo riguarda la richiesta di visualizzazione dei modelli di una particolare casa produttrice. É possibile effettuare la distinzione di queste due operazioni mediante il riconoscimento del nome dell'azione (riga 98), quest'ultimo impostato nelle pagine jsp.

La sezione Model, invece, include le classi utilizzate per la mappatura con le tabelle del database (create in precedenza e inserite all'interno della cartella MODEL) e altre classi Java sviluppate al fine di mettere a disposizione le interrogazioni effettuate

nel database per poter esportare le informazioni richieste.
I file creati sono: ModelUtil.java e UtilPortlet.java.

ModelUtil.java

```
package util;

import java.util.Iterator;
import java.util.List;

import model.Categoria;
import model.Marca;
import model.Modello;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;

public class ModelUtil {

   private static List<?> doQuery(String query) {
      Session session = UtilPortlet.beginSession();
      Transaction tx = null;

      List<?> list = null;

      try {
         tx = session.beginTransaction();
         list = session.createQuery(query).list();
         tx.commit();
      } catch (HibernateException he) {
         if (tx != null)
            tx.rollback();
         throw he;
      } finally {
         session.close();
      }
      Iterator<?> it = list.iterator();

      return list;
   }

   public static Iterator<Categoria> getCategoria() {
      String query =
          "select categoria from Categoria as categoria";

      return (Iterator<Categoria>) doQuery(query).iterator();
   }

    public static String getCategoriaName(String id) {
      String query = "select categoria.nome from Categoria as
             categoria where categoria.id="+id;
```

```
        return (String) doQuery(query).iterator().next();
    }

    public static Iterator<Marca> getMarca(String cat) {
        String query = "select marca from Categoria as categoria
               inner join categoria.marca as marca
               where categoria.id=" + cat;

        return (Iterator<Marca>) doQuery(query).iterator();
    }

    public static Iterator<Modello> getModello(String marca) {
        String query = "select modello from Marca as marca inner join
               marca.modello as modello where marca.id="+ marca;

        return (Iterator<Modello>) doQuery(query).iterator();
    }


}
```

La classe ModelUtil.java contiene i metodi utilizzati per eseguire le query sul database:

- `doQuery`: restituisce una lista contenente le informazioni raccolte dal database;
- `getCategoria`: riporta tutte le categoria presenti attualmente nella tabella Categoria;
- `getCategoriaName`: presenta il nome della categoria associata al corrispettivo id, quest'ultimo passato come parametro;
- `getMarca`: mostra tutte le marche delle case produttrici associate alla categoria scelta;
- `getModello`: visualizza la lista dei modelli associati alla marca selezionata.

All'interno degli ultimi quattro metodi sopra elencati, sono presenti le query scritte in linguaggio HQL utilizzate per importare i dati dal database.

UtilPortlet.java

```
1 package util;
2
3 import org.hibernate.HibernateException;
4 import org.hibernate.Session;
5 import org.hibernate.SessionFactory;
6 import org.hibernate.Transaction;
7 import org.hibernate.cfg.Configuration;
8
```

```
9  public class UtilPortlet {
10
11     public static Session beginSession() {
12        SessionFactory sessionFactory =
13            new Configuration().configure().buildSessionFactory();
14        Session session = sessionFactory.openSession();
15        return session;
16     }
17
18 }
```

La classe UtilPortlet.java è stata invece creata al fine di inserire tutti i metodi utili per il controllo dell'applicazione web: in questo frangente, l'unico metodo inserito è `beginSession()` che ha la funzione di creare una nuova sessione con il database per l'esecuzione delle interrogazioni sopra riportate.

