Università degli Studi di Padova
Facoltà di Ingegneria

Corso di Laurea in
Ingegneria Informatica

Framework ZEND per sviluppare applicazioni web in PHP

Framework ZEND to develope web application in PHP

Tesi di laurea di Pettenuzzo Giordano
22 Luglio 2013

Relatore : Prof. F. Filira

Anno Accademico 2012/2013

Nella stesura di questa tesi di laurea si è cercato di rispettere il più possibile il filo logico che connette, attraverso un legame del tipo simbiotico, l' argomento secondario in oggetto, ovvero il linguaggio PHP, con il vero obbiettivo della trattazione in esame: il framework ZEND. Infatti, come del resto si potrà vedere più avanti nella discussione, per costruire la struttura del framework ZEND viene utilizzato il PHP come linguaggio madre; perciò si può affermare con estrema serenità che il linguaggio PHP è la base di partenza per comprendere ZEND. Una tale affermazione risulta molto più solida e veritiera se si pensa al fatto che le persone che hanno realizzato il linguaggio PHP sono le stesse, se non una parte, del team che ha realizzato il framework ZEND. Quindi non resta altro che presentare le varie parti che andremo a trattare nel corso di questa relazione.

Per cominciare si trova una breve introduzione su che cosa sia un framework per spiegare, anche ad un utente inesperto, su quale argomento verterà l' intera trattazione, per quale motivo si è giunti all' impiego di questi strumenti e quali necessità soddisfino. In seguito ci si scosta dall' argomento principale per andare ad esporre il linguaggio di programmazione PHP, il quale andrà a costituire i mattoni per realizzare il framework ZEND, facendo riferimento al motovo per cui è nato, al suo svilluppo nel corso degli anni e alle caratteristiche per le quali è considerato uno dei migliori linguaggi per sviluppare applicazioni web. Dal terzo capitolo si comincia ad introdurre il framework ZEND ed il suo funzionamento andando a presentare le varie parti che lo compongono, soffermandosi maggiormente su quelle ritenute più importanti, utilizzando un esempio che viene riportato anche nella guida uffiaciale del framework stesso per migliorarne la spiegazione. Da qui si passa al cuore della tesi vera e propria: la fase pratica. In questa sezione vengono costruite, partendo da zero, due identiche applicazoni web che consentono la medesima operazione, ovvero l' autenticazione dell' utente, con l' unica differenza che mentre una viene realizzata con il linguaggio PHP puro l' altra utilizza il framework ZEND così da mettere a confronto i due modi di lavorare. A questo punto troviamo le conclusioni finali dove troveremo le nostre considerazioni personali, i vantaggi/svantaggi che derivano dall' impiego di questo particolare framework, nonchè i problemi che si possono incorrere nel suo utilizzo. Infine si desidera mettere in evidenza le due appendici nelle quali è presente il codice per intero delle due applicazioni generate così da non destare nessun dubbio in merito ai frammenti di codice già presentati nella sezione pratica a titolo esemplificativo.

Contents

Elenco degli algoritmi|a

List of Figures

1 INTRODUZIONE

Lo scopo di questa tesi è di dimostrare che anche senza una profonda conoscenza del linguaggio di programmazione php è possibile costruire un' applicazione web mediante un framework.

1.1 CHE COS' E' UN FRAMEWORK

Nella produzione del software, il framework è una struttura di supporto su cui un software può essere organizzato e progettato. Alla base di un framework c'è sempre una serie di librerie utilizzabili con uno o più linguaggi di programmazione, spesso corredate da una serie di strumenti di supporto allo sviluppo del software, come ad esempio un IDE, un debugger, o altri strumenti ideati per aumentare la velocità di sviluppo del prodotto finito.

1.1.1 PERCHÈ UN FRAMEWORK

Lo scopo di un framework è di risparmiare allo sviluppatore la riscrittura di codice già steso in precedenza per compiti simili. Questa circostanza si è presentata sempre più spesso man mano che le interfacce utente sono diventate sempre più complesse, o più in generale man mano che è aumentata la quantità di software con funzionalità secondarie simili.

> **ESEMPIO**:
>
> il tipo di interazione con l'utente offerta da un menu a tendina sarà sempre la stessa indipendentemente dall'applicazione cui il menu appartiene (o almeno questo è ciò che l'utente si aspetta); in casi come questo un framework, che permette di aggiungere la funzionalità di una finestra con un menu a tendina con poche righe di codice sorgente a carico del programmatore, o magari permettendogli di disegnare comodamente il tutto in un ambiente di sviluppo, permetterà al programmatore di concentrarsi sulle vere funzionalità dell'applicazione, senza doversi far carico di scrivere codice "di contorno".

Il termine inglese *framework* quindi può essere tradotto come intelaiatura o struttura, che è appunto la sua funzione, a sottolineare che al programmatore rimane solo da creare il contenuto vero e proprio dell' applicazione. Un framework è definito da un insieme di classi astratte e dalle relazioni tra esse, quindi istanziare un framework significa fornire un'implementazione di queste. Invece l'insieme delle classi concrete, proprie del' applicazione specifica, sono definite ereditando quelle astratte del framework assieme alle loro relazioni. In questo modo si ottiene un insieme di classi concrete collegate tra loro.

1.2 INTEGRATED DEVELOPEMENT ENVIROMENT (IDE)

Un IDE *integrated development environment* (ambiente di sviluppo integrato), conosciuto anche come *integrated design environment* o *integrated debugging environment* (rispettivamente ambiente integrato di progettazione e ambiente integrato di debugging), è un software che aiuta i programmatori nello sviluppo del codice. Normalmente consiste in un editor di codice sorgente, un compilatore e/o un interprete, un tool di building automatico, e (solitamente) un debugger. A volte è integrato con un sistema di controllo di versione e con uno o più tool per semplificare la costruzione di una GUI *Graphical User Interface* (Interfaccia grafica per utente). Alcuni IDE, rivolti allo sviluppo di software orientato agli oggetti, comprendono anche un navigatore di classi, un analizzatore di oggetti e un diagramma della gerarchia delle classi. Sebbene siano in uso alcuni IDE multi-linguaggio, come Eclipse, NetBeans e Visual Studio, generalmente gli IDE sono rivolti ad uno specifico linguaggio di programmazione, come Visual Basic o Delphi. IDE

1.2.1 STORIA

I primi IDE sono comparsi alla fine degli anni settanta:

- Interpreti BASIC dei primi personal computer.
- L'ambiente di programmazione Smalltalk.
- La Lisp machine.

Tuttavia questi sistemi comprendevano l'intero sistema operativo, e non permettevano l'esecuzione sulla stessa macchina di software scritto in altri linguaggi, se non riavviando il sistema. Nel 1983 viene commercializzato il primo IDE per Personal Computer che funziona come semplice applicazione, il Turbo Pascal della Borland, basato sul linguaggio Pascal. Dopo di allora sono nati numerosi IDE, inizialmente con interfaccia utente a carattere, poi di tipo grafico.

1.3 LIBRERIE (SOFTWARE)

In Informatica, una libreria è un insieme di funzioni o strutture dati predisposte per essere collegate ad un programma software attraverso un opportuno collegamento. Il collegamento può essere statico o dinamico; nel secondo caso si parla di *dynamic-link library*. Il termine libreria nasce da un'errata traduzione dell'inglese *library* (letteralmente biblioteca), ma ormai è così diffuso nel vocabolario dei professionisti da essere accettato quale esatta traduzione.

1.3.1 SCOPO ED UTILIZZO

Lo scopo delle librerie software è quello di fornire una collezione di entità di base pronte per l'uso ovvero riuso di codice, evitando al programmatore di

dover riscrivere ogni volta le stesse funzioni o strutture dati e facilitando così le operazioni di manutenzione. Questa caratteristica si inserisce quindi nel più vasto contesto del "richiamo di codice" all'interno di programmi e applicazioni ed è presente in quasi tutti i linguaggi. I vantaggi principali derivanti dall'uso di un simile approccio sono i seguenti:

- Si può separare la logica di costruzione di una certa applicazione da quella necessaria alla risoluzione di problemi specifici, quali il calcolo di funzioni matematiche o la gestione di collezioni;
- Le entità definite in una certa libreria possono essere riutilizzate da più applicazioni;
- Si può modificare la libreria separatamente dal programma, senza limiti alla potenziale vastità di funzioni e strutture dati man mano disponibili nel tempo.

1.3.2 LIBRERIE STANDARD

Quasi tutti i linguaggi di programmazione supportano il concetto di libreria e moltissimi includono delle librerie standardizzate (spesso chiamate proprio librerie standard del linguaggio in questione): si tratta di un insieme di funzioni e/o strutture dati che permettono di risolvere i problemi di programmazione più comuni. Ad esempio, molti linguaggi di programmazione hanno una libreria matematica, che consente di eseguire elevamenti a potenza, il calcolo dei logaritmi e così via; funzioni di I/O; funzioni e strutture dati per la gestione di collezioni di oggetti; e altre. Le librerie standard, rispetto a quelle non-standard, consentono una più agevole portabilità degli applicativi che le sfruttano; infatti, ogni produttore di compilatori è tenuto a includere una certa implementazione delle librerie standard; questo significa che le librerie sono potenzialmente supportate da tutte le piattaforme per le quali esiste un compilatore specifico. Viceversa, una libreria non-standard potrebbe non essere supportata su un certo sistema.

2 IL LINGUAGGIO PHP

PHP é un acronimo ricorsivo per *"Hypertext Preprocessor"* (preprocessore di ipertesti che originariamente significava *"Personal Home Page"*) oggi è un linguaggio di scripting interpretato, con licenza open source e libera, originariamente concepito per la programmazione Web ovvero per la realizzazione di pagine web dinamiche. Attualmente è utilizzato principalmente per sviluppare applicazioni web lato server ma può essere usato anche per scrivere script da riga di comando o applicazioni stand-alone con interfaccia grafica. L'elaborazione di codice PHP sul server produce codice HTML da inviare al browser dell'utente che ne fa richiesta. Il vantaggio dell'uso di PHP e degli altri linguaggi Web come ASP e .NET rispetto al classico HTML derivano dalle differenze profonde che sussistono tra Web dinamico e Web statico.

2.1 CENNI STORICI

Nato nel 1994 ad opera del danese Rasmus Lerdorf, PHP era in origine una raccolta di script CGI che permettevano una facile gestione delle pagine personali. Il significato originario dell'acronimo era *Personal Home Page*

> (secondo l'annuncio originale di PHP 1.0 da parte dell'autore sul newsgroup comp.infosystems.www.authoring.cgi).

Il pacchetto originario venne in seguito esteso e riscritto dallo stesso Lerdorf in C, aggiungendo funzionalità quali il supporto al database MySQL e prese a chiamarsi PHP/FI, dove FI sta per *Form Interpreter* (interprete di form), prevedendo la possibilità di integrare il codice PHP nel codice HTML in modo da semplificare la realizzazione di pagine Web dinamiche. In quel periodo, 50.000 domini Internet annunciavano di aver installato PHP. A questo punto il linguaggio cominciò a godere di una certa popolarità tra i progetti open source del web, e venne così notato da due giovani programmatori: Zeev Suraski e Andi Gutmans. I due collaborarono nel 1998 con Lerdorf allo sviluppo della terza versione di PHP (il cui acronimo assunse il significato attuale) riscrivendone il motore che fu battezzato **Zend** da una contrazione dei loro nomi. Le caratteristiche chiave della versione PHP 3.0, frutto del loro lavoro, erano la straordinaria estensibilità, la connettività ai database e il supporto iniziale per il paradigma a oggetti. Verso la fine del 1998 PHP 3.0 era installato su circa il 10% dei server web presenti su Internet e diventò a questo punto talmente maturo da competere con ASP, linguaggio lato server analogo a PHP sviluppato da Microsoft cominciando così ad essere usato su larga scala. La versione 4 di PHP, che fù la prima versione a permattere la programmazione ad oggetti, venne rilasciata nel 2000 e prevedeva notevoli migliorie. Attualmente siamo alla quinta versione, sviluppata da un team di programmatori, che comprende ancora Lerdorf, oltre a Suraski e Gutmans. La popolarità del linguaggio PHP è in costante crescita grazie alla sua flessibilità: nel Giugno 2001, ha superato il milione di siti che lo utilizzano. Nell'ottobre 2002, più del 45% dei server Apache usavano PHP. Nel gennaio 2005 è stato insignito del titolo di ***"Programming Language of 2004"*** dal

TIOBE Programming Community Index, classifica che valuta la popolarità dei linguaggi di programmazione sulla base di informazioni raccolte dai motori di ricerca. Nel 2005 la configurazione LAMP (Linux, Apache, MySQL, PHP) supera il 50% del totale dei server sulla rete mondiale. Nel 2008 PHP 5 è diventata l'unica versione stabile in fase di sviluppo. A partire da PHP 5.3.0, viene implementata una funzione chiamata *"late static binding"* che può essere utilizzata per fare riferimento alla classe chiamata in un contesto di eredità statica. A partire dal 5 febbraio 2008, a causa dell'iniziativa GoPHP5 sostenuta da una serie di sviluppatori PHP, molti dei progetti open-source di alto profilo cessano di supportare PHP 4 nel nuovo codice e promuovono il passaggio da PHP 4 a PHP 5.

2.1.1 PROGRAMMAZIONE LATO SERVER

Nelle reti informatiche, l'espressione lato server (*server-side*) fa riferimento a operazioni compiute dal server in un ambito client-server. Di solito, un server è un programma software, come un server web, che gira su una macchina remota (chiamata appunto "server") rimanendo in ascolto su determinate porte è raggiungibile da un computer client. Alcune operazioni devono essere compiute dal lato server perché richiedono l'accesso a informazioni o funzionalità non disponibili sul client, o richiedono misure di sicurezza che sarebbero inaffidabili se eseguite lato client. Le operazioni lato server includono anche trattamento e immagazzinamento di dati da client a server, perché possano essere disponibili a un gruppo di utenti. Si indica con questo termine il *database management system* DBMS che permette di memorizzare, modificare ed estrarre informazioni da un database al contrario del lato client. Quindi il servizio fornito dal lato server è quello di gestire il database con il server che fa da interfaccia con il lato utente. Quindi questo tipo di programmazione si definisce architettura a tre lati. Nell'ambito della programmazione web, si definiscono linguaggi lato server quei linguaggi di programmazione che vengono interpretati ed elaborati dal server il quale, successivamente, invia i risultati al client (il browser dell'utente). I linguaggi lato server più diffusi sono appunto PHP e ASP. Un programma scritto con questo tipo di linguaggi viene sempre elaborato sul server e mai reso disponibile all'utente, il quale può visualizzare solo il risultato del programma. Questo concetto è molto importante in quanto sta alla base della sicurezza e dell'affidabilità offerti dalla programmazione lato server. Ci sono operazioni per le quali PHP, e quindi un linguaggio di programmazione lato server, non è adatto, per citarne uno:

> **ESEMPIO:**
>
> il refresh del menù a tendina. In questo caso è molto meglio utilizzare un linguaggio come AJAX.

Perciò in una stessa pagina HTML possiamo avere parti di codice scritte in javascript, in PHP e in AJAX i quali si integrano benissimo.

2.1.2 PROGRAMMAZIONE LATO CLIENT

Nell'ambito delle reti di calcolatori, il termine lato client (client-side) indica le operazioni effettuate da un client in un rapporto client-server. Un rapporto di questo tipo è quello effettuato da un'applicazione, come un Browser, che avvia una connessione ad un server per poter funzionare. Si effettuano operazioni in modo client-side perchè richiedono risorse che non sono disponibili sul server ma che lo sono sul client, oppure perché al server manca la potenza per poter far funzionare lo script su tutti i client. Inoltre se non serve che i risultati siano salvati sul server, le operazioni risulteranno nettamente più veloci (ovviamente a seconda della potenza del client). Esistono vari programmi client, ad esempio i browser moderni che supportano sia il protocollo HTTP che l'FTP, che rendono uno script compatibile con più PC. Programmi che girano sul computer locale, senza mai inviare o ricevere dati in rete, non sono considerati programmi client, e così le operazioni di tali programmi non vengono considerate operazioni client-side.

2.2 CATTERISTICHE

PHP riprende per molti versi la sintassi del C, come peraltro fanno molti linguaggi moderni, e del Perl. È un linguaggio a tipizzazione debole (quindi nella dichiarazione delle variabili, che non è necassaria, non serve indicarne il tipo) e dalla versione 5 migliora ancora il supporto al paradigma di programmazione ad oggetti. Certi costrutti derivati dal C, come gli operatori fra bit e la gestione di stringhe come array, permettono in alcuni casi di agire a basso livello; tuttavia è fondamentalmente un linguaggio di alto livello, caratteristica questa rafforzata dall'esistenza delle sue moltissime API, *Application Programming interface*, oltre 3.000 funzioni del nucleo base. PHP è in grado di interfacciarsi a innumerevoli database tra cui MySQL, PostgreSQL, Oracle, Firebird, IBM DB2, Microsoft SQL Server, solo per citarne alcuni, e supporta numerose tecnologie, come XML, SOAP, IMAP, FTP, CORBA. Si integra anche con altri linguaggi/piattaforme quali Java e .NET e si può dire che esista un wrapper per ogni libreria esistente, come CURL, GD, Gettext, GMP, Ming, OpenSSL ed altro. Fornisce un'API specifica per interagire con Apache, nonostante funzioni naturalmente con numerosi altri server web. È anche ottimamente integrato con il database MySQL, per il quale possiede più di una API. Per questo motivo esiste un'enorme quantità di script e librerie in PHP, disponibili liberamente su Internet. La versione 5, comunque, integra al suo interno un piccolo database embedded, SQLite. Dispone di un archivio chiamato PEAR che mette a disposizione un framework di librerie riusabili per lo sviluppo di applicazioni PHP e di PECL che raccoglie tutte le estensioni conosciute scritte in C. PHP non ha ancora un supporto nativo per le stringhe Unicode o multibyte in quanto il supporto Unicode è in fase di sviluppo per una futura versione di PHP e consentirà di usare caratteri non ASCII in stringhe e nomi di funzioni, classi e metodi.

2.3 SICUREZZA

La percentuale di software non sicuro scritto in PHP, sul totale di tutte le falle nei software elencate dal *Common Vulnerabilities and Exposures*, ammontava al: 12% nel 2003, 20% nel 2004, 28% nel 2005, 43% nel 2006, 36% nel 2007, 34.8% nel 2008, 29.9% nel 2009 e 27.2% nel 2010. La maggior parte di questi punti vulnerabili possono essere sfruttati tramite remoto, ovvero senza accedere al computer che ospita l'applicazione vulnerabile. Le falle più comuni sono dovute al mancato adempimento delle best practice nella programmazione e da vulnerabilità presenti in codice scritto in versioni vecchie di PHP.

2.4 GESTIONE DEI PARAMETRI

Esistono diversi metodi per il passaggio dei dati fra le pagine, il php mette a disposizione i cookie, le sessioni, il metodo get e post utilizzando tre array di variabili: $_GET, $_POST e $_SESSION nativi di HTML. I cookie e le sessioni sono spesso utilizzati per gestire le informazioni del navigatore e le statistiche, ed hanno il grande difetto di non poter essere sfruttate se le impostazioni di navigazione del browser non consentono di scaricare i cookie nella memoria temporanea (salvo inserire scorciatoie particolari). I metodi GET e POST sono sempre disponibili, e pertanto sono da privilegiare quando possibile. Il primo tipo di parametro viene passato tramite la stringa che compare nella barra dell'indirizzo del browser; il secondo viene passato in background. Qui di seguito vedremo un pò più in dettaglio come vengono utilizzati questi due metodi di referenziazione delle pagine.

2.4.1 IL METODO GET

Il metodo get prevede che i nomi ed i valori da associare siano posti direttamente sulla barra di navigazione, nell'indirizzo URL della pagina richiamata. Tale procedura consente al navigatore e a chi può accedere al pc, di leggere il nome della variabile ed il suo valore, pertanto se i dati contengono informazioni personali necessiteranno di un'eventuale **criptazione**. La sintassi per utilizzare questo metodo è la seguente:

Algorithm 1 Query string 1

```
<a href="nome_pagina.php?" nome=valore>
```

Come vediamo dall'esempio è semplicissimo, basta ricordarsi del punto interrogativo e non sbagliare il nome o il valore della variabile. Se vogliamo inserire più nomi e valori, dobbiamo solo inserire il carattere & (e commerciale, che significa AND) fra le coppie di parametri. Come é mostrato qui di seguito

Algorithm 2 Query String 2

```
<a href="nome_pagina.php?" nome=valore & nome2=valore2 & nome3=valore3....>
```

Dopo aver inserito una di queste due righe (che per convenzione si chiamano query string) abbiamo a disposizione un array globale di nome $_GET che contiene tutti i parametri che abbiamo inserito nella query string ed è visibile nell'URL della nuova pagina. Da adesso nella nuova pagina che si è aperta abbiamo a disposizione tutte le variabili che contiene l'array:

Algorithm 3 Contenuto di $_GET

```
$_GET['nome'] ; //che equivale a 'valore'
$_GET['nome2'] ; //che equivale a 'valore2'
$_GET['nome3'] ; //che equivale a 'valore3'
```

2.4.2 IL METODO POST

Il metodo post si usa con i form e crea un array globale che si chiama $_POST. L'array generato conterrà delle chiavi che portano il nome (name="") delle caselle del form ed i valori saranno quelli che l'utente avrà inserito o cliccato (inseriti dal webmaster per i campi hidden).

Il metodo POST si differenzia dal metodo GET perché non è visibile nella barra degli indirizzi (anche se le variabili sono rintracciabili nel codice della pagina web), e perché con il metodo POST possiamo spedire non soltanto semplici variabili ma anche file, immagini ed altro materiale più consistente. La sintassi per utilizzare questo metodo è la seguente:

Algorithm 4 Instanziazione di $_POST

```
<form action="prova.php" method="post">
<input type="text" name="nome1">
<input type="checkbox" name="nome2" value="si">
<input type="submit" name="submit" value="invia">
</form>
```

Da questo momento abbiamo a disposizione l'array $_POST che sarà disponibile anche nella nuova pagina referenziata (ricordando che se una variabile non viene iserita, questa non sarà definita). Dopo aver spedito il form ed aver creato l'array globale $_POST nella pagina appena aperta è sufficente ricavare le variabili risalendo al nome della stessa contenuto nell'array per poter utilizzarne il contenuto come nel caso mostrato qui di seguito:

Algorithm 5 Contenuto di $_POST

```
$tuavariabile=$_POST['tuavariabile'] ;
```

2.5 PHP.INI

Il file di configurazione di PHP, chiamato php3.ini in PHP 3, e poi semplicemente php.ini nelle versioni successive, è letto all'avvio dell'interprete del linguaggio e fornisce le impostazioni dei vari moduli con cui l'interprete è stato compilato. Nella versione server modulare di PHP, questo avviene solo una volta, all'avvio del server web. Per le versioni CGI e CLI, invece, è invocato ad ogni richiesta. Per visualizzare tutte le opzioni di configurazione è possibile utilizzare la funzione phpinfo().

3 IL FRAMEWORK ZEND

3.1 CHE COS' E'

Gli strumenti nati per supportare il lavoro del programmatore sono sempre più diffusi. È una tendenza che si è manifestata anche nel mondo di PHP infatti nel giro di pochi anni sono nati diversi framework con l'obiettivo di aiutare gli sviluppatori e di rendere le applicazioni web più robuste, sicure e più veloci da sviluppare ed estendere. Zend Framework rientra in questa categoria.

Zend Framework (al quale ci riferiremo nel seguito di questa trattazione con la semplice abbreviazione di "ZF") è un *web application framework* open source creato per semplificare e rendere più efficace sotto ogni punto di vista la produzione di applicativi e servizi web PHP-based. Come vedremo, ZF contiene un insieme di componenti riutilizzabili che rispondono nella sostanza a tutti i requisiti richiesti per un completo applicativo web-based: questi componenti sono scritti interamente in PHP seguendo le migliori pratiche della programmazione ad oggetti.

Vi è ad esempio un robusto componente Model-View-Controller (MVC) per strutturare l'applicativo secondo le specifiche dettate dall'omonimo design pattern, uno per la gestione della ricerca, uno per l'autenticazione, uno per la gestione della localizzazione, un altro per la generazione dinamica di PDF, e molto altro ancora. Proprio a causa di questa sua caratteristica "modulare" è usuale riferirsi a questo framework come ad una "libreria di componenti".

Ciò che differenzia ZF da altri framework PHP è la sua particolare struttura "a compartimenti stagni": l'indipendenza delle sue componenti permette di utilizzare solo il necessario per il proprio progetto, ma allo stesso tempo, se utilizzate insieme, esse si integrano in un substrato incredibilmente potente e semplice da imparare, indirizzando il lavoro dello sviluppatore verso soluzioni basate sui principi di riutilizzo del codice, di estendibilità, leggibilità e manutenzione.

ZF è rilasciato sotto licenza BSD: questo permette al framework di essere inserito nella maggior parte dei progetti, siano questi open source o proprietari, con le minori restrizioni possibili per gli utilizzatori.

Prima di iniziare lo studio del framework è di vitale importanza capire che ZF non è la soluzione a tutti i nostri problemi di sviluppo, e soprattutto non è l'unica alternativa: ogni progetto ha i suoi requisiti e le sue caratteristiche, ed è solo dopo un attendo studio di questi, anche in relazione alle alternative proposte, che possiamo decidere l'architettura del nostro applicativo.

3.2 I VANTAGGI DI ZEND FRAMEWORK

Se la nostra decisione ricade nell'utilizzare un framework esistente allora:

> **Perché dovremmo decidere a favore di ZF rispetto ad altri framework PHP?**

Per rispondere a questa domanda dobbiamo conoscere le sue caratteristiche.

ZF si basa su alcuni concetti chiave che analizzeremo uno ad uno:

- Rapidità e semplicità di sviluppo
- Design moderno
- Facilità di apprendimento
- Documentazione completa
- Community driven

Ora cercheremo di vedere meglio questi concetti chiave che rendono ZF un così valido strumento a disposizione dei programmatori.

3.2.1 RAPIDITÀ E SEMPLICITÀ DI SVILUPPO

Zend Framework velocizza l'estensione o l'aggiunta di nuove funzionalità per i nostri progetti web. La rapidità diventa evidente quando dobbiamo implementare progetti che hanno bisogno di funzionalità aggiuntive: generazione dinamica di pdf, l'invio di mail, la gestione dei form e del multilingua. Tutto questo codice è già stato scritto e testato per noi, dunque tutto ciò che dobbiamo fare è scrivere esclusivamente il codice necessario per il nostro applicativo, includendo lì dove è necessario le classi messe a disposizione dal framework.

