UNIVERSITÀ DEGLI STUDI DI PARMA

FACOLTÀ DI INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

Sviluppo di un servizio web con specifiche WSRP per l'accesso ai laboratori

Development of a web service with WSRP specifications for access to laboratories

Relatore

Chiar.mo Prof. Ing. A. POGGI

Tesi di Laurea di

CLAUDIO PITZALIS

Anno Accademico 2010/2011

Tesi di laurea dedicata a

Anna

Chiara

Francesca

Maria Teresa

e a mia madre

Ringraziamenti

È sentito, oltre che doveroso, da parte mia, ringraziare il prof. Agostino Poggi che ha accettato il non facile compito di fare da relatore a uno studente del corso con didattica a distanza, quindi fuori dal classico schema.

Ringrazio molto tutto lo staff del CEDI, col quale si è creato con gli anni un rapporto di amicizia che ha favorito un clima adeguato per il proseguimento degli studi: quindi il dott. Emilio Iori, Norberto Vignali, l'Ing. Simona Bertè, la sig.ra Anna Veronese, ora in pensione, e in particolare Cesare Marchesini che tanto si è prodigato per farmi avere tutti gli strumenti necessari e che mi hanno permesso di mandare avanti il mio progetto anche a distanza.

Vorrei ringraziare anche tutti quei professori che nel corso della mia carriera hanno dimostrato la loro stima nei miei confronti, incoraggiandomi in tal modo a proseguire nelle mie fatiche: i professori Marino Belloni, Massimo Bertozzi, Stefano Caselli, Gianni Conte. Quest'ultimo ha sempre avuto una particolare attenzione nei confronti degli studenti "teledidattici".

Una parola di ringraziamento anche per il compianto prof. Eduardo Calabrese che ci ha lasciato da poco.

Ringrazio infine la stessa Università di Parma che mi ha dato questa grande opportunità.

Indice

Indice delle Figure

Indice dei Listati

Prefazione

Questa tesi nasce dalla necessità di proseguire in quella che è la fase di aggiornamento di alcuni servizi adattandoli al nuovo portale d'Ateneo, messo in linea nell'agosto del 2010, con lo scopo di rinnovare la metodologia di informare via web, aprendo contestualmente la fase della produzione d'informazioni da rendere disponibili sia alla comunità esterna come a quella interna all'Ateneo.

In estrema sintesi, l'obiettivo della tesi è di illustrare il percorso che ha portato a rispondere alla richiesta del CEDI (**CE**ntro universitario di servizi per la **D**idattica di **I**ngegneria) di sviluppare un'applicazione per il refactoring del servizio di accesso ai laboratori nelle facoltà di Ingegneria e di Architettura, con l'implementazione di elementi aggiuntivi, messi a disposizione degli amministratori del servizio medesimo.
Si tratta di un servizio che è attualmente inserito in una interfaccia diversa da quella del portale: il risultato finale, attraverso software stabilito dall'Ateneo, dovrà invece essere quello di uniformare il suddetto servizio alle caratteristiche richieste dal portale stesso.

Il fulcro del progetto sta nell'utilizzo, da una parte del framework open source **jaMVC**, di cui si tratta ampiamente nel capitolo 2, e dall'altro dello standard **WSRP** (Web Services Remote Portlet), di cui si tratterà in maniera esclusivamente accessoria nel capitolo 3.

Il primo, ancora poco documentato data la mancanza, a oggi di una community, è una sorta di estensione al pattern MVC (Model-View-Control) e illustra abbastanza bene come un framework di sviluppo web dovrebbe essere implementato: in una modalità indipendente da qualsiasi linguaggio di

programmazione specifica, mantenendo una interfaccia fissa che consente agli sviluppatori di scegliere la lingua giusta per l'applicazione web che vogliono costruire. Si parla, infatti, di "linguaggio agnostico".

Le specifiche del protocollo WSRP, che definiscono l'uso dei web services nella distribuzione delle informazioni ai portali Internet, permettono invece l'integrazione di contenuti remoti e applicazioni logiche all'interno di presentazioni (pagine web) 'end-user', che generalmente implicano significativi sforzi di programmazione. La piattaforma Alpha, su cui poggia il portale di Ateneo, grazie alle funzionalità basate sul protocollo WSRP, può importare su di sé dei web services già esistenti e messi a disposizione dai cosiddetti "producer". In altre parole, l'obiettivo di questa specifica è di consentire a un progettista dell'applicazione, o all'amministratore della stessa, di scegliere tra una ricca varietà di contenuti remoti conformi e di integrarli senza alcuno sforzo di programmazione.

Il codice sviluppato per questa tesi, in questo caso creato con linguaggio PHP, è pronto, con qualche essenziale modifica, per essere integrato nel portale di Ateneo, basato quest'ultimo sul CMS Drupal. A tale scopo è stato utilizzato un apposito portlet di prova per testare l'applicativo (il framework) come servizio a sé stante e successivamente il servizio stesso è stato agganciato a un nodo di un sito Drupal per valutarne il comportamento e il corretto funzionamento del protocollo WSRP. Oltre a ciò è stato messo a disposizione l'accesso a un DBMS Oracle.

Gli strumenti utilizzati per lo sviluppo del progetto, sono stati l'editor per files Notepad++, un client per interagire con il DBMS Oracle e un browser Web, privilegiando Chrome e Mozilla.

Lontani dal Centro di Calcolo Elettronico e dai laboratori di facoltà, sono stati preziosi il VPN Client della Cisco System e un client WebDav (in questo caso BitKinex) che hanno permesso di lavorare perfettamente anche a distanza. Per ultimo, ma non meno importante, è stato usato il sistema Subversion, per il controllo versione e il deposito del software: è stato pertanto installato il relativo client TortoiseSVN e creato quindi un repository SVN.

Capitolo 1

Come nasce il progetto

Questo capitolo vuole essere una breve sintesi del perché, dei protagonisti e dei primi passi che hanno portato allo sviluppo del progetto.

1.1 Il servizio attuale

Come già delineato in prefazione, l'idea della tesi nasce dalla necessità della direzione del CEDI, il **CE**ntro universitario di servizi per la **D**idattica di **I**ngegneria, di riformulare in qualche modo alcuni servizi già presenti e in particolare il servizio dedicato all'accesso ai laboratori informatici da parte dello studente.

Attualmente il servizio è attivo in una pagina web specifica, presso un sottodominio del portale di Ateneo, all'indirizzo http://laboratori.cedi.unipr.it/ e consiste, dopo alcune informazioni e l'accettazione del regolamento, nell'invio, da una parte, all'utente che ne ha fatto richiesta, di una email con username e password per l'accesso ai laboratori e, dall'altra, della scrittura lato server di un file di log per uso amministrativo e un file utente che verrà recuperato ed elaborato dai laboratori stessi per essere inseriti in una banca dati. Questo servizio è indirizzato unicamente agli studenti di Ingegneria e Architettura di Parma, per

questo, al momento dell'accesso si verrà trasferiti nel CAS (Servizio di Autenticazione Centralizzata) che provvederà a prelevare i dati utente dalla banca dati di Ateneo e consentire o meno l'accesso al servizio.

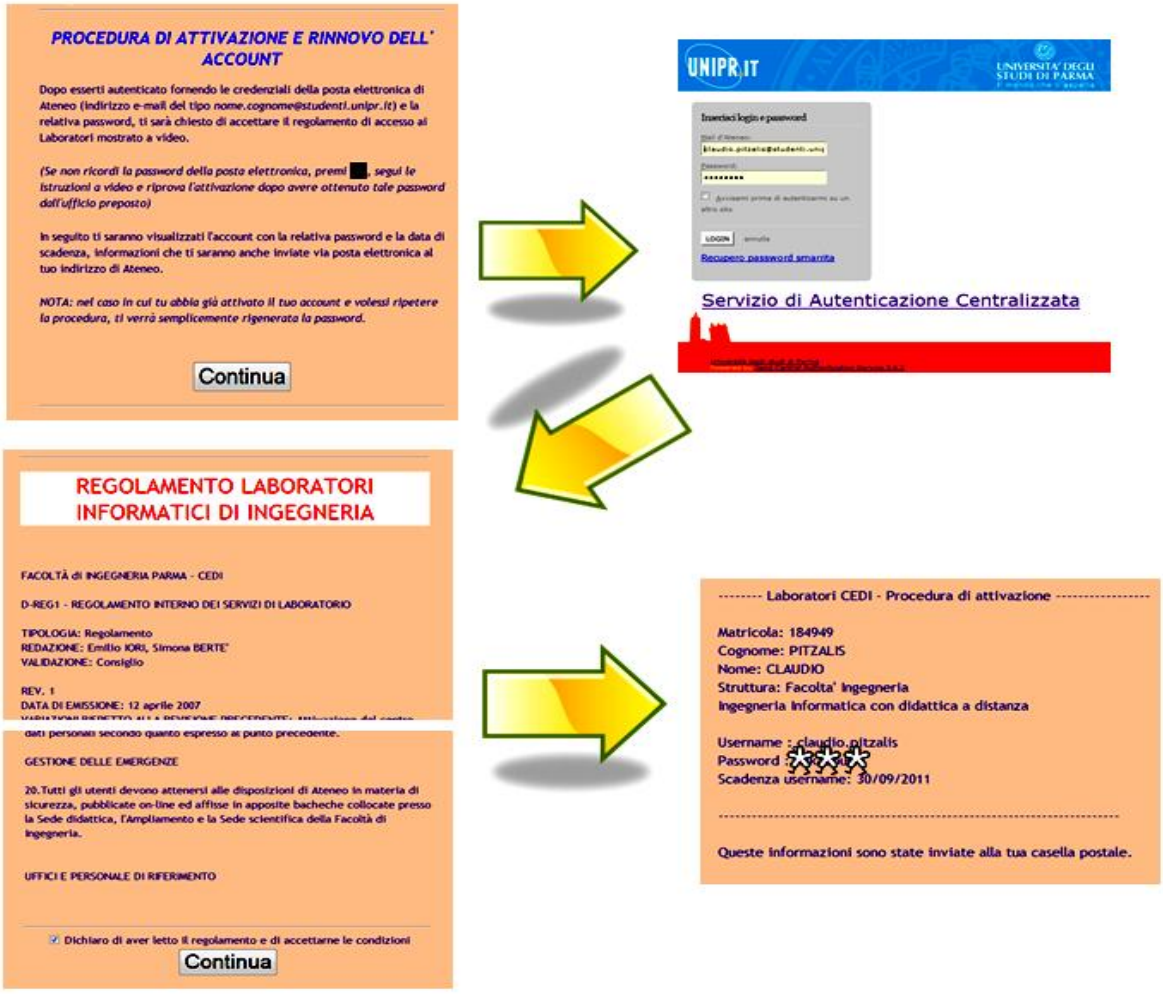

Figura 1.1. Pagine relative all'attuale servizio.

Come si può rilevare dalla figura 1.1, l'interfaccia utente è molto semplice, ma risponde in pieno alle necessità richieste dal servizio. Il codice invece, in parte html e in parte PHP, non è affatto banale, in quanto nella procedura di attivazione vengono implementate una serie di funzioni utili alla costruzione di password e username con tutti i relativi controlli, alla scrittura di più file su server, all'invio dei dati all'utente tramite sia pagina web immediata sia con una email di notifica e infine, come detto più sopra, al deposito di un file utente che verrà elaborato dai laboratori con tempi dipendenti dalla coda di richieste in sospeso.

1.2 La domanda del CEDI

Qualche mese fa, durante un'apposita riunione, sono state esposte dal direttore del CEDI, e dai suoi collaboratori e tecnici, le esigenze e le nuove modalità di accesso del servizio.

Il refactoring, come già detto, era da considerarsi la prima ed essenziale fase del lavoro: il servizio doveva essere conforme al nuovo portale di Ateneo e avrebbe dovuto quindi essere strutturato all'interno di un nodo Drupal, il CMS su cui è stato costruito il portale.

Successivamente si sarebbe dovuta valutare la possibilità di realizzare una modalità di accesso agli amministratori del sistema che permettesse loro di agire il più possibile dinamicamente su variabili e parametri di vario genere del servizio e consentisse una ricerca e una vista degli utenti registrati più agevole, altrimenti complessa nella versione attuale, dove per ogni registrazione viene inserita una riga in un file di testo.

Per ultimo è stato proposto di sottoporre un questionario di valutazione agli utenti secondo un algoritmo ancora da studiare: in pratica all'utente, dopo aver terminato il procedimento per l'accesso ai laboratori, dovrà essere somministrato un questionario, ma solo ad accessi successivi al primo e dopo un certo intervallo di tempo trascorso.

Vi erano alcuni presupposti, tra cui in particolare quello di elaborare un progetto secondo le caratteristiche della piattaforma Alpha adottata dall'ateneo. Per chiarire meglio il percorso intrapreso conseguentemente, è bene fare un accenno a questa piattaforma.

1.2.1 La piattaforma Alpha

Come accennato, l'Università di Parma ha sviluppato nel 2010 un portale web per il quale si è deciso di utilizzare Alpha, una piattaforma proposta dell'azienda Plurimedia (www.plurimedia.it) in collaborazione con CINECA (www.cineca.it).

In questa sezione si darà solo una breve descrizione dell'architettura della piattaforma: per un approfondimento si rimanda al sito di Plurimedia di cui viene esposto qui solo qualche estratto.

Al momento, la scelta prevalente della piattaforma Alpha ricade su elementi open source, che offrono servizi per sviluppare applicazioni web.

Si compone fondamentalmente dei seguenti applicativi:

- Apache HTTP server
- Jboss
- Drupal
- WSRP
- MVC
- DBMS Oracle Express

Nel seguito si farà riferimento solo alle componenti che sono strettamente legate al progetto.

Questa architettura relativa alla gestione servizi è destinata allo sviluppo e all'esecuzione di servizi web integrati con gli altri stack e con le altre componenti architetturali.

Si era già toccata la questione del "linguaggio agnostico" e infatti, una delle primarie peculiarità di questo stack, è proprio la completa astrazione dal linguaggio che viene usato per la realizzazione dei servizi web: con l'architettura utilizzata, qualsiasi linguaggio di sviluppo può essere utilizzato per la realizzazione di componenti applicative.

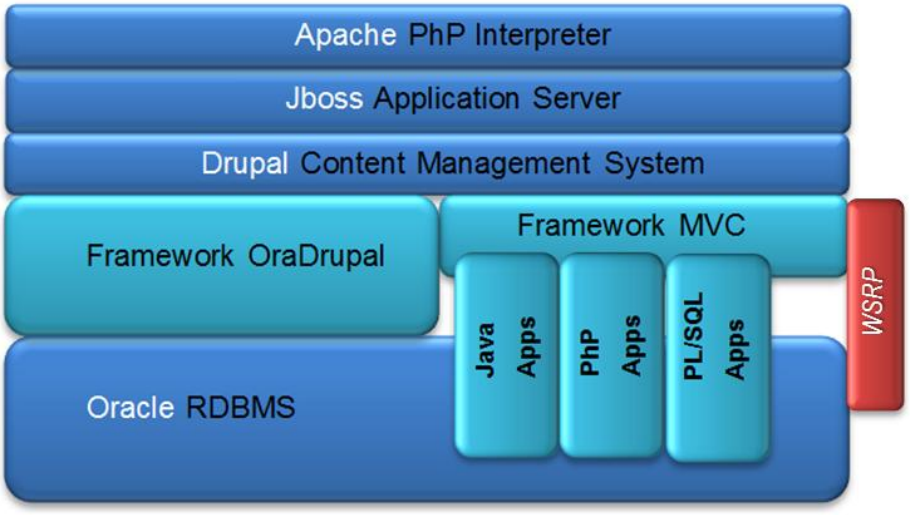

Figura 1.2 Lo stack delle applicazioni nella piattaforma Alpha[1]

[1] Documento di offerta tecnica. www.plurimedia.it

In figura sono rappresentati solo alcuni linguaggi utilizzabili dal framwork MVC: Java J2EE, PHP e PL/SQL. Questa integrazione di diverse tipologie di servizi realizzati con diversi linguaggi è possibile grazie alla completa compatibilità con lo standard WSRP V2 che consente di integrare servizi web non solo presenti all'interno dell'architettura, ma anche servizi web remoti cioè residenti su altri server in altri luoghi.