La parte inerente la presentazione della portlet (VIEW) rappresenta l'interfaccia grafica tra l'utente e l'applicazione e mette a disposizione del primo la possibilità di effettuare la ricerca dei vari modelli.
Le pagine web create per realizzare l'interfaccia grafica sono le seguenti:

Interfaccia Grafica della Portlet

init.jsp

```
<%@ taglib uri="http://java.sun.com/portlet_2_0" prefix="portlet"%>

<%@ page import="util.ModelUtil"%>

<%@ page import="model.Categoria"%>
<%@ page import="model.Marca"%>
<%@ page import="model.Modello"%>

<%@ page import="java.util.Iterator"%>

<%@ page import="javax.portlet.PortletURL" %>
<%@ page import="javax.portlet.PortletSession" %>
<%@ page import="javax.portlet.PortletPreferences" %>

<%@ page import="com.liferay.portal.kernel.util.ParamUtil"%>
<%@ page import="com.liferay.portal.kernel.portlet.*" %>



<portlet:defineObjects />

<%
   String redirect = ParamUtil.getString(request, "redirect");
   String portletPath = request.getContextPath();
   PortletPreferences preferences = renderRequest.getPreferences();
```

```
%>

<style type="text/css">
   <%@ include file="/css/test.css" %>
</style>
```

La pagina INIT.JSP si occupa di caricare tutte le classi utilizzate nelle pagine jsp successivamente create; inoltre, vengono definite delle variabili comuni a tutte le pagine web. L'uso di questo file apporta il vantaggio di non dover riscrivere in ciascuna pagina jsp le classi da utilizzare evitando così la ripetizione di codice.
Viene istanziata anche la variabile `preferences`, impiegata per caricare le variabili di configurazione della portlet (configurata quando si è in modalità EDIT).
Di seguito si riporta il file VIEW.JSP, commentando ogni sua parte di codice:

view.jsp

```
1  <%@ include file="init.jsp"%>
2
3  <script type="text/javascript"
4          src="<%=portletPath %>/js/jquery.msCarousel-min.js">
5  </script>
6
7  <div id="<portlet:namespace/>">
8
9  <%
10    String categoria = preferences.getValue("categoria","2");
11    String pathLoghi = categoria.equals("1")?
12    "/img/car/loghi/":"/img/moto/loghi/";
13 %>
```

In questo passaggio vengono caricate le librerie includendo il file INIT.JSP ed un file javascript, utilizzato per creare un effetto grafico al fine di migliorare l'interfaccia con l'utente; viene inoltre letta la categoria scelta attraverso la variabile `preferences` precedentemente creata.

```
14 <div class="containerCarousel">
15
16    <div class="frecciaContainer" id="preBig">
17       <img src="<%= portletPath %>/img/util/frecciasx.png">
18    </div>
19
20    <div id="carouselBig" class="mscarousel">
21    <%
22       Iterator<Marca> im = ModelUtil.getMarca(categoria);
```

```
23
24          while (im.hasNext()) {
25             Marca current = im.next();
26       %>
27       <div class="box">
28          <span id="<%=current.getIDMAR() %>">
29             <img src=
30             '<%=portletPath + pathLoghi + current.getSIMBOLO() %>'>
31             <p class="title"><%=current.getNOME() %></p>
32             <p class="descr"><%=current.getDESCRIZIONE() %></p>
33          </span>
34       </div>
35       <%
36          }
37       %>
38       </div>
39       <div class="frecciaContainer" id="nextBig">
40          <img src="<%=portletPath %>/img/util/frecciadx.png">
41       </div>
42    </div>
43
44    <div class="containerCarousel">
45
46       <div class="frecciaContainer" id="preSmall">
47          <img src="<%=portletPath %>/img/util/frecciasx.png">
48       </div>
49
50       <div id="carouselSmall" class="mscarousel">
51
52       <%
53          im = ModelUtil.getMarca(categoria);
54          while (im.hasNext()) {
55             Marca current = im.next();
56       %>
57          <span>
58          <img src=
59                '<%=portletPath + pathLoghi + current.getSIMBOLO() %>'>
60          </span>
61       <%
62          }
63       %>
64
65       </div>
66
67       <div class="frecciaContainer" id="nextSmall">
68          <img src="<%=portletPath %>/img/util/frecciadx.png">
69       </div>
70    </div>
71
72    <div id="result">
73    </div>
```