Ogni componente del framework ha inoltre una configurazione di default che lo rende utilizzabile fin da subito scrivendo pochissime righe di codice. Nella pratica questo si traduce in un grande risparmio di tempo: non è necessario impostare molte direttive di configurazione per ogni componente prima di poterlo integrare ed usare all'interno del nostro applicativo.

3.2.2 DESIGN MODERNO

Il successo di ZF è strettamente legato ad un lavoro attento e curato da parte del team di sviluppo, che ha utilizzato le più moderne e performanti tecniche di sviluppo software.

ZF è scritto interamente in PHP5 seguendo le tecniche dell'Object-Oriented Programming ormai consolidate da anni e sfruttando moderne tecniche di design del software, inoltre fa un uso massiccio dei design pattern per permettere la massima flessibilità agli sviluppatori alleggerendo di molto il loro lavoro. Il framework sfrutta appieno le caratteristiche positive di PHP, colmando i punti deboli di questo linguaggio con un'architettura e una struttura solida e davvero flessibile.

Ogni componente all'interno del framework ha pochissime dipendenze con ogni altro componente il che implica che ZF è estremamente flessibile; perciò lo sviluppatore è libero di utilizzare solo le parti del framework che sono strettamente necessarie al suo progetto.

3.2.3 FACILITÀ DI APPRENDIMENTO

La modularità e l'indipendenza dei componenti hanno un altro vantaggio per nulla trascurabile: potendoli studiare separatamente, la curva di apprendimento di ZF è decisamente leggera. Inoltre il design di ogni componente è tale per cui non siamo obbligati a capirne fin da subito l'intero funzionamento prima di poterlo utilizzare. Potendo procedere a piccoli passi e avendo un'idea generale del suo funzionamento possiamo, una volta che le nostre conoscenze ce lo concedono, permetterci di approfondire la nostra esperienza attraverso un procedimento relativamente semplice.

La separazione delle diverse responsabilità facilita e velocizza anche la ricerca e la riparazione dei bug che inevitabilmente sono presenti nelle nostre applicazioni: è più facile trovare ciò che stiamo cercando, perché possiamo delimitare già in partenza l'area in cui si presenta il problema.

3.2.4 DOCUMENTAZIONE COMPLETA

Può sembrare un elemento di poco conto, ma se ci fermiamo a riflettere sulla quantità di tempo che perdiamo quando utilizziamo strumenti poco documentati risulta chiaro il contrario. Per questo motivo fin da subito il team di sviluppo ha documentato in maniera attenta e completa tutto il codice del framework.

Zend Framework è accompagnato da due tipi di documentazione: quella per l'utente finale e quella dedicata alle API (*Application Program Interface*) per gli sviluppatori ZF. La prima è una raccolta ad alto livello delle caratteristiche principali del framework, mentre la seconda è generata utilizzando phpDocumenter e viene automaticamente aggiornata utilizzando uno speciale blocco (DocBlock) nel codice sorgente, che tra le altre cose può essere inserito in modalità di autocompletamento se sviluppiamo con un IDE tipo Eclipse (con il plugin PDT – PHP Develpment Tool) o Zend Studio, invogliando così la scrittura di codice ben documentato.

È inoltre possibile includere il manuale di ZF all'interno dell'ambiente di sviluppo, in modo da godere di tutte le informazioni disponibili durante la fase di implementazione.

3.2.5 COMMUNITY DRIVEN

La costante crescita della community di Zend Framework contribuisce di molto alla diffusione di questa tecnologia: essa rappresenta la fonte principale di ispirazione per lo studio di nuove funzionalità o per migliorare quelle esistenti, giocando dunque un ruolo di primo piano in questo contesto. I tutorial e le guide a disposizione sono in gran parte dedicati alla installazione e configurazione iniziale, essendo questo uno degli argomenti più discussi normalmente quando si lavora con i framework.

Il sito del progetto non è solo il punto di riferimento per la documentazione e per i download dei file, ma dispone anche di uno strumento validissimo per il ticketing e il tracking dei bug che gli utenti riscontrano durante lo sviluppo. Per

poter avere accesso a tutto questo è sufficiente registrarsi al sito per accedere a tale sistema e contribuire al miglioramento e allo sviluppo della piattaforma.

3.3 LA STRUTTURA DI ZEND FRAMEWORK

Lo ZF è composto da un' insieme di librerie, interamente sviluppate in PHP5, con una vasta gamma di funzionalità che permette di ottenere grandi risultati con una quantità limitata di codice. Tutti (o quasi tutti) i componenti di questa enorme libreria possono essere utilizzati scrivendo codice procedurale o semplicemente estendendo le classi messe a disposizione. Una volta installato il framework, viene aggiornata la *include_path,* come viene spiegato in questa stessa relazione nel paragrafo 3.5*,* in modo tale da poter includere le librerie da qualsiasi cartella del nostro sito. Quindi dopo aver copiato il pacchetto con tutta la libreria all' interno del nostro progetto contenente la cartella Zend con all'interno tutti i file contenenti le classi da cui è composto lo ZF, facendo attenzione ad aggiungvi il path come si vedrà in seguito nel peragrafo 4.3, sarà possibile utilizzare tutte le sue componenti semplicemente includendo in ogni pagina i file con le classi di cui si ha bisogno.

Per facilitare il problema dell' inclusione dei file giusti, la costruzione del framework stesso ci aiuta, infatti possiamo notare l' organizzazione dei file dentro la cartella Zend prendendo in considerazione una classe arbitraria come ad esempio Zend_Feed:

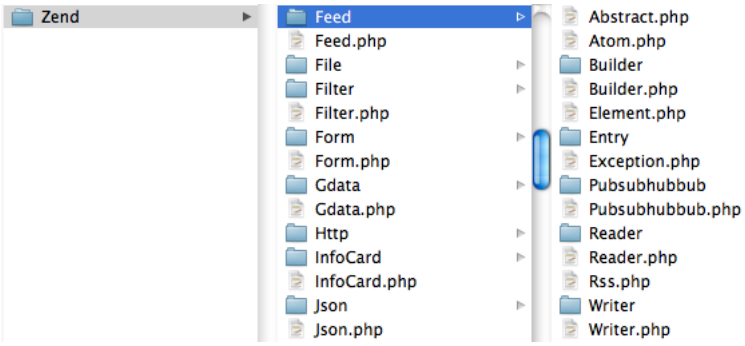

Figure 1: Struttura della libreria

Quindi se avessimo bisogno di utilizzare la classe per la gestione dei Feed RSS, dovremmo impiegare la classe è Zend_Feed_Rss. Per fare questo è necessari prima includere il file corretto semplicemente sostituendo gli underscore (_) con gli slash (/) come nell' esempio sottostante:

Algorithm 6 Utilizzo della libreria

```php
<?php
require 'Zend/Feed/Rss.php';
$channel = new Zend_Feed_Rss("http://www.firecode.it/blog/feed");
```

Anche se sarebbe bello riuscire a scrivere un paragrafo per ogni classe di questo enorme framework, mostrando le loro funzionalità partendo da un codice procedurale, per ora ci limiteremo a mostrarvi tutti i componenti raggruppati per utilizzo.

Model-View-Controller (MVC): Tutte le classi per sviluppare i vostri applicativi seguendo il pattern MVC.

- Zend_Application, Zend_Application_Bootstrap, Zend_Application_Module, Zend_Application_Resource
- Zend_Controller_Front, Zend_Controller_Action, Zend_Controller_Dispatcher, Zend_Controller_Plugin, Zend_Controller_Router
- Zend_Form Zend_Layout, Zend_View, Zend_View_Filter, Zend_View_Helper

Database: Tutto ciò che serve per interrogare il vostro database.

- Zend_Db
- Zend_Db_Adapter
- Zend_Db_Profiler
- Zend_Db_Select
- Zend_Db_Table

Internazionalizzazione L'ideale per la gestione di siti in multilingue che hanno bisogno di conversioni di date, misure, valuta o semplice traduzione.

- Zend_Currency
- Zend_Date
- Zend_Locale
- Zend_Measure
- Zend_Translate

Autenticazione, Autorizzazione e gestione Sessioni Un set completo di classi per gestire al maglio l'autenticazione dei vostri utenti, le autorizzazioni (o privilegi) che ognuno di loro possiede e il salvataggio dei dati attraverso le sessioni

- Zend_Acl
- Zend_Auth
- Zend_Session

Mail, Formati e Ricerche Addio al comando mail(); e ora la creazione di PDF non è più un problema

- Zend_Json
- Zend_Mail, Zend_Mime
- Zend_Pdf
- Zend_Search_Lucene

Core consentono di aggiungere potenzialità e prestazioni alle vostre applicazioni web

- Zend_Cache
- Zend_Config
- Zend_Console_Getopt
- Zend_Debug
- Zend_Filter
- Zend_Loader,
- Zend_Loader_Autoloader
- Zend_Log
- Zend_Memory
- Zend_Registry
- Zend_Validate
- Zend_Version

La presentazione di questa sequenza di componenti può sembrare inutile aprima vista, ma a noi serve per poter trarre le nostre considerazioni riguardo ai vantaggi/svantaggi che presenta l' utilizzo di una siffatta libreria. Infatti da quello che abbiamo potuto osservare fino a questo momento i nomi di tutte le componenti hanno la stessa struttura, del tipo:

Zend_NomeComponente_NomeSottocomponente_Ecc

Da questo possiamo capire che la struttura della libreria è di tipo posizionale cioè, per usare un particolare pacchetto, dobbiamo referenziarlo attraverso la sua posizione assoluta all' interno della libreria stessa. Perciò il fatto che tutte le componenti inizino con la clausola **Zend_** non è una abile scelta del settore marketing, ma significa che tutte le componenti che compongono il Framework Zend si trovano all' interno di quella specifica directory. Quindi per fare un esempio pratico prendiamo in esame la seguente:

Zend_Loader_Autoloader

La precedente dicitura indica semplicemente che la componente Autoloader si trova all' interno della sottodirectory Loader che a sua volta si trova all' interno della directory Zend. Questa inquadratura così rigorosa nella struttura delle librerie, anche se a prima vista può sembrare svantaggiosa, porta con se almeno 3 vantaggi:

1. Una volta afferrato il meccanismo di funzionamento si riesce agilmente ad usare qualiasi componente si voglia con estrema facilità in quanto sia per l' inclusione che per l' impiego si applica la setessa procedura (salvo qualche caso particolare vedi ad esempio Zend_Form nel paragrafo 3.11).

2. Dato che i componenti non sono collegati tra di loro e sono raggruppati in base alla funzione che svolgono, l' impiego di uno non impone necessariamente l' utilizzo di un' altro. Questo potrebbe essere il caso di Zend_Form e Zend_Validate. Infatti se realizzassimo i form della nostra applicazione con Zend_Form nulla ci vieterebbe di utilizzare un altro linguaggio, ovviamente compatibile con PHP, per effettuare i controlli sugli input. Anche se questa non è una buona soluzione data l' efficienza che questi due componenti relaizzano insieme è comunque una via percorribile.

3. L' innovazione più importante portata dall' incorrelazione tra le varie componenti è, oltre a quella già vista nel punto precedente, la possibilità di aggiungere funzionalità ad un sito già esistente (ad esempio come la getione di pdf) utilizzando Zend_Pdf anche se lo stesso sito non è realizzato con ZF. Questa opzione, che non è possibile effettuare con nessun altro framework disponibile oggigiorno in commercio, è la vera forza data dalla particolare architettura che costituisce questo potentissimo strumento che è lo Zend Framework.

Infine per dare un senso di omogeneita a tutto l' insieme troviamo la stessa sintassi, usata per le componenti, anche all' interno dell MVC, in particolar modo quando parliamo di Controller ed di Action. Quindi se osserviamo sulla barra del browser quando avviamo un progetto sviluppato con ZF, anche a livello di *testing*, nella linea di inserimento ad un certo punto durante l' esecuzione osserveremo una dicitura del tipo:

www.dominio.com/public/controller/action

Ovviamente quest' esempio è solamente un preambolo del comando dettagliato che verrà riproposto nel paragrafo 3.7, ma già adesso è possibile capire la procedura usata nell' esecuzione di un' applicazione sviluppata con ZF. Infatti, come verrà spiegato meglio nella sezione 4.3, tutti i progetti realizzati con Zend nascono nella directory *public*, che contiene il file *index.php*; dopodichè la gestione di tutto viene spostata ai vari *Controller* che attiveranno di volta in volta le *Action* necessarie per portare a termine le richieste dell' utente perchè, come verrà detto in seguito, in ZF non ci si sposta da un file all' altro come accade con altri linguaggi, bensì ci si sposta da *Controller* a *Controller* e da *Action* ad *Action*.

A questo punto possiamo dire di aver trattato a sufficienza questo argomento analizzando tutti i punti di interesse ai fini di comprendere com' è costituito e in che modo funziona il framework Zend per cui possiamo procedere oltre con l' esposizione della nostra tesi.

3.4 DA CHE COSA E' COMPOSTO

Nonostante sia una tecnologia relativamente recente, ZF è diventato rapidamente di fatto lo standard per lo sviluppo di applicativi enterprise in PHP. Se la maggior parte dei framework impone una specifica configurazione e l'inserimento nei nostri applicativi anche di quelle parti non strettamente necessarie, ZF si basa sul concetto per cui ogni componente può essere utilizzato indipendentemente dal resto del framework.

Questo approccio permette di concentrarsi sulla soluzione di specifici problemi di sviluppo del nostro progetto senza richiedere lo studio completo di un framework. Proprio per questi motivi ZF può essere impiegato sia in progetti PHP esistenti per introdurre, estendere o aggiornare specifiche funzionalità, sia in applicazioni enterprise di notevole complessità. In altre parole l'indipendenza dei componenti di ZF favorisce la loro integrazione e il loro utilizzo all'interno di altre librerie o framework.

ZF è costituito da una libreria di componenti che contiene tutto ciò di cui si ha bisogno per sviluppare la propria applicazione web. I componenti possono essere raggruppati in sei macro categorie:

- Core
- MVC
- Autenticazione ed Accesso
- Internationalization
- Servizi Web
- Interapplication communication

Le vedremo in dettaglio nelle prossime sezioni. Iniziando dalla componente Core e dall' innovativa componente MVC.

3.5 ZEND FRAMEWORK E L' APPOSITO TOOL DA RIGA DI COMANDO

Anche se attualmente è disponibile la versione 2.0 di Zend Framework, già dalla 1.8.0 è stato realizzato il componente **Zend_Tool** che permette di generare un progetto base, e non solo questo, lanciando un semplice comando da terminale. Premettendo che questo tutorial è scritto per gli utenti OS X, possiamo procedere utilizzando come SVN locale **MAMP** (Mac Apache MySQL PHP) disponibile gratuitamente presso *mamp.info* così come Zend Framework ha in dotazione Zend_Tool. Una volta che abbiamo installato correttamente MAMP, senza impostare nessuna particolare configurazione, possiamo procedere al download di Zend Framework posizionandolo ovunque si voglia. In realtà se recuperiamo la nostra copia via SVN risulta più semplice effettuarne l'aggiornamento in un secondo momento. Per fare questo, una volta aperto il terminale, dobbiamo seguire i seguenti comandi:

```
cd /Applications/MAMP/
mkdir svn
cd svn
mkdir zendframework
cd zendframework
svn checkout http://framework.zend.com/svn/framework/standard/trunk
```

Figure 2: Installazione di Zend Framework

Ora si deve aprire il file php.ini situato in /Applications/MAMP/conf/php5/php.ini e cercare la voce include_path (attorno alla riga 411 nella nostra versione) e aggiungere la posizione dello Zend Framework appena installato:

```
include_path = ".:/usr/lib/php:/usr/local/lib/php:/Applications/MAMP/
svn/zendframework/trunk/library"
```

Figure 3: Aggiunta dello ZF

A questo punto dobbiamo assicurarci di includere la directory di libreria altrimenti quando andremo ad operare con Zend Framework si otterrà soltanto una pagina vuota. Zend Framework è disponibile anche con uno script per la shell di comando che ha il compito di aiutare l' utente con il RAD (Rapid Application Deployment), inoltre è possibile creare anche un collegamento per Zend_Tool da terminale con l' aggiunta di un alias nel proprio profilo in /etc/profile come si può osservare:

```
alias zf=/Applications/MAMP/svn/zendframework/trunk/bin/zf.sh
```

Figure 4: Creazione dell' alias

Per aggiungere in modo per manente un alias nella "memoria" dobbiamo aggiungere soltanto un' altra istruzione dalla linea di comando che viene presentata qui sotto:

```
vi ~/.bash_profile
```

Figure 5: Salvataggio dell' alias

Una volta che abbiamo avuto accesso alla procedura dobbiamo, per poter salvare l' alias, entrare nella modalità "edit" premendo il tasto "A", così da incollare il nostro alias nella prima posizione disponibile spostandosi verso il basso con la freccietta della tastiera; quindi premere il tasto ESC digitando "WA" (tutti) e "Q" (quit) che salverà nel profilo della shell il comando di scelta rapida che sarà disponibile anche dopo la procedura di riavvio. Infine è necessario riavviare sia il teminale che l' SVN per rendere effettive le modifiche appena apportate.

Per verificare se Zend_Tool è stato correttamente installato proviamo a visualizzare la vesrione dello Zend Framework che è stato precedentemente installato con il seguente comando:

```
zf show version
```

Figure 6: Primo comando di zf

Se tutto è andato a buon fine, quello che si dovrebbe visualizzare nella shell di comando sarà una cosa del genere:

```
Zend Framework Version: 1.11.0dev
```

Figure 7: Versione di Zend Framework installata

Finora in questo paragrafo ci siamo occupati solamente dell' installazione e della configuarazione di Zend_Tool, ma adesso dobbiamo anche capire per quale motivo è stato realizzato, come funziona ed infine verificare un' eventuale utilità del suo impiego nella realizzazione di una web application con Zend Framework. Come già accenato in precedenza con questo strumento è possibile creare un progetto base (che nel gergo dei programmatori viene definito come "*Hello World*") semplicemente inserendo una banalissima istruzione dalla linea di comando vediamo come:

```
zf create project test
```

Figure 8: Nuovo progetto con Zend_Tool

Con questo comando viene realizzato non solo il nuovo progetto, ma anche tutta la struttura di cartelle e sottocartelle necessarie al suo funzionamento, e

che compongono un qualsiasi progetto sviluppato con Zend Framework. Ciò che viene creato non è solo un insieme di cartelle vuote, bensì sono presenti anche le componenti essenziali, praticamente obbligatorie, per lo svolgimento dell' applicazione (vedremo meglio nel cap. 4 in che modo) come ad esempio:

- il file boostrap.php
- l' application.ini
- l' IndecController.php
- l' index.php.

Infatti se proviamo ad eseguirlo usando il nostro SVN viene visualizzata la seguente schermata:

Figure 9: "Hello World" con Zend Framework

Oltre che per creare nuovi progetti Zend_Tool fornisce anche dei servizi per aggiungere altri componenti a quelli già esistenti come ad esempio solo per citarne alcuni tra i piu importanti e che verranno ripresi nel corso di questa relazione:

```
Model
  zf create model name module

View
  zf create view controller-name action-name-or-simple-name module

Controller
  zf create controller name index-action-included[=1] module

Action
  zf create action name controller-name[=Index] view-included[=1] module

Form
  zf enable form module
  zf create form name module

Layout
  zf enable layout
  zf disable layout
```

Figure 10: Comandi aggiuntivi di Zend_Tool

Quindi da quanto è emerso da questa breve pesentazone **Zend_Tool** risualta essere un potentissimo quanto utilissimo strumento a disposizione dei programmatori per facilitare e velocizzare enormemente la realizzazione di una web application con Zend Framework. Infatti grazie a questo strumento lo sviluppatore non deve più preoccuparsi di realizzare tutta quanta l' architettura per poter inserirvi il codice prodotto, ne tantomeno c' è la possibilità che sbagli a collocarli visto che vengono già inseriti automaticamente nella posizione corretta, rendendo così molto più agevole e leggero il compito del programmatore. Infine l' ultima considerazione che proponiamo in merito a questo argomento riguarda la velocità nell' apprendimento del suo funzionamento e la semplicità con la quale viene utilizzato tanto che anche uno sviluppatore alla sua prima esperienza con Zend Framework (tra i quali ci mettiamo anche noi) riesce comunque a costruirre la sua applicazione senza particolari inconvenienti.

3.6 LA COMPONENTE CORE

In questa componente si inseriscono tutti quegli elementi generici che non rientrano in nessuna delle altre categorie. Il componente che si occupa della generazione dei file in formato PDF, quello che gestisce l'invio di email e quello che gestisce il caching rientrano, ad esempio, in questo gruppo.

Zend_Pdf è il componente che ha il compito di generare dinamicamente i documenti in formato PDF: esso fornisce tutti gli strumenti necessari per controllare la posizione degli elementi che compongono la pagina.

Zend_Mail è invece il componente per la gestione della posta elettronica, sia in formato testuale che in HTML (*HyperText Markup Language*). Zend_Mail permette di inviare email sia attraverso il protocollo SMTP sia attraverso la funzione mail() nativa di PHP. A testimonianza della grande flessibilità e cura con cui ogni componente è stato progettato, Zend_Mail permette di aggiun-

gere facilmente metodi di trasporto alternativi, implementando l'interfaccia Zend_Mail_Trasport_Interface.

Gli utenti non sono disposti ad aspettare per le informazioni richieste, per questo il nostro prodotto deve poter garantire tempi di risposta rapidi e precisi. **Zend_Cache** è il componente che ci viene in aiuto in questo tipo di situazioni: esso fornisce un'interfaccia generica per implementare un sistema di caching con diversi strumenti (come database, dischi, ecc.). Questo approccio dimostra ancora una volta la grande flessibilità di ZF: possiamo partire implementando un piccolo sistema di caching che supporti le prime release del nostro applicativo, arrivando a sistemi più complessi solo quando si presenta una reale necessità.

3.7 MODEL-VIEW-CONTROLLER IN ZEND FRAMEWORK

Nella pratica vi sono due approcci per utilizzare Zend Framework:

1. Utilizzare alcuni dei suoi componenti all'interno della nostra applicazione: questa è la strada da seguire se abbiamo già un progetto esistente in cui vogliamo integrare il framework.

2. Sviluppare un'applicazione dall'inizio alla fine con ZF utilizzando il sottosistema **MVC** (Model-View-Controller) del framework come struttura base della web application. Queste tre componenti dell'omonimo pattern rappresentano rispettivamente il modello dei dati (Model), il rendering del modello dei dati (la View) e la logica che accetta gli input e che manipola il Model e la View (il Controller):

    - Model (data): il Model rappresenta i dati utilizzati dalla nostra applicazione;
    - View (presentation): la View si occupa del rendering dei dati rappresentati dal modello. È il risultato finale della "fusione" tra le azioni specificate dal Controller e i dati rappresentati dal Model.
    - Controller (business logic): il Controller si occupa di gestire tutti gli input e di manipolare sia le istanze dei Model che quelle delle View, curando le comunicazioni tra le due parti. Inoltre gestisce le azioni che l'utente può compiere sulla piattaforma per raggiungere il suo obiettivo: esso dunque rappresenta i verbi o gli eventi, come ad esempio "Crea", "Modifica", "Invia", ecc.

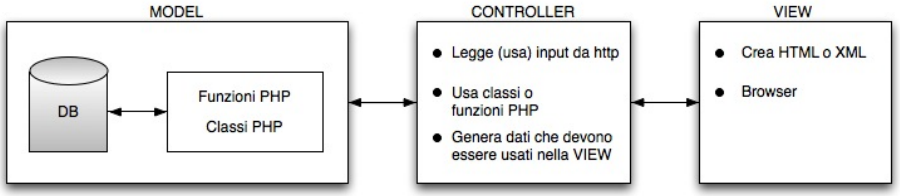

Figure 11: Model-View-Controller

In questa categoria quindi rientrano tutti i componenti fondamentali per l'implementazione del pattern MVC: vedremo in dettaglio qui di seguito quali sono e come possono essere usati.

3.7.1 IL CONTROLLER: ZEND_CONTROLLER_FRONT

Zend_Controller_Front implementa il pattern Front Controller: tutte le richieste passano attraverso questo elemento, che è responsabile dell'inoltro della richiesta e del ritorno del responso. Il workflow del Front Controller è gestito da diversi componenti:

1. Zend_Controller_Request_Abstract : rappresenta la richiesta non ancora processata; è responsabile di tutte le informazioni che circondano la richiesta, per tutto il restante processo.
2. Zend_Controller_Router_Interface : rappresenta il router, ovvero l'oggetto che ispezione la richiesta e determina quale controller e quale action devono essere eseguiti per processarla correttamente. Il router di default seziona l'URL secondo lo schema /controller/action/chiave/valore come nel seguente:

   > **ESEMPIO:** localhost/public/news/update/id/84 in cui siamo di fronte al controller "news", all'action "update", con la chiave "id" impostata sul valore "84″ .

3. Zend_Controller_Dispatcher_Interface : il ruolo del dispatcher è quello di prelevare le informazioni fornite dal router, instanziare il controller in questione ed eseguire l'action per processare la richiesta. Questo processo viene eseguito in loop per poter gestire quelle richieste che necessitano di complesse computazioni.
4. Zend_Controller_Response_Abstract : rappresenta l'oggetto "response" che raccoglie i risultati della richiesta ricevuta, gestita e completata.

3.7.2 IL MODEL: ZEND_DB

Zend_Db rappresenta l'interfaccia per la gestione di database SQL in ZF. Questo componente fornisce un livello di astrazione che permette di gestire diversi sistemi di base di dati utilizzando un insieme comune di tool. Questo

permette nella pratica di astrarre dal tipo di database utilizzato, portando anche a livello di storage una grande flessibilità e comodità.

Zend_Db_Table è la classe che implementa il pattern *Table Data Gateway* e il pattern *Row Data Gateway*, che permettono un approccio più generico alle gestione del database SQL. Questa è inoltre la classe che viene utilizzata per creare i modelli object-oriented; vediamo un esempio:

Algorithm 7 Esempio di classe per creare modelli

```php
<?php

class Model_Articolo extends Zend_Db_Table {

    protected $_name    = 'articolo';
    protected $_primary = 'id_articolo';

    function fetchLatest( $count = 5 ) {
        return $this->fetchAll( null,' id_articolo ASC', $count );
    }

/*
* Questo metodo restituisce il record che possiede l' "id_articolo" uguale al parametro
* $articoloID: creiamo una SELECT con una clausola WHERE sul parametro "id_articolo".
* Il metodo "fetchAll" lo ereditiamo da Zend_Db_Table , e si occupa di eseguire la query
* restituendo il record che viene salvato nella variabile "$articolo"
*/
    function fetchArticoloItem( $articoloID ) {

        $select = $this->select();
        $select->from($this, array('id_articolo', 'title'))
        ->where(' id_articolo = ?', $articoloID);
        $articolo = $table->fetchAll($select);
        return $articolo;
    }

    //Crea un nuovo record nella tabella 'articolo' per memorizzare le caratteristiche del  //nuovo oggetto
        function create ( $data ) {
        [...]
        $this->insert($data);
        [...]
    }
```

Supponiamo di avere una tabella MySQL articolo con chiave primaria id_articolo e due campi, title e description. La classe Model_Articolo rappresenta il modello per la tabella in questione: la prima cosa che notiamo è che estende Zend_Db_Table.