Per concludere, la piattaforma tecnologica adottata per il portale di Ateneo, oltre ad introdurre una connessione diretta fra dati dei sistemi informativi e portale, ha aperto prospettive interessanti in quanto rende disponibili i dati ufficiali a chiunque in Ateneo abbia necessità o intenzione di utilizzarli per la realizzazione di portali federati (Facoltà e Dipartimenti) o la produzione di servizi interattivi, relativamente semplificata dall'adozione del framewrok MVC, fornendo ai futuri sviluppatori di servizi WSRP le basi per incrementare nuovi servizi autonomamente. Oltre a questo, considerando la compatibilità con il protocollo WSRP v.1 e v.2, è stata fornita con queste basi la possibilità di uno sviluppo congiunto, con altri atenei, di servizi per gli utenti e facilitare il riuso di codice già sviluppato da altri.

Proprio su quest'ultimo punto si poggia la realizzazione del progetto di tesi: la creazione di un servizio e il suo possibile riuso.

1.2.2 Drupal e il framwork MVC

Un altro dei presupposti del CEDI era l'utilizzo del framework jaMVC, ma prima di entrare nel merito di questo framework in particolare, è giusto porsi alcune domande: perché utilizzare un framework quando Drupal permette con estrema facilità la creazione di pagine o di blocchi anche complessi all'interno del portale? Inoltre, con una buona dimestichezza di questo CMS, è anche possibile realizzare moduli appositi più o meno complessi che consentono di caratterizzare e personalizzare il servizio voluto, e soddisfare correttamente tutte le esigenze del servizio stesso. Ed effettivamente questa è stata la primissima idea conseguente alla richiesta del CEDI.

Brevemente, Drupal è un Content Management System (CMS - Sistema di Gestione dei Contenuti), vale a dire un gestore di contenuti e di siti web dinamici, ed è realizzato in PHP, che è un linguaggio di scripting lato server. Tutti i CMS web sono concettualmente abbastanza simili: costituiti, cioè, da una serie di script lato server che, interagendo con un database, realizzano così, in corso di esecuzione (a run time), le pagine web che mostrano infine i contenuti. E questi contenuti non sono semplicemente memorizzati in cartelle lato server del proprio hosting, come avviene normalmente, ad esempio, con le pagine html, ma sono conservati nel database, e una eventuale loro modifica, rimozione o aggiunta sarà effettuata sempre attraverso interfacce web.
Drupal è open-source software, distribuito sotto licenza GPL ("GNU General Public License", quindi può essere liberamente scaricato, distribuito e installato) ed è mantenuto e sviluppato da una comunità di migliaia di utenti e sviluppatori. Questo è sicuramente uno dei grandi punti di forza dei più conosciuti CMS.

È interamente sviluppato in PHP, uno dei linguaggi più usati nel web e utilizza come base di dati MySQL o PostgreSQL in modo nativo, ma è utilizzabile con ogni tipo di database server. Una peculiarità interessante dei sistemi CMS è di facilitare ampiamente l'organizzazione e la cooperazione nella creazione di documenti e contenuti di vario genere, il cui utilizzo non è necessariamente limitato alla gestione di siti web, anche se attualmente proprio questo ne è l'uso più frequente.[2]

Premesso questo, è intuitivo arrivare alla conclusione che un qualsiasi tipo di servizio sarebbe facilmente implementabile all'interno del portale di Ateneo, poiché questi utilizza proprio il CMS Drupal.
Vi sono però da considerare alcuni aspetti: la scelta del CMS integrata nell'architettura Alpha, non a caso, è ricaduta su uno dei più potenti e innovativi sistemi CMS Open Source presenti oggi sul mercato, ma la community di Drupal ha realizzato un processo di certificazione dei moduli opzionali: se un team di professionisti realizza un modulo di Drupal ed intende pubblicarlo nell'ambito della community, tale modulo viene sottoposto a verifiche di qualità rigorose

[2] *Valutazione pacchetto software Drupal* - Relazione per Ingegneria del Software A e Reti di Calcolatori A Docente Prof. Agostino Poggi, Studente Claudio Pitzalis – A.A. 2009/2010 Progetto-Drupal-2010_184949.pdf

prima di essere accettato e pubblicato dalla community stessa. Tra l'altro per filosofia sono accettati unicamente moduli sotto licenze GPL e gratuiti.
Oltre a ciò, va aggiunto che, essendo Drupal in continua e rapida evoluzione (vengono rilasciate nuove versioni nel giro di brevi archi temporali), creando appropriati moduli, pagine o blocchi, relativi a un servizio, se ne possono perdere in parte o per intero le funzionalità al successivo rilascio e aggiornamento di una nuova versione, costringendo lo sviluppatore a intervenire nuovamente sul codice per riadattare il servizio al portale.
Altra ed evidente motivazione che porta a ragionare su un modello di servizio slegato dall'interfaccia Drupal, può essere la decisione di dismettere l'uso di questo CMS e passare ad altro tipo di gestore di siti web: in questo caso il servizio diventerebbe inattivo e bisognerebbe riformularne completamente il codice in base alla nuova struttura del portale.

Ecco pertanto la motivazione che ha portato alla scelta di un framework MVC: il servizio sarà nelle condizioni di funzionare in maniera autonoma e indipendente dal portale di destinazione ma, grazie anche alle specifiche WSRP, potrà essere inserito nel portale stesso di Ateneo garantendone il funzionamento anche nei casi sopra menzionati. Il framework jaMVC risponde sia a questa caratteristica come alla citata filosofia della piattaforma Alpha, di utilizzare cioè unicamente software open source.

Nei capitoli 2 e 3 verranno sviluppati più ampiamente gli argomenti legati al MVC, in particolare jaMVC, e al protocollo WSRP.

1.3 Una prima risposta

Prima ancora di addentrarsi nei codici e nei software proposti, è stato presto necessario costruire un modello di diagramma di navigazione dell'attuale servizio, per meglio ricostruirne successivamente le varie fasi di accesso, le implementazioni e la struttura da presentare all'utente. In questo caso si tratta di un refactoring: si ricostruisce il servizio così come si presenta attualmente, ma con software differente.

Contemporaneamente, non bisognava trascurare il modello che poteva rendere evidente le migliorie richieste del servizio in produzione: questo sarebbe servito, accompagnato da uno pseudo-codice, a interpretare al meglio le nuove funzioni da svolgere.

1.3.1 Diagramma di navigazione per refactoring

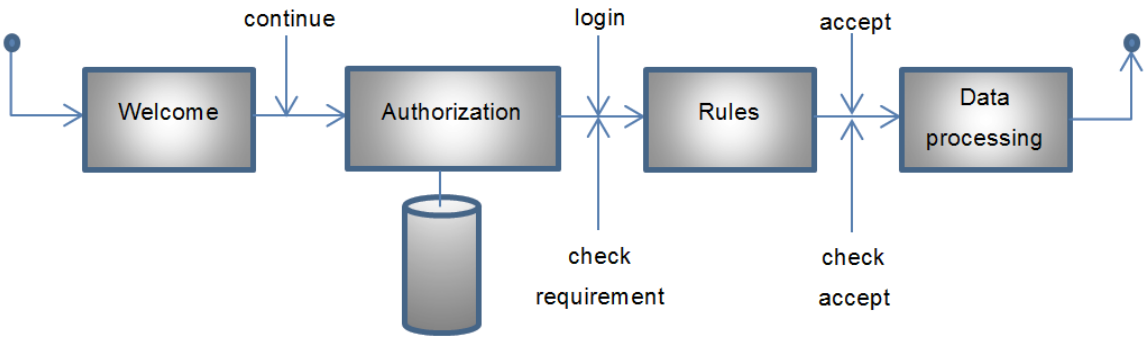

Figura 1.3 Diagramma di navigazione del servizio vigente

Dalla figura 1.3 si evince immediatamente la corrispondenza con la figura 1.1 di pag. 2: dopo la pagina iniziale di informazioni e di benvenuto, si passa, dopo la pressione di un box, alla pagina di autorizzazione e di login, dove viene fatto il check dei requisiti richiesti: essere studente dell'ateneo di Parma e far parte delle facoltà di architettura o di ingegneria. Per questo, il servizio è affiancato da un sistema di login gestito tramite l'applicativo Apache DS (anche questo Open Source):

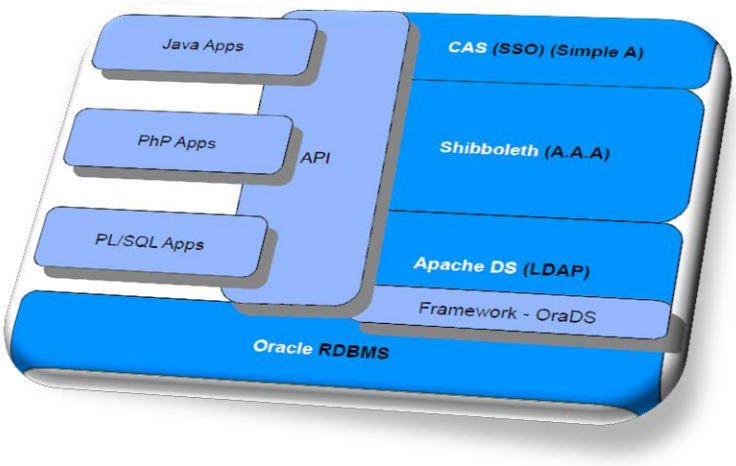

Figura 1.4 Stack gestione identità della piattaforma Alpha

Come si può vedere in figura, questo stack di Alpha indirizza le tematiche legate alla gestione utenti. E' previsto, infatti, un LDAP per la gestione utenti tramite l'applicativo Apache DS.

Molto brevemente, e solo per ciò che può concernere questo progetto, è da segnalare come, grazie al framework realizzato da Plurimedia denominato OraDS, l'applicativo ApacheDS scarica i dati all'interno del DB Oracle presente in architettura e non su file system come avviene normalmente. Lo stack prevede anche altre componenti finalizzate a gestire la tematica dell'Identity Management nella sua completezza: una di esse si chiama CAS ed è un sistema open source per la gestione del Single Sign On cioè per la gestione delle credenziali di accesso.[3]

Infine, presente in questo stack di Alpha, vi è Shibboleth che rappresenta ormai uno standard per la gestione delle autorizzazioni e dei gruppi di utenza.
Detto in parole povere e semplificando all'eccesso, il sistema preleva i dati dell'utente eventualmente inserito nella banca dati (verifica delle credenziali d'accesso tramite CAS) e in caso di risposta positiva vengono controllati i dati necessari per le successive autorizzazioni (Shibboleth).
Sia pure non approfondite in questa sede, la conoscenza di queste componenti è fondamentale per sviluppare i migliori servizi a favore soprattutto del lato amministrativo del portale.

Tornando al diagramma di navigazione, una volta che l'utente è autorizzato ad accedere al servizio, si scatenano una serie di eventi di cui la pagina messa a video è solo la punta di un iceberg: quello che succede sotto, e di cui è già stata data una anticipazione, è fondamentale per la costruzione del nuovo codice e che si valuterà più avanti; codice che, specialmente per quanto riguarda l'implementazione di alcune funzioni scritte in php, sarà in parte simile.

1.3.2 Diagramma per l'evoluzione del servizio

Dalla figura 1.5 nella pagina successiva, si evince immediatamente che intanto abbiamo almeno due *use case*: uno per semplice utente del servizio (generalmente studente) e uno per l'amministratore del servizio. In seguito, quest'ultimo case

[3] *L'architettura Alpha* – Settembre 2010 - Plurimedia s.r.l.
http://www.plurimedia.it/sites/www.plurimedia.it/files/Alpha-v1.pdf

sarà ulteriormente suddiviso tra amministratore completo e amministratore "in sola lettura dati".

Inoltre si può notare l'utilizzo di tabelle database per inserire e prelevare dati, in sostituzione di file log e utente scritti sul lato server e poco duttili per ricerche o modifiche dei parametri inseriti.

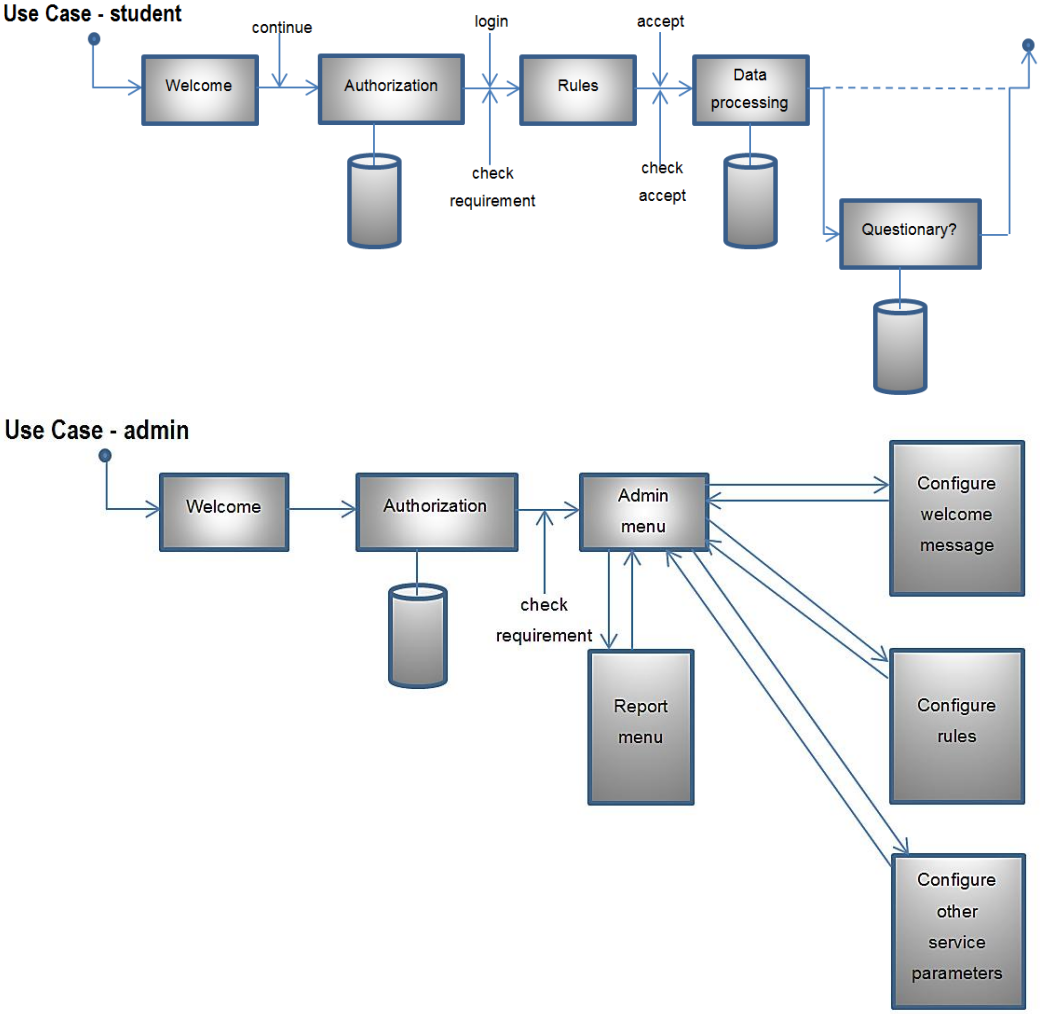

Figura 1.5 Bozza diagramma di navigazione del servizio evoluto

Per l'utente finale quindi non vi sarà, in apparenza, alcuna modifica del servizio: verrà mantenuta approssimativamente la stessa impostazione e rimarrà invariato l'algoritmo di creazione password e username. Inoltre l'invio della email di notifica avrà la stessa struttura. In base però a un criterio di valutazione dell'intervallo di tempo trascorso tra due accessi, potrà essere proposto un questionario di soddisfazione (che esula da questa tesi).

Nella sostanza, invece, sarà creata una tabella nel database Oracle, dove saranno inseriti di volta in volta i dati degli utenti, e per questo anche Shibboleth avrà un suo ruolo: dopo l'autenticazione al primo accesso al servizio, nella tabella saranno inseriti innanzitutto i dati già esistenti prelevati dalla banca dati centrale di ateneo tramite appunto Shibboleth, come nome, cognome, matricola (eventuale), codice facoltà, struttura e tipo di utenza (studente, tecnico, operatore o amministratore del sistema) e in seconda istanza i dati creati dal servizio stesso, come data primo accesso, data ultimo accesso (in questo caso coincidenti), username, password e un codice di controllo utile per stabilire se l'utente debba o meno accedere al questionario. Per accessi successivi al primo vi sarà invece solo un aggiornamento di questi dati.