Questa porzione di codice si occupa dell'intera visualizzazione della pagina prelevando i dati dal database attraverso le oppor-

tune classi java. Come si può osservare dal sorgente sopra riportato, grazie all'utilizzo del pattern MVC non è presente alcuna riga di codice relativa alla logica dell'applicazione; una volta ottenuta la lista delle marche associate a una determinata categoria (riga 22), è sufficiente iterarla e creare per ciascuna iterazione la sezione di codice html atta ad impostare la grafica per illustrare in modo adeguato i dati prelevati.

```
74 <script type="text/javascript">
75
76    jQuery(document).ready(function() {
77       jQuery("#carouselBig span").each(function() {
78          jQuery(this).click(function() {
79
80             // creo l'url
81             baseurl='<portlet:renderURL windowState=
82             "<%= LiferayWindowState.EXCLUSIVE.toString() %>">
83                <portlet:param name="jspPage"
84                         value="/viewElenco.jsp" />
85                </portlet:renderURL>'
86
87             // prelevo l'id della marca
88             var app = jQuery(this).attr("id");
89
90             // eseguo la richiesta ajax
91             jQuery.ajax({
92                async : true,
93                type : 'post',
94                url : baseurl,
95                data : ({ 'marcaId' : app }),
96                success: function(data) {
97                   jQuery("#result").html(data);
98                }
99             });
100         });
101      });
102   });
103
104   var BigController = jQuery("#carouselBig").msCarousel({
105      boxClass:'div',
106      width:860,
107      height:280,
108      callback:bigSlideControl,
109      showMessage:true,
110      messageOpacity:1
111   }).data("msCarousel");
112
113   var SmallController = jQuery("#carouselSmall").msCarousel({
114      boxClass:'span',
115      width:860,
116      height:120,
117      callback:thumbSlideControl,
```

```
118         scrollSpeed:500
119      }).data("msCarousel");
120
121      //add click event
122      jQuery("#nextBig").click(function() {
123         BigController.next();
124      });
125
126      jQuery("#preBig").click(function() {
127         BigController.previous();
128      });
129
130      jQuery("#nextSmall").click(function() {
131         SmallController.next();
132      });
133
134      jQuery("#preSmall").click(function() {
135         SmallController.previous();
136      });
137
138      jQuery("#carouselSmall span").click(function(arg) {
139         var target = this;
140         var counter = jQuery("#carouselSmall span").index(target);
141         BigController.goto(parseInt(counter));
142      });
143
144      function activateThumb(no) {
145         jQuery("#carouselSmall span").removeClass("active");
146         jQuery("#carouselSmall span:eq("+no+")").addClass("active");
147      }
148
149      function bigSlideControl(arg) {
150         var oBigController = arg;
151         var currentItem = oBigController.getCurrentID();
152         activateThumb(currentItem);
153         if(currentItem==0) {
154            jQuery("#preBig").css({opacity:0.4});
155         } else {
156            jQuery("#preBig").css({opacity:1});
157         }
158
159         if(SmallController!=undefined) {
160            SmallController.goto(parseInt(currentItem));
161         }
162      }
163
164      function thumbSlideControl(arg) {
165         var oController = arg;
166         var currentItem = oController.getCurrentID();
167         if(currentItem==0) {
168            jQuery("#preSmall").css({opacity:0.4});
169         } else {
170            jQuery("#preSmall").css({opacity:1})
171         }
```

```
172     }
173
174 </script>
175
176 </div>
```

Codice Javascript - jQuery

L'insieme di righe sopra riportato è attinente alle varie funzioni javascript digitate utilizzando la libreria jQuery. Oltre alle procedure utilizzate per la creazione dell'effetto lista scorrevole (carosello) (figura 25), è presente anche la funzione AJAX relativa al caricamento dei vari modelli (righe 76 - 102); per la richiesta asincrona, devono essere specificati i seguenti parametri:

- pagina da caricare, specificata attraverso la renderUrl;
- tipo di richiesta (post o get);
- parametri da passare alla pagina richiesta;
- punto della pagina corrente dove dovranno essere inseriti i dati una volta che la richiesta è completata.