Inoltre, per ogni modello che definiamo dobbiamo dichiarare le variabili protected $_name che richiama il nome reale della tabella e $_primary che richiama la chiave primaria della tabella stessa.

La parte rimanente del codice della classe è composta da tre funzioni, fetchLatest($count = 5), fetchArticoloItem($articoloID) e create($data): la prima permette di ricavare gli ultimi articoli inseriti, specificati in numero dal parametro $count, che è impostato a 5 come valore di default. La seconda fun-

zione invece restituisce un oggetto che contiene l'id e il titolo dell'articolo che ha come identificatore quello passato come parametro. La terza ed ultima funzione create($data) la vedremo invece in dettaglio nel prossimo paragrafo. Nei primi due casi le query sono delle semplici SELECT sulla tabella articolo; nel primo caso ad esempio l'SQL generato dal codice PHP è:

Algorithm 8 SQL genarato con PHP

```
SELECT * FROM "articolo" ORDER BY articolo_id DESC LIMIT 0,5
```

Zend_Db_Table fornisce quindi un'interfaccia flessibile ed intuitiva per la gestione delle nostre tabelle: per inserire un articolo ad esempio è possibile utilizzare la funzione insert messa a disposizione dalla classe stessa ed ereditata dal nostro model. Supponiamo di avere creato il seguente controller ArticoloController:

Algorithm 9 Esempio di classe di controllo

```
class ArticoloController extends Zend_Controller_Action {

    public function init() {
        /* qui inizializziamo il controller*/
        [...]
    }

    /* Questa è la action chiamata di default*/
    public function indexAction() {
        $this->view->headTitle( 'Elenco Articoli' );
        $articoliData = new Model_Articolo();
        $articoli = $articoliData->fetchLatest();
        $this->view->articoli = $articoli;
    }

    /* Medoto che gestisce la richiesta di creazione di un articolo*/
    public fuction createAction() {
        [...]
        //controlliamo che la richiesta sia pervenuta attraverso il metodo POST
        if (  $this->getRequest()->isPost() ) {
            //Recuperiamo i parametri inviati attraverso il form
            $title = $this->getRequest()->getParam("article_title");
            $description = $this->getRequest()->getParam("article_description");
            //creiamo il model per l'oggetto Articolo
            $article_model = new Model_Articolo();
            //l'array che contiene i dati da passare al model per la creazione del record
            $data = array("title" => $title, "description" => $description);
            //richiamiamo il metodo "create" del model per creare un oggetto Articolo
            $article_model->create($data);
            [...]
        }
    }
}
```

Quando il Front Controller riceve la richiesta di creazione articolo:

www.dominio.com/public/articolo/create

richiama il controller articolo e la action create: viene dunque eseguito il metodo createAction() visto precedentemente nella classe ArticoloController. Come abbiamo visto nei commenti sul codice, viene creato il model per l'oggetto articolo, e successivamente richiamata la funzione create: vediamo in dettaglio come il model crea un nuovo record nel database.

Per inserire un articolo richiamiamo la funzione insert messa a disposizione dalla classe Zend_Db_Table:

Algorithm 10 Funzione di inserimento

```
[...]
$this->insert($data);
[...]
```

Questa istruzione corrisponde ad una INSERT SQL: ZF si occupa di mappare per noi i dati sui campi della tabella articolo e di scrivere la query completamente. Le query dovrebbero essere lanciate all'interno di un blocco try {} catch {} per poter gestire correttamente eventuali errori (che possono riguardare sia la connessione che l' iserimento).

3.7.3 LA VIEW: ZEND_VIEW

Zend_View è il componente responsabile della gestione della "vista" per la realizzazione del pattern MVC in ZF. Il Front Controller crea un'istanza di Zend_View attraverso la quale è possibile impostare tutte le variabile necessarie per gestire il flusso di dati generato dal model ed instradato dal controller: Zend_View riceve tale flusso e crea il codice XHTML per gestire la visualizzazione della risposta alla richiesta dell'utente.

Vediamo un esempio proseguendo con la visualizzazione di un articolo. Supponiamo che l'utente voglia leggere un articolo e che la richiesta giunta alla nostra web application sia della forma seguente:

www.dominio.com/public/articolo/read/id/8284

Ormai siamo in grado di "leggere" questa richiesta: il controller articolo richiama il metodo per gestire la action read passando a quest'ultima il parametro id per recuperare un determinato oggetto.

Algorithm 11 Funzione di lettura gestita dal controller

```
class ArticoloController extends Zend_Controller_Action {

        [...]
        /* il controller chiede al modello di recuperare i dati di un determinato articolo,
        quello con id uguale al parametro passato, e successivamente assegna alla
        vista l'oggetto ricevuto dal model */

        public function readAction( $articleID ) {
            [...]
            $articoloModel = new Model_Articolo();
            $articolo = $articoloModel->fetchArticoloItem( $articleID );
            $this->view->articoloRecord = $articolo;
            [...]
        }
        [...]
        }
```

Come indicato nei commenti, il controller richiama il metodo read passandogli il parametro id dell'articolo richiesto: nel corpo del metodo viene creato un oggetto Model_Articolo sul quale si richiama il metodo fetchArticoloItem($articleID). Questo metodo restituisce un record, nelle specifico quello che corrisponde all'id passato come parametro.

La variabile $articolo dunque conterrà il record che possiede l'id uguale al parametro: a questo punto non rimane che passare i dati alla vista, assegnando alla variabile $articoloRecord, accessibile appunto nella view, il record recuperato.

Vedremo nella sezione finale un esempio completo di come comunicano la view, il controller e il model implementando un semplice sistema di login.

3.7.4 PROGRAMMAZIONE PHP VS PARADIGMA MVC

L' algoritmo che andremo ad implementare tratta un operazione "classica", se si vuole definirla così, per le tipologie di applicazioni che vengono sviluppate con questi linguaggi di programmazione. L' azione che andremo a eseguire riguarda il recupero di notizie da un database per poi andarle a visualizzare sulla pagina web.

Procedendo con ordine vediamo anzitutto come si svilupperebbe questo algoritmo solo con l' utilizzo di PHP:

Algorithm 12 Esempio con PHP

```php
<?php
    $db = mysql_connect($my_host, $my_username, $my_password) or die ("Sever error connection");
    mysql_select_db($db_name,$db) or die ("Database error connection");
    $result=mysql_query("SELECT * FROM notizie ORDER BY notizia_date desc");
?>
<html>
    <body>
        <h1>NOTIZIA</h1>
        <?php while($row = mysql_fetch_object($result))
        { ?>
        <h2><?php echo $row->titolo; ?></h2>
        <p>
            <?php echo $row->body; ?>
        </p>
        <?php } ?>
    </body>
</html>
```

Come possiamo vedere il codice generato non è poi così difficicle da realizzare, però da queste poche righe possiamo comunque dedurre delle importanti considerazioni. Una programmazione di questo genere rende difficile mantenere uno script di questo tipo soprattutto difronte ad eventuali modifiche future e alla possibilità di riutilizzare il codice creato per altre azioni simili. In particolare queste problematiche diventano più evidenti se la visualizzazione, anzichè riguardare una sola pagina, riguarda più pagine all' interno della stessa applicazione. Queste valutazioni possono sembrare di poco conto per un utente inesperto, ma per un programmatore questi problemi e la loro risoluzione costituiscono una parte fondamentale di una programmazione efficace, tanto che all' università si tengono corsi che trattano proprio di questi argomenti. Inoltre poniamo l' attenzione sul fatto che la realizzazione di un tale codice porta già di per sè a commettere quantomeno errori di tipo sintattico in quanto il passaggio continuo che avviene tra i due linguaggi rende difficile non solo la stesura dello stesso, ma anche il suo controllo e al presentarsi di un eventuale errore ne rende difficile l' identificazione e la conseguente correzione.

A questo punto non resta altro che analizzare come si potrebbe realizzare lo stesso algoritmo utilizzando ZF e, ovviamente l' MVC. Con questo approccio ci sono due rilevanti modifiche che il nostro sript subisce e che risaltano agli occhi in modo evidente:

1. Lo spostamento del frammento di codice che recupera le notizie dal database in un componente riutilizzabile (solitamente una classe o una funzione in php) detto **model**

2. La separazione della chiamata alla funzione che recupera le notizie dal file html. Questo comporta il vantaggio di poter modificare il codice html senza preoccuparsi dei file html e come sono stati creati.

Quello che si è cercato di spiegare precedentemente non è altro che: la separazione del controllore dalla view, cioè vengono separati i controlli effettuati sui

dati acquisiti dalla loro visualizzazione sulla pagina html. Queste poche parole non fanno altro che focalizzare l' attenzione sul vero centro del nostro discorso e quindi la vera innovazione portata da questo componente. Perciò per dissolvere qualsiasi ambiguità o dubbio che potrebbero ancora essere presenti andiamo ad esemplificare in maniera più concreta con l' utilizzo di uno schema quanto è stato detto finora in maniera astratta:

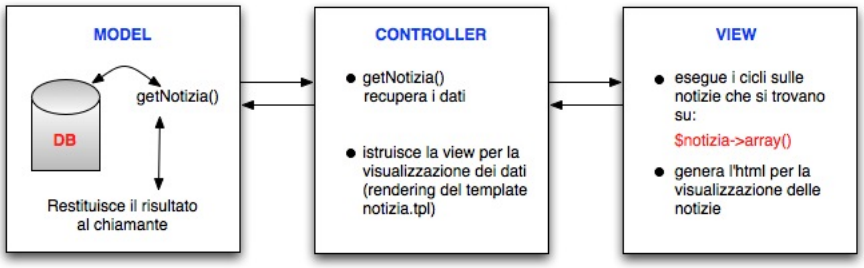

Figure 12: Separazione dell' MVC

Un lettore attento dovrebbe essersi già accorto che con questo tipo di architettura non è più necessaria la duplicazione del codice, in pratica viene realizzato perfettamente il paradigma del riutilizzo del codice. A questo punto non dovrebbero esserci piu incertezze sulle modalità di funzionamento dell' MVC perciò passiamo tranquillamente alla realizzazione dell' algoritmo vero e proprio. Qui di seguito vengono riportate le realizzazioni delle tre componenti dell' MVC che andranno ad implementare il nostro script. In ordinesi ha:

- IL MODEL che effettua la connessione al database e prepara le notizie in modo che possano essere utilizzate per le prossime operazioni (in questo caso la visualizzazione anzichè un' elaborazione),

Algorithm 13 Esempio di MODEL

```php
<?php
    function getNotizia()
    {
        $db = mysql_connect($my_host, $my_username, $my_password) or die ("Sever error connection");
        mysql_select_db($db_name,$db) or die ("Database error connection");
        $result = mysql_query("SELECT * FROM notizie ORDER BY notizia_date desc");

        $notizia = array();
        while($row = mysql_fetch_object($result))
        {
            $notizia[] = $row;
        }

        return $notizia;
    }
?>
```

- IL CONTROLLER che recupera le informazioni da esaminare e predispone il template in modo che questo possa essere correttamente visualizzato,

Algorithm 14 Esempio di CONTROLLER

```php
<?php
    $notizia = getNotizia();
    /*
     * display_template è una funzione fittizia per fare il rendering
     * del template notizia.tpl passato come argomento
     *
     * questa funzione non realizza la visualizzazione ma come
     * devono essere visualizzati i dati
     *
     */
    display_template('notizia.tpl');
?>
```

- LA VIEW che effetua infine la visualizzazione del template sulla pagina html.

Algorithm 15 Esempio di VIEW

```php
<html>
    <body>
        <h1>>NOTIZIE</h1>
        <?php
            foreach ($notizia as $row)
            {?>
                <h2>><?php echo $row->titolo; ?></h2>
                <p><?php
                    echo $row->body; ?>
                </p><?php
            }?>
    </body>
</html>
```

La prima differenza sostanziale che si può osservare tra le due realizzazioni è la differenza nel numero degli script creati. Cio è dovuto al fatto che ZF come del resto tutti i linguaggi di programmazione moderni, compreso lo stesso php, implementano la programmazione ad oggetti. Nella parte di questo esempio riguardante php si è deciso tuttavia di non adottare tale architettura, anche se supportata, semplicemente perchè la programmazione ad oggetti di php, per quanto a prima vista risulti simile nella struttura, non è la stessa cosa che viene implementata con l' utilizzo dell' MVC di ZF. Per capire il nostro punto di vista si deve ritornare allo scopo per cui è stato presentato questo esempio, che no è quello di impiegare ad ogni costo la programmazione ad oggetti, ma quello di mostrare le differenze tra l' utilizzo di php e di un framework che lo implementa. Perciò nell' ambito di questa trattazione la programmazione ad oggetti va vista solo come uno strumento e non come la soluzione del problema, tanto più che

il suo impiego nella prima parte avrebbe solo confuso le idee. Probabilmente da questo piccolo esempio, anche ad un programmatore esperto, non sarà comunque evidente del perchè stiamo parlando di una differenza così profonda in quanto non si è fatto grande uso delle potenzialità di ZF. L' importante è ricordare che il MODEL-VIEW-CONTROLLER viene inserito come struttura portante di ogni progetto sviluppato tramite ZF e che oguna delle sue componenti mette a disposizione un' ampia serie di funzioni e classi già implementate. Quindi programmare con ZF senza implementare l' MVC all' interno della *web application* non è il modo corretto di lavorare con questo strumento, soprattutto se uno sviluppatore vuole sfruttarne al massimo tutte le potenzialità, a meno che non si deva soltanto aggiungere delle funzionalità ad una applicazione già esistente per cui grazie alla versatilità e alla divisione dei vari elementi di ZF, di cui abbiamo già parlato nel capitolo 3.2, è possibile utilizzare unicamente le componenti necessarie senza andare modificare l' intera struttura del progetto. Questi sono tutta una serie di motivi per cui sviluppare applicazioni con ZF risulta relativamente facile ed immediato, anche ad utenti non esperti, una volta che si è compreso l' architettura dell' MVC.

NOTA: il codice presentato in questo esempio non ha la pretesa di essere eseguibile ne tantomeno di funzionare correttamente, è più che altro una sorta di pseudo-codice che ha il compito di indicare come dovrebbero essere eseguiti i vari passaggi necessari alla realizzazione dell' applicazione nel modo in cui sono stati concepiti dai creatori del framework stesso. Se infatti uno sviluppatore programma correttamente, ovvero nel modo canonico pensato per Zend, la struttura del suo codice dovrebbe rispecchiare quella evidenziata negli algoritmi appena visti come, del resto, abbiamo fatto noi nella parte del capitolo 4 dedicata allo sviluppo di un progetto con ZF.

3.8 COMPONENTI DI AUTENTICAZIONE ED INTERNAZIONALIZZAZIONE

3.8.1 AUTENTICAZIONE E ACCESSO

L'autenticazione è il processo di identificazione di un utente nell' applicazione, solitamente attraverso l'inserimento di una coppia di token (user/password). Il controllo degli accessi è invece quel processo attraverso cui si controlla ciò che un utente autenticato può o non può fare nell'applicazione, come ad esempio cancellare record o accedere a risorse riservate.

Zend_Auth e **Zend_Acl** sono i componenti che ZF mette a disposizione per la gestione rispettivamente del processo di autenticazione e di controllo degli accessi. Zend_Auth rispecchia la grande flessibilità di Zend Framework: visto il numero importante di sistemi diversi esistenti per autenticare un utente, questo componente è stato progettato per permette al programmatore di implementare il sistema più adeguato alle sue necessità se nessuna delle soluzioni alternative proposte risulta adatta.

Zend_Acl utilizza un approccio RBAC (*Role-Based Access Control*) per l'implementazione del controllo degli accessi. RBAC è un sistema generico di controllo basato su associazioni ruolo – risorsa: un ruolo rappresenta qualsiasi cosa che intende accedere al sistema mentre una risorsa è qualsiasi informazione disponibile. Per una applicazione web normalmente un ruolo è un gruppo di utenti mentre una risorsa può essere una pagina .html, una pagina .php, un'immagine, un record del database, ecc.

3.8.2 INTERNATIONALIZATION

Zend Framework supporta diversi livelli di internazionalizzazione, permettendo di avere un controllo pressocché totale sulle impostazione della localizzazione, sui set di caratteri e ovviamente sul multilingua. **Zend_Locale**, Zend_Measure e Zend_Currency sono i componenti principali responsabili della corretta corrispondenza tra la lingua impostata e gli idiomi utilizzati, mentre Zend_Translate si occupa del processo di traduzione testuale delle lingue disponibili nell' applicazione: rispetto alle funzioni native di PHP, **Zend_Translate** semplifica e velocizza il processo di traduzione, anche grazie alla perfetta integrazione con altri componenti come ad esempio Zend_View e Zend_Form. Detto questo cerchiamo ora di analizzare più dettagliatamente quanto abbiamo espresso in maniera astratta nelle righe precedenti.

Di default, Zend_Form e relativi validatori restituiscono delle stringhe di testo in lingua inglese in caso i vari campi non siano validati, qui di seguito vedremo in pochi e brevi passaggi come tradurre questi messaggi di errore in maniera praticamente automatizzata utilizzando le traduzioni create dallo stesso team di Zend Framework. Scaricando il pacchetto completo di Zend Framework per eseguirne l' installazione, oltre che al framework in se stesso possiamo trovare diverse altre directories, in particolare quella di nostro interesse per questo specifico ambito si chiama '*resources*'. All' interno di resources possiamo trovare la cartella '*languages*' con al suo interno diverse altre directories, ognuna delle quali identifica una particolare lingua. Procedendo con il copiare l' intera directory 'languages' ed incollandola all' interno del nostro progetto, allo stesso livello delle directories 'application' e 'library' come viene del resto anche consigliato dalla documentazione ufficiale di Zend (*zend translate using structure*), la struttura della nostra applicazione dovrebbe assomigliare alla seguente:

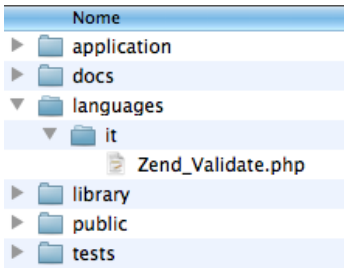

Figure 13: Struttura progetto

Ora che abbiamo copiato i file al posto giusto dobbiamo procedere alla realizzazione di una nuova funzione _init* all'interno della classe Bootstrap. Quindi una volta aperta inseriamo il seguente codice:

Algorithm 16 Nuova funzione del Boostrap

```
protected function _initLanguage()
{
    $localeValue = 'it';

    $locale = new Zend_Locale($localeValue);
    Zend_Registry::set('Zend_Locale', $locale);
    $translationPath = dirname( APPLICATION_PATH ) . DIRECTORY_SEPARATOR . 'languages' .
DIRECTORY_SEPARATOR . $localeValue;

    $translate = new Zend_Translate('array', $translationPath, $localeValue);

    Zend_Registry::set('Zend_Translate', $translate);
    Zend_Validate_Abstract::setDefaultTranslator($translate);
    Zend_Form::setDefaultTranslator($translate);
}
```

Il nuovo metodo _initLanguage() appena creato istanzia le classi Zend_Locale e Zend_Translate e le inserisce all'interno del registro. Da notare che il valore di Zend_Locale (il primo parametro) è forzato per essere 'it'. Per un sistema dove sia previsto lo switch automatico di questo valore in base alla provenienza dell'utente dovrete provvedere alla realizzazione di un plugin, ma questa parte va oltre lo scopo che ci siamo prefissati di presentare in merito a questo argomento. La classe Zend_Translate accetta diversi parametri come costruttore, il primo che possiamo vedere indica il tipo di adapter da utilizzare, che nel nostro caso abbiamo optato per un 'array'. Questo sta ad indicare che le stringhe che verranno tradotte saranno salvate in un file contenente un array. A nostro avviso questo approccio è il più semplice da utilizzare, anche se l'unica pecca che potrebbe presentarsi riguarda la corretta manipolazione del suddetto array che non tutti potrebbero essere in grado di effettuare. Ovviamente l' array dovra essere restituito utilizzando l' operatore 'return'. Nella documentazione ufficiale di Zend (*zend translate adapter*) è presente anche un listato degli adapters disponibili e visitandola troveremo la conferma che l' array è lastruttura più semplice da utilizzarsi per questa procedura come già riportato in precedenza.

Le ultime due righe di codice di questo metodo fanno il lavoro di traduzione dei form:

Algorithm 17 Istruzioni per la traduzione dei Form

```
Zend_Validate_Abstract::setDefaultTranslator($translate);
Zend_Form::setDefaultTranslator($translate);
```

L'oggetto *translate* appena creato sarà utilizzato da tutte le istanze di Zend_Form e Zend_Validator per la traduzione dei messaggi di errore. Per provare la veridicità di quanto abbiamo appena detto è sufficiente avviare un qualsiasi

Form, al quale progetto siano state apportate queste modifiche, o crearne uno ad hoc per la fase di test e osserveremmo tutti i messaggi d' errore tradotti in italiano. Allo stesso modo è possibile cambiare la lingua dei validatori semplicemente modificando il valore di *$localeValue* con una qualsiasi altra lingua.

OLTRE AI FORM

L'oggetto *translate* appena creato potrà essere utilizzato anche in qualsiasi file della view della nostra applicazione utilizzando il **Translate_view_helper**. Allo scopo di effettuare una prova, creiamo all'interno della directory 'it' (languages/it/) un file chiamato Custom.php (o qualsiasi nome preferiate) ed inseriamo al suo interno questo codice:

Algorithm 18 Custom.php

```php
<?php

  return array(
      'test' => utf8_encode('Questa è una stringa di traduzione di prova')
  );
```

Questo file appena creato sarà automaticamente analizzato e le traduzione inclusa nell'oggetto *translate*. Inserendo quindi, in un qualsiasi file della view della nostra applicazione, il seguente codice:

Algorithm 19 Istruzione per la traduzione della vista

```php
$this->translate('test');
```

Otterremo a video qualcosa del genere:

```
// Questa è una stringa di traduzione di prova
```

Figure 14: Risulatato della traduzione della vista

A questo punto possiamo tranquillamente affermare di aver concluso questa trattazione sui componenti di internazionalizzazione anche se solamente per i più utilizzati e diffusi nella realizzazione delle *web application* svillupate con Zend e da quello che si è visto nel corso dello sviluppo non è per nulla difficile o complicato implementarli tanto che anche uno sviluppatore alle prime esperienze con ZF poterbbe impiegarli nei sui progetti. La versiome che abbiamo presentato con gli array può essere inegrata facilmente anche per un sito che non necessita, per vari motivi, di alcuna funzione di internazionalizzazione tanto è semplice e immediata la sua realizzazione così da fornire un ulteriore servizio all' utente finale.

3.9 COMPONENTI PER SERVIZI WEB E COMUNICAZIONI

3.9.1 SERVIZI WEB

L'integrazione e la comunicazione tra web service è ormai di vitale importanza per una web application: negli ultimi anni, soprattutto con l'avvento dei social network, è diventato praticamente impossibile pensare ad una applicazione che non sia in grado ad esempio di leggere un Feed RSS o di interfacciarsi con Facebook, Flicker o Twitter.

È diventata pratica comune rilasciare API che gli sviluppatori possono utilizzare per integrare all'interno delle proprie applicazioni servizi esterni di Google, Yahoo!, Facebook, e di tutti quei social network dedicati alla condivisione del contenuto, come digg.com, slide_share.com, delicious.com, e tanti altri ancora.

Zend Framework, sfruttando le caratteristiche della sua struttura a componenti, permette di integrare rapidamente e in maniera assolutamente trasparente qualsiasi tipo di web service: i componenti con il prefisso Zend_Service sono proprio quelli che rientrano in questa categoria.

Zend_Service_Delicious, Zend_Service_Yahoo, Zend_Service_Slide_Share sono solo alcuni esempi: l'integrazione di servizi web esterni è sostanzialmente completa. Ogni componente è dedicato ad un particolare servizio, con le sue determinate caratteristiche e specifiche.

3.9.2 INTERAPPLICATION COMMUNICATION

La comunicazione tra applicazioni viene implementata principalmente attraverso tre tecniche: SOAP (*Simple Object Access Protocol*), XML-RPC e REST (*Representational State Transfer*).

Zend Framework offre strutture solide per supportare sia questi formati sia soluzioni personalizzate in caso di requisiti particolari. Inoltre, per quelle applicazioni che si avvalgono di Ajax, è stato introdotto il componente Zend_Json per gestire ogni aspetto della comunicazione attraverso il protocollo JSON (*JavaScript Object Notation*), particolarmente indicato per gestire il dialogo tra Javascript e PHP nelle applicazioni Ajax.

3.10 LA CONNESSIONE AL DATABASE

PHP ha un buon supporto per i database, ma ha il difetto di fornire troppe soluzioni alternative per attuare l' interfacciamento con i server di dati. In mezzo a questa quantità veramente enorme di classi e funzioni una buona scelta sarebbe quella di optare per una via univoca da perseguire in tutti i propri applicativi. La soluzione migliore è sicuramente quella di affidarsi ad un layer di astrazione potente e funzionale, che ci permetta di interfacciarci con sistemi diversi ed architetture differenti; PHP offre anche questa soluzione attraverso l' utilizzo dei PDO. Questo strumento si focalizza solamente sui database e offre una buona serie di funzionalità. La soluzione proposta dallo Zend Framework

si basa appunto su questa libreria, che cerca di migliorare con l'aggiunta di interessanti routine di utilità.

3.10.1 IL MODULO ZEND_DB

Il modulo **Zend_Db** rappresenta il layer di astrazione basato su PDO utilizzato ovviamente da Zend framework per accedere ai database. L' utilizzo di PDO come libreria nativa permette al layer di astrazione di fornire l' accesso a tutti i sistemi di database supportati dalla libreria stessa. Tra questi ricordiamo i più usati come: MySQL, PostgreSQL, Microsoft SQL Server, SQLite ed molti altri ancora. La creazione di un oggetto per interfacciarsi ad un database è eseguita tramite l'utilizzo del pattern *factory*:

Algorithm 20 Connessione ad un db tipo MySQL

```
<?php

/* Presuppongo che il Framework Zend sia nel vostro include_path */
require_once 'Zend/Db.php';

// Opzioni di connessione
$options = array ('host'     => '127.0.0.1',
                  'username' => 'utente',
                  'password' => 'password',
                  'dbname'   => 'test');

$db = Zend_Db::factory('pdoMysql', $options);

?>
```

La funzione statica *factory* della classe Zend_Db accetta come parametri una stringa rappresentante il driver di connessione che si desidera utilizzare ed un array di opzioni specifiche per il driver che sono necessarie per connettersi correttamente alla fonte di dati richiesta. Se per esempio avessimo voluto connetterci invece ad un database SQLite, che non richiede l'autenticazione e l'utilizzo di un server, avremmo utilizzato il codice seguente:

Algorithm 21 Connessione ad un db tipo SQLite

```
<?php

require_once 'Zend/Db.php';

$options = array ('dbname' => 'test');

$db = Zend_Db::factory('pdoSqlite', $options);

?>
```

Il risultato di queste operazioni è un' implementazione specifica del driver utilizzato dell' interfaccia **Zend_Db_Adapter_Abstract**. Tramite questa interfaccia è possibile accedere alle funzionalità aggiuntive fornite dal framework in questione e ad alcune delle funzionalità esposte da PDO.