Per l'amministratore invece, una volta verificate le autorizzazioni necessarie, si dovranno presentare varie opportunità: la vista completa, attraverso un report, degli utenti con i relativi dati, la ricerca di uno o più utenti, tramite query da inviare al database, la modifica del messaggio di accesso al servizio, del regolamento, dell'intestazione e del piè pagina. In più l'amministratore dovrà aver accesso diretto ad alcuni parametri per il controllo e la modifica. Insomma l'amministratore dovrà avere tutti gli strumenti per rendere il più possibile 'customizzabile' il servizio.

Nel seguito saranno esposte le modalità di sviluppo e le soluzioni trovate per rispondere alla domanda, grazie soprattutto al framework jaMVC.

Capitolo 2

Il framework jaMVC

Dopo una dovuta introduzione a quella che è la "filosofia" del pattern MVC, in questo capitolo si entrerà nel merito del framework jaMVC, con alcuni dettagli fondamentali, e si potranno vedere alcuni frammenti di codice con questo sviluppato.

2.1 Il pattern MVC: i concetti fondamentali

Per molte tipologie di software, la progettazione e la gestione delle applicazioni Web è resa più agevole seguendo dei paradigmi predefiniti. Il più diffuso da qualche anno è il pattern MVC (Model View Controller) che divide l'intero progetto in tre livelli. È quindi un modello di sviluppo di applicazioni che mira a separare gli elementi di logica (Model), presentazione (View) e controllo (Control).

1. **Model**: ciò che si vuole rappresentare e che può essere modificato dall'utente. È la rappresentazione logica specifica dei dati su cui opera un'applicazione: gestisce, dando loro in qualche modo significato, i dati e fornisce i metodi per accedervi in modo astratto e consistente.

Normalmente la progettazione del modello è guidata sostanzialmente dalla struttura del database e quindi dalle modalità di accesso ad esso.

2. **View**: modalità in cui si desidera rappresentare il modello, o parte di esso. Il report che mostra i dati utente può essere rappresentato tramite una form, ma si potrebbe anche essere interessati a rappresentare solo l'andamento degli accessi al servizio tramite un grafico. Il view dunque presenta il Model in una forma adatta all'interazione. Tipicamente è una *user interface*, come ad esempio la pagina html che mostra la lista degli utenti.
3. **Control**: risponde agli eventi, tipicamente generati dagli utenti. Invoca cambiamenti di stato del Model e indirizza la View. In altre parole presenta le varie possibilità d'interazione. Se l'utente, ad esempio, sceglie di visualizzare gli utenti che hanno un determinato nome, allora ci sarà la logica che andrà a selezionare dalla tabella degli utenti quelli scelti e li passerà alle Viste. Altrimenti, potremmo voler modificare i dati completi di un utente cliccando sul suo nome su una lista.

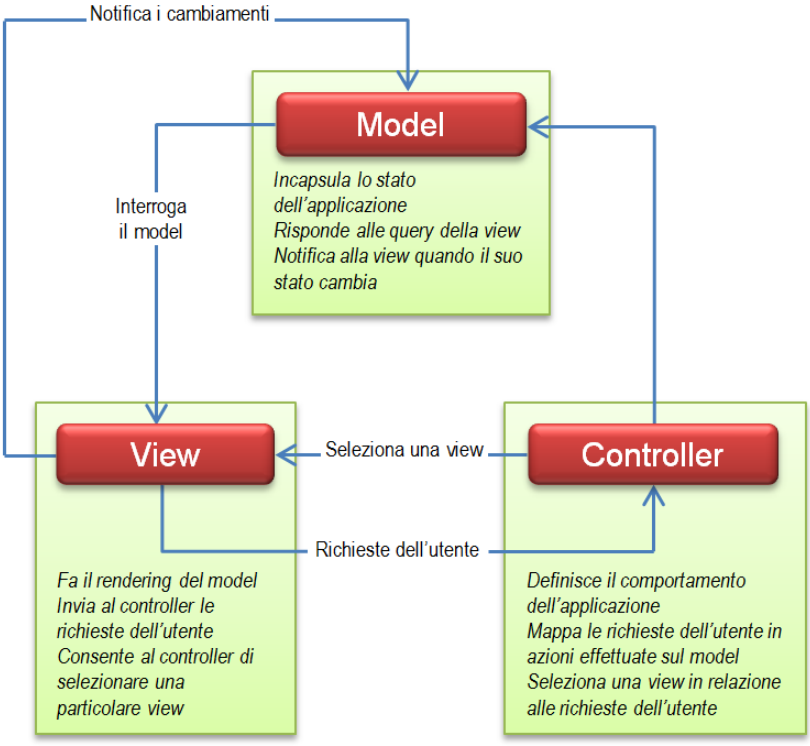

Figura 2.1 Schema del pattern MVC[4]

[4] http://www.fis.unipr.it/dokuwiki/doku.php?id=alessio.cavalieri:php-mvc - Alessio Cavalieri.

È chiaro come sia quest'ultimo componente quello più significativo nello sviluppo di una Web application: il **Controller** è il motore dell'applicazione e averlo separato dalla visualizzazione dei dati e dal modello ha reso la sua progettazione più semplice, permettendo di concentrare gli sforzi sulla logica del funzionamento.

2.1.1 Come opera MVC

- L'utente interagisce con l'interfaccia (es. clicca un link) generando quindi una richiesta http.
- Il Controller riceve la richiesta, inizializza il sistema, filtra i dati, e individua un'azione.
- Il Controller chiama il Model e in base alla specifica richiesta ne altera lo stato.
- In base alla logica dell'applicazione viene scelta una View.
- L'interfaccia utente aspetta successive richieste dall'utente e inizia un nuovo ciclo.

Nella figura 2.1 è schematizzato il funzionamento di una richiesta eseguita utilizzando il paradigma MVC.

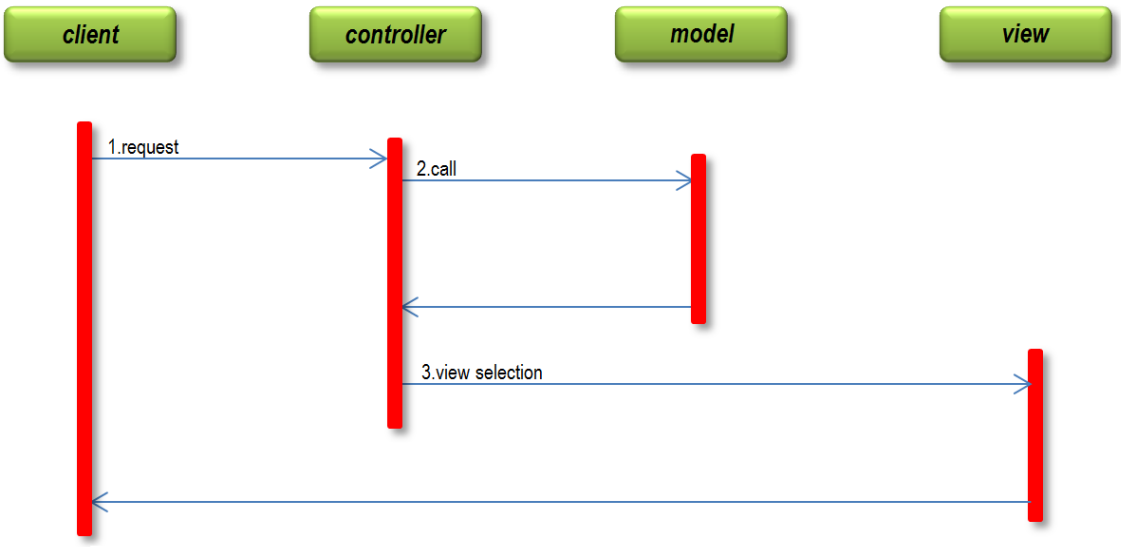

Figura 2.2 Sequenza di una richiesta in MVC

2.1.2 Vantaggi e svantaggi del pattern MVC

Questo approccio ha numerosi vantaggi:

- Disaccoppiamento delle componenti.
- La separazione in livelli di codice permette di:
  - . Organizzare la struttura del codice
  - . Lavorare in team con competenze diverse
  - . Supportare diversi livelli temporali di cambiamento
  - . view multiple per uno stesso modello
  - . Aggiungere o cambiare view con finalità diverse
- Il controller permette di avere un punto unico di accesso
- L'architettura MVC è quasi uno standard; anche cross-platform
- Riusabilità

Esistono anche alcuni svantaggi che sono tuttavia superabili:

- Adatto soprattutto a progetti medio/grandi.
- Architettura sostanzialmente complessa.
- Flessibilità dipendente dal framework utilizzato.[5]

2.2 Il jaMVC

Dopo questa breve introduzione al paradigma MVC, si può finalmente passare alla descrizione del framework jaMVC che verrà utilizzato per la scrittura del codice.

Essendo di recente sviluppo, questo framework è praticamente privo di una documentazione adeguata, specialmente perché, come anticipato, manca, per il momento, una community che può allargare le conoscenze di questo pacchetto. La poca documentazione presente è data dalla pagina wiki nel repository del sorgente e da qualche appunto preso in alcuni corsi relativi, interni all'Ateneo, e gentilmente offerto d'ausilio a questo progetto.

[5] *Design Patterns: Element of Reusable Object-Oriented Software* – Gamma, Helm, Johnson, e Vlissides (GoF) Addison-Wesley, 1995.

Per l'infrastruttura di un'applicazione con jaMVC sono richiesti fondamentalmente due server.

Nel primo server devono essere inclusi:

- Server web http con mod_php (modulo di Apache per eseguire script php)
- Php con versione 5.2.12 (interprete PHP) o superiore
- Moduli:
  - ✓ Dom (XML)
  - ✓ Xsl (trasformazioni XSLT)
  - ✓ Pdo (PHP data object, servono per accedere al DB, una parte di questi dipendono dal DB)

XSL, acronimo di eXtensible Stylesheet Language, è il linguaggio di descrizione dei fogli di stile per i documenti in formato XML. Lo standard XML prevede che i contenuti di un documento siano separati dalla formattazione della pagina in cui verranno pubblicati. L'XSL permette di visualizzare lo stesso file XML in formati diversi: come pagina web, come pagina stampabile oppure come traccia per un'esposizione orale.

L'XSL incorpora tre linguaggi:

- XSL Transformations (XSLT): il linguaggio di trasformazione dell'XML;
- XSL Formatting Objects (XSL-FO): usato per l'applicazione degli stili e del modo di apparizione a un documento XML.
- XML Path (XPath): è usato nei fogli di stile XSLT per descrivere come accedere alle parti di un documento XML.[6]

In parole povere, l'XLS converte i tag XML in HTML, oppure gli XML in altri formati.

Dopo questo software di base, ci vuole finalmente il framework jaMVC, sviluppato da Andrea Gariboldi, http://www.plurimedia.it/persone/andreagariboldi e disponibile presso il repository dedicato: è sufficiente fare un checkout dell'ultima revisione presso questo repository con il comando svn:

```
svn checkout http://jamvc.googlecode.com/svn/trunk/php/core
svn checkout http://jamvc.googlecode.com/svn/trunk/php/srv_test
svn checkout http://jamvc.googlecode.com/svn/trunk/php/ui
```

[6] *WIKIPEDIA*

```
svn checkout http://jamvc.googlecode.com/svn/trunk/php/wsrp
svn checkout http://jamvc.googlecode.com/svn/trunk/xsl
svn checkout http://jamvc.googlecode.com/svn/trunk/resources
```

Infine bisognerà installare le cartelle sulla piattaforma di sviluppo precedentemente preparata.

Il pacchetto comprende anche altri linguaggi diversi dal PHP, ma in questa sede non se ne terrà conto.

Il secondo server ospiterà invece il database.

Per sviluppare un servizio si deve creare, sotto il framework, una cartellina che avrà il nome del servizio. Questa cartellina avrà il seguente contenuto di files e cartelle:

Figura 2.3 Contenuto cartellina di un servizio jaMVC

Per comodità, e per un utile riferimento all'operatore, nel repository è stato inserito un servizio di esempio: srv_test, che permette di dare una prima valutazione sul codice e sull'interfaccia usata di default. Il progetto stesso di tesi si è sviluppato riscrivendo e ampliando il codice già scritto, e vista la mancanza di una documentazione adeguata, la presenza di un servizio, sia pure minimale, ha consentito di semplificare la lettura del codice stesso, altrimenti piuttosto ostica.

Questa è un'applicazione web con un unico punto di accesso, index.php: questa modalità si definisce applicazione web ***model 2***. Le ***model 1*** invece hanno molti punti di accesso.

Avere un unico punto di accesso può essere utile, ad esempio, per contare gli accessi.

A questo punto, per la definizione di un codice adeguato, si pensi a una banalissima applicazione che ha un pulsante 'start': a livello model si dice che se si preme il button si scatena l'evento 'start'. A livello control si dice invece che quando si scatena l'evento 'start' si deve caricare una nuova pagina, quella, per esempio, dove è possibile inserire alcuni dati.

Il Controller in concreto ha il compito di capire quali eventi si sono verificati e che cosa fare quando accadono gli eventi:

- Ascolta gli eventi
- Decide le pagine da mostrare (view)
- Invoca la logica di business (model)

In questo modo le applicazioni sono separate in blocchi, dove la manutenibilità risulta in questo modo più semplice. Inoltre questo approccio è molto adatto allo sviluppo in gruppo perché alcune persone si possono occupare del model, altre del view e altre del control.

Una delle prescrizioni suggerite sul Controller è che, essendo già dentro al framework, non sarebbe necessario svilupparlo, almeno a livello teorico . In realtà, con la scrittura di servizi un po' più 'corposi' e con una conoscenza più approfondita del software, si vedrà più avanti come può essere invece utile intervenire anche su di esso.

Comunque, in linea di massima, si può considerare, almeno per i non addetti ai lavori, la possibilità di occuparsi unicamente del model e del view.

Ritornando e semplificando ulteriormente quello detto più sopra, un'applicazione MVC, e nel particolare un'applicazione con jaMVC, si costruisce stendendo prima l'elenco dei requisiti e i relativi use case. Occorre perciò individuare gli attori, vale a dire i profili utente, e tutte le azioni che gli utenti devono potere eseguire con l'applicazione. E questo, in maniera schematica, è già stato fatto ed esposto in precedenza.

In questa fase è utile costruire il diagramma di navigazione, il flusso cioè dei passaggi tra le varie pagine e degli eventi che scatenano i passaggi. Ogni pagina può essere rappresentata da un'icona e sulle frecce c'è l'evento che scatena il passaggio da una pagina all'altra. Il diagramma non è detto che sia fatto da uno sviluppatore, ma può esserne incaricato anche un esperto di presentazione di dati. Le pagine presenti nel diagramma corrispondono una ad una alle pagine web contenute nella View. Ogni pagina a livello view avrà dietro la sua corrispondente pagina model. Il model potrebbe essere inutile specialmente se la pagina svolge semplicissimi compiti, ad esempio di sola presentazione testuale.

Per piccole applicazioni, il Model lavora direttamente sui dati nel database, per applicazioni enterprise, invece oltre al diagramma di navigazione, serve l'object model e poi il diagramma entity relationship, che non vengono trattati in questa sede.

2.3 Il setting.php

Per quanto detto finora sul diagramma di navigazione è opportuno introdurre subito il file *setting.php*, che merita una sezione a parte, poiché rende subito evidenti le caratteristiche di base del software.

Contrariamente a quanto suggerito nei corsi (dedotto dagli appunti acquisiti), dove questa fase viene inserita al termine della progettazione, qui invece si può definire la radice del progetto dove sono indicate le regole di navigazione.

2.3.1 Le regole di navigazione

Siamo nella cartella principale dell'applicazione e nel file *settings.php*.

In questo file intanto si indica il nome del servizio.

```
$service = 'cedi2'; //nome del servizio
```

che, come si vedrà in seguito, sarà utilizzato dal protocollo WSRP.

Poi si definirà la prima pagina da caricare all'avvio dell'applicazione.

```
$home = '#check_user'; //a seconda dell'utente la home page sarà differenziata
```

In realtà la home non è sempre predefinita: è possibile caricare pagine diverse a seconda dell'utente, come avviene in questo caso. Si sarebbe potuto indicare direttamente, come home, una pagina in particolare, ad esempio

```
$home = 'admin_page';
```

che avrebbe portato direttamente alla pagina, in questo caso, di amministrazione.
Più in generale, la sintassi è:

```
$home = 'idpagina';
```

ed è dunque la pagina alla quale si vuole essere reindirizzati come home page del servizio. **index.php**, che è la prima pagina effettivamente eseguita quando viene fatta una richiesta di utilizzo del servizio, reindirizza, tramite il metodo '*show()*' che si analizzerà più avanti, l'utente alla pagina contenuta nella variabile `$home`. In realtà, inserendo il framework all'interno di un portlet tramite il protocollo WSRP, l'index.php non verrà affatto preso in considerazione, e questo ha una sua logica evidente. Ma se ne tratterà in una sezione adeguata.