Di seguito si inserisce il file jsp VIEWELENCO.JSP utilizzato per soddisfare la richiesta AJAX appena descritta:

viewElenco.jsp

```
1  <%@ include file="init.jsp"%>
2
3  <span class="title">MODELLI</span>
4
5  <%
6     String categoria = preferences.getValue("categoria","2");
7     String pathMod = categoria.equals("1")?
8            "/img/car/mod/":"/img/moto/mod/";
9  %>
10
11 <%
12    String marca = renderRequest.getParameter("marcaId");
13    Iterator<Modello> im = ModelUtil.getModello(marca);
14    int i=0;
15    while (im.hasNext()) {
16       i=(i+1)%2;
17       Modello current = im.next();
18 %>
19    <div class="mod color<%=i%>" id=<%=current.getIDMOD()%>>
20       <span class="img">
21          <img src='<%=portletPath + pathMod + current.getFOTO()%>'>
22       </span>
23       <span class="nome"><%=current.getNOME()%></span>
24       <span class="descr"><%=current.getDESCRIZIONE()%></span>
25    </div>
26 <%
27    }
28 %>
```

Una volta individuata la categoria prescelta (riga 6) e la marca selezionata (riga 12), è necessario caricare la lista dei modelli attraverso la classe `ModelUtil` ed in seguito iterarla, affinchè vengano visualizzati nella pagina.

Foglio di Stile della portlet

Per migliorare l'utilizzo del software da parte dell'utente, è opportuno associare alle pagine jsp un foglio di stile per poter migliorare l'aspetto grafico con il quale l'applicazione si presenta. Il foglio di stile è rappresentato dal file TEST.CSS:

test.css

```
.portlet-Laurea .mscarousel .child {
   position:relative;
}

.portlet-Laurea .mscarousel .child .set,
.portlet-Laurea .mscarousel .set {
   float:left;
   position:relative;
}
.portlet-Laurea .mscarousel {overflow:hidden}
.portlet-Laurea .mscarousel img{border:0}
.portlet-Laurea .mscarousel .child .clear{clear:both}
.portlet-Laurea .hand{cursor:pointer}


.portlet-Laurea .containerCarousel {
   background-color:#e5e5e5;
   border-bottom:2px solid;
   margin:5px 0;
   overflow:hidden;
   padding:5px 0;
   width:980px;
   -moz-border-radius:15px;
   -webkit-border-radius: 15px;
   border-radius: 15px;
}


.portlet-Laurea #carouselSmall,
.portlet-Laurea #carouselBig {
   margin:5px;
   padding:10px 5px;
}

.portlet-Laurea #carouselBig {
   float:left;
   position:relative;
   -moz-border-radius: 10px;
   -webkit-border-radius: 10px;
   border-radius: 10px;
```

```
}

.portlet-Laurea #carouselBig .box {
   width:865px;
}

.portlet-Laurea #carouselBig span img {
   float:left;
   margin:10px 20px;
   max-height: 240px;
   max-width: 300px;
}

.portlet-Laurea #carouselBig span {
   position:relative;
   float:left;
   width:860px;
   cursor:pointer;
   text-align: justify;
}

.portlet-Laurea #carouselBig span p {
   position: relative;
   float: left;
}

.portlet-Laurea .title {
   font-size:40px;
   text-decoration:underline;
   width: 520px;
   font-family:sans-serif;
}

.portlet-Laurea .descr {
   position: relative;
   float: left;
   width: 520px;
   font-family:sans-serif;
   font-size:13px;
   font-style:italic;
}

.portlet-Laurea .active {
   -moz-border-radius:5px 5px 5px 5px;
   border:3px solid #666666;
}

.portlet-Laurea #carouselSmall {
   float:left;
   position:relative;
}

.portlet-Laurea #carouselSmall {
   -moz-border-radius:15px;
```

```
   -webkit-border-radius: 15px;
   border-radius: 15px;
}

.portlet-Laurea #carouselSmall span {
   height:100px;
   width:100px;
   padding:5px;
   margin:0px 5px;
   float: left;
   cursor: pointer;
   text-align:center;
}

.portlet-Laurea #carouselSmall span img {
   max-height:100px;
   max-width:100px;
}

.portlet-Laurea #preBig img,
.portlet-Laurea #nextBig img {
   width:40px;
   margin:5px;
   float:left;
   position:relative;
   top:120px;
}

.portlet-Laurea #preSmall img,
.portlet-Laurea #nextSmall img {
   width:30px;
   margin:10px;
   float:left;
   position:relative;
   top:55px;
}

.portlet-Laurea #result {
   position:relative;
   top:12px;
   width:980px;
   overflow-x:hidden;
   overflow-y:auto;
}

.portlet-Laurea #result .title {
   float:left;
   font-size:40px;
   padding:7px 380px;
}

.portlet-Laurea #result .mod {
   margin:10px;
   overflow:auto;
```

```
   padding-bottom:10px;
   float: left;
}

.portlet-Laurea #result .color0 {
   background-color: WhiteSmoke;
   border:2px solid #004791;
}

.portlet-Laurea #result .color0 .nome {
   background-color:#004791;
   color:whitesmoke;
}

.portlet-Laurea #result .color1 {
   background-color:#004791;
   color: WhiteSmoke;
}

.portlet-Laurea #result .color1 .nome {
   color: #004791;
   border: none;
}

.portlet-Laurea #result .img,
.portlet-Laurea #result .nome,
.portlet-Laurea #result .descr {
   position:relative;
   float: left;
}

.portlet-Laurea #result .img {
   margin:0 40px 0 0;
   padding:10px;
   top:50%;
}

.portlet-Laurea #result img {
   max-height:200px;
   max-width:200px;
}

.portlet-Laurea #result .nome {
   -moz-border-radius:20px 0 20px 0;
   -webkit-border-radius:20px 0 20px 0;
   border-radius:20px 0 20px 0;
   background-color:WhiteSmoke;
   font-size:30px;
   margin:12px 0 14px 0;
   padding:10px 20px;
   text-transform:uppercase;
}

.portlet-Laurea #result .descr {
```

```
    font-family:sans-serif;
    text-align:justify;
    width:660px;
}
```

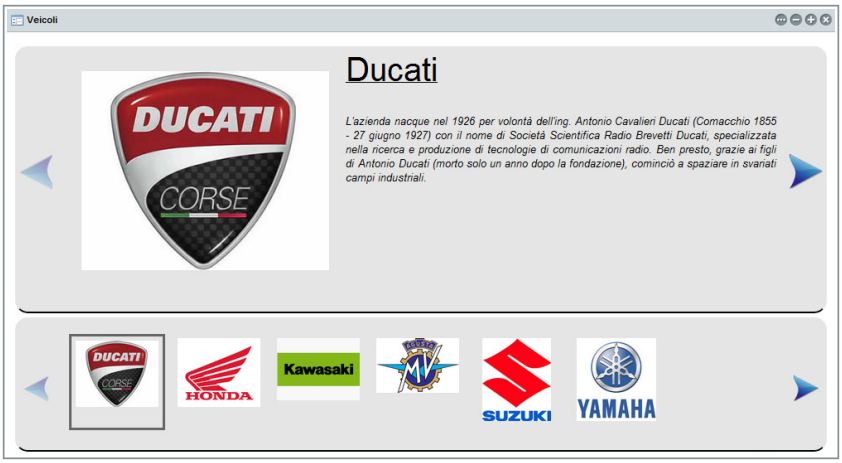

Figura 25: Portlet catalogo - Interfaccia utente

Come si può osservare dall'immagine 25, la finestra è suddivisa in due contenitori: nel primo è presente la descrizione dell'azienda selezionata, il nome ed il simbolo associato; nel secondo sono elencate tutte le aziende presenti nella base dati, ciascuna rappresentata dal proprio stemma.
Una volta selezionata la casa costruttrice, sotto l'elenco dei marchi compare una finestra nella quale vengono elencati i corrispondenti modelli; per ciascuno di questi viene illustrata la foto, il nome e fornita una breve descrizione (figura 26).