Un altro modo sicuramente utilizzato dalla maggior parte dei programmatori esperti, anche se per questo non significa che si tratta di un sistema complesso e complicato, sfrutta il file di configurazione (**application.ini** oppure **config.ini**) dell' intero progetto nel quale andranno inserite le opzioni specifiche per il driver di cui abbiamo parlato prima. Pertanto se sceglissimo questa via dovremmo procedere con la modifica di questo file aggiungendvi una nuova sezione, alla quale daremo il nome di **staging**:

Algorithm 22 Modifica del file application.ini

```
[staging : production]
resources.db.adapter = 'Pdo_Mysql'
resources.db.params.host = 'localhost'
resources.db.params.username = 'my_username'
resources.db.params.password = 'my_password'
resources.db.params.dbname = 'test'
```

La formula **[staging : production]** indicha semplicemente che la nuova sezione eredita i valori della *production* e li sostituisce quando necessario, nel caso ovviamente di equivalenza. In altre parole potremmo dire, rifacendoci ad una similitudine con il linguaggio ad oggetti di JAVA, che la sezione *staging* appena creata in un certo senso è figlia della sezione *production* dalla quale eredita e ne modifica i parametri. Una volta eseguita questa procedura la nostra connessione al database diventa:

Algorithm 23 Connessione ad un db utilizzando il file di configurazione

```
<?php
$config = new Zend_Config_Ini(ROOT_DIR.'/application/configs/application.ini','staging');
$db = Zend_Db::factory($config->resources->db);
?>
```

Naturalmente, anche se abbiamo presentato il suo impiego solo per il database di MySql, questo modo di lavorare si adatta perfettamente anche a tutti gli altri drivers della famiglia PDO con le dovute modifiche dettate dallo specifica scelta fatta dallo sviluppatore come abbiamo visto nel caso di SQLite.

3.10.2 INTERROGARE IL DATABASE

Una volta ottenuta l' interfaccia richiesta possiamo iniziare a lavorare per effettuare le interrogazioni sulla nostra base di dati. Ogni query SQL fornita direttamente al modulo verrà passata al layer PDO sottostante che si occuperà di eseguirla e restituire l' eventuale risultato sotto forma di **PDOStatement**. Vediamo come eseguire una semplice query di selezione su un database di prova:

Algorithm 24 Esecuzione di una query

```php
<?php

// Connessione al server già effettuate

/* Query eseguita quotando manualmente i valori */

$sql = $db->quoteInto('SELECT * FROM articles WHERE creation_date > ?','2006-06-05');
$result = $db->query($sql);
$rows = $result->fetchAll();

var_dump($rows);

/* Stessa query eseguita utilizzando i placeholder */

$result = $db->query('SELECT * FROM articles WHERE creation_date > :creation_date',
                     array('creation_date' => '2006-06-05'));
$rows = $result->fetchAll();

var_dump($rows);

?>
```

Come potete notare l'esecuzione di una query è un procedimento molto semplice. Il risultato restituito è un'istanza di *PDOStatement* che viene utilizzata per recuperare e gestire i risultati. Un argomento molto importate su cui soffermarsi un attimo è il **quoting** dei parametri passati alle query. Nell'esempio fornito non è molto utile, ma è fondamentale quando si opera su dati potenzialmente pericolosi perché provenienti da fonti non sicure. Il quoting di una stringa viene effettuato da Zend_Db utilizzando i metodi *quote* o *quoteInto* forniti dall' adapter, oppure, nel caso si utilizzino i *placeholder*, in modo automatico sui parametri:

Algorithm 25 Esempio della funzione quote()

```php
<?php

//    Quoto un valore scalare (stringa, numero, ecc …)
echo $db->quote('Come "va" ?');

/*
* Il valore restituito sarà '"Come \"va\"?"', circondato da apici doppi
*
*/

// Quoto un array
echo $db->quote(range(3));
/*
* Il risultato è una serie di stringhe separate da virgola:
* '"1", "2", "3"'
*/

// Quoto un valore e lo inserisco in una stringa
echo $db->quoteInto('id = ?', 1);
/* id = "1" */

?>
```

Come si può notare la funzione *quote* si occupa di trasformare il valore passato come argomento in una stringa che non potrà nuocere in alcun modo

quando viene utilizzato all' interno di una query SQL. Utilizzando *quoteInto* invece si effettua il quoting di tutti i parametri della funzione dopo il primo, e si sostituiscono (nell'ordine di passaggio) i risultati ai punti di domanda presenti nella stringa usata come primo argomento. Nel caso si utilizzino dei *placeholder* durante la chiamata della funzione *query* su un adapter, i valori dei *placeholder* specificati nell' array vengono tutti quotati prima dell' assegnazione.

3.10.3 PREPARED STATEMENT

Il modulo Zend_Db offre anche il supporto ai *prepared statement*. Questa teconologia permette (se fornita dal sistema al quale ci si connette) di precompilare delle query specificate parzialmente lasciando dei parametri variabili che potranno essere inseriti in seguito. Grazie alla precompilazione della query tutte le successive chiamate verranno eseguite in modo più efficiente. Vediamo un esempio:

Algorithm 26 Precompilazione della query

```php
<?php

// Precompilo la query
$stmt = $db->prepare('SELECT * FROM articles WHERE creation_date > :creation_date');

// Assegno il parametro alla query precompilata …
$stmt->bindValue('creation_date', '2006-06-05');

// … ed eseguo lo statement
$stmt->execute();
$rows = $stmt->fetchAll();
var_dump($rows);

// Eseguo nuovamente la query con diversi parametri
$stmt->bindValue('creation_date', '2006-03-01');
$stmt->execute();
$rows = $stmt->fetchAll();

var_dump($rows);

?>
```

Il metodo **prepare** si occupa di precompilare la query e restituire un oggetto *PDOStatement* con il quale eseguire le operazioni necessarie. Utilizzando **bindValue** è possibile assegnare un determinato valore ad un *placeholder*, mentre con **execute** viene eseguita la query usando i parametri specificati in precedenza. L' eventuale risultato è salvato all' interno dell' oggetto, e può essere recuperato con le normali funzionalità di PDO.

3.10.4 INSERIRE, MODIFICARE E CANCELLARE I RECORD

Il modulo Zend_Db espone una serie di funzionalità per velocizzare le operazioni di modifica delle tabelle del database. Per quanto riguarda il processo di inserimento di un nuovo record possiamo appoggiarci al metodo **insert**:

Algorithm 27 Inserimento di un nuovo record: insert

```php
<?php

$row = array (
    'title'         =>    'Perchè non vogliamo morire',
    'author'        =>    'Pinco Pallino',
    'creation_date' =>    '2006-06-05',
    'content'       =>    ''
);

$rows_affected = $db->insert('articles', $row);
$last_insert_id = $db->lastInsertId();

?>
```

Il metodo **insert** accetta come argomenti la tabella su cui operare ed un array contenente i valori da assegnare alle colonne della tabella per questo inserimento. Il risultato dell' operazione di inserimento, che si occupa automaticamente di effettuare il *quoting* dei dati, è il numero di righe modificate dall' esecuzione dell' istruzione. L' id dell' inserimento può essere recuperato usando il metodo **lastInsertId** dell' adapter.

Effettuare l' operazione di aggiornamento dei dati è altrettanto semplice:

Algorithm 28 Aggiornamento di un record: update

```php
<?php

$data = array ('content' => '… some content …');

$where = $db->quoteInto('creation_date = ?', $any_date);
$rows_affected = $db->update('articles', $data, $where);

?>
```

Usando il metodo **update** possiamo passare, oltre all' array di valodi da modificare, anche una clausola condizionale opzionale per limitare le operazioni di aggiornamento. Questa clausola non verrà quotata automaticamente quindi dovremo occuparci di fallo manualmente come nell'esempio proposto.

Infine l' operazione di eliminazione:

Algorithm 29 Eliminazione di un record: delete

```php
<?php

$where = $db->quoteInto('creation_date = ?', $any_date);
$rows_affected = $db->delete('articles', $where);

?>
```

Il metodo **delete** accetta due argomenti: il primo è una stringa con il nome della tabella, il secondo è una stringa che contiene un' espressione SQL che viene utilizzata come criterio per le righe da eliminare i quali valori non sono quotati, perciò questa responsabilità ricade sul programmatore. Il valore restituito è il

numero di righe interessate dall'operazione di eliminazione. Se invece si omette il secondo argomento, il risultato è che tutte le righe della tabella del database vengono eliminate.

Arrivati a questo punto possiamo dire, dal nostro punto di vista, di aver concluso la trattazione riuguardante il modulo Zend_Db focalizzandci soprattuto sulle funzionalità esposte dai driver PDO e su alcune procedure generiche che ogni sviluppatore avrà modo di utilizzare nei propri script.

3.11 LA GESTIONE DEGLI UTENTI

La gestione degli utenti può essere suddivisa in due macro: l'autenticazione ed il controllo degli accessi.

Il primo è quel processo che permette di identificare un individuo in base alle credenziali che fornisce, normalmente tramite una coppia di valori (nome utente – password).

Il secondo è invece quel processo che permette di distinguere tra quello che un utente autenticato può o non può fare nell'applicazione. ZF dispone di due componenti per gestire entrambi i processi, rispettivamente Zend_Auth e Zend_Acl: nelle prossime sezioni vedremo un esempio pratico del loro utilizzo.

L'esempio che andremo ad analizzare riguarda una situazione decisamente comune per una web application, ovvero la gestione degli utenti: chiunque interagisca con la nostra applicazione può essere considerato un utente che ricopre un determinato ruolo (es. Anonimo, Editore, Amministratore, ecc.). In base al ruolo ricoperto l'utente può disporre di determinate risorse e di determinate azioni per interagire con l'applicazione.

Una web application complessa può avere diversi ruoli per categorizzare le utenze, ma nel nostro esempio prenderemo in considerazione una situazione che ci permetta di studiare le casistiche basilari; configureremo i seguenti ruoli:

- Registrati: utenti semplicemente registrati al sito che non hanno alcun privilegio se non quello di leggere contenuto "protetto"
- Editore: utenti che hanno i privilegi per gestire il contenuto (creare, modificare, eliminare)
- Amministratore: l'utente che gestisce il sito, che quindi può accedere a qualsiasi funzionalità ed a qualsiasi contenuto

Una volta effettuata la connessione al database con uno dei metodi che abbiamo presentato nel paragrafo precedente avremo bisogno di un modo per memorizzare le varie tipologie di utenti che usufruiranno della nostra applicazione; per il nostro esempio supponiamo di dedicare una tabella del database per il salvataggio permanente delle utenze. La tabella potrebbe avere la seguente struttura:

Algorithm 30 Struttura tabella utenti

```
CREATE TABLE `utenti` (
`userid` int(11) NOT NULL auto_increment,
`username` varchar(50) default NULL,
`password` varchar(250) default NULL,
`firstname` varchar(50) default NULL,
`lastname` varchar(50) default NULL,
`role` varchar(25) default NULL,
`status` tinyint(2) default 0,
PRIMARY KEY (`userid`)
) DEFAULT CHARSET=utf8;
```

Lanciando questa query SQL creiamo la tabella utenti nella base di dati. Come abbiamo visto, per gestire la tabella dobbiamo creare un model Utente, rappresentato dal file Utente.php situato all'interno della directory application/models/ ottenendo in questo modo qualcosa del genere:

Algorithm 31 Il modello utente

```
<?php
class Model_Utente extends Zend_Db_Table_Abstract {
            /**
            * nome della tabella
            */
            protected $_name = 'utenti';

            /**
            * chiave primaria
            */
            protected $_primary = 'userid';
}
```

Più avanti vedremo in dettaglio l'implementazione del form per la registrazione degli utenti, e quindi il processo di creazione di un'utenza. Per ora supponiamo di avere già popolato la nostra tabella avendo così già degli utenti registrati nella nostra applicazione, e di aver già inoltre configurato l'utenza di amministratore.

3.11.1 IL MODEL ED IL CONTROLLER

Come prima cosa creiamo un nuovo metodo nel model per visualizzare la lista di tutti gli utenti registrati. Nel file application/models/Utente.php aggiungiamo dunque il seguente codice:

Algorithm 32 Funzione di visualizzazione utenti

```
public static function getUtenti() {

        $select = $this->select();
        $select->from($this,array('username','lastname', 'firstname'))
            ->order('lastname ASC');
        return $this->fetchAll($select);

}
```

In questo metodo non facciamo altro che creare una SELECT per interrogare la tabella "utenti": la query restituisce tutti i record della tabella (comprensiva dei campi: username, lastname e firstname) ordinati per cognome in maniera ascendente. Una volta impostato il modello, si può passare alla creazione del controller, che ci permetterà di gestire le utenze, ovvero del file application/controllers/UtenteController.php attraverso il seguente comando:

Algorithm 33 Creazione del controller

```
zf create controller utente
```

Il passo successivo è implementare nel controller la funzione che gestisce la action list. Anche questa volta utilizziamo Zend_Tool, e da linea di comando, sempre posizionandoci nella directory di progetto, lanciamo la seguente riga:

Algorithm 34 Creazione dell' action list

```
zf create action list utente
```

Una volta aperto il file UtenteController.php appena creato, all'interno del metodo listAction generato automaticamente da Zend_Tool, basta aggiungere queste poche righe di codice per recuperare la lista degli utenti ed assegnarla alla vista:

Algorithm 35 Funzione di recupero utenti

```
public function listAction () {
    $ utentiCorrenti = Model_Utente::getUtenti();
    if ($utentiCorrenti->count() > 0) {
      $this->view->utenti = $utentiCorrenti;
    } else {
      $this->view->utenti = null;
    }
}
```

Ormai dovremmo essere in grado di capire questo metodo: la prima cosa che facciamo è di richiamare sull'oggetto model il metodo getUtenti() analizzato precedentemente e di salvare il risultato nella variabile $utentiCorrenti. A questo punto eseguiamo un veloce controllo per verificare che effettivamente ci siano utenti nel database. In questo caso assegnamo alla view la variabile utenti che conterrà un oggetto che rappresenta l'elenco di tutti gli utenti registrati nella nostra base di dati, in caso contrario assegniamo null a tale variabile.

3.11.2 LA VIEW

Passiamo ad occuparci della view. Per visualizzare la lista degli utenti dobbiamo creare il file application/views/scripts/utente/list.php che conterrà il codice HTML della vista:

Algorithm 36 Visualizzazione degli utenti

```
<h3>Lista degli utenti</h3>
<?php
    //questa è proprio la variabile che abbiamo impostato nel controller
    if( $this->utenti != null ) {
        echo "<ul class='data'>"
        foreach ( $this->utenti as $utente ) {
            echo "<li>" . $utente->firstname;
            echo  $utente->lastname . " - " . $utente->username;
            echo "</li>";
        }
        echo "</ul>";
?>

<?php } else { ?>

        <p>La lista degli utenti attualmente risulta vuota.</p>

<?php } ?>
```

La vista visualizzerà un elenco puntato in cui ogni elemento conterrà il nome, il cognome e l'username di ogni utente presente nel nostro database. Chiaramente siamo liberi di impostare il layout come meglio crediamo, ad esempio inserendo codice javascript per rendere più accattivante la pagina.

Riassumendo: abbiamo creato il controller UtenteController, il model Utente e la vista per la action list. Notiamo che questo file è stato inserito all'interno della directory delle viste, precisamente nella directory utente: è qui infatti che ZF andrà a cercare il file per visualizzare la action list del controller utente. È quindi di vitale importanza rispettare il naming imposto dal framework: questa forse è l'unica vera limitazione di questo strumento.

Anche se è solo un esempio introduttivo possiamo comunque iniziare ad intravedere la comodità e la ben organizzata struttura di cui dispone ZF. Generalmente quando si vuole sviluppare un'applicazione complessa non viene tutto

seguito da una singola persona ma il tutto viene seguito da un team perciò: lo sviluppatore "*frontend*", quello cioè che si occupa dell'interfaccia utente, è libero di lavorare tranquillamente indipendentemente da ciò che succede nel controller e nel model, perché tutto ciò di cui ha bisogno è sapere che nella view è a disposizione l'elenco di tutti gli utenti grazie al riferimento $this->utenti . A loro volta gli sviluppatori che si occupa della logica di business (controller e model) sono liberi di riorganizzare il *backend* senza per forza interrompere il lavoro di chi si sta occupando dell'interfaccia utente.

3.12 ZEND_AUTH: ACCESSO DEGLI UTENTI

Ora che abbiamo chiuso il ciclo della comunicazione tra il controller, il model e la vista, vediamo come controllare l'accesso alla lista degli utenti. Quello che vogliamo ottenere è semplice: solo l'amministratore può visualizzare e gestire la lista delle utenze; per fare questo supponiamo di avere a disposizione un form per il login (lo vedremo nell' ultima parte di questa sezione) che invia i dati inseriti dall'utente per essere successivamente confrontati con quelli presenti nella base dati, finalizzando così il processo di autenticazione. Una volta che l'utente risulta autenticato possiamo controllare i permessi e conseguentemente ciò a cui può o non può accedere.

La prima cosa da fare è verificare nel controller se ci troviamo di fronte un utente autenticato o meno, e in caso positivo passarne l'identità alla view:

Algorithm 37 Funzione di verifica autenticazione

```
public function indexAction() {

     //recuperiamo l'istanza di Zend_Auth
    $auth = Zend_Auth::getInstance();

  if ( $auth->hasIdentity() ) {

     $this->view->identity = $auth->getIdentity();

      }
}
```

Zend_Auth implementa il pattern singleton, cosicché le informazioni circa l'utenza possono essere recuperate ovunque sia necessario nell'applicazione. Una volta che abbiamo l'istanza di Zend_Auth possiamo controllare se è impostata una identità, e in tal caso passarla alla vista. Qui possiamo utilizzare questa informazione ad esempio per escludere la visualizzazione della lista delle utenze a tutti gli utenti anonimi:

Algorithm 38 Algoritmo di controllo idoneità utente

```
[...]
<?php if ( null == $this->identity ) { ?>
  //l'identità non esiste, dunque non visualizziamo la vista
  [...]
  <?php } else { ?>
  [...]
  //l'identità esiste, possiamo continuare con gli ulteriori controlli (verificare i privilegi dell'utente)
<?php } ?>
[...]
```

Quello che succede in questo codice è abbastanza intuitivo: se l'istanza della identità risulta uguale a "null" allora ci troviamo di fronte ad un utente non registrato o anonimo, e possiamo decidere come comportarci (ad esempio: invece di visualizzare la lista degli utenti possiamo visualizzare un messaggio del tipo "Per accere alla risorsa selezionata devi essere loggato al sistema").

Con Zend_Auth è disponibile una serie di "adapter" che forniscono una soluzione quasi totalmente implementata per i meccanismi di autenticazione più diffusi ed utilizzati:

- Zend_Auth_Adapter_Http
- Zend_Auth_Adapter_Digest
- Zend_Auth_Adapter_DbTable
- Zend_Auth_Adapter_InfoCard
- Zend_Auth_Adapter_Ldap
- Zend_Auth_Adapter_OpenId

Ad esempio, gli "adapter" Http e Digest implementano il meccanismo di autenticazione memorizzando i dati in un file del disco e comunicando attraverso il protocollo HTTP.

L' "adapter" DbTable invece permette di implementare il meccanismo di autenticazione memorizzando i dati delle utenze in una tabella del database: proprio come abbiamo visto nel nostro esempio.

Grazie alla natura del sistema di "adapter" di Zend_Auth passare da un "adapter" ad un altro è un processo relativamente semplice: in altre parole, la struttura ed i meccanismi implementati in Zend_Auth permettono di passare da un sistema di autenticazione ad un altro in maniera semplice e pulita.

3.13 ZEND_ACL: IL CONTROLLO DEGLI ACCESSI

Zend_Acl fornisce una semplice ma robusta implementazione per la gestione delle liste di controllo degli accessi (*Access Control Lists* – ACLs). ACL rappresenta una soluzione flessibile per il controllo degli accessi: tutto ruota intorno ai concetti di "ruolo", "risorsa" e "privilegio":

- un “ruolo” normalmente rappresenta un gruppo o una classe di utenti,
- una “risorsa” rappresenta qualcosa all’interno del sistema che necessita di una protezione (come un’azione in un controller o dati nel database) e
- un “privilegio” rappresenta una tipologia di accesso, come ad esempio “scrittura”, “lettura”, “creazione” o “modifica”.

Per impostare il controllo degli accessi con Zend_Acl abbiamo bisogno di una lista di ruoli ed una lista di risorse. La prima cosa da fare dunque è impostare i ruoli che rappresentano gli utenti nella nostra applicazione. Creiamo innanzitutto una istanza di Zend_Acl :

Algorithm 39 Istaziamento di Zend_Acl

```
[...]
        $acl = new Zend_Acl();
[...]
```

Ora che abbiamo l’istanza di Zend_Acl possiamo aggiungere i ruoli:

Algorithm 40 Creazione dei ruoli

```
[...]
        $acl = new Zend_Acl();

        //creiamo il ruolo "registrato"
        $acl->addRole(new Zend_Acl_Role('registrato'));

        //creiamo il ruolo "admin"
        $acl->addRole(new Zend_Acl_Role('admin'));
[...]
```

A questo punto abbiamo creato i ruoli “registrato” e “admin” richiamando il metodo “addRole()” fornito dal componente Zend_Acl. Come vediamo dal codice, a questo metodo passiamo un’istanza di Zend_Acl_Role: questa è una classe molto semplice che memorizza il nome del ruolo. Due importanti precisazioni: una volta che il ruolo è stato creato non è più possibile cambiarne il nome, inoltre tale nome deve essere univoco nel contesto della ACL.

I ruoli possono essere organizzati secondo una struttura gerarchica di relazioni padre-figlio, in cui il ruolo figlio eredita i permessi del ruolo padre. Supponiamo ad esempio di aggiungere il ruolo “editore” al nostro applicativo:

Algorithm 41 Istanziamento di un ruolo

```
[...]
        $acl->addRole(new Zend_Acl_Role('editore'), 'registrato');
[...]
```

In questa situazione il nuovo ruolo “editore” ha come padre il ruolo “registrato”, dunque un “editore” eredita tutti i privilegi di un utente “registrato”, oltre chiaramente ai permessi specifici che saranno impostati per il ruolo di “editore”.

Come abbiamo già visto una “risorsa” è qualcosa che vogliamo proteggere: normalmente in ZF una “risorsa” è un controller o una “action” di un controller. Vorremmo ad esempio che solo gli utenti amministratori possano visualizzare la lista di utenti memorizzati nel nostro database: Zend_Acl_Resource è il componente che ci viene in aiuto nella definizione di “risorse”, ed è una classe semplice quanto Zend_Acl_Role, infatti anche in questo caso il solo compito che gli è assegnato è quello di memorizzare il nome della risorsa.

Vediamo come si può procedere per il nostro esempio:

Algorithm 42 Memorizzazione della risorsa utente

```
[...]
        $acl->add(new Zend_Acl_Resource('utente'));
[...]
```

Anche in questo caso utilizziamo il metodo “add” della classe Zend_Acl, ma questa volta gli passiamo una istanza della classe Zend_Acl_Resource che rappresenta la risorsa “utente”. Anche in questo possiamo organizzare gerarchicamente le risorse che vogliamo proteggere, esattamente con la stessa tecnica che abbiamo adottato per la definizione dei ruoli.

Ora che abbiamo definito i ruoli e le risorse ci resta l’ultimo passaggio per configurare l’oggetto Zend_Acl: dobbiamo specificare quali permessi ha un determinato ruolo su una determinata risorsa, cioè dobbiamo definire i “privilegi”. Il “privilegio” è un tipo di accesso richiesto, e nella maggior parte dei casi sono legati alle operazioni che possiamo compiere sulla web application, come creare, modificare o eliminare delle informazioni.

Zend_Acl dispone di due metodi per impostare i privilegi: “allow()” e “deny”:

- “allow()” permette di impostare una regola di accesso
- “deny()” viene utilizzato per rimuovere o negare privilegi

Vediamo con alcuni esempi come si dovrebbe procedere:

Algorithm 43 Assegnazione di privilegi

```
[...]
        //(1)
        $acl->allow('admin', 'utente', 'list');

          //(2)
    $acl->allow('registrato', 'messaggiInterni', 'read');

        //(3)
        $acl->allow('editore', 'messaggiInterni', array('create', 'modify',
'delete');
[...]
```

La gestione dei privilegi segue la struttura gerarchica eventualmente impostata durante la configurazione dei ruoli, per cui un ruolo figlio, oltre ad avere i propri privilegi, eredita anche quelli del ruolo padre. Nell'esempio subito sopra, la seconda regola assegna agli utenti di tipo "registrato" la possibilità di eseguire la action "read" del controller "messaggiInterni", supponendo che la action "read" non faccia altro che stampare a video una lista di messaggi interni. La terza regola, invece, imposta i permessi degli utenti che rientrano nel ruolo di "editore": questi possono creare, modificare ed eliminare i messaggi interni, ma essendo figli della categoria "registrato" ereditano chiaramente anche il privilegio di leggerli, e dunque non è necessario ridefinire questa regola per un "editore". Con la prima regola invece assegnamo agli utenti con ruolo "admin" la possibilità di accedere alla funzionalità "list" del controller "utente": in altre parole, solo gli utenti "admin" potranno eseguire l'action "list" implementata dal controller "utente", e dunque vedere stampata a video la lista degli utenti registrati nella nostra web application.

Ora che abbiamo tutti gli elementi possiamo apprezzare come sia semplice, intuitivo ed efficace il meccanismo di protezione delle risorse in ZF. Grazie all'implementazione del pattern **Front Controller** possiamo impostare i nostri controlli di sicurezza in un solo posto per tutta la web application, appoggiandoci ad una serie di plugin dedicati per la gestione di queste funzionalità.

3.14 ZEND_FORM

Zend_Form è il componente introdotto dalla versione 1.5 di ZF che permette di gestire in maniera strutturata, flessibile e rapida lo sviluppo di form. Questi come sappiamo rappresentano uno degli strumenti più utilizzati per ricevere comunicazioni dagli utenti delle nostre applicazioni web: pertanto avere a disposizione uno strumento che velocizza non solo lo sviluppo del form, ma anche i controlli e le validazioni degli input, è qualcosa di estremamente comodo e apprezzato dagli sviluppatori.