Tornando al codice, l'asterisco posto davanti a una definizione, indica invece che ci si trova di fronte automaticamente alla gestione e al controllo della navigazione tramite il corrispondente controllore posto all'interno del file *navigation.php*, nella cartella **Control.** E qui cominciano a tornare i ragionamenti fatti precedentemente.
In questo specifico caso, nel file *navigation.php*, si troverà un meccanismo di controllo che dirigerà la home page da una parte o dall'altra a seconda del tipo di utenza che si collegherà al sistema. È molto importante attribuire sempre dei nomi efficaci, che rendano l'idea di come il codice andrà implementato.
Si troverà quindi nel suddetto file *navigation.php,* la funzione `goto_check_user()` che si occuperà di indirizzare l'utente nella pagina di attivazione di accesso al servizio, nella pagina di amministrazione o, in caso di requisiti mancanti, in una pagina di avviso.
Quindi, quando si desidera procedere a un controllo della navigazione per pagine non predefinite, basterà inserire il simbolo cancelletto davanti al nome del controllore, il quale si troverà poi implementato col suffisso 'goto_' nel file *navigation.php.*

Proseguendo col *setting.php*, le regole di navigazione sono definite con un **array** php, nonché con la variabile **`$nav_config`**. Questo vettore viene opportunamente inizializzato per definire i cambi pagina all'esecuzione di una procedura di gestione evento: c'è un elemento dell'array per ogni pagina dell'applicazione e ogni elemento, ovvero ogni pagina, è a sua volta un array che definisce, per ogni evento che porta con sé, quale pagina deve caricare.

Così, nella sezione di listato seguente, si può individuare, ad esempio, come nell'elemento chiamato 'admin_page', che fa riferimento al file admin_page.tpl.php della cartella **view** e al file admin_page.php nella cartella **model**, corrisponde un array, dove, per ogni evento, si viene indirizzati alla pagina indicata.

```
. . .
$service = 'cedi2'; //nome del servizio
$home = '#check_user'; //a seconda dell'utente la home page sarà differenziata

   /*Diagramma di navigazione*/
$nav_config = array
(
       'info' => array (
                              'proceed'  =>  'rules'    //l'evento proceed porterà alla pagina di lettura regolamento
                    ),
       'rules' => array (
                              'accept'=>'service_activation'
//l'evento accept porterà alla pagina di attivazione del servizio
                                                          ),
       'admin_page' => array (
                     'activation'=>'service_activation',
                     'add'     => 'form1',
                     'modify' => 'form',
                     'send_email' => 'send_email_page',
                     'search' => 'search_result_page',
                     'info_modify' => 'info_modify_page',
```

```
'rules_modify' => 'rules_modify_page',
'header_modify'=>'header_modify_page',
'footer_modify'=>'footer_modify_page',
'vars_modify' => 'vars_modify_page'
                                    ),
```

Listato 2.1 Un primo estratto del file *setting.php*

Per chiarire meglio la questione, e a titolo esemplificativo, nella pagina di amministrazione si troverà un pulsante ‘add’ che porterà alla pagina utile ad aggiungere dati utente al database, oppure, premendo il tasto relativo all’evento ‘send_email’ ci si troverà rimbalzati nella pagina per l’invio di una email all’utente selezionato, la send_email_page, e così via.

Ancora, scorrendo nella lettura del codice si scoprirà che l’elemento send_email_page di cui sopra e a cui si è stati indirizzati, conterrà ancora un array dove per ogni elemento, corrispondente a un evento scatenante, come la pressione di un button, si verrà nuovamente reindirizzati a un’altra pagina:

```
'send_email_page' => array
                    (
                          'invia' => 'admin_page'
                    ),
```

Listato 2.2 Un elemento del vettore $nav_config in *setting.php*

In pratica, premendo il tasto ‘invia’, oltre a eseguire una serie di azioni atte a inviare l’email, si viene convogliati nuovamente nella pagina iniziale di amministrazione.

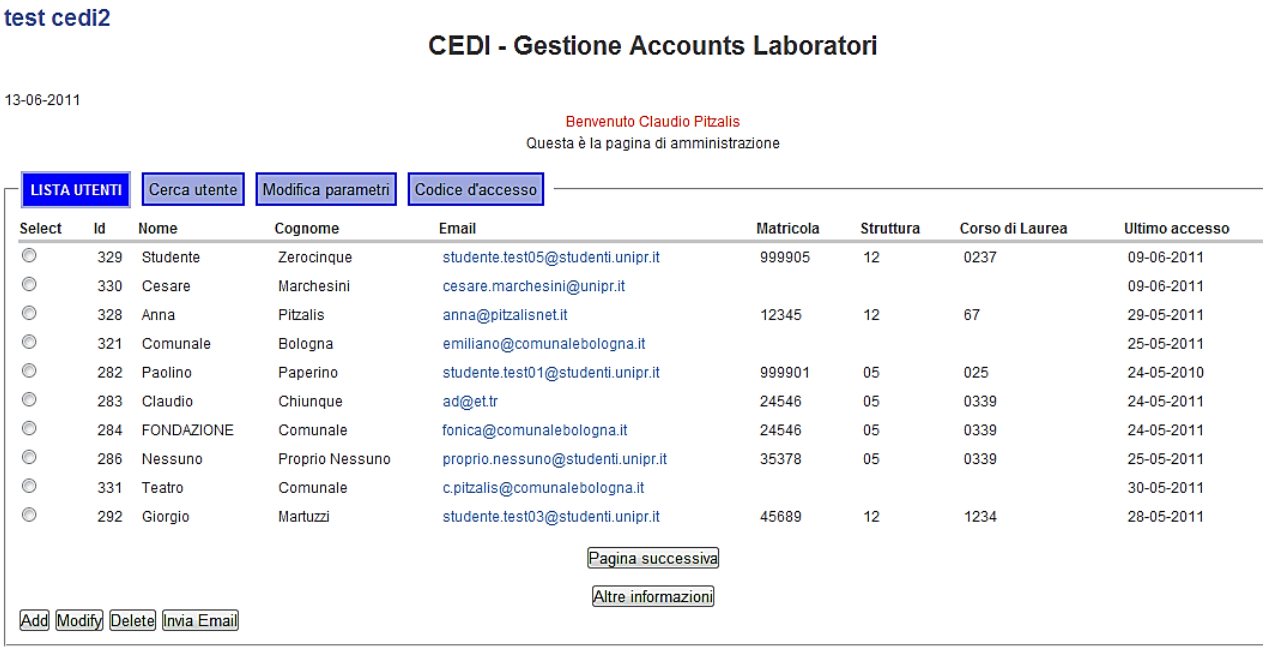

Figura 2.4 Pagina iniziale di amministrazione del servizio

Per ulteriore chiarezza, nella figura 2.4 qui sopra, viene anticipata la prima pagina che si presenta all'amministratore del servizio: si possono notare i pulsanti che scatenano gli eventi sopra citati.

Per concludere questa parte, si aggiunge che non è necessario descrivere le funzioni che non richiedono il cambio di pagina.

2.3.2 Il controllo sui dati

Sempre nel *settings.php* saranno definiti i controlli sui dati che devono essere inseriti nei campi. E per questo verrà utilizzato ancora una volta un vettore bidimensionale: **`$check_config`**. A ogni evento può essere associata una funzione: questo risulta utile quando si vogliono eseguire controlli sui dati di input o sulle variabili d'ambiente prima di procedere con le altre operazioni del servizio. Le funzioni di default associate all'evento si trovano nella libreria **check.inc** all'interno della cartella **include**, mentre le funzioni definite dallo sviluppatore del servizio vengono inserite nella cartella **control** nel file *check.php.*

```
$check_config = array
(/*nella pagina 'rules'...*/
  'rules' => array
```

```
(   /*all'evento:accept...*/
   'accept' => array
    (
  /*viene controllato che il box di nome accept sia
stato spuntato... */
   'accept' => array (
  /*utilizzando la funzione check_notempty()*/
   'type' => 'check_notempty',
   'message' => $messages['no_check_box']
        )
);
```

Listato 2.3 Un secondo estratto del file *setting.php*

In questo frammento di codice, sempre parte del *setting.php*, i commenti inseriti rendono piuttosto bene l'idea del meccanismo di funzionamento del controllo degli eventi: in particolare ci si trova di fronte alla parte che fa riferimento alla pagina di regolamento ('rules') per l'accesso ai laboratori. Alla pressione del tasto 'accept' viene controllato se il box relativo è stato spuntato tramite il metodo check_notempty(), che, essendo uno dei metodi predefiniti nel pacchetto, sarà posizionato nel file *check.inc*.

Qui sotto è visibile parte della pagina a cui si fa riferimento, *rules.tpl.php*, dove, per poter proseguire, è necessario spuntare il box.

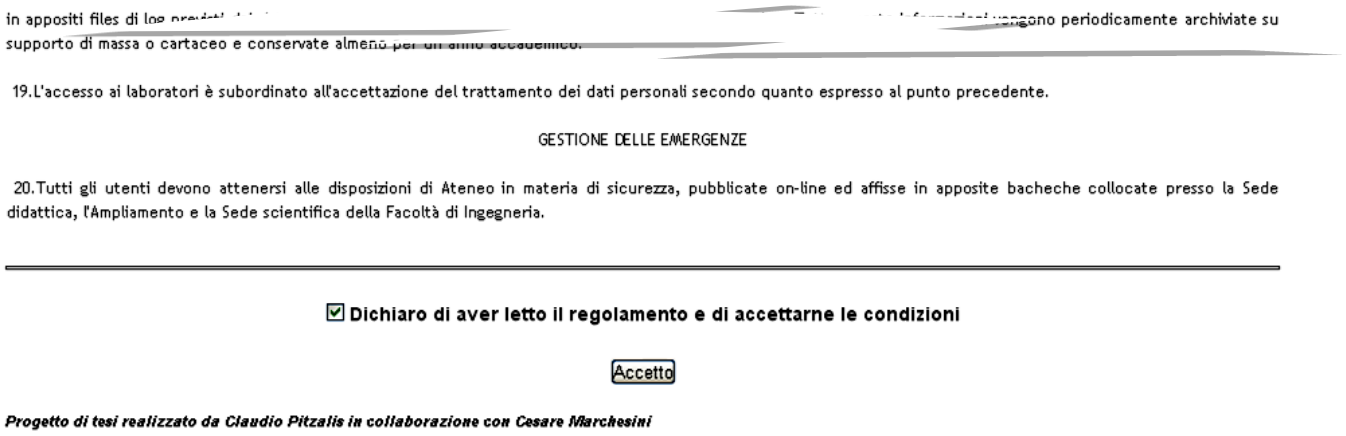

Figura 2.5 Pagina per l'accettazione del regolamento scritto con jaMVC

Si noti come la pagina riprenda quella già vista nella figura 1.1 di pag. 2.

Proseguendo nella descrizione del codice, sono presenti anche indicazioni utili sul meccanismo: per ogni evento di cui si desidera far partire il controllore, viene creato un vettore in cui

- prima di tutto viene inserito l'elemento chiave che prende il nome utilizzato, ad esempio, per il box, il radio o la text-area presenti nella pagina html situata nella cartella **view**. In questo caso l'elemento è **`'accept'`**
- il secondo elemento, il cui indice **`'type'`** è comune a tutte le richieste di controllo, fa riferimento appunto al tipo di controllo da eseguire e rimanda quindi al metodo e alla sua implementazione: in questo caso si tratta di una funzione predefinita già scritta nel pacchetto jaMVC, ma gli sviluppatori potranno inserire controlli specifici e dedicati all'interno del file *check.php* come detto sopra. Se ne vedrà almeno un esempio in seguito.

  C'è da aggiungere che ogni elemento può essere a sua volta un vettore con due o più codici di controllo: per esempio all'interno di un form vi può essere necessità di controllare sia l'esistenza del dato inserito come la sua correttezza.

```
'form1' => array
              (
'add_email'  => array
                    (
    'type' => array('check_email','check_no_duplicate'),
    'message'=>array($messages['email_error'],$messages['email_already_present'])
                                                          ),
```

Listato 2.4 Controllo di inserimento dati nel form

Nell'esempio si intuisce come nel form per l'aggiunta dell'email di un nuovo utente, sarà necessario fare un controllo prima della corretta struttura dell'indirizzo (e a questo pensa un metodo predefinito), poi della presenza di eventuali duplicati nel database (questa implementazione è stata invece aggiunta dallo sviluppatore).

Nel caso vi siano più elementi che richiedono lo stesso tipo di controllo, è possibile utilizzare una variabile che contiene il codice richiesto, così da non doverlo riscrivere ogni volta.

Un esempio:

```
$id_exists = array
          (
            'id' => array
                  (
                    'type' => 'check_exists',
                    'message' => $messages['no_select_row']
                       )
             );
```

e nel seguito del codice ritroveremo la variabile utilizzata in questo modo:

```
'admin_page' => array
              (
                  'modify' => $id_exists,
                  'delete' => $id_exists,
                      'send_email' => $id_exists,
                      'search' => array (
                                              'insert_name' => array
                                                    (
```

Listato 2.5 Variabile come controllore

- Come terzo elemento si ha la componente relativa al messaggio di errore, eventualmente commesso, che sarà mostrato all'utente. È possibile inserire sia direttamente il messaggio nel vettore, come è anche possibile puntare al vettore contenente tutti i messaggi, situato all'inizio dello stesso file *setting.php*. in questo progetto è stata adottata la seconda soluzione.

```
$messages = array (
     'no_check_box' => 'Per poter proseguire occorre accettare il regolamento
          spuntando il sottostante checkbox.',
     'no_select_row' => 'È necessario selezionare una riga.',
     'add_user' => 'Aggiungi utente',
     'no_select_name' => 'Devi indicare un nome',
     'no_select_surname' => 'Devi inserire il cognome',
     'email_error' => 'Il formato dell\'email non è corretto.',
     'email_already_present' => 'ATTENZIONE. L\'email è già presente nel
          database.',
     'no_insert_object' => 'Devi inserire l\'oggetto',
     'no_insert_text' => 'Manca il contenuto del messaggio',
     'at_least_one_data' => 'Devi inserire dati almeno in una casella di
          testo',
     'no_insert_date' => 'Devi inserire un giorno e un mese: (dd/mm).',
```

Listato 2.6 Il vettore dei messaggi d'errore $messages

Questi due blocchi fondamentali del *setting.php*, soprattutto se scritti seguendo la logica del servizio che si vuole offrire, permettono di avere subito a portata di

mano un riferimento per la costruzione delle pagine e l'implementazione dei metodi richiesti. Durante la progettazione del servizio, in maniera interattiva, vengono presi uno per uno gli elementi degli array e su questi sono costruite le altre parti del codice.

In definitiva, questa parte può essere scritta, anche con pseudo-codice, dal committente del servizio e il rimanente, e più corposo lavoro, può essere più facilmente svolto da sviluppatori esperti in Php che sapranno soddisfare adeguatamente le richieste abbozzate nel diagramma di navigazione espresso dal *setting.php*.

Si termina questa sezione indicando la presenza, in questo file, di altre due variabili importanti: una, **`$conn`**, per la connessione al database, e l'altra, **`$wsrp_settings`**, che contiene i parametri del servizio: nome servizio, nome visualizzato e titolo, utili per rendere accessibile il servizio, come si approfondirà nel prossimo capitolo, tramite un portale come un portlet WSRP.

```
$conn = new PDO('oci:dbname=//localhost:1521/XE', 'password',
     'nome_db');
$conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
$conn->setAttribute(PDO::ATTR_CASE, PDO::CASE_LOWER);
$conn->setAttribute(PDO::ATTR_STRINGIFY_FETCHES, FALSE);

$wsrp_settings =  array
(
    'id' => $service,
    'title' => array
                (
                     'it' => $service,
                     'en_US' => $service
                ),
    'display-name' => array
                     (
                        'it' => $service,
                        'en_US' => $service
                     )
);
```

Listato 2.7 Le variabili $conn e $wsrp_settings

2.4 Model e view: i metodi, le implementazioni e gli eventi

Nella sezione precedente si è già data un'infarinatura di quello che può essere la costruzione di una pagina html: il risultato finale mostrato all'utente attraverso l'evolversi degli eventi che di volta in volta si presentano nel diagramma di navigazione. Vediamo quindi più in dettaglio, con qualche esempio, il contenuto e le funzioni delle cartelle **model** e **view** che soddisfano, insieme al **control**, al paradigma MVC.