Al fine di semplificare l'utilizzo della portlet e di renderla graficamente più piacevole da vedere, per la selezione dei diversi marchi sono state create, attraverso l'utilizzo della libreria jQuery, due liste scorrevoli (caroselli), uno annesso all'altro in modo tale che la selezione di un marchio da una lista faccia automaticamente spostare l'evidenziazione della preferenza nell'altra. Per la visualizzazione dei vari modelli è stata effettuata la scelta di alternare i colori di bordo, sfondo e testo con lo scopo di distinguere più chiaramente un veicolo da un altro.

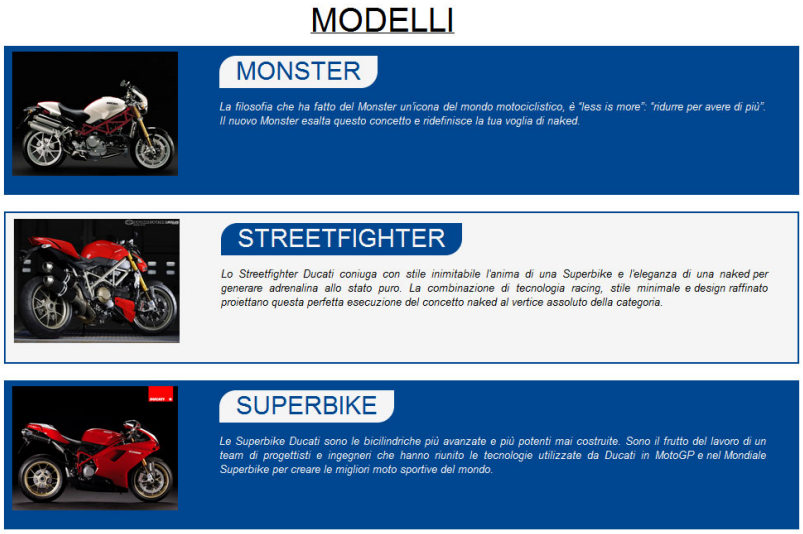

Figura 26: Portlet catalogo - Elenco Modelli

Portlet Modalità Edit

Per impostare la categoria, è necessario visualizzare la portlet in modalità EDIT. Gli eventi e i file di impostazione della portlet sono i medesimi utilizzati nella modalità VIEW, l'unica differenza riguarda l'interfaccia grafica tramite la quale è possibile effettuare la propria scelta, la quale è generata dal file EDIT.JSP.

edit.jsp

```
1  <%@ include file="init.jsp"%>
2
3
4  <div id="edit">
5
6  <p class="title">Selezionare la categoria</p>
7
8  <form action="<portlet:actionURL
9        name="salvaCategoria">
10       </portlet:actionURL>"
11 method="post">
12
13    <span class="nome">Nome Categoria</span>
14    <span>
15       <select class="categoria" name="categoria">
16       <%
17          Iterator<Categoria> itNome = ModelUtil.getCategoria();
18          while (itNome.hasNext()) {
19             Categoria app = itNome.next();
20       %>
21             <option value=<%=app.getIDCAT() %> >
22             <%=app.getNOME() %></option>
23       <%
```

```
24          }
25       %>
26       <%
27          String categoriaAttuale =
28                preferences.getValue("categoria","2");
29       %>
30       <p class="nome">Categoria Attuale =
31          <%=ModelUtil.getCategoriaName(categoriaAttuale) %>
32       </p>
33    </span>
34    <div class="button">
35       <input type="submit" value="Invia">
36    </div>
37
38 </form>
39
40 </div>
```