Ogni programmatore web che si rispetti sa che durante lo scambio di dati questi devono essere sempre controllati, siano essi in entrata (input) o in uscita (output), per mantenere il controllo della piattaforma ed un livello di sicurezza

adeguato. **Zend_Filter** e **Zend_Validate** sono due componenti indipendenti da Zend_Form ma utilizzati da quest'ultimo proprio per filtrare e validare il flusso di dati della nostra applicazione.

I meccanismi alla base del funzionamento di Zend_Form sono molto intuitivi: ogni istanza di Zend_Form viene trattata come un contenitore di altri oggetti, nello specifico di tipo Zend_Form_Element. Esiste un oggetto Zend_Form_Element per ogni elemento HTML tipico dei form, come ad esempio la selezione, la casella di testo, l'area di testo, la checkbox e così via.

La struttura di un form in ZF è dunque relativamente semplice: un oggetto Zend_Form crea un form HTML vero e proprio dando anche accesso a tutti i suoi attributi standard, e all'interno di tale oggetto è possibile definire tanti oggetti Zend_Form_Element quanti necessari per gestire gli elementi HTML dei form. Vediamo come implementare un form per il login.

In ZF l'implementazione di un form di divide in due fasi:

- "Reindirizzare" il form
- Processare il form

Oltre alla divisione appena esposta, non vi sono ulteriori linee guida per la gestione dei form, quindi siamo liberi di organizzare il nostro codice e le nostre classi come meglio si addice alla struttura e al tipo di applicativo che dobbiamo sviluppare. Possiamo ad esempio decidere di scrivere il form completamente in HTML e affidare a ZF solo la seconda fase del processo, diversamente possiamo scegliere di implementare i nostri form come oggetti di ZF, precisamente come istanze della classe Zend_Form.

3.14.1 IL FORM VISTO COME CLASSE

In questa parte seguiremo uno degli esempi più consigliati dalle diverse letterature dedicate, e cioè quello di creare una directory application/forms in cui includeremo i nostri form.

In questa cartella dunque creiamo il file LoginForm.php con la seguente classe:

Algorithm 44 Creazione della classe per il login

```
<?php

class Form_LoginForm extends Zend_Form {
        [...]
}
```

Questa è la classe che ci permetterà di istanziare e configurare l'oggetto form. Quando il form viene costruito, la classe richiama il metodo init() ereditato da Zend_Form, ed è qui dove possiamo, ad esempio, aggiungere al form gli elementi HTML.

Per effettuare il login abbiamo bisogno di due dati in input da parte dell'utente: il nome utente e la password (le credenziali di cui abbiamo parlato in precedenza e utilizzate per il processo di autenticazione), entrambi gestiti con due caselle di testo:

Algorithm 45 Funzione di login

```
[...]
        public function init() {

                // (1) Impostiamo gli attributi "action", "method" e "id" del form
                $this->setAction('/login/submit/')
                        ->setMethod('post')
                        ->setAttrib('id', 'loginForm');

                /********************** Textbox nome utente ************************/
                // (2) creiamo un elemento di tipo "text".....
                $nomeutente = $this->createElement('text', 'username');

                //...gli associamo una etichetta
                $nomeutente->setLabel('Nome utente:');

                // (3) impostiamo i filtri e i validatori per questo elemento HTML
                $nomeutente->setRequired(TRUE)
                             ->addFilter('StringTrim')
                             ->addValidator('alnum');

                //Impostiamo un messaggio di errore personalizzato
                $nomeutente->getValidator('alnum')->setMessage('Inserisci nel nome solo lettere e numeri');

                $nomeutente->setAttrib('size', 35);

                //Aggiungiamo l'oggetto appena creato nel nostro form
                $this->addElement($nomeutente);

                /********************** Textbox password ************************/

                $password = $this->createElement('password', 'password');
                $password->setLabel('Password:');
                $password->setRequired(TRUE)
                          ->addValidator('stringLength', true, array(8))
                          ->addFilter('StringTrim');

                $password->getValidator('stringLength')->setMessage('La password deve contenere 8 caratteri');

                $this->addElement($password);

                //Aggiungiamo al form il pulsante di invio
                $this->addElement('submit', 'submit', array('label' => 'Entra'));

        }
[...]
```

Vediamo ora di illustrare, cercando di darne una più esaustiva spiegazione, i passi fatti nel precedente frammento di codice per creare un semplice form di login:

1. La prima cosa che abbiamo configurato sono alcuni degli attributi fondamentali per il funzionamento del form stesso. Abbiamo quindi impostato l'azione che si occuperà di processare il form (nel nostro caso /login/submit/), il metodo (method) con cui inviamo i dati al server (nel nostro caso utilizzeremo il metodo POST piuttosto che il GET) ed infine l'attributo identificatore "id".

2. Il passo successivo è stato quello di creare il campo di input di tipo testuale per l'inserimento del nome utente. Abbiamo utilizzato il metodo createElement di Zend_Form passandogli come parametri il tipo di elemento che vogliamo creare (text nel nostro caso) ed il suo identificativo (username), utile per poter recuperare dal lato server il valore inserito dall'utente. Il processo di creazione continua impostando una etichetta per il campo, in modo da fornire all'utente un adeguato feedback sul campo che sta compilando.

Il risultato finale in codice HTML è simile al seguente:

Algorithm 46 Form per l'accesso dell'utente

```
<form id="loginForm" enctype="application/x-www-form-urlencoded" action="/ZFHtmlIt/public/login/submit"
... method="post">
        <div id="loginDiv">
                <ul>
                        <li>
                        <label for="username" class="leftalign required">Nome utente</label>
                        <input type="text" name="username" id="username" value="" />
                        </li>
                        <li><label for="password" class="leftalign required">Password </label>
                        <input type="password" name="password" id="password" value="" /></li>
                        <li class="submit">
                        <input type="submit" name="Entra" id="Entra" value="Entra" />
                        </li>
                </ul>
        </div>
</form>
```

Oltre alla creazione e configurazione base, nel codice PHP visualizzato sopra, possiamo anche notare che ad ogni elemento abbiamo associato anche un filtro e un validatore: questo sarebbe stato il terzo punto (3.) della precedente spiegazione, ma vista l' importanza di tale argomento si è deciso di dedicargli un apposito peragrafo.

3.14.2 FILTRAGGIO E VALIDAZIONE DELL' INPUT

La validazione e il filtraggio dell'input sono due funzionalità indispensabili e da implementare con molta attenzione nelle web application perchè in questo contesto mettiamo in gioco gran parte della sicurezza dei nostri applicativi.

Come abbiamo visto nella parte introduttiva Zend_Filter e Zend_Validation, anche se indipendenti, sono stati integrati in Zend_Form, proprio per facilitare la gestione di questi meccanismi.

Nel codice in cui abbiamo creato il form abbiamo esteso i campi "nome utente" e "password" per includere il filtraggio e la validazione attraverso le interfacce addFilter() e addValidator(). Andando a prendere uno stralcio del codice precedente abbiamo rispettivamente:

Algorithm 47 Filtraggio e validazione dell' input dell'utente

```
       [...]
//Impostiamo i filtri e i validatori per il campo "username"
$nomeutente->setRequired(TRUE)
            ->addFilter('StringTrim')
            ->addValidator('alnum');

//Impostiamo un messaggio di errore "customizzato"
$nomeutente->getValidator('alnum')->setMessage('Il nome utente può contenere solo lettere e numeri');

[…]
//Impostiamo i filtri e i validatori per il campo "password"
$password->setRequired(TRUE)
         ->addFilter('StringTrim')
         ->addValidator('stringLength', true, array(8));

//"Customizziamo" il messaggio
$password->getValidator('stringLength')->setMessage('La password deve contenere 8 caratteri');
       [...]
```

Come possiamo osservare innanzitutto rendiamo obbligatori entrambi i campi attivando la direttiva setRequired: infatti per effettuare il login gli utenti devono necessariamente inviare sia un nome utente che una password validi.

A questo punto impostiamo i filtri: in entrambi i casi impostiamo quello che elimina gli spazi nelle stringhe di testo (StringTrim), ma ZF fornisce tutta una serie di filtri già implementati che possiamo utilizzare a seconda delle necessità, come ad esempio Anum, Alpha, Digits, ecc. che restituiscono una stringa eliminando rispettivamente tutti i caratteri tranne quelli alfanumerici, quelli alfabetici e le cifre.

Attraverso il metodo addValidator() invece impostiamo i validatori: per il campo username abbiamo impostato un validatore che permette l'inserimento solo di caratteri alfanumerici, mentre per il campo password abbiamo impostato un validatore che non accetta stringhe inferiori agli 8 caratteri. Per finire, in entrambi i casi grazie al metodo setMessage() abbiamo personalizzato il messaggio di feedback che riceve l'utente nel caso in cui il rispettivo validatore riscontri dei valori non accettabili.

ZF ci fornisce la possibilità di scrivere filtri e validatori in base alle specifiche necessità dei nostri form nel caso in cui quelli di default non sia sufficienti.

3.14.3 VISUALIZZAZIONE DEL FORM

Vediamo ora il codice per la visualizzazione e l'elaborazione del form.

Un passo fondamentale è quello di configurare l'autoloader per impostare il *namespace* **Form_** e collegarlo alla directory che abbiamo creato apposta per contenere i nostri form. In breve, è come se dovessimo "attivare" la directory /application/forms/.

Nel file application/Bootstrap.php dunque aggiungiamo il seguente metodo:

Algorithm 48 Funzione per configurare l'autoloader

```
protected function _initAutoload() {
        // Add autoloader empty namespace
        $autoLoader = Zend_Loader_Autoloader::getInstance();
        $resourceLoader = new Zend_Loader_Autoloader_Resource(array(
            'basePath'  => APPLICATION_PATH,
            'namespace' => '',
            'resourceTypes' => array(
                'form' => array(
                    'path'=> 'forms/',
                    'namespace' => 'Form_',
                )
            ),
        ));
        // viene restituto l'oggetto per essere utilizzato e memorizzato nel bootstrap
        return $autoLoader;
}
```

Questo metodo viene chiamato per configurare l'autoloader, che a questo punto saprà come leggere e trattare le classi contenute nella directory application/forms e che hanno il nome che inizia con il prefisso Form_.

Nel controller Login (file application/controllers/LoginController.php) aggiungiamo la action che si occupa di gestire la richiesta di visualizzazione del form, che potrebbe essere simile alla seguente:

Algorithm 49 Funzione per la richiesta di visualizzazione

```
[...]
    public function formAction() {

        //Creo un oggetto form configurato come abbiamo precedentemente visto
        $frmLogin = new Form_LoginForm();

        //Assegnamo alla view una istanza del nostro oggetto form
        $this->view->form = $frmLogin;
    }
[...]
```

Passiamo ora ad analizzare il codice della view creando il file form.php nella

directory application/views/scripts/login/).

Algorithm 50 Visualizzazione del form

```
    [...]
    <h2>Area Riservata</h2>
    <p>Per accedere all'area riservata effettuare il login:</p>
    <?php
        echo $this->form->render();
        // echo $this->form; produce lo stesso risultato
    ?>
[...]
```

Come possiamo vedere questa view è molto semplice: non facciamo altro che visualizzare il form così come è stato configurato nei passi precedenti.

3.14.4 ELABORAZIONE DEL FORM

Per chiudere il cerchio rimane infine da gestire l'elaborazione del form stesso. Riprendiamo il metodo formAction() scritto poc'anzi nel controller Login, e lo estendiamo come segue:

Algorithm 51 Elaborazione del form su formAction()

```
[...]
    public function formAction() {
        //Creo un oggetto form configurato come abbiamo precedentemente visto
        $frmLogin = new Form_LoginForm();
        //Verifichiamo che la richiesta sia di tipo POST
        if ( $this->getRequest()->isPost() ) {
           //Se il form è valido, lo processiamo
           if ( $frmLogin->isValid($_POST) ) {
                //recuperiamo i dati così .....
                $nomeUtente = $this->getRequest()->getParam('username');
                $password      = $this->getRequest()->getParam('password');
                //o così, a seconda di come ci troviamo più comodi
                //$dati = frmLogin->getValues();
                // processiamo la richiesta
           }
        }
        //Assegnamo alla view una istanza del nostro oggetto form
        $this->view->form = $frmLogin;
    }
[…]
```

Andiamo ad analizzare la serie di controlli effetuati sui dati inseriti dall'utente:

- Il primo controllo verifica che la richiesta ricevuta dal controller Login sia stata effettuata come ci aspettavamo attraverso il metodo POST.

- Il controllo successivo invece verifica che il form sia stato validato correttamente.

Se risultano verificati entrambi i controlli si può procedere con il processare i dati ricevuti, nel nostro caso il nome utente e la password.

Come vediamo dal codice, possiamo recuperare i parametri inviati dal form in due modi: singolarmente attraverso il metodo getParam() dell'oggetto "richiesta" (request) o utilizzare il metodo getValues del form che li recupera tutti insieme restituendoli in un'unica struttura dati.

3.14.5 CONCLUSIONI

L'esempio che abbiamo visto tratta una minima parte delle funzionalità e delle caratteristiche dei componenti Zend_Form, Zend_Filter e Zend_Validate. Non possiamo approfondire oltre il discorso in quanto lo scopo di questa trattazione non è quello di far apprendere in dettaglio, partendo da zero, ogni singolo componente di questo framework perchè la trattazione diverrebbe pesante soprattutto per un programmatore alle prime armi con ZF sortirebbe l'effetto contrario di quello desiderato. Qello che ci si vuole prefiggere con queta tesi è quello di far capire, con pochi rudimenti basilari, quanto sia veloce, semplice e performante progettare applicazioni con ZF. Però è mia opinione che valga la pena fare altre due brevissime osservazioni in merito:

- La prima riguarda i "decoratori": questi particolari oggetti permettono di personalizzare praticamente ogni caratteristica dell'aspetto dei form e dei loro sotto-elementi. Attraverso i decoratori e qualche riga di CSS si possono creare degli effetti davvero interessanti.

- La seconda riguarda la gestione grafica del form. ZF è predisposto per supportare alcuni dei più utilizzati framework javascript dedicati allo sviluppo di applicationi RIA (*Rich Internet Application*). Ad esempio il supporto a Dojo è praticamente nativo: all'interno di ZF infatti è disponibile un componente, Zend_Dojo, attraverso il quale possiamo far interagire i due framework. Potremmo ad esempio riscrivere il form che abbiamo visto utilizzando Zend_Dojo per la creazione delle caselle di testo, ottenendo così i tipici effetti delle interfacce utente di ultima generazione.

4 REALIZZAZIONE DI UN' APPLICAZIONE FUNZIONANTE

In questo capitolo si andrà a costruire un' applicazione web per creare l' accesso di un utente registrato, o quanto meno la struttura base di questa operazione, tramite una password. L' idea è quella di realizzare tale procedura utilizzando dapprima solo PHP per poi sviluppare la medesima operazione impiegando Zend Framework in modo tale da far vedere quali possano essere le varie differenze procedurali e funzionali dei due approcci. In questo modo saremo in grado di trarre le nostre conclusioni, che presenteremo nel prossimo capitolo (5), sulle potenzialità o lacune, nonchè i vantaggi/svantaggi, che un programmatore potrebbe incorrere nel caso decidesse di usare ZF per costruire le sue applicazioni.

Qui di seguito viene riportato uno schema dell' architettura che dovrà assumere il nostro progetto per un corretto funzionamento:

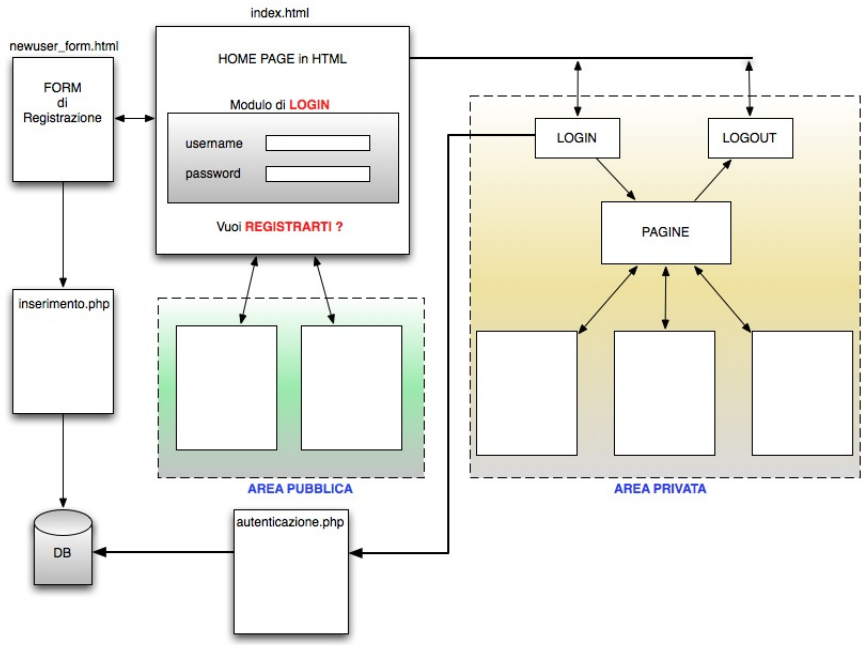

Figure 15: Schema di progetto

Perciò la parte che andremo a sviluppare comprenderà una HOME PAGE nella quale l' utente dovrà scegliere se registrarsi sul nostro sito oppure, se lo è già, effettuare il login. A seconda della scelta fatta verra aperto un form per l' iserimento dei dati del caso richiesto che referenzieranno rispettivamente un file PHP: inserimento.php, autenticazione.php. Questi due file si occuperanno di:

- Manipolare i dati in modo che possano essere correttamente elaborati dal

database,

- interfacciarsi correttamente con il database per effettuare le operazioni richieste dall' utilizzatore attraverso delle query di inserimento o recupero dati.

Infine quando l' utente deciderà potrà effettuare il LOGOUT richiamando un file sempre in PHP (logout.php) che si occuperà di chiudere le connessioni attive e di conseguenza modificare opportunamente le informazioni sulla base dati.

4.1 COSTRUZIONE DEL DATABASE

La prima cosa da fare, una volta che si ha ben chiaro che cosa deve fare l' applicazione che stiamo sviluppando e come vogliamo che funzioni, è la creazione del database. Questa struttura ha il compito di immagazzinare tutte le informazioni che sono di interesse per la gestione della nostra applicazione. Nello specifico noi siamo interessati esclusivamente a raccogliere informazioni riguardanti gli utenti che accedono al nostro sito che possono essere sia di natura personale, in modo da poter individuare un particolare utente, sia di natura funzionale in modo da mantenere una sorta di bilancio statistico sull 'intero sistema. Per questo motivo la nostra base dati risulta essere abbastanza semplice, infatti comprende soltanto due componenti, o se vogliamo tabelle. Abbiamo utilizzato il termine "abbastanza semplice" dal momento che la complessita di un argomento dipende essenzialmente da due fattori: da quanto sia vasto e da quanto dettagliatamente lo si vuole decrivere.

Dopo questa delucidazione puramente semantica riportiamo lo schema relazionale (logico) per il nostro database:

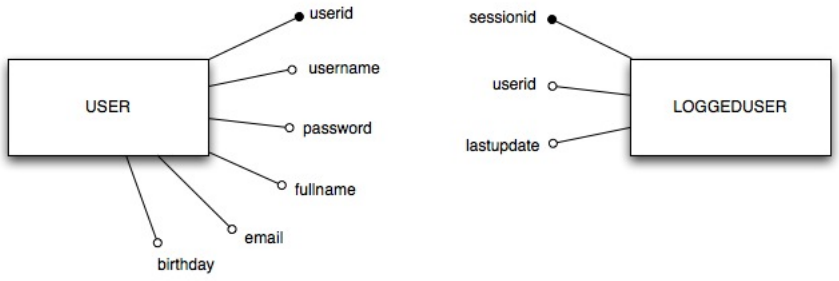

Figure 16: Schema E-R

A questo punto possiamo creare le tabelle che costituiscono il nostro database con il codice SQL che viene riportato qui di seguito:

```
create table users (
        userid int auto_increment primary key,
        username varchar(50) not null,
        pasword varchar(50) not null,
        fullname varchar(150),
        email varchar(200) not null,
        birthday date,
        index (username)
);
```

Figure 17: Struttura tabella user

```
create table loggeduser (
        sessionid varchar(100) primary key,
        userid int not null,
        lastupdate datetime not null
);
```

Figure 18: Struttura tabella loggeduser

Ora che il database è creato e pronto ad essere utilizzato possiamo passare alla fase di progettazione del programma vero e proprio che occuperà di ricevere e salvare i dati inseriti dall' utente per poi utilizzarli successivamente per effettuare l' operazione di login.

4.2 SVILUPPO CON PHP

Come già accenato all' inizio di questo capitolo l' applicazione di cui si è parlato verrà realizzata in due versioni. In questa sezione descriveremo il funzionamento di quella implementata solo con l' utilizzo di PHP. Chiaramente, vista la semplicità dell' operazione trattata, non utilizzeremo il paradigma ad oggetti in quanto per il linguaggio PHP non è necessario implementarlo, di congeguenza l' architettura che ne deriverà sarà costituita da un insieme di file correlati fra loro. Ricordando lo schema di figura 3, il quale indica la linea guida da seguire per poter completare il nostro applicativo, possiamo osservare che il primo file che viene utilizzato è l' index.html. Non è un caso che questo sia il primo file ad essere caricato in quanto funge da loader per l' intero programma; ciò significa che qualsiasi web application per poter essere eseguita dovrà iniziare con un file denominato index.html. Il nostro file index.html, oltre a caricare l' intero programma, ha il compito di dare il benvenuto all' utente e mettere a disposizione due scelte, come viene mostrato nella figura qui di seguito:

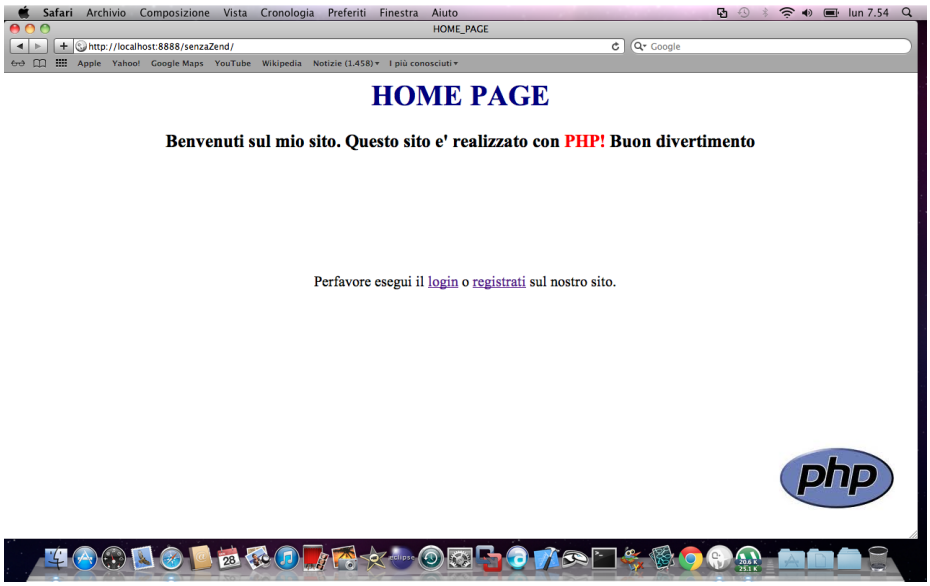

Figure 19: Pagina iniziale

Come si può osservare la pagina di benvenuto risulta essere estremamente semplice anche per un principiante in materia tanto che non si ritiene necessario approfondirne ulteriormente ne la costruzione ne la funzionalità. Al primo avvio l' unica operazione possibile, per un utente che non sia già presente nel database dell' applicazione, è la registrazione; quindi attraverso il link presente nella pagina di benvenuto viene richiesto il file newuser_form.html che apre la pagina di registrazione. In questa pagina si trovano una serie di campi che l' utente deve copilare per poter effetuare con successo la registrazione e così accedere all' area privata del sito. Come possiamo osservare nella prossima figura nella pagina di registrazione le uniche operazioni possibili sono, a parte la compilazione dei campi, l' invio dei dati inseriti o la loro cancellazione.

Figure 20: Pagina di Registrazione

Nemmeno questa pagina di per se presenta delle notevoli difficoltà implementative; però si ritiene necessario ai fini della trattazione mostrare come viene costruito un form per l' inserimento di dati e conseguentemente il metodo di trasmissione realizzato con PHP. Così, mettendo in luce questo procedimento, risulterà più semplice confrontare le differenze presenti tra i due metodi in esame:

Algorithm 52 Costruzione form in PHP

```
<form method="GET" action="inserimento.php" align="center">
        <table><tr><td><label>username<span> *</span> :</label></td>
                    <td><input name="username" type="text" size="40" maxlength="50"/></td></tr>
                <tr><td><label>password<span> *</span> :</label></td>
                    <td><input type="password" maxlength="50" size="40" name="password" /></
td></tr>
                <tr><td><label>full name :</label></td>
                    <td><input name="fullname" type="text" size="40" maxlength="150" /></td></
tr>
                <tr><td><label>e-mail<span> *</span> :</label></td>
                    <td><input name="email" type="text" size="40" maxlength="200" /></td></tr>
                <tr><td><label>data di nascita :</label></td>
                    <td><input name="birthday" type="text" size="40" maxlength="10" value =
"aaaa/mm/gg"/></td></tr>
                    <tr><td><button type="submit">Vai</button> <button type="reset">Cancella</
button></td></tr>
        </table>
</form>
```

Procedendo con ordine da questo frammento di codice possiamo vedere che il metodo di trasmissione utilizzato è il metodo GET anzichè il metodo POST, così da minimizzare ulteriormente gli sforzi di programmazione. Invece per quanto riguarda i campi di inserimento a colpo d' occhio si osserva che i con-

trolli riguardano principalmente le lunghezze, intese sia come lunghezze dell' input visualizzato, sia come lunghezze massime dei campi da inserire. Infatti in questa fase i controlli riguardano se non altro l' estetica più che mirare ad un' ottimizzazione dell' input per un uso sucessivo, che sono comunque possibli a scapito però di un appesantimento del codice con una conseguente perdita di chiarazza.

Una volta compilati i vari campi i dati vengono inviati al file inserimento.php che ha il compito di controllarli ed elaborarli per poterli inserire all' interno del database così da ultimare la fase di registrazione. Siccome però non esiste una funzione in PHP che possa fare esattamente quello che ci serve e come lo vogliamo dobbiamo accontentarci di utilizzare le numerose funzioni messe a disposizione e cercare di adattarle alle nostre esigenze. Per quanto ci riguarda possiamo distinguere due tipi di controllo per le due tipologie di dati presenti nel nostro form:

- obbligatori
- non obbligatori.