2.4.1 Componenti dell'interfaccia

In generale, quando si passa all'implementazione delle view (le cosiddette 'viste') si potrebbe partire da un template fatto da un grafico e da questo produrre l'html. Una volta fatto l'html si crea il template Php, che ha estensione ".tpl.php" per posizionarsi nella cartella **view**. È necessario avere particolare cura e attenzione sull'uso dei nomi e degli id che si danno agli oggetti, perché il gestore degli eventi e delle chiamate di funzioni utilizzano proprio i nomi specificati nella parte view. Ciò sarà sempre più chiaro nel proseguimento delle varie fasi del progetto.
Se si hanno più view con parti di codice html in comune, relative a componenti di grafica (tipo barre di navigazione, menù, ecc.), il framework in questo aiuta: infatti è possibile mettere dei tag XML al posto di parti di codice html comuni. L'utilità dei tag, che generalmente sono definiti nei fogli di stile, è evidente, perché consentono di astrarre la view dall'html puro.
A parte qualche caso particolare, nei template di solito non c'è codice vero e proprio: c'è solo appunto un po' di html e di tag. Per questo può essere implementato anche da chi non ha dimestichezza con il Php.

A tale proposito, prima di proseguire, è bene dire che nello sviluppo di questo particolare servizio per l'accesso ai laboratori, non si è tenuto conto della grafica, sia perché non essenziale, viste le funzioni che deve svolgere, sia perché generalmente è una parte che occuperebbe spazi temporali ampi ed è legata al

gusto personale. Inoltre, specialmente quando lo sviluppatore, come in questo caso, lavora in autonomia, è bene che si occupi fondamentalmente degli aspetti relativi al funzionamento del codice in sé.

Sarà sempre possibile anche a posteriori, per uno sviluppatore grafico, elaborare e definire al meglio gli aspetti visivi, senza per questo intaccare il funzionamento del software relativo al servizio stesso.

Di seguito si tenterà di dettagliare le componenti grafiche dell'interfaccia utilizzate.

Come riferimento sarà utilizzata ancora una volta la pagina dell'amministratore, dove si trovano la maggior parte dei tag proposti dal pacchetto.

```
<tabs id="admin">
     <tab label="Lista utenti">
     <report id="users" rows4page="10">
            <thead>
              <tr>
                    <th>Select</th>
                    <th>Id</th>
                    <th>Nome</th>
                    <th>Cognome</th>
                    <th>Email</th>
                    <th>Matricola</th>
                    <th>Struttura</th>
                    <th>Corso di Laurea</th>
                    <th>Ultimo accesso</th>
              </tr>
          </thead>
          <tbody>
              <tr>
                    <td><radio name="id"/></td>
                    <td>{id}</td>
                    <td>{nome}</td>
                    <td>{cognome}</td>
                    <td><a event="modify" title="Modifica i dati
     di {nome} {cognome}" query="id={id}">{email}</a></td>
                    <td>{matricola}</td>
                    <td>{id_facolta}</td>
                    <td>{id_laurea}</td>
```

```
                    <td>{last_access_d}</td>
                </tr>
            </tbody>
        </report>
            <button event="add" label="Add"/>
            <button event="modify" label="Modify"/>
            <button event="delete" label="Delete"/>
            <button event="send_email" label="Invia Email"/>
        </tab>
        <tab label="Cerca utente" >
            <h3>Cerca utente registrato</h3><br /><br />
                        "position:absolute;left:310px;width:200px;"
name="insert_email"                                    /><br /><br />

        </tab>
</tabs>
```

Listato 2.8 Il template admin_page.tpl.php nella cartella view

Nella figura 2.4 di pagina ventitré è possibile vedere il risultato ottenuto da questo codice: scorrendo uno per uno i vari tag, si nota che `<tabs…` apre quella che è la 'cornice' dell'interfaccia e ogni singolo `<tab>` corrisponderà invece alle diverse viste che si possono ricevere al click del relativo box. Si ha perciò che il tab di default, col label inerente a esso, mostrerà la lista utenti (`<tab label="Lista utenti">`) insieme alle operazioni eseguibili su di essa.

Così, più avanti, si possono notare gli altri tab con la loro definizione e costruzione e che consentono di visualizzare le altre funzioni offerte all'amministratore del servizio: il tab successivo, ad esempio, `<tab label="Cerca utente" >`, come si può vedere dalla figura successiva, apre un form su cui poter compiere una ricerca utenti accurata all'interno del database. Nel form troveremo altri tag per il riempimento dei campi input, come

```
<text style = "position:absolute;left:310px;width:200px;"
 name = "insert_email" label = "Cerca per indirizzo email"/>
```

dove, oltre a inserire il nome che dovrà essere coerente con la chiamata alla funzione, è possibile gestire anche le dimensioni della text area.

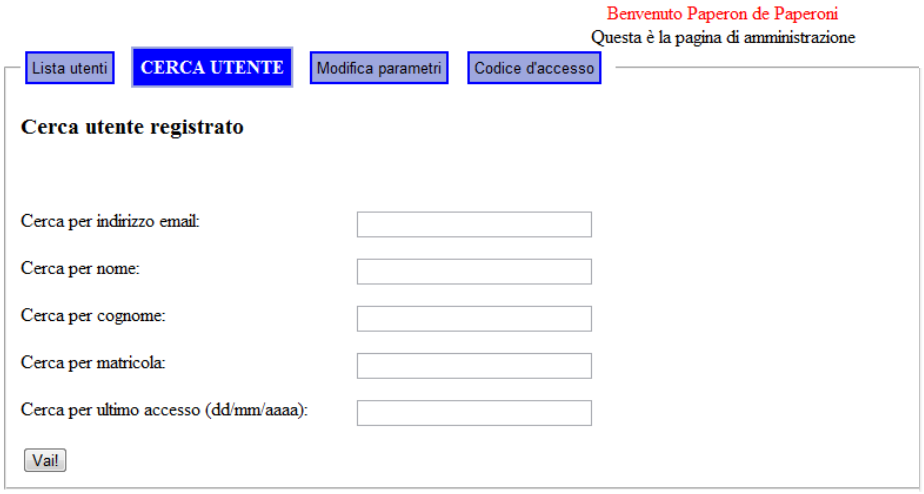

Figura 2.6 Interfaccia amministrativa per ricerca utente

Esistono anche altri campi input:

```
<email name="email" label="Email"/>
<password name="idvariabile"/>
<textarea name="idvariabile" rows="numerorighe" cols="numerocolonne"/>
```

che si descrivono da soli.

Per i rimanenti tab si rimanda alla lettura del codice completo, che ha di notevole, una volta compresa la definizione di ogni tag, di essere piuttosto semplice, quasi auto-esplicativo.

Quindi, riassumendo, ci si trova in una pagina di template all'interno della cartella view, si è visto che i tab aiutano a definire una struttura gerarchica all'interfaccia e permettono, pur rimanendo all'interno di una stessa pagina, di creare quante viste si vuole. Rimane ora da vedere cosa si può inserire tra `<tab>` e `</tab>`.

Tornando alla nostra lista utenti, all'interno del tab ci si imbatte nel tag `<report…>` che è una componente dell'interfaccia utente, basata su XSL, per la quale si apre una breve parentesi.

Nel pacchetto jaMVC, nella sottocartella **ui** dentro la cartella **includes**, si possono trovare le UI (User Interface): sono delle routine php già fatte che il framework mette a disposizione, ad esempio come nel caso del report, per accedere ai dati. In particolare si trovano l'implementazione della UI *report* e quella per le liste (*list*). Per il servizio richiesto è stata utilizzata solo l'interfaccia utente report. A ogni modo è anche possibile costruire delle proprie UI.

Secondo il codice proposto, il report svolge le seguenti funzioni: intanto l'id del report è l'attributo per gestire l'univocità del tag e poter avere una funzione di riempimento; in questo caso 'users', identificherà il report nel model di questa pagina (admin_page.php).

`rows4page="10"` individuerà invece il numero di tuple che si desidera mostrare (qui si otterrà, per ogni pagina, una lista di 10 utenti). Proseguendo, viene creata una semplice tabella costruita con banale html: tra i tag `<th>` e `</th>` vi sono inserite le intestazioni delle colonne, mentre tra i tag `<td>` e `</td>` vengono inseriti i nomi dei campi: le parentesi graffe sono necessarie, infatti per riempire i report viene utilizzata la funzione *query_users()* nel model, che interpreta il nome ricevuto come il nome del campo della query di riempimento.

query_users() è dunque la funzione che viene invocata per riempire il report di nome *users* ed occorre che la funzione restituisca delle tuple coerenti con la struttura del report dichiarata nel view.

Per dare l'idea della 'raffinatezza' del codice, si fa notare come il nome della funzione *query_users()* sia strettamente legata all'id del report: in un'altra view è stato costruito un altro report con id 'search_users' e nel model corrispondente si troverà la funzione *query_search_users()* che avrà metodi simili. Questi metodi sono supportate dalla UI tramite la funzione di base *ui_report_fetch()* incastonata all'interno del file *report.inc* nella cartella poco sopra descritta.

```
function query_users()
{
   return ui_report_fetch("select * from lab_access_1");
}
```

Listato 2.9 Funzione query per il prelievo dei dati nel DB

jaMVC, come si nota dal frammento di codice qui sopra, offre dunque utilissime funzioni in grado di prelevare con estrema semplicità i dati richiesti nel database: qui vengono prelevati i dati di ogni utente dalla tabella di dati nel database da inserire dinamicamente nel report 'users' nella view admin_page.tpl.php.

La funzione `ui_report_fetch( "$query_string", array($var1 , $var2));` restituisce le tuple risultanti dall'interrogazione $query_string. Quest'ultima è la variabile contenente la stringa SQL per eseguire l'interrogazione, mentre $var1 e

$var2 forniscono un esempio di passaggio di valori alla query dove i punti interrogativi '?' nella stringa sono sostituiti con i parametri del vettore, come si può constatare dalla funzione *query_search_users()* qui sotto:

```
function query_search_users()
     {
     return ui_report_fetch("select * from lab_access_1
          where ? = ?",array('email',get_page_value('insert_email')));
     }
```

Listato 2.10 Passaggio di valori alla query tramite '?'

Tornando al report, nella prima colonna si trova un pulsante radio nel cui corrispondente codice c'è l'attributo *name* che serve per poter gestire il parametro nel Model, dove avrà lo stesso valore che nel form. Da notare come il nome "id" dato al radio sarà utilizzato anche dal controllore per verificare se un pulsante è stato selezionato o meno (si veda listato 2.5 a pagina 26).

2.4.2 Gli eventi

Si è già parlato più volte di 'eventi' o di qualcosa che 'scatena gli eventi': scorrendo ancora il codice si trovano eventi sia via link che eventi via button che potranno dare bene l'idea del concetto.

```
<a event="modify" title="Modifica i dati
          di {nome} {cognome}" query="id={id}">{email}</a>
```

Evento via link

```
<button event="add" label="Add"/>
```

Evento via button

Nel primo caso 'query' è obbligatorio e serve a passare dei dati come parametro alla funzione di gestione dell'evento. Se non serve specificare una query, si utilizza query="" . Il codice presentato scatena l'evento "modify" e nel contempo passa alla funzione *on_modify()*, definita nel model, il parametro 'id' selezionato, attraverso funzioni per il passaggio di variabili, come get_selected_value() che vedremo in seguito.

Per il gestore degli eventi esistono delle convenzioni: se si ha un evento che si chiama ***add***, come nel secondo caso dell'evento via button, e se si definisce nel

Model una funzione che si chiama ***on_add()*** allora il controller la invoca. Le funzioni che gestiscono gli eventi sono detti event handlers.

L'evento il più delle volte porta a una nuova pagina: nel primo caso porta a una pagina con un form di modifica dei dati dell'utente selezionato, nel secondo caso invece si verrà indirizzati a un form per l'aggiunta di un utente.

Per verificare fin quanto detto finora, basti notare nel *setting.php* come alla voce 'add' si venga dirottati alla pagina 'form1', mentre la voce 'modify' porti alla pagina 'form' (listato 2.1 pagina 22).

Prevalentemente gli eventi sono definiti dallo sviluppatore e le loro implementazioni si trovano nella pagina, dove l'evento viene scatenato, cioè, per esempio, nel model dove è posizionato il link o il button. Quindi abbiamo, in questo caso, nel view admin_page.tpl.php, il link 'modify' e il button 'add' e nel model admin_page.php i metodi on_add() e on_modify(). Non sempre sono necessarie le implementazioni di queste funzioni: spesso l'evento è solo un reindirizzamento di pagina che svolge il controller tramite l'array $nav_config in *setting.php*. Infatti, qui sotto si vede come l'implementazione di on_modify() contiene codice attivo, mentre on_add() è vuoto. Non è necessario inserire nel model una funzione di fatto non implementata, ma è consigliabile lasciarla per futuri eventuali sviluppi o modifiche. È semplicemente una questione di regola e di migliore correttezza di programmazione.

```
function on_add()
   {

   }
function on_modify()
{
set_page_query_values(db_query("select   *   from   lab_access_1   where
        id= ?",array(get_value('id'))));
}
```

Listato 2.11 Metodi degli eventi 'add' e 'modify'

Oltre agli eventi utente (cioè definiti dallo sviluppatore) ci sono anche degli eventi predefiniti, come ad esempio "on_<nome_pagina>_enter", ed è un evento che il controller scatena quando entro in una pagina. Un altro è *on_<nome_pagina>_exit*: se la pagina si chiama form, allora la funzione si chiamerà *on_form_exit*.

Nel progetto tema della tesi, questa convenzione sui nomi sarà usata di frequente. Un'altra volta si può costatare come il pacchetto jaMVC renda veramente semplice la costruzione di un servizio per il programmatore: il framework lavora in base alle convenzioni (conventions over configurations), per cui se si chiama una funzione in un certo modo, si è in grado di sapere già che quella sarà chiamata dal controller in determinate situazioni.

Esistono anche degli eventi speciali: l'evento *back*, che porta una pagina indietro, e *home* che reindirizza alla pagina home:

```
<a event="home"query=""></a>
```

oppure

```
<button event="back"label="Torna indietro"/>
```

Nel file *navigation.inc*, dentro la cartella **includes**, viene definito tutto ciò che riguarda gli eventi generici dell'applicazione, tramite le funzioni *execute_<nome_evento>()*:

```
function execute_page_enter($action)
    {
       if (file exists('model/'.$action.'.php'))
                                    .$action.'.php');
     }

function execute_page_exit($action)
    {
       if (file_exists('model/'.$action.'.php'))
         get_php_contents('model/'.$action.'.php');

       remove_page_value($name,$act...
    }
```

Listato 2.12 Frammenti del file navigation.inc

Un valido esempio può essere la pagina iniziale d'informazioni che si presenta allo studente:

```
function on_info_enter()
{
  /*Richiamo il file dal server per inserire i dati nella pagina*/
              $file = 'resources/info_file';
              if (!($read_info_file = fopen($file, 'r')))
              add_info("Non riesco ad aprire il file: ".$file);
```

```
            else
            {
            $info = fread($read_info_file, filesize($file));
            fclose($read_info_file);
            set_value('info',$info);
            }
}
```

Listato 2.13 Esempio di funzione d'ingresso pagina.

Nell'esempio qui sopra, all'apertura della pagina **view** 'info.tpl.php', verrà eseguita la funzione 'info.php' posta nel **model**: si richiama e si apre il file conservato sul server, *info_file*, dentro la cartella resources, che contiene il testo delle informazioni, per essere inserito in una variabile di contesto, $info. Questa variabile sarà richiamata nel view per essere stampata a video.