Il sorgente EDIT.JSP contiene principalmente un modulo html all'interno del quale vengono visualizzate le categorie disponibili e quella attualmente selezionata. L'elenco delle categorie viene fornito al momento del caricamento della pagina, iterando la lista restituita dal metodo `getCategoria()` descritto nella classe MODELUTIL.JAVA. Anche in questo caso, per migliorare l'aspetto grafico della pagina web, è stato utilizzato il foglio di stile TEST.CSS al quale sono state aggiunte le seguenti istruzioni:

test.css

```
.portlet-Laurea #edit {
   padding:20px;
   -moz-border-radius:15px;
   background-color: #004791;
   color: WhiteSmoke;
   overflow: auto;
}

.portlet-Laurea #edit .title {
   font-size:30px;
   text-transform:uppercase;
}

.portlet-Laurea #edit .nome {
   color:red;
   position:relative;
   padding:5px 30px;
   background-color:yellow;
   -moz-border-radius:10px;
   -webkit-border-radius:10px;
   border-radius:10px;
   font-size:18px;
```

```
   float: left;
   background-color: WhiteSmoke;
   color: #004791;
   margin:0px 0px 15px;
}

.portlet-Laurea #edit .categoria {
   background-color:whiteSmoke;
   float:left;
   font-size:16px;
   font-weight:bold;
   margin:0 24px;
   padding:3px 14px;
   text-transform:uppercase;
}

.portlet-Laurea #edit .button {
   float:left;
   font-size:15px;
   width:900px;
}

.portlet-Laurea #edit input[type="submit"] {
   -moz-border-radius:10px;
   -webkit-border-radius:10px;
   border-radius:10px;
   padding:3px 18px;
}
```

L'interfaccia grafica della portlet in modalità EDIT risulta essere la seguente:

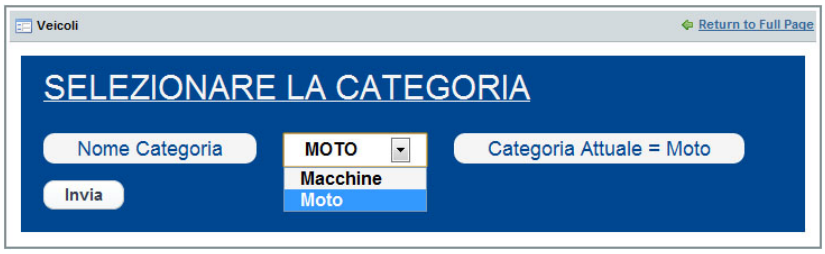

Figura 27: Portlet catalogo - Configurazione

Una volta selezionata una tra le categorie disponibili dal menù a discesa, attraverso il pulsante Invia viene trasmessa la scelta fatta alla portlet la quale la cattura attraverso il metodo `processAction` (4.2.3). Selezionando per esempio la categoria Macchine, l' applicazione web si presenta con la seguente schermata:

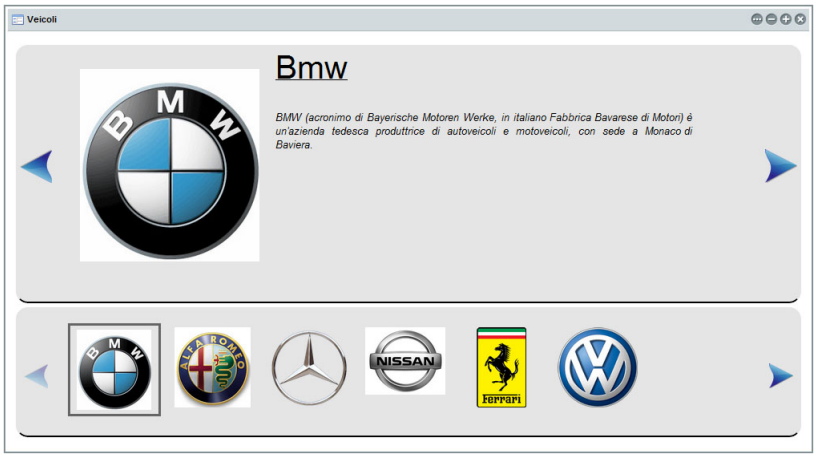

Figura 28: Portlet catalogo - Elenco Autoveicoli

5 CONCLUSIONI

In questa tesi sono stati descritti tutti i passaggi chiave utilizzati durante il periodo di tirocinio per la creazione di pagine web all'interno di un portale e, nel dettaglio, sono state analizzate le varie operazioni necessarie per la generazione delle nuove applicazione web da inserire successivamente all'interno di Liferay.
Per poter rendere il più comprensibile possibile il procedimento per la creazione di una applicazione web, è risultato opportuno inserire una parte teorica inerente al funzionamento di un portale e di una portlet partendo dalle varie definizioni delineando poi il ciclo di vita delle portlet e i collegamenti tra le stesse.
In seguito, è stato necessario chiarire la distinzione tra il concetto di programmazione lato client da quello lato server, un nodo assai importante per capire quali linguaggi di programmazione utilizzare per determinare le varie funzionalità che è possibile implementare nella propria applicazione web.
Successivamente è stata inserita una breve descrizione di Liferay e delle varie funzionalità, rilevanti per capire quali strumenti offre questo sofisticato portale e quali è invece necessario aggiungere per semplificare il più possibile la creazione delle portlet, come per esempio l'interfacciamento di una portlet con la propria base dati, utilizzata per il reperimento delle proprie informazioni.
Infine, è stato presentato un esempio semplice di creazione di una portlet allo scopo di rendere disponibile una guida chiara e allo stesso tempo esauriente per la generazione di una applicazione web.

Le istruzioni presenti riguardano quindi:

- installazione del software;
- creazione del database per immagazzinare i dati;
- impostazioni di configurazione per manipolare i dati presenti nel database;
- creazione della parte logica dell'applicazione;
- creazione dell'interfaccia dell'applicazione.

Tramite questa relazione si vuole perciò creare e fornire un valido manuale consultabile per comprendere argomenti particolari quali la portlet e il portale, cercando di superare le diverse difficoltà che si possono incontrare durante lo sviluppo delle pagine web annesse alla propria applicazione internet. All'interno di questa tesi sono state elencate e descritte tutte le azioni necessarie per garantire la corretta creazione della propria applicazione rendendo così disponibile, in un'unica guida, sia i concetti teorici che quelli pratici utilizzati durante il periodo di tirocinio.

BIBLIOGRAFIA

[1] Paolo Atzeri, Stefano Ceri, Stefano Paraboschi, Riccardo Torione, *Basi di dati - Modelli e linguaggi di interrogazione Terza Edizione*, McGraw-Hill.

[2] Cay Horstmann, *Concetti di informatica e fondamenti di Java*, Apogeo.

[3] Richard L. Sezov, Jr., *Portal Administrator's Guide*, Liferay, Inc. http://docs.liferay.com/portal/5.2/official/liferay-administration-guide.pdf

[4] Developing with Liferay, http://www.liferay.com/community/wiki/-/wiki/tag/development

[5] Gavin King, Christian Bauer, Max Rydahl Andersen, Emmanuel Bernard, Steve Ebersole, *Hibernate Reference Documentation*, http://docs.jboss.org/hibernate/stable/core/reference/en/pdf/hibernate_reference.pdf.

[6] OpenXava Documentation, http://openxava.wikispaces.com/index_en.

[7] jQuery Documentation, http://docs.jquery.com/Main_Page

[8] Html Reference, Javascript Reference, http://www.w3schools.com/

[9] MySQL 5.1 Reference Manual, http://dev.mysql.com/doc/refman/5.1/en/index.html