Il primo tipo è destinato ai campi fondamentali per la gestione del database e per poter individuare ciascun utente che si connetta sul nostro sito come avviene per lo username del quale viene presentato il seguente frammento di codice:

Algorithm 53 Controllo sull 'input del tipo obbligatorio

```
if($_GET['username'] == ''){
        // ...
        // Errore. Il campo username e' vuoto. Devi inserire un username
}
elseif(strpbrk($_GET['username'], ' ,£,$,%,&,/,(,),=,?,^,>,<,!,\,ç,@,#,+,*,[,],§,°')){
        // ...
        // Errore. Devi inserire una username corretta
}
else{
        // Successo. Recupero il dato
        $username = $_GET['username'];
}
```

Come si può osservare ci sono due modi in cui questo campo genera errori: il primo è ovviamente quando non viene inserito nulla da parte dell' utente, mentre l' altro avviene quando il dato non rispetta le specifiche desiderate dal realizzatore del sito stesso; altrimenti se le precedenti condizioni vengono rispettate l' input viene recuperato dalla variabile globale GET e predisposto per essere elaborato successivamente.

Il secondo tipo invece è destinato ai campi non necessari al funzionamento del database, come possono essere in questo caso i dati anagrafici, in merito ai quali viene riportato l' esempio del nome completo dell' utente:

Algorithm 54 Controllo sull' input del tipo non obbligatorio

```
if($_GET['fullname']!=''){
        // Recupero il dato
        $fullname=$_GET['fullname'];
        if(!preg_match('/^[a-zA-Z]/', $fullname)){
                // ...
                // Errore. Devi inserire un nome valido
        }
}
```

Dall' analisi di questo breve frammento di codice, la prima differenza che si nota subito rispetto al caso precedente è il recupero dell' informazione, se presente, prima di effettuare l' eventuale controllo. Diversamente da quanto succede per un dato del tipo obbligatorio questa tipologia può dare errore soltanto quando il campo non rispetta le condizioni di validità.

Dal momento che non siamo in grado di effettuare i controlli e visualizzarne gli errori nella stessa pagina del form di registrazione si è adottata la specifica di progetto di non bloccare l' analisi degli input appena uno dia esito negativo per notificarlo, ma di proseguire con la verifica fino alla fine e solo allora notificare tutti gli errori occorsi. Per realizzare questa direttiva,come si può vedere, si è utilizzato un contatore (count) e un array (message):

Algorithm 55 Struttura per il controllo dell' input

```
$count=0;
$message = array("","","","","");

if($_GET['username'] == ''){
        $count=$count+1;
        $message["1"] ="Il campo username e' vuoto. Devi inserire un username<br>";
}
elseif(strpbrk($_GET['username'], ' ,£,$,%,&,/,(,),=,?,^,>,<,!,\,ç,@,#,+,*,[,],§,°')){
        $count=$count+1;
        $message["1"] ="Devi inserire una username corretta<br>";
}
else{
        $username = $_GET['username']; }
// ...
// Altri controlli sugli input dell' utente
// ...
if($_GET['fullname']!=''){
        $fullname=$_GET['fullname'];
        if(!preg_match('/^[a-zA-Z]/', $fullname)){
                $count=$count+1;
                $message["3"] ="Devi inserire un nome valido<br>";
        }
}
// ...
// Ancora altri controlli
// ...
if($count=='0'){
        // Successo. Connessione al database per l' inserimento del nuovo utente
        // ...
}
else{
        // Errore. Il Form di registrazione non è stato compilato correttamente
        // ...
}
```

In questo modo aggiornando di quando in quando il contatore e l' array ci troveremo a disposizione degli strumenti in grado di fornirci informazioni riguardo a quanti e quali errori sono stati commessi dall' utente al momento della registrazione. Un lettore attento potrebbe volgere l' attenzione sulla discutibile numerazione impiegata per aggiornare l' array nei campi obbligatori osservando che per lo username, come mostrato nell' esempio, viene utilizzato due volte lo stesso identificativo. Questo tuttavia non non si ritiene uno sbaglio in quanto è impossibile, per come sono stati creati i campi, che uno di loro presenti entrame le tipologie di errore; potrebbe esserlo per un progetto molto più elaborato il quale necessita di un numero considerevole di tipologie d' errore e conseguentemente anche di una struttura adeguata di contenimento, ma per il livello al quale siamo interessati questa soluzione resta comunque valida. Quindi una volta terminata l' analisi di tutti gli inputnon resta da verificare il valore assunto dal contatore. Se questo è diverso da zero significa che i dati presentano uno o più anomalie, pertanto non resta altro se non informare l' utente del fallimento dell' operazione di registrazione comunicandogli quali errori ha commesso. Ciò può essere fatto agilmente come viene mostrato qui di seguito attraverso la visualizzazione dell' array contenente i tipi d' errore utilizzando un banalissiomo ciclo for:

Algorithm 56 Visualizzazione errori

```
for ($i = 1; $i <= 5; $i++) {
        if($message["$i"] != ""){
                echo $message[$i];
        }
}
```

Chiaramente tale procedura è resa molto semplice grazie alle strutture ausiliarie che si sono impiegate, il contatore e l' array appunto, nonche anche dalla particolare numerazione utilizzata per aggiornarlo. come del resto viene raffigurato. A questo punto viene riportata la schermata visualizzata nella condizione d' errore nella quale oltre a mostrare la lista degli errori commessi mette a disposizione un link per poter rieffettuare la la procedura di registrazione qualora l' utente lo desiderasse.

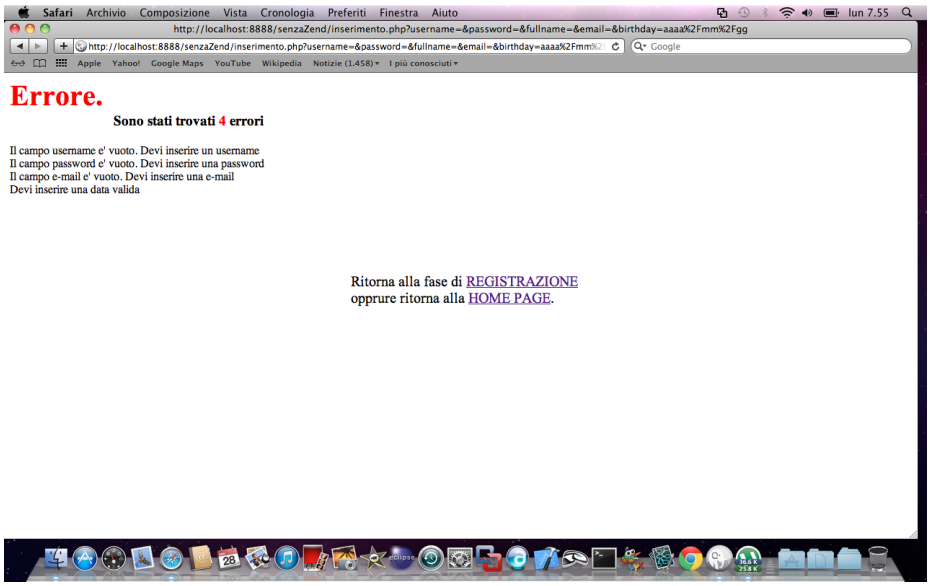

Figure 21: Pagina d' errore

Contrariamente a quanto detto per il contatore in precedenza, se questo risulta uguale a zero significa che è rimasto inalterato dalla condizione iniziale per cui non sono stati commessi errori; quindi possiamo effettuare tranquillamente la connessione al database in questo modo consapevoli di avere dei dati validi da inserire

Algorithm 57 Connessione al database

```
$db = mysql_connect($my_host, $my_username, $my_password) or die ("Sever error connection");
      mysql_select_db($db_name,$db) or die ("Database error connection");
```

A questo punto se non ci sono errori di connessione, che non dipendono certo dal codice che abbiamo prodotto, possiamo eseguire la seguente query d' inserimento dati nella tabella user:

Algorithm 58 Query di registrazione

```
$query="INSERT INTO users (username, password, fullname, email, birthday) VALUES
('$username','$pwd','$fullname','$email','$birthday')";

$result=mysql_query($query,$db);
if(!"$result"){
        print("Errore nell' inserimento dei dati nel database");
}else{
        mysql_close($db);
        // ...
}
```

Se tutto va a buon fine, a meno di un errore interno al database, l' operazione di resistrazione è avvenuta correttamente come mostra la schermata seguente ed all' interno della tabella user precedentemente creata per la base di dati è presente un nuovo record con i dati appena inseriti dall' utente, il quale rappresenta un profilo sul cui poter eseguire l' atra operazione messa a disposizione per questa trattazione: il login.

Figure 22: Pagina di registrazione avvenuta

Una volta terminata la registrazione finalmente possiamo effttuare l' operazione di login cliccando direttamente sul link visualizzato nella fase di avvenuta registrazione anzichè tornare all' home page. Come si può notare la pagina che viene aperta, contenente il form per effettuare questa operazione, è per tutto ed in tutto simile a quella già vista in fase di registrazione con le ovvie differenze. Pertanto non si ritiene necessario dilungare oltre il discorso con un' analisi che rispecchierebbe quella già sostenuta in precedenza per la fase di registrazione.

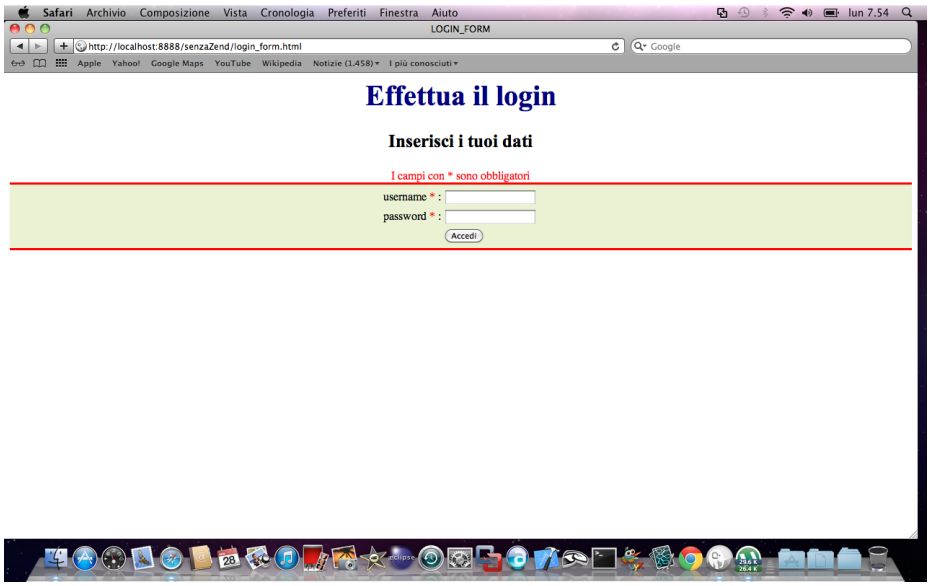

Figure 23: Pagina di login

I dati stavolta vengono inviati al file autenticazione.php il quale ha il medesimo compito del file inserimento.php, cioè controllare ed elaborare i dati affinchè l' operazione di login possa essere portata a termine con successo. Quindi una volta controllata la validità dei dati inseriti con la medisima modalità vista per l' operazionedi registrazione, la quale fornisce lo stesso output d' errore nel caso in cui gli input non rispettino le condizioni prestabilite, possiamo interrogare la base dati con la seguente query di ricerca in modo da trovare il record desiderato:

Algorithm 59 Query di ricerca utente

```
$query="SELECT * FROM users WHERE username='$username' AND password='$pwd'";
$result=mysql_query($query);
$count=mysql_num_rows($result);
$array_result=mysql_fetch_row($result);
$id=$array_result[0];
```

Come si può vedere dal frammento di codice l' effetto di questa query non è solo quello di interrogare la base dati, ma anche recuperare l' id del record trovato nel caso positivo, il quale sarà necessario come vedremo nel segito della relazione per poter terminare correttamente la fase di login. Inoltre si è utilizzato un contatore che ha lo scopo di contenere questa volta il numero di record, quindi profili utente, che hanno dato esito positivo alla query appena eseguita e che pertanto sono candidati all' operazione di login. Chiaramente per una base dati che si rispetti i valori assumibili da questo contatore sono soltanto l' 1 e lo 0, che rispettivamente rappresentano il caso successo ed insuccesso; se invece si presentasse un qualsiasi altro valore, ovviamente maggiore di 1 in quanto

parlare di valore negativo non ha senso, significa che sono presenti più record con la coppia username/password identica cosa che renderebbe inaffidabile il database ed inconsistenti i dati in esso registrati. Quindi utilizzando questo contatore in una struttura di controllo siamo in grado di gestire agilmente le due possibilità citate sopra come mostrato:

Algorithm 60 Struttura per la verifica della ricerca

```
if ($count==1){
        // Utente trovato
        // Completamento login => Creazione/Aggiornamento dati in loggedusers
        // Reindirizzamento area riservata
}
else{
        // Nessun riscontro trovato. Utente non presente nel database
}
```

Ora non dobbiamo fare altro che gestire entrambi gli eventi presentati precedentemente. Il più semplice da trattare è certamente il caso in cui il contatore sia uguale a 0 il che significa, come del resto già accennato, che nel nostro database non sono presenti record corrispondenti ai parametri richiesti; precui non resta altro da fare se non informare l' utente di un' insuccesso come viene mostrato ad esempio nella figura sottostante:

Figure 24: Pagina di utente mancante

Il caso con il contatore uguale a 1 è leggermente più complicato in quanto si deve curare l' inserimento dei dati nella tabella degli utenti loggati, creare le variabili di sessione e reindirizzare l' utente nella pagina riservata. Per poter

realizzare questo dobbiamo dapprima capire se l' utente che sta tentando l' accesso in questo momento si è già "loggato" in precedenza oppure se è la sua prima volta attraverso l' esecuzione di questa semplice query:

Algorithm 61 Query di ricerca utente "loggato"

```
$query1="SELECT * FROM loggedusers WHERE userid='$id'";
$result1=mysql_query($query1);
$count1=mysql_num_rows($result1);
```

Come si sarà già capito anche questa volta viene usato un contatore che ha la medesima caratteristica e funzione di quello già impiegato nella fase di ricerca del profilo richiesto dall' utente. Pertanto a seconda del valore assunto dal contatore, anche questa volta, 0 oppure 1 si dovrà rispettivamente inserire un nuovo record o aggiornarne uno già esistente come viene mostrato qui di seguito:

Algorithm 62 Inserimento/Aggiornamento tabella loggedusers

```
if($count1==1){
        $query2="UPDATE loggedusers SET lastupdate='$date' WHERE userid='$id'";
        $result2=mysql_query($query2);
}else{
        $query3="INSERT INTO loggedusers (userid,sessionid,lastupdate) VALUES('$id',
'$username','$date')";
        $result3=mysql_query($query3);
}
```

Arrivati a questo punto dobbiamo creare le variabili di sessione da poter recuperare una volta aperta la pagina riservata così da non perdere le informazioni dell' utente che si è collegato sul nostro sito. Perciò qui sotto riportiamo la struttura base per poter utilizzare le sessioni in PHP, nonchè la posizione esatta all' interno del nostro codice delle istruzioni usate per crearene le variabili:

Algorithm 63 Utilizzo delle sessioni

```
// Apertura di una sessione da usare prima di qualsiasi output a video
session_start();
// ...
// corpo del programma
// ...
if ($count==1){
        // ...
        $_SESSION["username"] = $_GET['username'];
        $_SESSION["pwd"] = $_GET['password'];
        // ...
}
// ...
// Chiusura della sessione
session_write_close();
```

Come si può osservare usare le sessioni è, almeno a questo livello, abbastanza semplice ad eccezione dell' istruzione che ne determina l' apertura la quale va

inserita assolutamente prima di qualiasi output a video, questa è un' unica condizione sulla posizione da rispettare per poterla usare correttamente. Un ultima accortezza in merito a questo fatto riguarda proprio gli output o quello che in PHP viene considerato tale, infatti anche uno spazio all' inerno dello stesso codice viene considerato come un oggetto da visualizzare.

L' ultima cosa che vediamo a riguardo della procedura di login e precisamente all' interno del file autenticazione.php è l' istruzione impiegata per il reidirizzamento alla pagina riservata, che abbiamo chiamato pagina.php, una volta che il login è avvenuto con successo:

Algorithm 64 Istruzione di reindirizzamento

```
header ("location: pagina.php");
```

Così viene aperta finalmente l' ultima pagina dela nostra applicazione web la quale presenta il seguente aspetto:

Figure 25: Pagina riservata

L' unica cosa degna di nota in questa pagina riguarda il controllo delle variabili di sessione. Infatti, come viene presentato qui di seguito, una volta aperto il file pagina.php se le variabili che andiamo a recuperare non esistono significa che nessuno ha effettuato il login per accedere a quest' area e quindi l' utente in questione non possiede i diritti per visualizzarne il contenuto, pertanto viene immediatamente reindirizzato con la consueta istruzione all' home page senza visualizzare nulla dell' area riservata.

Algorithm 65 Controllo delle variabili di sessione

```
// Prima è necessario aprire la connessione
session_start();
if (!isset($_SESSION['username']) && !isset($_SESSION['pwd']))
{
        header("location:index.html");
        exit;
}
```

Contrariamente se questo controllo da esito positivo invece è possible recuperare i valori delle suddette variabili con una semplicissima assegnazione che vi mostriamo:

Algorithm 66 Recupero variabile di sessione

```
$user = $_SESSION['username'];
```

Giunti a questo punto non ci rimane altro che analizzare l' operazione di logout; nella pagina riservata infatti è presente il link per poterla effettuare. Ovviamente è solamente in quest' area che è presente questa procedura in quanto è l' unica che necessita dell' operazione di login per poervi accedere. Una volta lanciata l' esecuzione viene aperto il file logout.php del quale si riporta l' intero algoritmo tanto sono così poche le righe di codice che lo compongono:

Algorithm 67 Operazioni per effettuare il logout

```
// Prima è necessario aprire la connessione
session_start();
if (isset($_SESSION['username']) && isset($_SESSION['pwd'])){
        session_unset();                // Libero le variabili
        session_destroy();              // Distruggo la sessione
        session_write_close();          // Chiudo la sessione
        session_regenerate_id(true);    // Rigenero l' id per le sessioni future
        header ("location:index.html");
}
```

Com' è evidente questa operazione non visuallizza alcunchè in quanto il suo unico scopo è quello di disconnettere l' utente e predisporre la sessione ad un prossimo accesso. Un' ultima cosa sulla quale si vuole porre l' attenzione del lettore è la condizione di esecuzione di tale procedura, infatti se non esistessero le variabili di sessione che ci aspettiamo di aver usato nel corso dell' applicazione non sarebbe necessario effettuare il logout in quanto non sarebbe presente nessun utente da disconnettere e reidirizzare all' home page.

Così, sperando di essere stati sufficientemente esaustivi nel discorso, siamo giunti alla conclusione della trattazione per quanto riguarda la costruzione di una piccola *web application* realizzata usando il linguaggio PHP che implementa le funzioni di *signin*, *login* e *logout*. Tutto il codice completo di ciò che è stato mostrato in questo paragrafo sarà presentato nell' appendice A.

4.3 SVILUPPO CON ZEND

In questo paragrafo vediamo come si può costruire la stessa applicazione descritta precedentemente utilizzando *Zend framework.* Una volta che il software è correttamente installato all' interno del nostro sistema, processo per il quale nella bibliografia è presente un riferimento ad un blog davvero utile in merito, possiamo procedere con l' implementazione della nostra *web application.* Quindi non ci resta altro che creare il nostro *Zend project* inserendo l' istruzione dalla shell di comando come viene mostrato nella figura seguente:

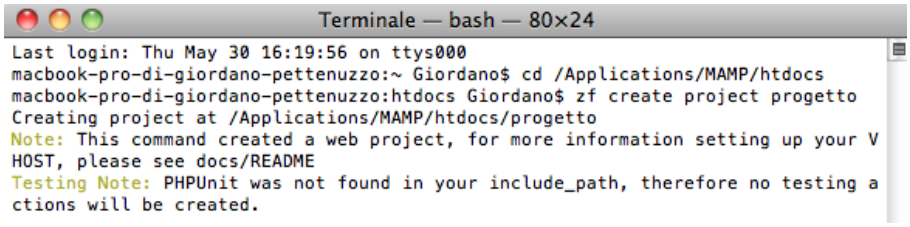

Figure 26: Creazione di un progetto con Zend

Con l' istruzione appena eseguita si ottiene il seguente risultato:

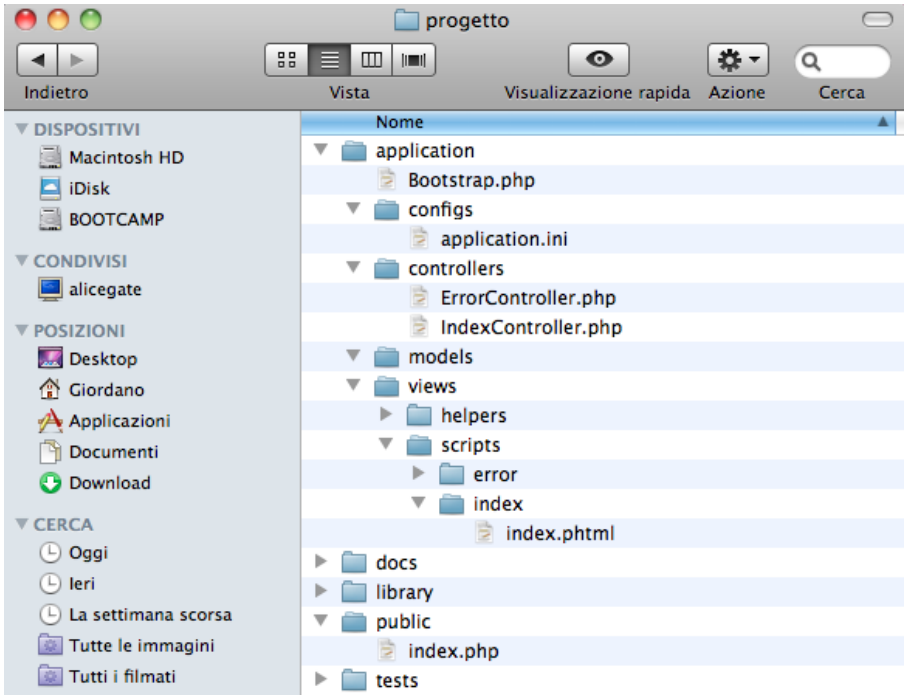

Figure 27: Risultato della creazione del progetto

Per una maggiore comprensione si consiglia, soprattutto per la fase iniziale, di leggere comunque il blog sopracitato in quanto il comando che viene utilizzato per la creazione del nostro progetto, come si può vedere nella precedente immagine, deriva dall' installazione del framework seguendo passo passo le istruzioni ivi presentate. Ci sono molte istruzioni simili a quella che abbiamo utilizzato noi per realizzare il progetto che possono essere applicate direttamente dalla shell di comando, non solo per la creazione di controller o viste ad esempio, ma anche allo stesso modo per l' abilitazione/disabilitazione e configurazione di parametri e funzioni tanto che un programmatore esperto potrebbe costruire l' intera struttura dell' applicazione utilizzando soltanto questo strumento.

Allo stesso modo per quanto detto nella sezione riguardante il linguaggio PHP il primo file che viene caricato, ossia quello con la funzione di "main" per l' intero sito, è il file **index.php** che questa volta è situato all' interno della cartella *public* nella directory di progetto. Tuttavia in questo caso non troviamo, come nel caso precedente, la costruziuone della pagina di benvenuto, ma una serie di istruzioni che hanno lo scopo di:

- definire/configuare i percorsi dove reperire i file di Zend Framework (*application, models, library*),
- inizializzare l' MVC e collegarne il layout (usando *Zend_Layout*),
- configurare il database,
- avviare l' esecuzione del programma istanziando e lanciando i *controller.*

Un' ultima cosa da ricordare prima di poter cominciare ad introdurre quanto è stato fatto per realizzare questa versione della nostra applicazione riguarda i meccanismi di chiamata che, a differenza di quanto accadeva nel linguaggio PHP, con Zend non viengono più effettuati passando da file a file, bensì ci si sposta all' interno del progetto da controller a controller e da azione ad azione. Inoltre, siccome sviluppiamo questo sito ex novo, non è da dimenticare il funzionamento generale di Zend il quale basa tutta la sua architettura sull' MVC (*Model - View - Controller*), di cui si è abbondantemente parlato nel capitolo 3.5 di questa relazione.

La prima operazione da anlizzare è la configurazione del database. Per eseguire questa operazione il framework Zend mette a disposizione varie soluzioni che variano a seconda della necessità di chi deve servirsene ed ovviamente rende possibile la connessione con qualsiasi tipo di **DBMS** (*data base managment sistem*). Il metodo consigliato e più utilizzato per effettuare tale procedura è sicuramente l' impiego del file di configurazione, anch' esso creato all' inizio con il progetto e che solitamente prende il nome di **application.ini,** accoppiato assieme ad un adattatore della classe **Zend_Db_Adapter**. Questa classe fornisce una semplice interfaccia database SQL che viene utilizzata per collegare l' applicazione php con un particolare DBMS. Visto che di questi strumenti ne esistono diverse marche, allora esiste anche un adattatore per ognuno di loro che ne facilita l' utilizzo. Di conseguenza dopo aver detto tutto questo viene presentata la realizzazione della nostra configurazione.

Algorithm 68 Configurazione del database

```
// set up database
require_once 'Zend/Registry.php';
$config = new Zend_Config_Ini(ROOT_DIR.'/application/configs/application.ini', 'staging');
$dbAdapter = Zend_Db::factory($config->resources->db);
$registry = Zend_Registry::getInstance();
Zend_Registry::set('configuration',$config);
Zend_Registry::set('dbAdapter',$dbAdapter);
```

Come si può osservare la prima istruzione si occupa di configurare il percorso e l' *enviroment* (la sezione in cui inserire i parametri di connessione all' interno del file di configurazione), mentre la seconda come abbiamo detto ha il compito di creare l' interfaccia di connessione allo specifico DBMS. Le altre righe di codice hanno invece il compito di salvare i due parametri appena creati in modo che possano essere successivamente recuperati per permettere alle funzioni contenute nel Model di connettersi al database e eseguire le query richieste. Infine, per completare questa procedura, visto che nel nostro caso il database è realizzato con **MySql** il file di configurazione viene modificato nel seguente modo:

Algorithm 69 File di configurazione

```
[staging : production]
resources.db.adapter = 'Pdo_Mysql'
resources.db.params.host = 'localhost'
resources.db.params.username = 'root'
resources.db.params.password = 'root'
resources.db.params.dbname = 'authentication'
```

Da quello che si è potuto vedere in questa prima fase è quanto sia facile, anche se non come in PHP, effettuare la connessione al database; però emerge altresì l' estrema semplicità con la quale si riesce gestire più database simultaneamente all' interno della stessa apllicazione, anche costruiti su diversi DBMS, cosa che diventerebbe problematica usando il linguaggio PHP puro.