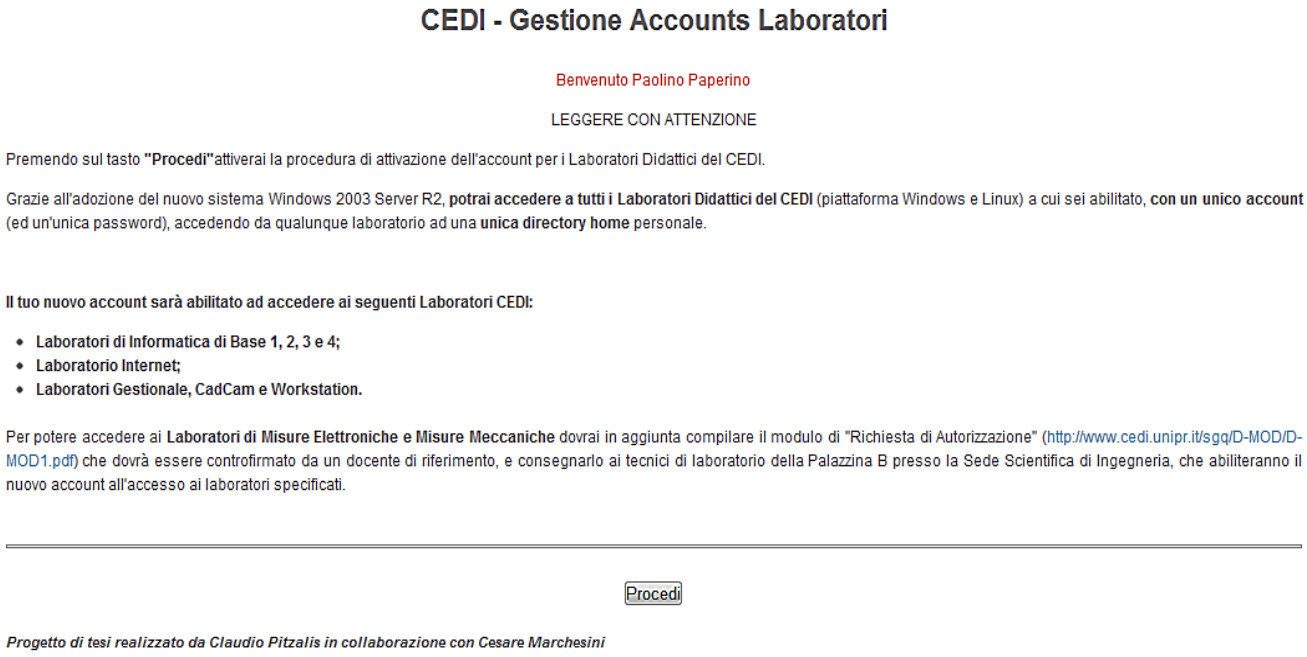

Figura 2.7 Pagina iniziale di informazione per studente

Anche qui è possibile fare un confronto tra questa pagina, creata col framework jaMVC, e la figura 1.1 a pag.2 che rappresenta invece il servizio attuale.

2.5 Gestione delle variabili

Si è già vista qua e là, nei frammenti di codice proposti, la necessità di passare alcuni dati o variabili da una pagina all'altra o dal database. Il framework gestisce i dati con il concetto di contesto e ogni dato in ogni tipo di contesto è identificato da un **nome** e dal suo **valore**.

I contesti di utilizzo delle variabili sono tre:

- **Contesto di Sessione**: vale per tutta la sessione (cookie). Nel contesto di sessione sono gestite le variabili globali di sessione, comprese quelle specificate in *settings.php*. La libreria per la gestione di queste variabili è *session.inc* nella cartella **includes.**

  Dettagliando le funzioni principali di questa libreria, si hanno

  - ✓ *set_session_value('name','value')*:
    setta la variabile di sessione 'name' con il valore di 'value'. Se value non è specificato lagge il valore corrente della variabile.
  - ✓ *set_session_values('name','array_values')*:
    stessa cosa di *set_session_value('name','value')*, ma per vettori.
  - ✓ *get_session_value('name')*:
    ritorna il valore della variabile 'name'; nel caso sia un vettore ritorna il primo elemento.
  - ✓ *get_session_values('name')*:
    ritorna il vettore referenziato da 'name'.
- **Contesto di Pagina**: è tutto ciò che rimane vivo tra un evento *enter* e un evento *exit* di una pagina, visti prima; cioè si apre quando si carica la pagina e si chiude quando la stessa si chiude. Le funzioni di gestione delle variabili di pagina sono contenute nella libreria *page.inc* posizionata nella stessa cartella della precedente libreria.

  Quelle di più comune utilizzo sono:

  - ✓ *set_page_value('name','value')*:
    setta la variabile di pagina 'name' con il valore di 'value'.
  - ✓ *set_page_values('name','array_values')*:
    stessa cosa di *set_page_value('name','value'),* ma per vettori.

- ✓ *get_page_value('name')*:
  ritorna il valore della variabile 'name', se è un vettore ritorna il primo elemento.
- ✓ *remove_page_value('name',$action=false)*:
  esegue il destroy della variabile 'name'.
- ✓ *set_page_query_values($stmt)*:
  carica come variabili di pagina i valori estratti dal DB con la query inserita nella variabile $stmt.

- **Contesto di Richiesta**: le variabili passate con i metodi get e post possono essere lette con le funzioni contenute in *request.inc*. Valgono solo per la singola richiesta.
  - ✓ *get_value('name')*:
    ritorna il valore della variabile 'name'.
  - ✓ *get_values('name')*:
    ritorna il vettore 'name'.
    Similmente ai contesti precedenti esistono le funzioni *set_value('name','value')* e *set_values('name','array_values')*.

Esiste inoltre una serie speciale di funzioni per la gestione delle variabili. Le funzioni di questo gruppo permettono di selezionare automaticamente l'ambito della variabile da utilizzare. In particolare, la funzione di utilità *get_selected_value('name')*, presente nel file *navigation.inc*, ancora dentro la cartella **includes**, cerca le variabili prima nel contesto di richiesta, poi di pagina e infine nel contesto di sessione. In altre parole, dopo aver tentato di recuperare il dato nel contesto di richiesta, cercherà di operare in ambito di pagina; se la variabile di pagina, il cui nome è stato dato come parametro, non esiste, la funzione cercherà la variabile in ambito di sessione. Se anche questa variabile non esiste, sarà restituito un valore nullo.

```
function get_selected_value($name)
    {
        $retval= null;
        if (exists_par($name))
          $retval= get_value($name);
        elseif (exists_page_par($name))
          $retval= get_page_value($name);
```

```
        elseif (exists_session_par($name))
        {
          $retval= get_session_value($name);
          set_page_value($name,$retval);
        }
          return $retval;
    }
```

Listato 2.14 Implementazione del metodo get_selected_value()

Si possono eventualmente creare altre funzioni, definite dallo sviluppatore, utili alla gestione delle variabili; ad esempio per il progetto è stato molto utile definire la semplicissima funzione *set_session_query_values($stmt)*, altrimenti non implementata, che consente di prelevare i dati richiesti da una tabella del database a cui è stata inviata la query $stmp, e inserirli direttamente in variabili equivalenti di sessione. In mancanza di questa si sarebbe dovuto sopperire utilizzando prima *set_page_query_values($stmt)*, che carica i dati nella pagina, e successivamente caricare gli stessi in sessione tramite *set_session_values_from_page($names)*. Qui sotto vi è illustrata l'implementazione di *set_session_query_values($stmt)* che è stata costruita modificando di poco quella corrispondente relativa al contesto di pagina.

```
function set_session_query_values($stmt)
    {
        $arr = db_fetch_array($stmt);

        if (is_array($arr))
        foreach ($arr as $n => $v)
          if (is_resource($v))
            set_session_value($n,stream_get_contents($v));
          else
            set_session_value($n,$v);
    }
```

Listato 2.15 Il metodo get_session_query_value() creato dallo sviluppatore

Questa funzione viene utilizzata, ad esempio, all'ingresso dell'utente nel servizio: vengono prelevati i suoi dati dal database e tenuti per tutta la durata della sessione, pronti per ogni eventuale utilizzo. Nel frammento di codice sottostante, si individua come, tramite la funzione get_wsrp_user(), che preleva i dati utente tramite il modulo wsrp e Shibboleth, questi vengano, dopo un passaggio al contesto di richiesta, passati alla sessione.

```
$user->{"uid"}= get_wsrp_user();
        set_value('email',$user->{"uid"});
/* Prelevo i dati dell'utente che si è autenticato e li inserisco nel
contesto di sessione*/
set_session_query_values(db_query("select * from lab_access
                          where email = ?", array(get_value('email'))));
```

Listato 2.16 Prelievo dati utente e inserimento in sessione

Una volta avuto l'accesso, si riceverà un messaggio di benvenuto con nome e cognome utente, come visto ad esempio in figura 2.7, grazie al semplice codice inserito nella view relativa:

```
echo "<font color = \"red\">Benvenuto ".get_session_value('nome') ."
".get_session_value('cognome')."</font><br />";
```

Sono presenti altre funzioni per la gestione delle variabili, come ad esempio *remove_session_value('name',$action=false)*, anch'essa utilizzata nel servizio, ma per le quali si rimanda alla lettura del codice.

2.6 header, footer e messaggi utente

Prima di concludere questo capitolo, è bene dare un'occhiata ad alcune parti abbastanza importanti riguardanti il funzionamento del sistema.

Nella cartella view si possono notare due file: header.php e footer.php. Essi contengono rispettivamente il codice dell'intestazione e del piè pagina della vista. Per chiarire bene, partiamo da **index.php**: è il primo file che viene eseguito al caricamento del servizio, e solitamente contiene l'inclusione delle librerie jamMVC e il metodo show(), incluso *in navigation.inc*, che reindirizza alla home page del servizio. Analizzando l'implementazione della funzione show(), tra le altre cose si notano le funzioni head() e foot() che in praticano leggono il contenuto dei file header e footer precedentemente citati. L'uso di questi file risulta a abbastanza evidente. Si consideri una normalissima pagina html: nell'header c'è tutta l'intestazione fino al tag <body> compreso, nel footer può semplicemente trovarsi </body></html> a rispetto della chiusura di una pagina html. All'interno sarà posto il servizio sviluppato.

L'utilità di questo meccanismo è indiscutibile: si scrive una volta per tutte questa sezione di codice, che verrà inserita automaticamente in ogni pagina del servizio.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="it">
  <head>
    <title>Laboratori di Informatica - CENTRO DIDATTICO INGEGNERIA
    </title>
    <script src="<?php resource_url('jquery.js'); ?>"></script>
    <script src="<?php resource_url('jquery.form.js'); ?>"></script>
    <script src="<?php resource_url('jquery.datePicker.js');?>">
</script>
    <script src="<?php resource_url('jamvc.js'); ?>"></script>
  </head>
  <body>
   <center>
   <h1>CEDI - Gestione Accounts Laboratori</h1><br /></center>
```

Listato 2.17 Esempio di header.php

Nell'esempio qui sopra si ha anche parte di codice oltre il body: ciò consentirà di avere per ogni pagina del servizio una intestazione costantemente presente.

Lo stesso discorso si può ripetere per il piè pagina.

Nel servizio oggetto della tesi, si è sviluppato un comodissimo costrutto che consente all'amministratore di modificare, tra gli altri parametri, anche queste parti, senza necessità di entrare direttamente nel codice, ma attraverso l'interfaccia web.

Figura 2.8 Tab per la modifica dei parametri nella pagina amministrativa

Dalla figura qui sopra, oltre a verificare la presenza dell'header e del footer, si possono notare alcune delle possibilità al momento offerte all'amministratore, tra cui, appunto, la modifica dei precedenti parametri.

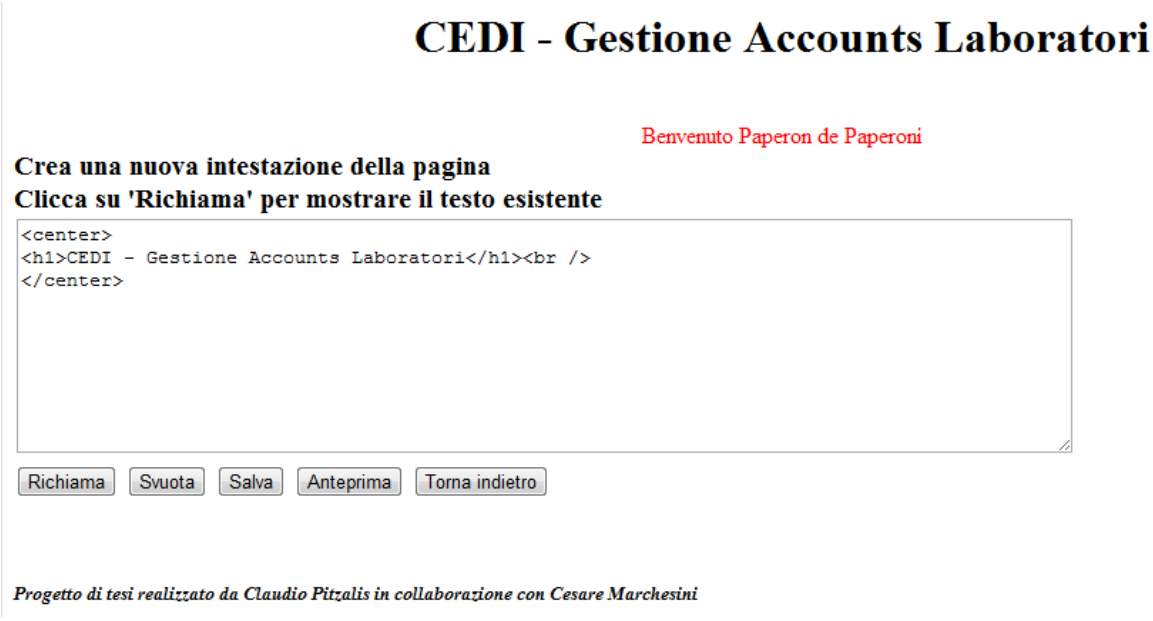

Figura 2.9 Pagina per la modifica della pagina d'intestazione

In alcune pagine, anche in quelle viste fino ad ora, vi può essere la necessità di inviare un messaggio all'utente, che sia esso di errore, di conferma o altro.
Senza soffermarsi troppo sull'argomento, basti dire che in ogni pagina view è sufficiente inserire, nella sezione dove si desidera scrivere il messaggio, il tag <message />. Qui, per esempio, verranno posti i messaggi di controllo di cui è già stato esposto discutendo sul *setting.php*

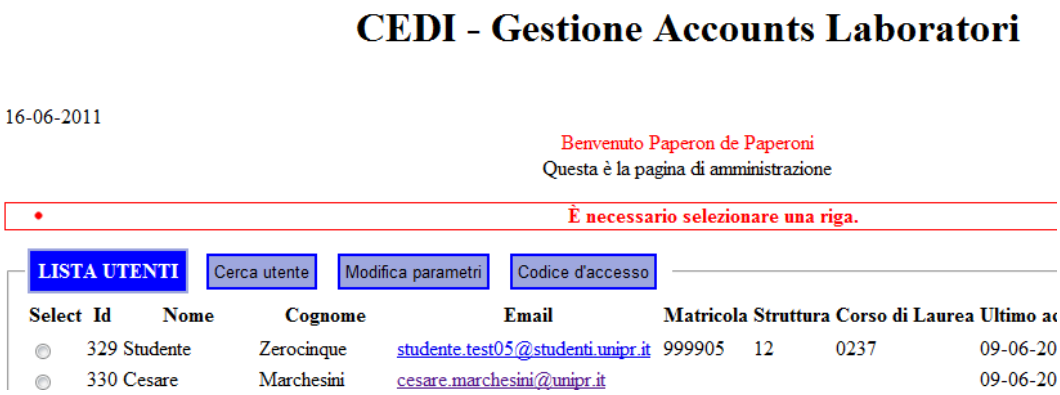

Figura 2.10 Messaggio di errore

Dalla figura precedente si può verificare sia quanto detto a pag. 26, corrispondente ai listati 2.5 e 2.6 sulla variabile di controllo $id_exists, sia come viene mostrato il messaggio, in questo caso di errore. Qui sotto parte del codice relativo.

```
<?php echo date('d-m-Y');
        echo "<center>";
        echo       "<font       color       =       \"red\">Benvenuto
".get_session_value('nome')                                          ."
".get_session_value('cognome')."</font><br />";
        echo 'Questa &#232; la pagina di amministrazione<br />';
?>
        <messages/></center>
```

Listato 2.18 Esempio di logica per l'invio del messaggio

Altri metodi per inviare messaggi sono ad esempio aggiungere nel codice di implementazione degli eventi che si scatenano delle funzioni come le seguenti:

- *void add_info(string message)*:
  aggiunge un messaggio INFO all'utente
- *void add_warning(string message)*: aggiunge un messaggio WARNING all'utente
- *void add_error(string message)*: aggiunge un messaggio ERROR all'utente
- *void add_fatal(string message)*: aggiunge un messaggio FATAL error all'utente

L'esempio qui sotto illustra molto bene il meccanismo con cui viene inviato un messaggio all'amministratore dopo aver cancellato un utente dal database.