Giunti a questo punto siamo in grado di proseguire nel presentare lo svolgimento del progetto, magari aggiungendo eventualmente qualche delucidazione in più sui vari passaggi eseguiti, soprattutto su quelli un pò più delicati o di maggior interesse così da allontanare qualsiasi perplessità o dubbio ancora presenti. Quindi una volta che è stato eseguito il file index.php presente all' intermo della directory pubblica la gestione di tutta l' applicazione passa al controller e precisamente all' ***index action*** dell' ***Index Controller***, i quali sono generati automaticamente con il comando di creazione del progetto visto all' inizio. Per quanto riguarda la nostra *home page,* come si può vedere qui di seguito o per quanto già sappiamo, è abbastanza semplice pertanto nell' index-Index (action-Controller) si trovano soltanto le assegnazioni per la **view** di ciò che deve essere visualizzato.

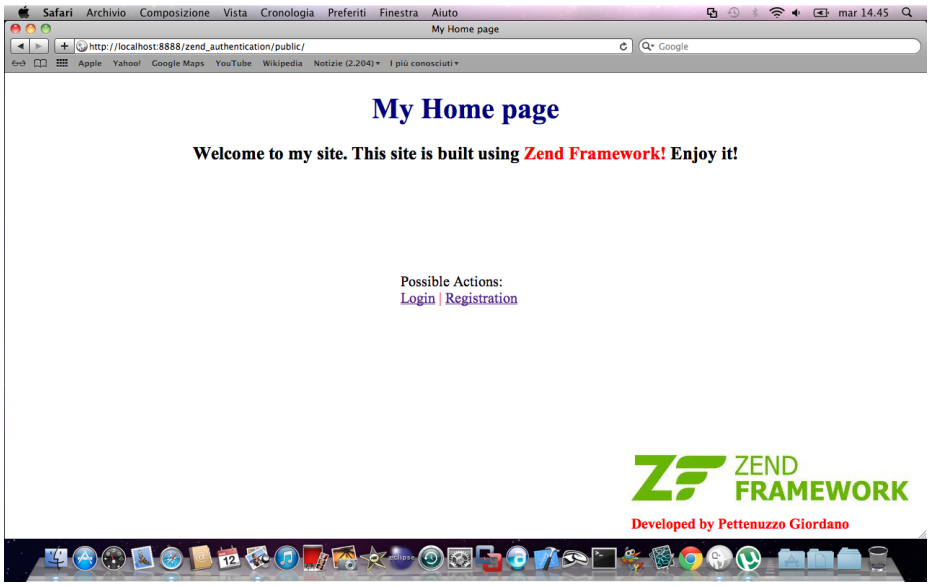

Figure 28: Pagina iniziale

L' unica nota che si vuole riportare riguarda il modo in cui vengono effettuati i metodi di chiamata (link) ad altre pagine, che come già detto in precedenza, differiscono dal metodo classico usato da PHP, ed appunto per tale motivo, se ne riporta un esempio nel seguente frammento di codice:

Algorithm 70 Esempio di passaggio tra controller e action

```
<a href="<?php echo $this->url(array('controller'=>'Login','action'=>'index'));?>">Login</a>
```

Così si puo vedere com' è possibile raggiungere, all' interno della stessa applicazione, una particolare action di uno specifico controller. La cosa cambia se invece si deve raggiungere una pagina esterna al progetto in quanto cambia il modo in cui effettuare il "lincaggio", ovvero cambia la sintassi dell' istruzione, visto che non sono presenti ne controller ne tantomeno action da referenziare anche se la pagina in questione è realizzata anchessa con il framework *Zend* perchè, come abbiamo detto all' inizio di questo paragrafo, il primo file ad essere eseguito è l' index.php e non l' Index Controller.

Una cosa che ormai dovrebbe essere chiara anche solo dopo questa breve esposizione, visto il modo in cui è stata analizzata nella parta teorica, riguarda la struttura dell' MVC ed in particolar modo il rapporto di connessione tra le parti che lo compongono. Infatti, per quanto detto finona, risulta che:

> **Ad ogni *Controller* è associata una *View* ed ad ogni *Action* è associato un file .phtml che riportano lo stesso nome del componente ai quali sono accopiati.**

Quindi tra il controller e la view il rapporto di esistenza è del tipo 1 a 1 e per rimarcare questo concetto viene presento il seguente schema:

Figure 29: Rapporto tra controller e view

Ora vediamo com' è possibile realizzare il form di registrazione con questo framework. Come abbiamo già visto nel paragrafo 3.11 i form vengono trattati come fossero delle classi, per fare un esempio pratico costruire un form in Zend assomiglia molto ad usare le classi con JAVA, pertanto per chi non è a digiuno di programmazione ad oggetti non presenta nessuna complicazione, ed inoltre in questa operazione veniamo aiutati dalla classe già predisposta all' uso: Zend_Form. Quindi all' interno della cartella **forms** creata assieme al progetto andremo amettere l' oggetto che ci interessa ottenere e che chiameremo: RegistrationForm.php. Se seguiamo le istruzioni della parte teorica del capitolo 3 il codice che implementa il nostro form dovrebbe avere una sintassi simile a questa:

Algorithm 71 Form di Registrazione

```
class forms_RegistrationForm extends Zend_Form {
    public function __construct($options = null) {
        // Costruttore di default
        parent::__construct($options);
        // => Non serve mettere: $this->setMethod('post');
        $this->setName('registration');

        $username = new Zend_Form_Element_Text('username');
        $username->setLabel('User name:*')
                 ->setRequired(true)
                 ->addFilter('StripTags')
                 ->addFilter('StringTrim')
                 ->addValidator('NotEmpty');
        $password = new Zend_Form_Element_Password('password');
        $password->setLabel('Password:*')
                 ->setRequired(true)
                 ->addValidator('NotEmpty');
        // ...
        // Altri input
        // ...
        $submit = new Zend_Form_Element_Submit('submit');
        $submit->setLabel('Registration');
        $cancel = new Zend_Form_Element_Reset('reset');
        $cancel->setLabel('Cancel');
        // Aggiungiamo gli elementi che verranno utilizzati nel controllore
        $this->addElements(array($fullname, $username, $password, $email, $birthday,
$submit, $cancel));
```

Come possiamo osservare dal frammento di codice appena visualizzato anche non utilizzando il costruttore di default è possiblie con una banalissima istruzione, diversamente da quanto sarebbe necessario usando solo il linguaggio PHP, utilizzare il metodo di trasissione POST che è molto più elegantee riservato rispetto al suo antagonista. Questo è uno dei motivi per cui nella precedente applicazione si è preferito optare per il metodo GET, il quale comunque funziona perfettamente tanto quanto il metodo POST. Con l' ultima istruzione gli oggetti inseriti vengono caricati e messi a disposizione per il controller così da poter essere elaborati e successivamente visualizzati. A questo punto ci spostiamo all' interno del controllore RegirtationController.php e specificatamente all' action index così da vedere come fare per poter utilizzare il form che abbiamo appena creato.

Algorithm 72 Struttura per l' utilizzo del form

```
class RegistrationController extends Zend_Controller_Action
{
        function indexAction()
        {
                // ...
                $form = new forms_RegistrationForm();
                // ...
                $this->view->form = $form;
                // ...
                if ($this->_request->isPost()) {
                        // ...
                        if ($form->isValid($formData)) {
                                // ...
                                // Registrazione effettuata con successo
                                // ...
                        } else {
                                $form->populate($formData);
                        }
                }
        }
// ...
```

Come già accennato poco fa, a proposito dei form che in Zend vengono visti come classi di oggetti, per poter usare un form si ricorre all' operatore **new**, proprio come avviene in un qualunque linguaggio orientato ad oggetti, così da creare una nuova istanza dell' oggetto in questione. Com' è riportato nel codice soprastante, una volta che abbiamo creato il nuovo oggetto possiamo elaborarlo a nostro piacimento all' interno del controllore e nel nostro caso, visto che si tratta di un form per la registrazione, non ci rimane altro che passarlo alla view così che possa effettuarne la visualizzazione per l' utente. Effettivamente è la *view* che si occupa della visualizzazione e ovviamente il file che si occupa del *rendering* del form di registrazione si trova, come abbiamo già spiegato, all' interno della view nella directory riservata al Registration Controller. Ci sono due modi per effetuare la visualizzazione del form e dipendono dal modo in cui questo viene passato dal controllore, infatti è possibile passarlo per intero, come abbiamo fatto noi, oppure passarlo componente per componente così da avere un maggior controllo sulla gestione della pagina. Pertanto ecco come nel nostro caso ne viene realizzato il rendering:

Algorithm 73 Visualizzazione del form di registrazione

```
<?php echo $this->form ;?>
```

Quello che si ottiene infine, una volta che viene portato a termine l' esecuzione di tutta la procedura, viene mostrato nell' immagine seguente:

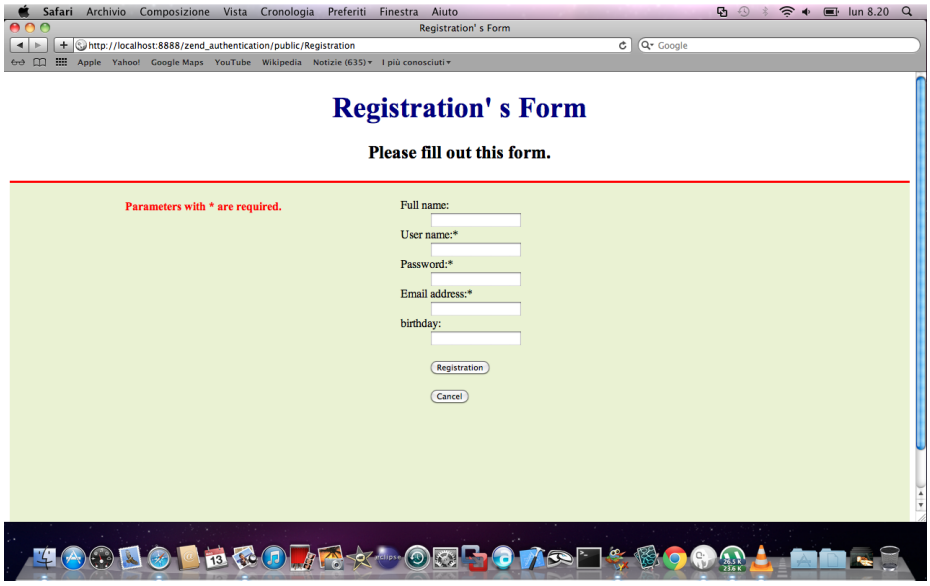

Figure 30: Pagina di registrazione

Chiaramente, com' è accaduto nel caso precedente, anche ora i dati che l' utente inserisce devono rispettare una certa forma e struttura, perciò sono necessari ovviamente dei controlli sull' input. Diversamente dal caso in PHP, nel quale i controlli venivano effettuati direttamente dal programmatore adattando a seconda del caso le funzioni del linguaggio stesso, con l' impiegodi un framework ed in particolare di questo si hanno a disposizione una serie di validator e filtri creati appositamente da Zend e già predisposti per eseguire il controllo sugli input. Anche se questi elementi sono già stati presentati nella parte teorica se ne riporta comunque una breve sequenza come ulteriore dimostrazione sul metodo di utilizzo:

Algorithm 74 Utilizzo di validatori e filtri

```
$fullname = new Zend_Form_Element_Text('fullname');
$fullname->setLabel('Full name:')
         ->addFilter('StripTags')
         ->addFilter('StringTrim')
         ->addValidator('regex', false, array('/^[a-zA-Z]/'))
         ->addValidator('stringLength', false, array(6, 20))
         ->addValidator('alnum', false, array(' '));

$username = new Zend_Form_Element_Text('username');
$username->setLabel('User name:*')
          ->setRequired(true)
          ->addFilter('StripTags')
          ->addFilter('StringTrim')
          ->addValidator('NotEmpty');

$password = new Zend_Form_Element_Password('password');
$password->setLabel('Password:*')
          ->setRequired(true)
          ->addValidator('NotEmpty');

$email = new Zend_Form_Element_Text('email');
$email->setLabel('Email address:*')
      ->addFilter('StringToLower')
      ->setRequired(true)
      ->addValidator('NotEmpty')
      ->addValidator('EmailAddress');

$birthday = new Zend_Form_Element_Text('birthday');
$birthday->setLabel('birthday:')
         ->addValidator('Date'); ;
```

Aggiungendo questi oggetti otteniamo una serie di controlli sugli input che il programmatore ha stabilito di implementare nell' applicazione per il corretto funzionamento del database. Inoltre, da quello che si può osservare, è estremamente semplice utilizzarli in quanto basta aggiungere all' input il validatore o filtro desiderato richiamandone il nome e, tenendo presente che della realizzazione se ne sono occupati i creatori di Zend, siamo certi della loro correttezza e funzionalità. In effetti il team di Zend ha predisposto una vasta gamma di validatori e filtri in modo da gestire ogni genere di input inserito dall' utente come ad esempio:

- date,
- e-mail,
- stringhe di caratteri o numeri;

così da non appesantire il carico di lavoro dello sviluppatore. Tutto questo diventa ancora più semplice se il programmatore ha una minima conoscenza di PHP in quanto, da ciò che si è visto, molti di questi validatori ricalcano il nome, e quindi l' effetto, di alcune funzioni di PHP (StringToLower, StringTrim, StringLenght, ecc); altri invece esegono procedure per le quali non esiste una funzione nemmeno in PHP come il validatore NoEmpty. Un ultima cosa da dire riguardo a questi strumenti riguarda la loro visualizzazione la quale avviene in automatico a meno che non ci si deva preoccupare del "prettiness". Perciò per

noi che non gestiamo la loro presentazione i *validator* vengono visualizzati nel modo seguente:

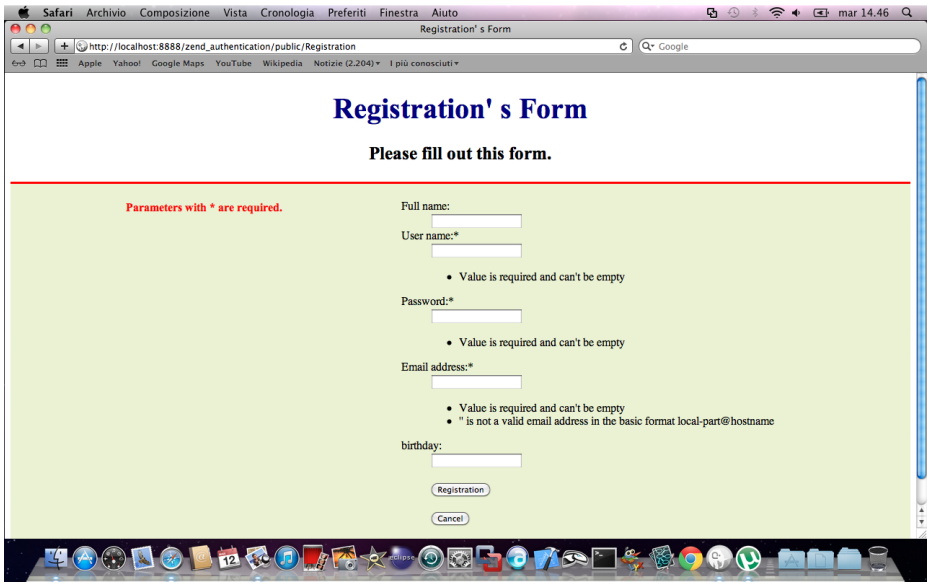

Figure 31: Pagina di registrazione - utilizzo validatori

Questa realizzazione ci fornisce almeno due vantaggi rispetto alla realizzazione eseguita con PHP oltre al fatto di non dover costruire i controlli.

1. Non dovendo preoccuparci della visualizzazione dei validatori non dobbiamo nemmeno creare delle strutture di controllo per gestire gli eventuali scorrettezze commesse durante l' inserimento dei dati, ne tantomento realizzare un sistema di rendering per informare l' utente degli errori effettuati.

2. Visto che la visualizzazione dei validatori avviene automaticamente sotto al campo nel caso esso non rispetti le specifiche designate dal programmatore, com' è già stato mostrato, all' utente viene fornita l' informazione dell' errata compilazione del dato immediatamente senza che questi venga ridirezionato ad un' altra pagina come succedeva nel caso precedente.

In questo modo viene migliorata sia la chiarezza che la funzionaltà dell' applicazione da parte dell' utente il quale può correggere l' inesattezza direttamente sul form errato senza dover reinserire anche i campi corretti. Inoltre si ha un considerevole miglioramento anche sull' *output* della pagina visualizzata che viene resa molto più dinamica e interattiva.

Obiettivamente per quanti generi di input riusciamo a gestire con i *validator* ci sono dei controlli per i quali non è possibile predisporre alcuna funzione risolutiva visto che dipendono da un' analisi soggettiva della specifica applicazione che si sta realizzando. Nel nostro caso una tale condizione riguarda l' unicità

della username così che ogni utente possa essere identificato senza entrare in conflitto con gli altri e così mantenere consistenti i dati contenuti nel database. Per effettuare questa procedura abbiamo fatto ricorso al terzo elemento di cui è composto l' MVC e del quale non abbiamo ancore parlato: il **model**. Ricordando quanto detto nel paragrafo 3.5, ed in particolare la Figura 1 in esso presentata, in questo elemento si trovano classi e funzioni php che all' occorrenza vengono utilizzate dai controller per eseguire le loro eleborazioni. Com' è successo con i form anche i model vengono trattati come delle classi, percui una volta creato l' oggetto all' interno della directory models del progetto possiamo cominciare ad implementare le nostre funzioni nel seguente modo:

Algorithm 75 Creazione del Model

```
class Users extends Zend_Db_Table_Abstract {

        protected $_name = 'users';

        // ...

        function checkUnique($username) {
                $select = $this->_db->select()
                            ->from($this->_name,array('username'))
                            ->where('username=?',$username);
                $result = $this->getAdapter()->fetchOne($select);
                if($result){
                        return true;
                }
                return false;
        }

        // ...
```

Nel frammento di codice appena presentato è possibile osservare uno dei modi messi a disposizione dal Framework Zend per effettuare le *query* su un database, infatti oltre al metodo "classico" di php è possibile utilizzare un metodo di interrogazione di più alto livello, come nel caso del codice presentato, oppure delle funzioni precompilate che sono in grado di eseguire delle operazioni cosiddette "standard" quali ad esempio l' iserimento o la cancellazione di dati. Quindi per adoperarlo seguiamo lo stesso procedimento impiegato con i form andando così ad aggiungere ulteriori informazioni al codice del controller già mostrato:

Algorithm 76 Utilizzo del Model

```
// ...
// Creazione dell' oggetto users
$DB = Zend_Registry::get('dbAdapter');
$users = new Users(array('db' => $DB));
// ...
if ($form->isValid($formData)) {
        $data = $form->getValues();
        if($users->checkUnique($data['username'])){
                // ...
                // gestione del risultato
                // ...
```

L' unica nota che si ritiene di dover dire in merito all' algoritmo soprastante riguarda la prima istruzione nella quale viene effettuato il recupero dei dati relativi alla connessione del database, infatti quando creiamo l' istanza del model ad esso passiamo proprio questo parametro perchè, come mostrato nelle Figure 1 e 2, se si opera nel modo corretto dovrebbero essere le funzini contenute nel model ad interrogare la base dati.

Quindi se anche quest' ultimo controllo fallisce quello che si ottiene è l' uteriore visuallizzazione di un messaggio di errore sulla schermata del form di registrazione come viene mostrato qui di seguito:

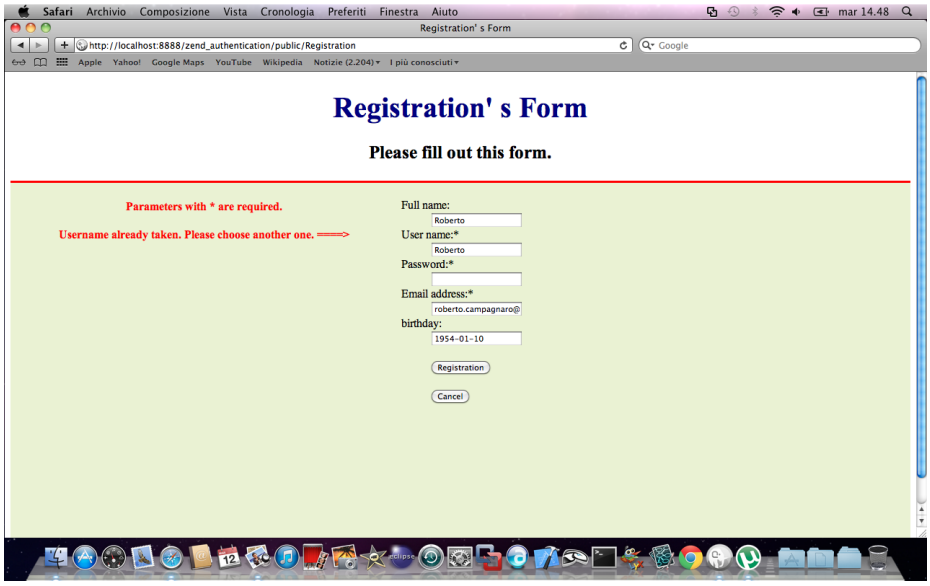

Figure 32: Pagina di registrazione - ulteriore controllo

Contrariamente se non viene commesso nessun errore da parte dell' utente, il che significa che i dati inseriti sono del tutto corretti, l' unica cosa rimasta da fare è terminare la fase di registrazione della nostra applicazione con due semplici istruzioni:

Algorithm 77 Successo nella registrazione

```
// Usiamo la funzione già disponibile di Zend per l' inserimento dei dati
$this->insert($data);
// Ci spostiamo all 'action di successo della registrazione
$this->_helper->redirector('success');
```

Mentre la prima istruzione effettua l' operazione di inserimento dell' input nella base dati utilizzando la funzione *insert($param)* già predisposta da Zend, la seconda invece ha i compito di far spostare l' esecuzione del programma

nell' action "succes" restando comunque sempre all' interno del Registration Controller. Infatti è quest' action che si occupa di informare l' utente che l' operazione di registrazione ha dato esito positivo visualizzando un messaggio di successo. Qui di seguito presentiamo direttamente il risultato di questa action senza andare ad analizzare il codice che la costituisce in quanto si tratta solamente di assegnazioni per la view.

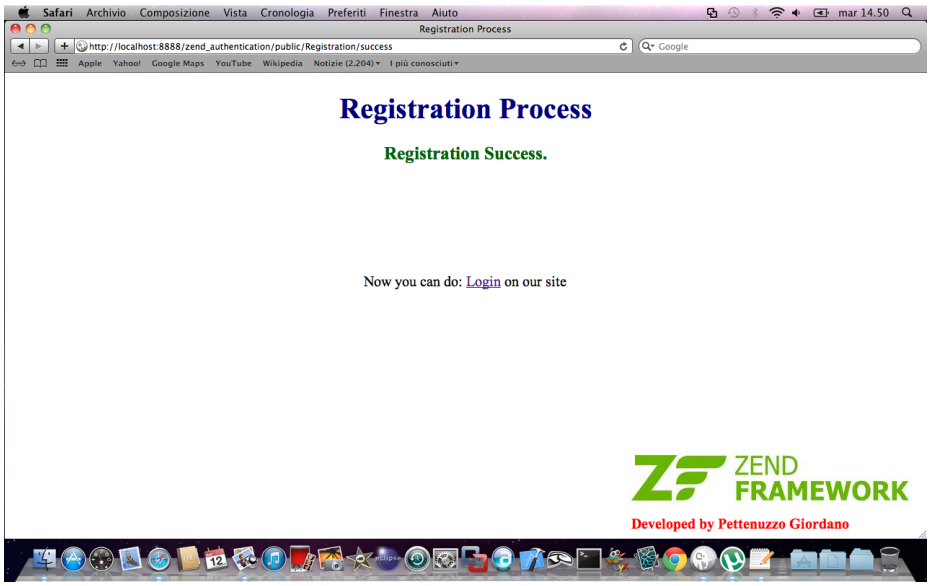

Figure 33: Pagina di registrazione avvenuta

Con quest' ultima schermata si conclude una volta per tutte l' operazione di registrazione, per cui possiamo passare tranquillamente ad analizzare la fase di *login*. Per quanto riguarda la preparazione del form per l' inserimento dei dati si eseguono gli stessi passaggi svolti nella fase di registrazione. Quindi implementiamo la classe forms_LoginForm all' interno della directory forms che poi verrà utilizzata nel LoginController, esattamente come abbiamo già fatto in precedenza, così da poter essere infine passata alla view correlata che si occuperà della sua visualizzazione. Quello che si ottiene viene mostrato nell' immagine seguente:

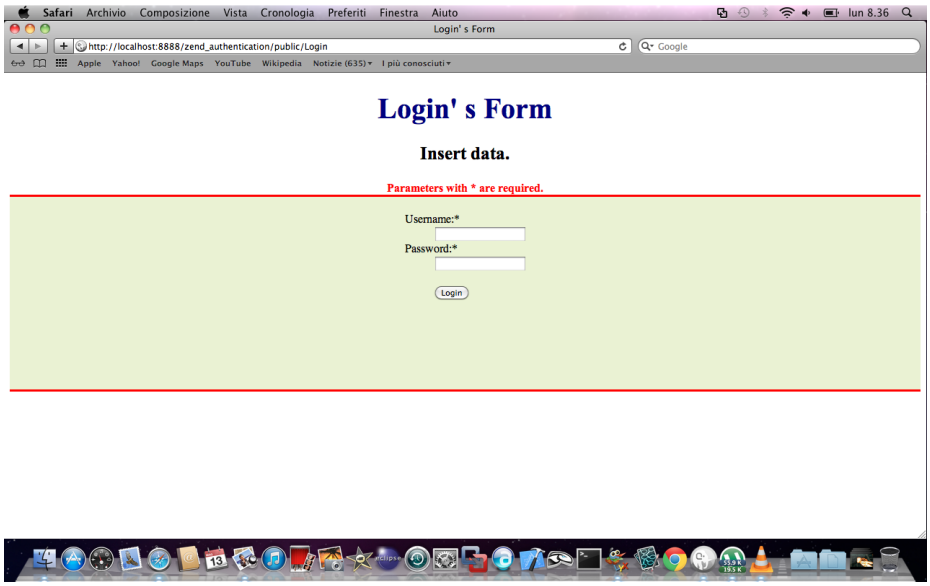

Figure 34: Pagina di login

Allo stesso modo vengono applicati anche i validatori con le loro condizioni di validità ed esistenza. A questo proposito infatti qui sootto viene riportato un esempio di quello che viene visualizzato in caso d' errore da parte dell' utente.