```
function on_delete_found()
  {
 db_query("delete lab_access_1 where id = ?",array(get_value('id')));
 add_info("Utente cancellato.");
  }
```

Listato 2.19 Utilizzo della funzione add_info() per l'invio di messaggi

E nell'immagine seguente il risultato dell'evento.

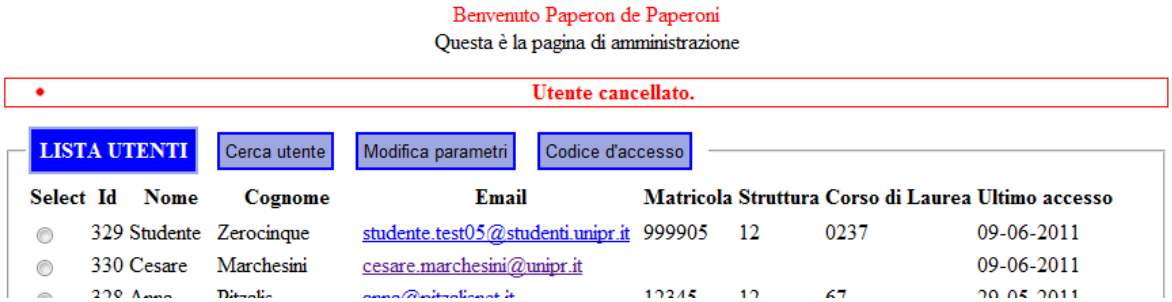

Figura 2.11 Risultato dell'evento 'Delete'

Per terminare questo capitolo, viene indicata la presenza di altre cartelle che compongono e completano l'ambiente di sviluppo del framework jaMVC, sulle quali però si tralasciano gli approfondimenti.

- **help**: contiene le eventuali pagine di help per il servizio.
- **resources**: contiene le risorse utilizzate dal servizio WSRP, come, ad esempio, file di stile CSS, librerie javascript, e immagini caricate nelle pagine web.
- **xsl**: contiene le librerie utilizzate dal protocollo XSLT (Extensible Stylesheet Language Transformations). Per ogni pagina viene creato un file XML che unifica moduli View e Model del pattern MVC, questo file viene poi trasformato in una pagina XHTML dal protocollo XSLT. All'interno si trova *default.xsl* che è usato appunto di default: se si desiderano alcune formattazioni particolari è possibile scrivere un xsl personalizzato e posizionarlo sotto questa cartella **xsl** dell'applicativo.
- **includes**: contiene le librerie del servizio WSRP. Alcune di queste sono già state citate precedentemente. In alcune versioni di questo pacchetto, la cartella si chiama, non a caso, **core**.

Capitolo 3

jaMVC come servizio WSRP

In questo capitolo sarà illustrato com'è stato possibile inserire l'applicazione framework all'interno di un'interfaccia Drupal grazie alle specifiche WSRP: problemi e soluzioni.

3.1 I servizi WSRP

Si è già fatto ripetutamente cenno al WSRP (Web Services for Remote Portlets): un protocollo di rete standard utilizzato nella comunicazione tra portali web e portlet remoti. I portlet sono componenti web che offrono servizi all'interno di pagine web. Il codice generato viene integrato con la pagina web che richiede il servizio.

Questa architettura ha il vantaggio di poter creare servizi indipendenti dalla pagina Web: il servizio viene creato una sola volta e pubblicato, mentre i portali web, previa autenticazione, ne richiedono l'utilizzo.

Lo standard WSRP è stato progettato per aggregare i contenuti dei portali in modo standard. I portali che implementano questo protocollo potranno accedere, visualizzare e utilizzare risorse come le Portlet che risiedono su Portali remoti.

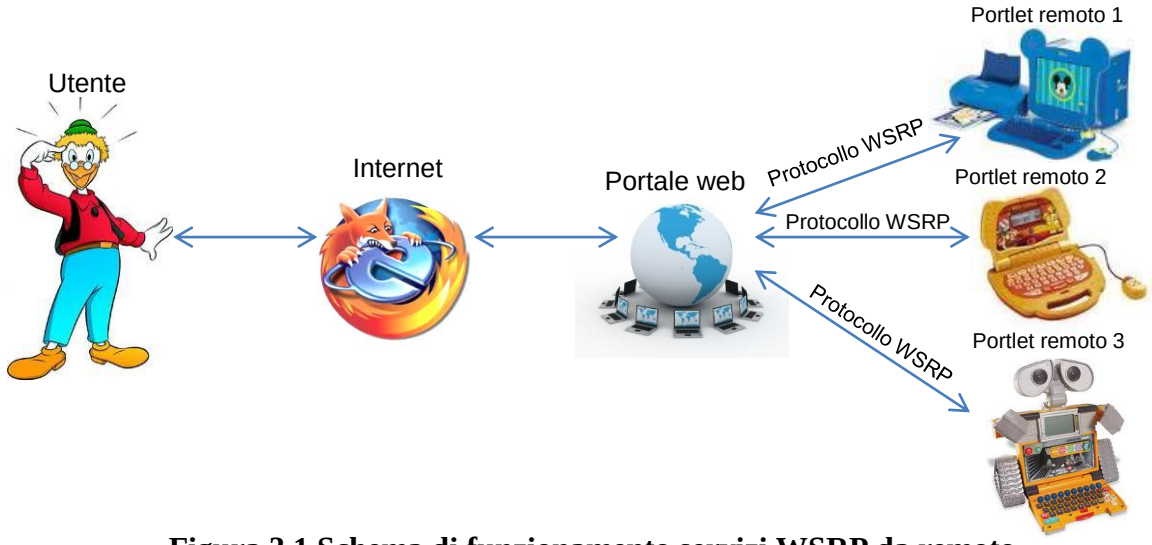

Figura 3.1 Schema di funzionamento servizi WSRP da remoto

Il WSRP mette a disposizione una piattaforma base per avere interoperabilità nella pubblicazione e consumo delle Portlet remote. In particolare definirà un protocollo comune e un insieme di interfacce per i web services presentation-oriented.

Lo scenario sarà composto da:

- Portal server (Producer) che mettono a disposizione Portlet in modalità remota accessibili mediante lo standard WSRP
- Portal server (Consumer) che 'consumeranno' Portlet remote e aggregheranno tali contenuti nel proprio portale.

L'aspetto interessante è che Producer e Consumer potranno essere implementati in piattaforme differenti.[7]

L'applicazione jaMVC costruita per il servizio richiesto, può essere facilmente accessibile direttamente tramite http dove è stato testato in tutte le sue sfaccettature e di cui sono state presentate alcune schermate nei capitoli precedenti. Ma per soddisfare appieno la domanda del CEDI, questo servizio deve essere inserito nel portale di ateneo, che, come è stato già spiegato è stato redatto col CMS Drupal che nativamente non contiene il modulo WSRP.

[7] *Progetto e realizzazione di Servizi Web basati su tecnologia Portlet* – Tesi di Laurea di Roberto Pellegrino. UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA A.A. 2005/06

Questo modulo è stato invece sviluppato dallo stesso Andrea Gariboldi che ha creato il jaMVC, e l'ha lasciato a disposizione della community inserendolo nel portale di Drupal.org: http://Drupal.org/project/wsrp. Occupa uno spazio di appena 90KB e una volta inserito in una valida directory **modules** di Drupal, la configurazione risulta molto semplice: come amministratore del CMS, che in questo caso è il **consumer**, bisogna inserire i nomi dei **producers** nella pagina di configurazione dedicata al modulo WSRP. Quindi, nel caso del nostro servizio, è stato incluso, come producer, l'indirizzo http://mvcstudenti.cce.unipr.it/php/cedi2, utilizzato per la piattaforma di sviluppo del servizio jaMVC.
Dopo questo, è possibile creare nodi e blocchi referenziando il portlet WSRP: per gli operatori Drupal quindi è sufficiente dirigersi sulla gestione dei blocchi per crearvi dei contenuti e lì potranno inserire il nodo sulla base del portlet WSRP.

Nella sezione 3.3 sarà visibile il nodo Drupal utilizzato come prototipo di quella che sarà l'interfaccia definitiva, e messo a disposizione per testare il servizio.

3.1.1 Discordanze

Al primo tentativo di utilizzo del software, questa volta come servizio WSRP, ciò che subito risulta essere evidente è la mancanza dell'intestazione e del piè pagina (*header* e *footer*), ma con un'analisi più approfondita del sistema si capisce che questo ha una sua logica stringente: esistono già un'intestazione e un fine pagina, in quanto, di fatto, ci si trova davanti a delle pagine precostituite con un <html>, un <head> e un <body> e le rispettive chiusure.
Da un lato questo potrebbe non essere un problema, dal momento che il servizio vero e proprio è bene che sia all'interno di queste due sezioni, e infatti per il servizio in sé, se scritto bene, il problema non si pone; dall'altro però è indiscutibile l'utilità che possono avere queste due parti per quello detto nel capitolo precedente. Inoltre, nel footer, ad esclusivo titolo di amministrazione, era stato inserito un utile codice javascript che, insieme alla funzione *dump_vars()* implementata nel pacchetto, consentiva di leggere, a richiesta, il debug del sistema con tutte le variabili presenti nei vari contesti. La scelta di inserire questa parte di

codice nel piè pagina era dovuta proprio alla volontà di non interferire col servizio.
A prova di tutto questo è stata eseguita una eliminazione provvisoria del file *index.php*: la conseguenza è stata che il servizio chiamato direttamente tramite http non era più attivo, mentre sotto il protocollo WSRP non mostrava alcuna variazione.

Un altro problema riscontrato è stato la mancata richiesta di autorizzazioni all'accesso, il cui codice si trova nel file *authx.inc* nella solita cartella **includes**. Ciò è dovuto al fatto che il modulo WSRP contiene già in sé un suo metodo per il controllo d'accesso utente, *authx.wsrp.inc* in una cartella **wsrp,** esterna però al servizio quindi non gestibile se non da un amministratore del portale.

Infine, lo stile, comunque poco curato durante lo sviluppo, è risultato diverso. Specialmente il carattere di default si è modificato con quello previsto dal CSS del tema assegnato a Drupal. Anche questo ha una sua ragione di essere.

3.1.2 Soluzioni

La soluzione trovata per la mancata visualizzazione dell'header e del footer è stata risolta dopo aver spulciato quello che può essere considerato il lifecycle del sistema: il metodo *show()* (si veda pag.40). Sono pertanto state eseguite piccole modifiche al codice, inserendo un paio di funzioni che verificano o meno la presenza del protocollo WSRP e agiscono di conseguenza.

Naturalmente bisogna prima considerare due situazioni differenti, e per ognuna di esse è bene definire due tipi di header e di footer: una per il servizio http, quindi con l'intestazione completa che risponda alle esigenze di una corretta scrittura html; l'altra invece tralasciando la parte di intestazione (head) e inserendo unicamente ciò che si desidera mostrare all'utente, entro il body. Questo anche se le pagine verrebbero comunque rappresentate apparentemente senza errori, tranne poi scoprire che ci si trova con pagine html senza <body> o, al contrario, averne due. Questo è dovuto al fatto che i recenti browser sono spesso a prova di errori, almeno quelli più grossolani.
Quindi, la logica che ha portato alla modifica del metodo show(), è semplicemente quella di lasciare le cose come prima nel caso di servizio diretto tramite http, di

inserire invece diverso header e diverso footer, che, come abbiamo visto nel capito precedente, sono customizzabili dall'amministratore, e che avranno motivo di esistere se il servizio diventa un producer WRSP.

```
//Aggiunta per l'header in Drupal
function head_wsrp()
        {
                if (function_exists('get_wsrp_user'))
                {
                        if (file_exists('view/header.php'))
                                o(get_php_contents('view/header.php'));
                }
        }
//Aggiunta per il footer in Drupal
        function foot_wsrp()
        {
                if (function_exists('get_wsrp_user'))
                {
                        if (file_exists('view/footer.php'))
                                o(get_php_contents('view/footer.php'));
                }
        }
```

Listato 3.1 Metodi aggiunti per la lettura dell'header e del footer

Nel listato precedente sono visibili i metodi delle funzioni che verranno inserite in *show()*. La procedura utilizzata non è certamente molto elegante: si va a vedere se esiste una certa funzione legata al WSRP e si prosegue di conseguenza, ma questo dipende dalle conoscenze effettive che si hanno del *core* del portale, i cui gestori potranno invece trovare una chiave differente.

3.1.3 Gestione autorizzazioni

Si è già vista in precedenza la chiamata al metodo *get_wsrp_user()*. Pur essendo ancora abbastanza oscuro il funzionamento, questa funzione svolge un importante ruolo di collegamento col sistema di autenticazione Shibboleth: senza entrare troppo nel dettaglio, quando ci si autentica nel sito Drupal di ateneo, vengono incorporate in sessione una serie di variabili utili legate all'utente (nome, cognome, numero di matricola, ecc.). Purtroppo il framework, lavorando in

'remoto', non può assumere su di sé queste variabili direttamente, ma *get_wsrp_user()* fa, in qualche modo, da tramite, rigenerando solo alcuni di questi dati e convogliandoli al servizio.

Allo stato attuale *get_wsrp_user()* rilascia solo l'indirizzo email dell'utente registrato, che viene utilizzato come username, costringendo il servizio a recuperare gli altri dati utente in apposito database. Questo lascia pensare a una perdita di tempo per il server, che di fatto deve richiamare gli stessi dati due volte.
Non potendo ovviamente accedere al database centralizzato, per il servizio di test di questo servizio, è stato perciò necessario creare una tabella apposita provvisoria all'interno del database Oracle messo a disposizione.

Il mancato funzionamento, invece, del file *authx.inc*, viene sopperita dal già citato *navigation.php*, che tra le altre cose si occupa dell'accesso utente.
Nel listato 2.16 di pag.40 risulterà ora ancora più chiara la funzionalità nell'uso di *get_wsrp_user()*.

3.2 Il risultato

Dalle schermate successive sarà possibile confrontare i risultati ottenuti con quelli dati dal portale del servizio web diretto, visti nei capitoli precedenti.

Si può immediatamente costatare come il concetto di framework venga rispettato: il servizio, pur 'vivendo' anche autonomamente, ora è agganciato a un portale diverso dall'originario.

Certamente anche dal punto di vista estetico, il servizio, altrimenti piuttosto freddo, ne guadagna qualcosa, senza per altro avere avuto cura di questo aspetto.

Figura 3.2 Pagina senza credenziali d'accesso

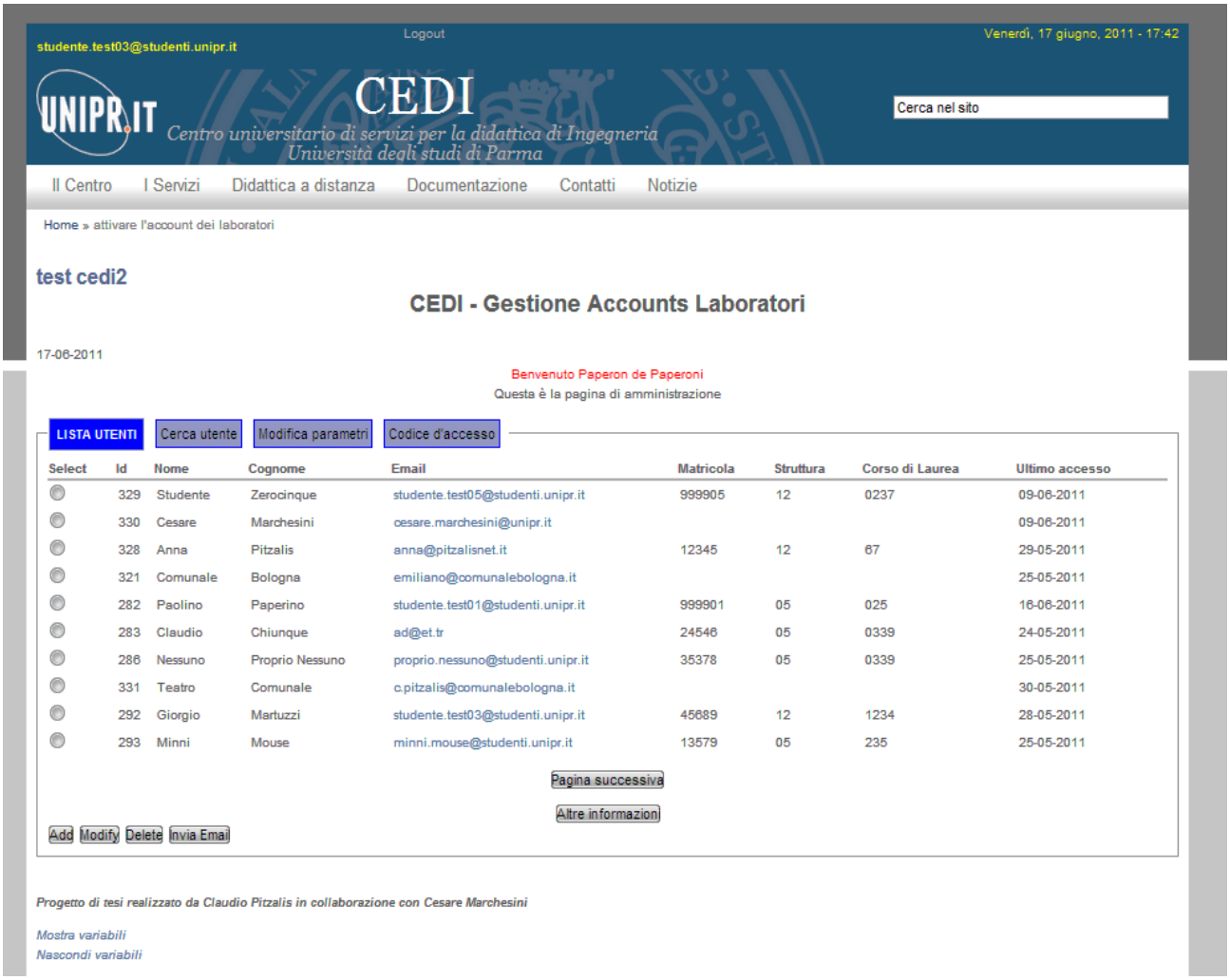

Figura 3.3 Pagina d'accesso dell'amministratore

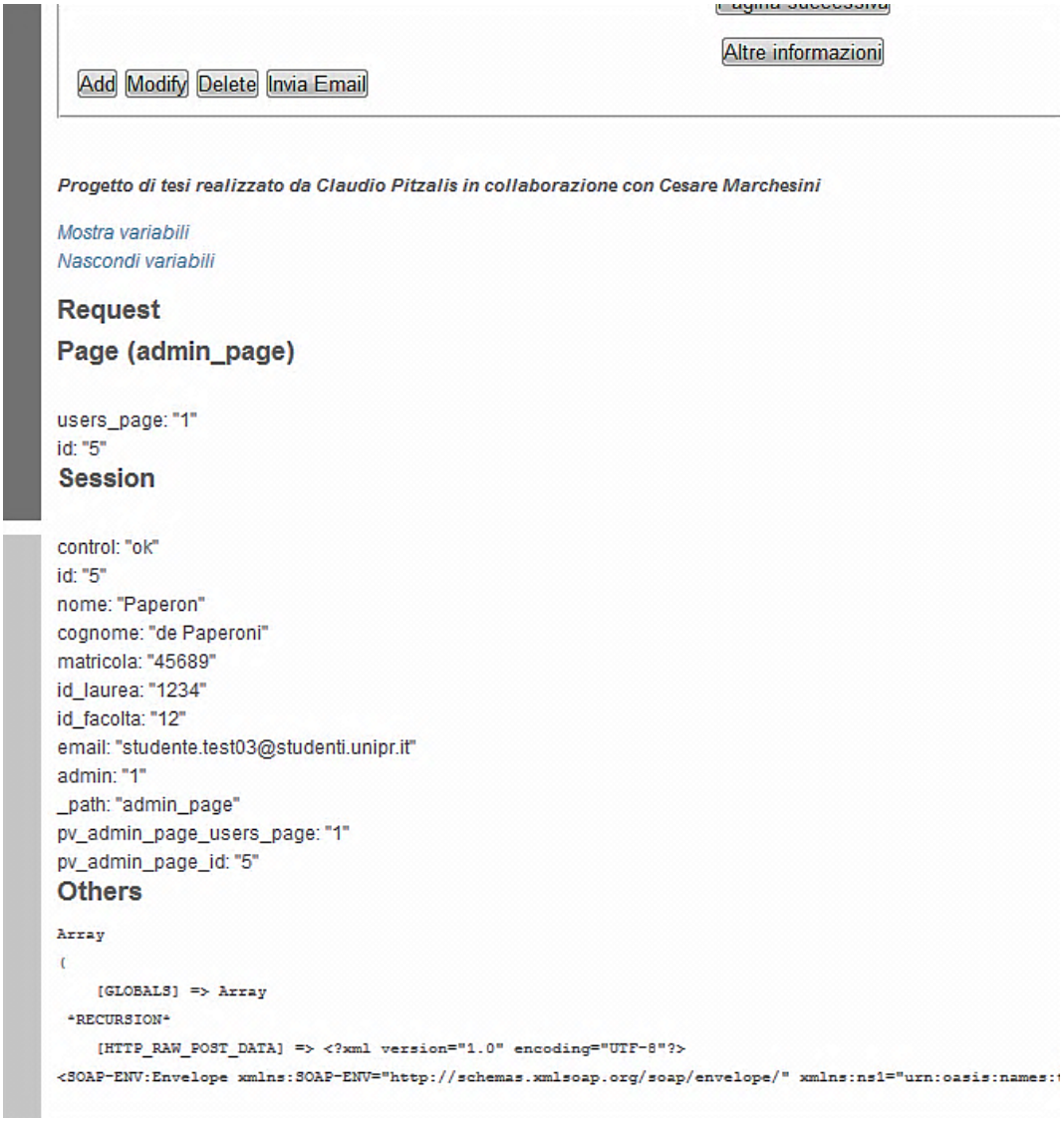

Figura 3.4 Parte del debug visibile all'amministratore

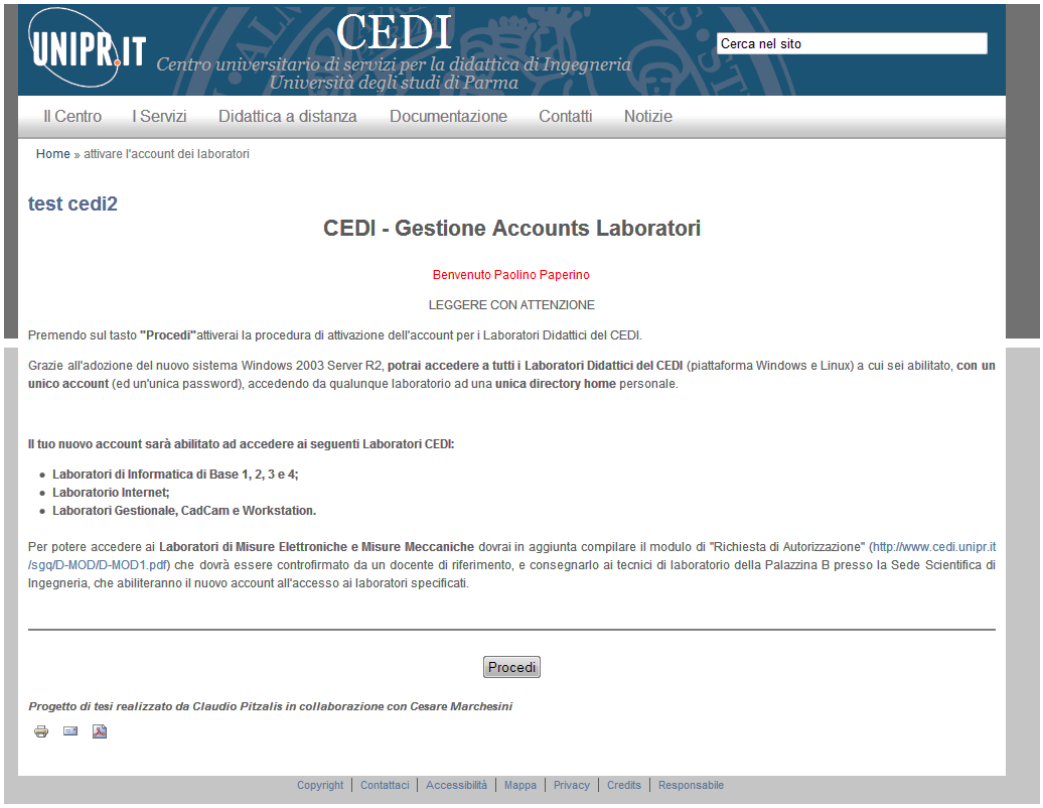

Figura 3.5 La nuova pagina per l'accesso ai laboratori dello studente

Figura 3.6 Nuovo aspetto dell'output nella procedura di attivazione

Conclusioni

Dopo un primo impatto piuttosto difficile, dovuto soprattutto al fatto che il pacchetto software da utilizzare proposto ha una letteratura pressoché nulla, e né forum né newsletter potevano in qualche modo venire in aiuto, è stato possibile proseguire grazie alla curiosità e alla sfida di toccare argomenti nuovi, personalmente non trattati approfonditamente durante gli anni universitari, come il paradigma MVC e le specifiche WSRP, queste ultime piuttosto complesse da esaminare. Nel prosieguo del lavoro, è risultato persino piacevole costruire e ricostruire da zero un servizio con i pochi strumenti a disposizione, o meglio, rintracciando gli strumenti all'interno dello stesso pacchetto, in quanto una lettura attenta del codice, pur non commentato, permette, con l'applicazione, di scoprirne i grandi pregi intrinsechi.

jaMVC è tutt'altro che un programma WYSIWYG (What You See Is What You Get), si potrebbe anzi dire che ne è l'antitesi, ma offre opportunità interessanti ai programmatori, specialmente grazie al supporto WSRP che consente di testare portali "consumer" e "producer" verificandone l'efficacia nell'integrazione, indipendentemente dall'implementazione e dall'architettura.
Questo framework permette dunque lo sviluppo di applicazioni che operano come producer grazie al protocollo WSRP e rendendo agibili le proprie funzionalità grazie a un'interfaccia prestata dal portale.

Per quanto riguarda il refactoring del servizio proposto in questa tesi, che ha la presunzione di voler essere anche un piccolo manuale introduttivo al jaMVC, rilevandone la mancanza, è stato sviluppato molto codice, di cui però in questa sede è stato considerato solo qualche stralcio, ma che promette e premette a ulteriori sviluppi e miglioramenti.

Non è chiaro invece se questo pacchetto continuerà a essere sviluppato e aggiornato dall'autore, così come il modulo wsrp per Drupal, che a oggi risulta valido solo per le versioni 6.xx del CMS e, come è stato possibile testare, non è affatto compatibile con le nuove versioni. Inoltre, come detto ripetutamente, la mancanza di una community unita alla presenza di pochissimi servizi attivi con jaMVC, al momento giocano a sfavore di questa scelta.

Se l'Ateneo decidesse di sostituire il servizio di accesso ai laboratori con l'attuale progetto, dopo le dovute modifiche, o seguire la stessa linea con altri servizi ancora da rifattorizzare, sarebbe altamente consigliato migliorare il passaggio dei dati tra il processo di autenticazione Shibboleth e il framework. Questo per alleggerirne il sistema, in quanto già il CMS Drupal ha, per sua intrinseca costituzione, un continuo accesso al DBMS.

Indipendentemente dagli sviluppi futuri, lo studio di questo modello di programmazione per web, è una valida occasione di sperimentazione, di ampliamento delle conoscenze e di approfondimento, da una parte del paradigma MVC, e, dall'altra, della gestione dei contenuti nei siti a grande dimensione, come è il portale d'Ateneo, dove è necessario tenerne separate le varie parti.

Glossario

APACHE HTTP SERVER, o più comunemente **Apache.** È la piattaforma server Web modulare più diffusa, in grado di operare da sistemi operativi UNIX-Linux e Microsoft.

CINECA, è un consorzio, senza scopo di lucro e con sede a Casalecchio di Reno (BO), formato da 47 Università italiane (dalle 4 originarie), da OGS, CNR e dal Ministero dell'Università e della Ricerca.

CMS, C*ontent Management System*. È uno strumento usato dai webmaster per gestire un sito web evitando di dover programmare dinamicamente l'intero software lato server che gestisce il sito.

FRAMEWORK, letteralmente *Struttura*. Include software di supporto, librerie di codice utilizzabili con uno o più linguaggi di programmazione, un linguaggio per gli script e altri software che possono aiutare a organizzare i vari elementi di un progetto.

DRUPAL, è un CMS modulare scritto in linguaggio PHP e distribuito sotto licenza GNU GPL.

DBMS, *Database Management System*, è un sistema informatico progettato per gestire un Database, ovvero un insieme di numerosi dati strutturati.

J2EE, *Java 2 Enterprise Edition*, è la versione enterprise della piattaforma Java.

JBoss, è un Application Server open source che implementa l'intera suite di servizi Java EE. Essendo basato su Java, JBoss è un Application Server multipiattaforma, utilizzabile su qualsiasi sistema operativo che supporti Java.

LDAP, *Lightweight Directory Access Protocol*, è un protocollo standard per l'interrogazione e la modifica dei servizi di directory.

spiegazione del termine. Testo della spiegazione. Testo della spiegazione.

MVC, *Model-View-Control*, è il nome di un design pattern. È il pattern a cui fa riferimento il framework jaMVC.

PHP, *Hypertext Preprocessor*, preprocessore di ipertesti. È un linguaggio di scripting interpretato, con licenza open source e libera.

PL/SQL, *Procedural Language/Structured Query Language*, è un linguaggio di programmazione proprietario (per database di Oracle Corporation), procedurale, server-based ed estensione dell'SQL.

PORTLET, componente web basato sulla tecnologia Java, gestito da un Portlet Container che esegue richieste da parte dell'utente e genera contenuti dinamici.

PORTALE, un portale Web è un sito web che costituisce un punto di partenza, una porta di ingresso ad un gruppo consistente di risorse di Internet o di una Intranet.

PostgreSQL, è un completo database relazionale ad oggetti rilasciato con licenza libera.

SUBVERSION, noto anche come **SVN**, che è il nome del suo client a riga di comando, è un sistema di controllo versione.

VPN, *Virtual Private Network*, è una rete di telecomunicazioni privata instaurata tra soggetti che utilizzano un sistema di trasmissione pubblico e condiviso come per esempio Internet.

WebDav, *Web-based Distributed Authoring and Versioning*, si riferisce a un set di istruzioni del protocollo HTTP, che permettono all'utente di gestire in modo collaborativo dei file in un server remoto.

Web Service, Secondo la definizione data dal W3C un Web service è un sistema software progettato per supportare l'interoperabilità tra applicazioni diverse in rete.

WSRP, *Web Services for Remote Portlet*, è lo standard che definisce l'interazione tra Producer e Consumer al fine di permettere l'utilizzo da parte del Consumer di Portlet remote.

XML, *Xtensible Markup Languager*, permette la definizione (sintattica) della struttura dei documenti sul web. È lo standard di trasmissione dati tra applicazioni diverse.zione del termine.

Bibliografia

Matt Butcher, *Learning Drupal 6 Module Development*, Packt Publishing Ltd., 2008

E.Gamma, R.Helm, R.Johnson, e J.Vlissides (GoF), *Design Patterns: Element of Reusable Object-Oriented Software* – Addison-Wesley, 1995

Roberto Pellegrino, *Progetto e realizzazione di Servizi Web basati su tecnologia Portlet* – Tesi di Laurea - Università degli Studi di Modena e Reggio Emilia, A.A. 2005/06 .

Claudio Pitzalis, *Valutazione pacchetto software Drupal* - Relazione per Ingegneria del Software A e Reti di Calcolatori A Docente Prof. Agostino Poggi, A.A. 2009/2010

Plurimedia s.r.l., *L'architettura Alpha* – Settembre 2010

John K. VanDyk, *Pro Drupal Development – Second Edition*, Apress, 2008.

Siti internet di riferimento

http://Drupal.org

http://www.fis.unipr.it/dokuwiki/doku.php?id=alessio.cavalieri:php-mvc

http://code.google.com/p/jamvc/

http://www.plurimedia.it/

e…

http://www.wikipedia.org/