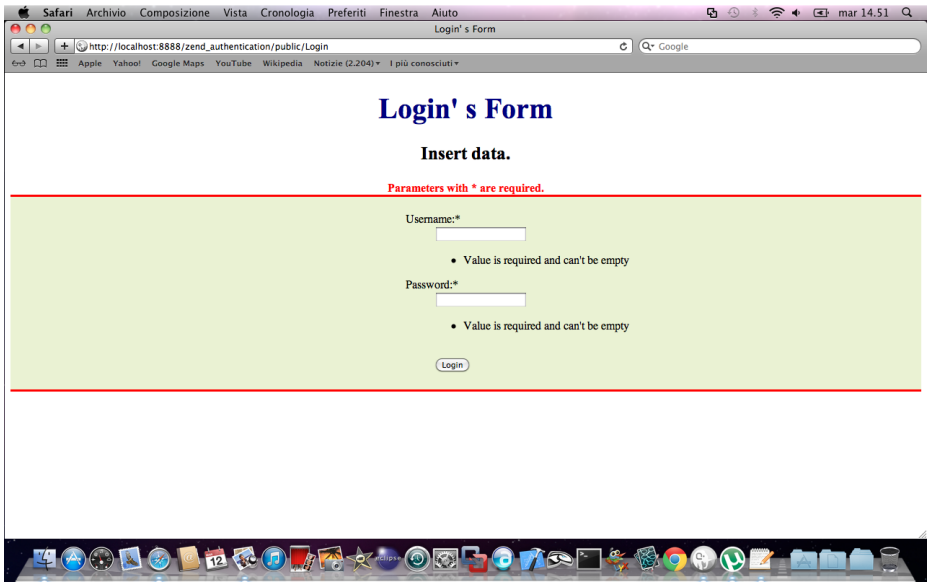

Figure 35: Pagina di login - utilizzo validatori

Ora invece vediamo in che cosa differisce questa procedura rispetto alla precedente. Innanzitutto non basta semplicemente acquisire dei valori corretti inseriti dall' utente, ma si deve anche eseguirne l' autenticazione per poter completare l' operazione di login. Per effettuare questo Zend mette a disposizione la classe *Zend_Auth_Adapter_DbTable* che può essere usata come segue:

Algorithm 78 Autenticazione utente

```
$auth = Zend_Auth::getInstance();
// ...
// Inizializzazioni come nel caso di registrazione
// ...
if ($form->isValid($formData)){
        $data = $form->getValues();

        $authAdapter = new Zend_Auth_Adapter_DbTable($users->getAdapter(),$users->getName();
        $authAdapter->setTableName('users')
                    ->setIdentityColumn('username')
                    ->setCredentialColumn('password');
        $authAdapter->setIdentity($data['username'])
                    ->setCredential(md5($data['password']));

        $result = $auth->authenticate($authAdapter);
        // ...
        // Studio del risultato
        // ...
```

Grazie a questo strumento ci ritroviamo attraverso l' esecuzione di poche istruzioni nella condizione di avere nella variabile *$result* il risultato dell' operazione di autenticazione che attende solo di essere esaminato. Infatti ancora non sappiamo in quale situazione ci troviamo (successo o fallimento) e perciò dobbiamo analizzare il contenuto della suddetta variabile. Di nuovo Zend ci viene in aiuto fornendo tutta una serie di **API** per l'autenticazione che comprendono anche adattatori di autenticazione per poter effettuare l' identificazione, cioè determinano se un soggetto è in realtà quello che pretende di essere sulla base di un insieme di credenziali. In questo modo anche l' analisi del risultato si può svolgere gilmente come viene mostrato qui sotto:

Algorithm 79 utilizzo delle API

```
switch ($result->getCode()){
// CASO 1
        case Zend_Auth_Result::FAILURE_IDENTITY_NOT_FOUND:
                // Gestione dell' evento
        break;
// CASO 2
        case Zend_Auth_Result::FAILURE_CREDENTIAL_INVALID:
                // Gestione dell' evento
        break;
// CASO 3
        case Zend_Auth_Result::SUCCESS:
                // Gestione dell' evento
        break;
// CASO 4
        default:
                // Gestione dell' evento
        break;
}
```

Per quanto riguarda i casi 1, 2 e 4 la loro gestione nel nostro caso risulta essere estremamente semplice visto che si tratta solamente di una visualizzazione d' errore direttamente sul form per informare l' utente che l' operazione di login non ha avuto buon fine. Per questo motivo viene riportato solamente l' effetto di uno solo di questi eventi.

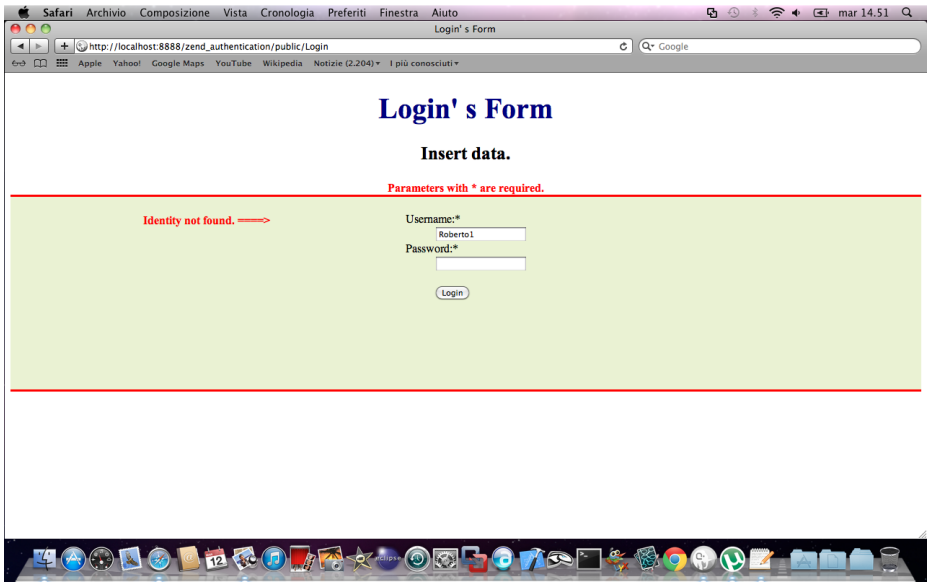

Figure 36: Pagina di login - visulaizzazione CASO 1

Diversa invece è la gestione del caso 3 dal momento che deve occuparsi del completamento della processo. Per questo motivo deve anzitutto identificare lo stato in cui si trova l' utente che ha appena effettuato il login, predisporre le variabili in modo tale che possano essere registrate all' interno della base dati e di conseguenza modificare correttamente le occorrenze all' interno delle tabelle facendo attenzione a mantenere la consistenza dei dati. Tutto quello che è stato appena detto può sembrare un' operazione lunga e complessa, ecco perchè se ne riporta il codice:

Algorithm 80 Aggiornamento database

```
// Preparazione dei parametri d' inserimento
$user = $authAdapter->getResultRowObject();
$userid = $user->id;
$sessionid = $user->username;
$date = new Zend_Date();
$lastupdate = $date->get('YYYY-MM-dd HH:MM:ss');
$param = array('sessionid' => $sessionid,'userid' => $userid, 'lastupdate' => $lastupdate);
$loggedUsers = new LoggedUsers(array('db' => $DB));

if($loggedUsers->findUser($data['username'])== 1){
        // Utente gia loggato in passato => solo aggirnamento dati
        $loggedUsers->updateUser($lastupdate, $sessionid);
}else{
        // Primo login dell' utente => crezione di un nuovo record
        $loggedUsers->create($param);
}
// ...
```

Anche se sono stati aggiornati i campi del database comunque l' operazione di login non può dirsi ancora conclusa, infatti rimane ancora in sospeso la realizzazione della sessione che viene illustrato qui sotto:

Algorithm 81 Creazione della sessione

```
// Costruzione della sessione
if($result->isValid()){
        $storage = new Zend_Auth_Storage_Session();
        $storage->write($authAdapter->getResultRowObject());
} else {
        $this->view->errorMessage = "Something was wrong with your authentication.";
}
// Reindirizzamento
$this->_helper->redirector('loggedin');
```

La creazione di questa struttura permette di identificare immediatamente l' utente connesso una volta che si accede all' area riservata della nostra applicazione. Quindi, com' è stato già mostrato, non resta altro da fare per poter terminare la fase di login con successo che spostarsi nel controller "loggedin" restando sempre all' interno del LoginController. Il compito di questo controller è quello di recuperare i parametri di sessione ed informare l' utente del completamento dell' operazione. Qui di seguito viene presentato il frammento di codice che esegue questa procedura:

Algorithm 82 Controllo sessione e visualizzazione

```
// Controllo apertura sessione
$storage = new Zend_Auth_Storage_Session();
if ($storage->IsEmpty()){
        $this->_helper->redirector('index', 'Index');
        exit;
}
// Recupero parametri per la visualizzazione
$user = $storage->read();
$this->view->pageTitle = "Login Success";
// ...
```

Per un' operazione come quella del login, e anche per tante altre, è molto utile usare le sessioni perchè questi strumenti fungono, come in questo caso, da controllore sugli accessi visto che chiunque acceda alla pagina privata senza aver effettuato con successo la procedura del login, e quindi aver avviato una nuova sessione, verrà direttamente inviato all' home page senza vedere alcunchè dell' area riservata. A questo punto non rimane altro che mostrare l' effetto dell' ultima sequenza di istruzioni che hanno il compito di predisporre la view che visualizzerà l' ultima pagina della nostra applicazione:

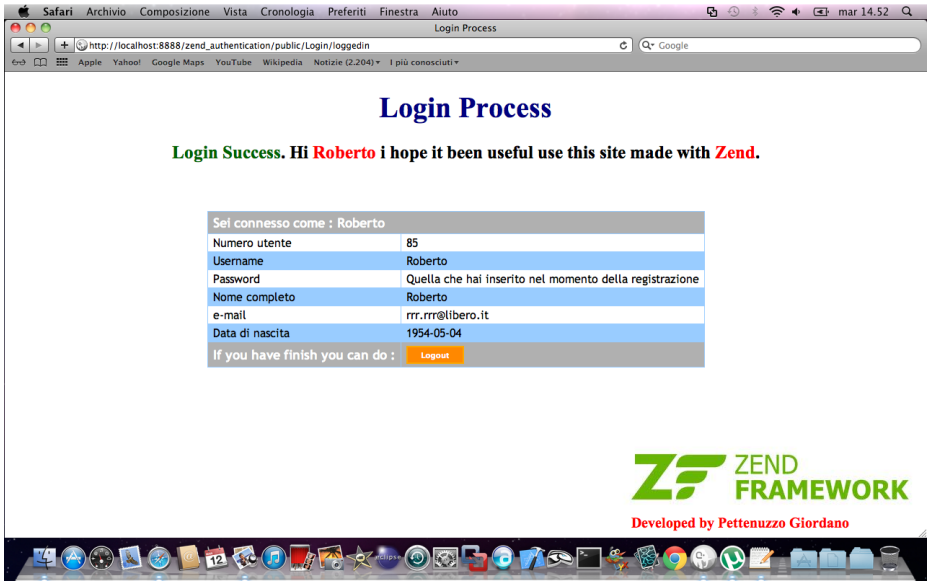

Figure 37: Pagina di login avvenuto

Infine l' ultima cosa che ci rimane da vedere per poter dire di aver terminato questa presentazione è il modo in cui è possibile effettuare l' operazione di *logout*. Questa action, raggiungibile per mezzo di un link mascherato da bottone ed inserito nella vista dell' avvenuto successo per la fase di login, è contenuta all' interno del Login Controller ed ha lo scopo di:

- disconnettere l' utente cancellando le variabili di sessione,
- eliminare i parametri d' autenticazione predisponendoli per un successivo login,
- riportare l' utente nella home page di partenza.

Come nel caso visto precedentemente riguardante l' analisi dell' applicazione realizzata con il linguaggio PHP il tutto può essere eseguito nel modo seguente.

Algorithm 83 Oprerazione di logout

```
// Esecuzione del logout
$storage = new Zend_Auth_Storage_Session();
$storage->clear();
Zend_Auth::getInstance()->clearIdentity();
// Ritorno all' home page
$this->_helper->redirector('index', 'Index');
```

Da quello che abbiamo potuto vedere però all' interno di questa action non è presente nessuna assegnazione per la vista collegata in quanto questa operazione non ha la necessità di visualizzare alcunchè. Quest' affermazione tuttavia si trova in netto contrasto con la **configurazione standard di Zend** e con un' altra affermazione fatta in precedenza nel corso di questa stessa trattazione durante fase di registrazione secondo la quale ad ogni action è associato un elemeto .phtml che ne effettua la visualizzazione del risultato. Per cui se provassimo a lanciare la nostra applicazione il comilatore di Zend segnalerebbe subito l' errore. Per ovviare a questa situazione dobbiamo per forza modificare manualmente l' impostazione predefinita di Zend come viene mostrato qui sotto:

Algorithm 84 Disabilitazione della vista

```
// Disabilitazione vista
$this->_helper->layout->disableLayout();
$this->_helper->viewRenderer->setNoRender(true);
```

In questo modo grazie a questo stratagemma viene disabilitato il layout ed esclusa la vista dell' action collegata, per cui l' esecuzione del logout non presenta più nessun problema e di conseguenza possiamo portare a termine con successo tutto il sito web che abbiamo realizzato.

A questo punto possiamo dire conclusa anche questa sezione tranne per il fatto che, per dare un senso di completezza al tutto, si ritiene di dover mostrare il risultato di quello che è stato prodotto durante la costruzione dell' intera applicazione visualizzando le modifiche apportate all' intera struttura del progetto rispetto a cio che è stato presentato al momento della sua creazione.

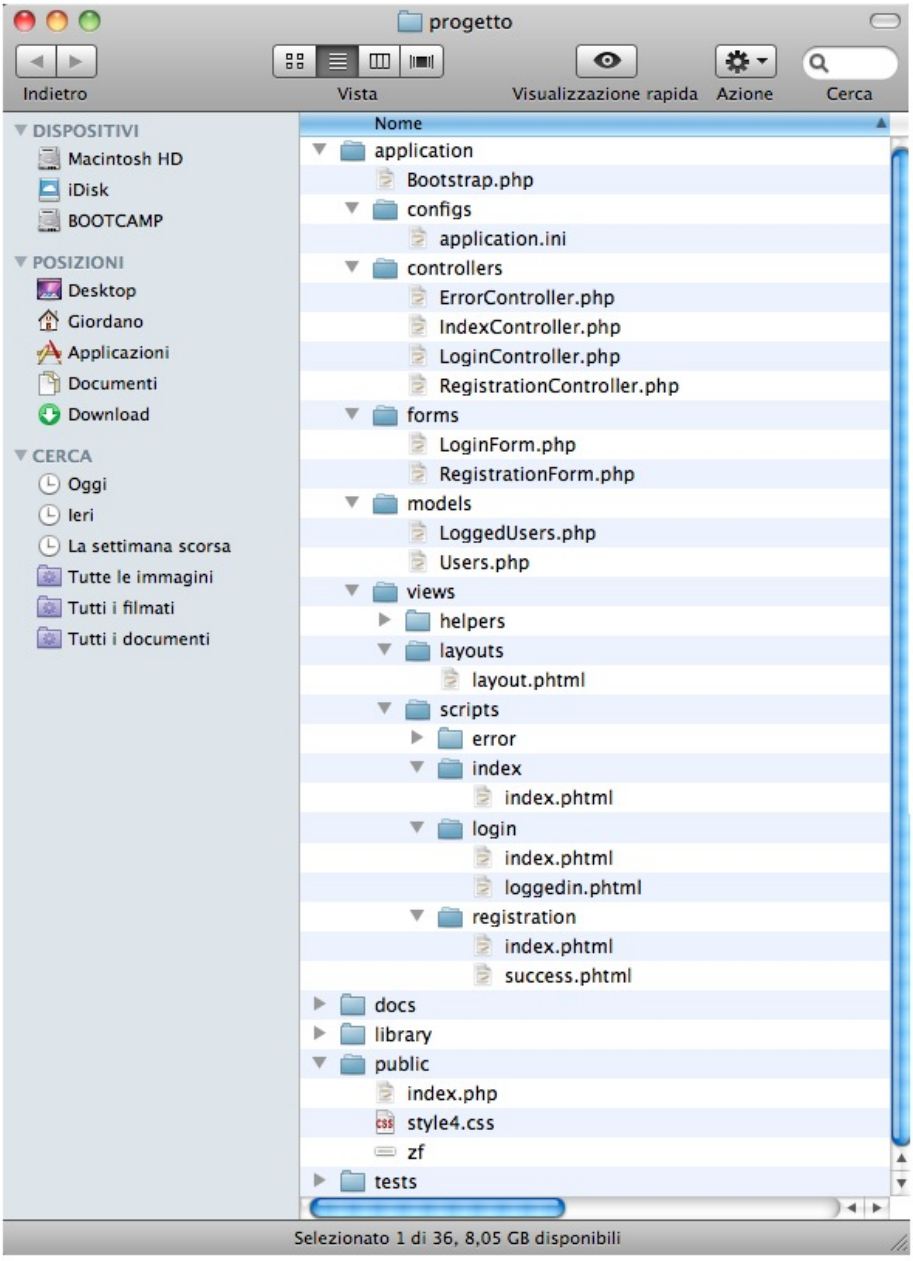

Figure 38: Progetto alla fine dello sviluppo

Tutti i frammenti di codice che sono stati presentati in questo paragrafo saranno mostrati per intero nell' appendice B.

4.4 CONFRONTO TRA I DUE METODI

Nei due precedenti paragrafi abbiamo presentato separatamente i due metodi per realizzare la medesima applicazione così da dare una struttura logica a tutto il discorso analizzando dall' inizio alla fine le due implementazioni, inoltre si è scelto di percorrere questa strada anche per non creare confusione nel lettore mischiando i due procedimenti. Però tenendo presente lo scopo di questa tesi, che ha il compito di verificare se vale o no la pena utilizzare il Framework Zend per sviluppare applicazioni web, è necessario metterli a confronto così da evidenziare le varie differenze di costruzione dei due modi di lavorare. Infatti da quello che è stato realizzato finora non risulta ancora chiaro quali possano essere i vantaggi/svantaggi nell' usare un framework, ed in particolare questo framework in esame, piuttosto che un semplice linguaggio di programmazione quale il PHP. Quindi è proprio in questo paragrafo che procederemo con il suddetto confronto dei punti per noi più significativi:

Costruzione del form

La prima cosa che andremo a confrontare sarà la realizzazione del form per l' inserimento dei dati.

1. Realizzazione con PHP

Algorithm 85 Form in PHP

```
<form method="GET" action="inserimento.php" align="center">
   <table class = "cover"><caption><span>I campi con * sono obbligatori</span></caption>
   <tr><td width = 36%></td>
        <td><table>
           <tr><td><label>username<span> *</span> :</label></td>
               <td><input name="username" type="text" size="40" maxlength="50"/></td>
           </tr>
           <tr><td><label>password<span> *</span> :</label></td>
               <td><input type="password" maxlength="50" size="40" name="password"/></td>
           </tr>
           <tr><td><label>full name :</label></td>
               <td><input name="fullname" type="text" size="40" maxlength="150" /></td>
           </tr>
           <tr><td><label>e-mail<span> *</span> :</label></td>
               <td><input name="email" type="text" size="40" maxlength="200" /></td>
           </tr>
           <tr><td><label>data di nascita :</label></td>
               <td><input name="birthday" type="text" size="40" maxlength="10" value = "aaaa/mm/
gg"/></td>
           </tr>
           <tr><td></td>
               <td><button type=”submit”>Vai</button> <button type="reset">Cancella</button></td>
           </tr>
        </table></td>
   </tr></table class ="cover">
</form>
```

2. Realizzazione con ZEND

Algorithm 86 form in ZEND

```
// Realizzazione del form
class forms_RegistrationForm extends Zend_Form{
    public function __construct($options = null){
        parent::__construct($options);
        $this->setName('registration');
        $fullname = new Zend_Form_Element_Text('fullname');
        $fullname->setLabel('Full name:')
        $username = new Zend_Form_Element_Text('username');
        $username->setLabel('User name:*')
        $password = new Zend_Form_Element_Password('password');
        $password->setLabel('Password:*')
        $email = new Zend_Form_Element_Text('email');
        $email->setLabel('Email address:*')
        $birthday = new Zend_Form_Element_Text('birthday');
        $birthday->setLabel('birthday:')
        $submit = new Zend_Form_Element_Submit('submit');
        $submit->setLabel('Registration');
        $cancel = new Zend_Form_Element_Reset('reset');
        $cancel->setLabel('Cancel');
        $this->addElements(array($fullname, $username, $password, $email, $birthday, $submit,
$cancel));
    }
}
// Utilizzo del form
$form = new forms_RegistrationForm();
$this->view->form = $form;
// Visualizzazione del form
echo $this->form ;
```

Come possiamo vedere dai due frammenti di codice appena presentati possiamo affermare che a livello di linee di codice più o meno i metodi si equivalgono, ma la differenza sostanziale tra i due riguarda l' organizzazione logica. Infatti mentre nel primo caso oltre alla costruzione del form ci si deve occupare anche della sua visualizzazione. Questo non sarebbe nemmeno un grosso problema fintantochè il form resta di piccole dimensioni; ma dobbiamo tenere conto che, all' aumentare sia della dimensione quanto della varietà dei campi da inserire, si complica notevolmente la gestione dell' estetica della pagina visualizzata tanto che il problema non è più la costruzione del form bensì la sua visualizzazione.

Invece nel secondo caso la costruzione del form è del tutto separata dalla visualizzazione che viene gestita interamente da un altro componente del framework (la VIEW) il quale scopo è esclusivamente quello di effettuare il rendering delle pagine. Inoltre per gestire meglio l' estetica è possible, diversamente da quanto abbiamo fatto noi per motivi di semplicità, visualizzare singolarmente ogni componente del form posizionandolo in qualsiasi parte della pagina si voglia anzichè visualizzarlo come se si trattasse di un unico blocco.

Controllo degli input

Ora invece analizziamo come vengono eseguiti i controlli sugli input in entrambe le parti.

1. Realizzazione con PHP

Algorithm 87 Controllo degli input con PHP

```
// Campo facoltativo
if($_GET['fullname']!=''){
        $fullname=$_GET['fullname'];
        if(!preg_match('/^[a-zA-Z]/', $fullname)){
                $count=$count+1;
                $message["3"] ="Devi inserire un nome valido<br>"; }
}
// Campo obbligatorio
if($_GET['email']==''){
        $count=$count+1;
        $message["4"] ="Il campo e-mail e' vuoto. Devi inserire una e-mail<br>";
}elseif(strpbrk($_GET['email'], ' ') || !strpbrk($_GET['email'], '@') || !strpbrk($_GET['email'],
'.') || strrpos($_GET['email'], ".") < strrpos($_GET['email'], "@")){
        $count=$count+1;
        $message["4"] ="Devi inserire un' e-mail valida<br>";
}else{
        $email=$_GET['email']; }
// ...
if($count=='0'){
        // Registrazione effettuata con successo. Visualizzazione per l' utente
}else{
        // Errore nell' inserimento dei campi. Visualizzazione errori:
for ($i = 1; $i <= 5; $i++){
        if($message["$i"] != ""){
                echo $message[$i];
        }
}
```

2. Realizzazione con ZEND

Algorithm 88 Controllo degli input con ZEND

```
// Campo facoltativo
$fullname = new Zend_Form_Element_Text('fullname');
$fullname->setLabel('Full name:')
         ->addFilter('StripTags')
         ->addFilter('StringTrim')
         ->addValidator('regex', false, array('/^[a-zA-Z]/'))
         ->addValidator('stringLength', false, array(6, 20))
// Campo obbligatorio
$email = new Zend_Form_Element_Text('email');
$email->setLabel('Email address:*')
      ->addFilter('StringToLower')
      ->setRequired(true)
      ->addValidator('NotEmpty')
      ->addValidator('EmailAddress');
```

Quindi anche se nel primo caso le numerose funzioni di PHP ci vengono in aiuto per il controllo degli input, è comunque compito del programmatore preoccuparsi della gestione di questi input costruendo una struttura apposita per contenerli così da effettuarne la visualizzazione, come del resto viene mostrato anche nel codice soprastante. Questi controlli sono tanto più complessi tanto più restrittive sono le condizioni che vengono poste ai singoli campi di inserimento. Inoltre, come se non bastasse questo, è necessaria un' ulteriore fase di *testing* per verificare che i controlli realizzati implementino realmente quanto è stato richiesto dal committente del sito; infatti anche se chi li ha realizzati può

definirsi un esperto del settore l' eventualità che commetta errori, anche banali (ad esempio di digitatura), è una possibilità più che plausibile.

Dall' altro lato con Zend grazie all' utilizzo di validatori e filtri, che vanno aggiunti all' interno della costruzione del form, viene mantenuto sottocontrollo in modo semplice ed elegante tutto ciò che riguarda ogni singolo campo d' inserimento. Come si è potuto vedere, attraverso l' uso di poche parole chiave, è possibile implementare e migliorare gli stessi controlli che si sono realizzati nella costruzione con PHP. Il vantaggio di utilizzare questi strumenti deriva dal fatto che il codice che esegue tali controlli è già stato realizzato e testato dal team creatore di Zend Framework; quindi abbiamo l' assoluta certezza della sua correttezza e pertanto al programmatore non resta altro da fare che applicarli correttamente. Inoltre il team di ZF non si è solo adoperato per li implementazione dei controlli come abbiamo appena detto, ma si è anche preoccupato del *rendering*. Infatti lo sviluppatore non deve più preoccuparsi della visualizzazione degli errori nel caso in cui non vengano rispettate le clausole imposte dai validatori perchè sono già stati realizzati nel migliore dei modi e vengono sollecitati in modo automatico senza neppure l' aggiunta di una riga di codice.

Connessione al database

Adesso ci preoccupiamo di mettere a confronto i due modi per effettuare la connessione al database che nel nostro caso è stato realizzato con MySql.

1. Realizzazione con PHP

Algorithm 89 Connessione al database con PHP

```php
$my_host='localhost';
$my_username='my_username';
$my_password='my_password';
$db_name='my_db_name';

$db = mysql_connect($my_host, $my_username, $my_password) or die ("Sever error connection");
mysql_select_db($db_name,$db) or die ("Database error connection");
```

2. Realizzazione con ZEND

Algorithm 90 Connessione al database con ZEND

```php
// File config.ini
[staging : production]
resources.db.adapter = 'Pdo_Mysql'
resources.db.params.host = 'localhost'
resources.db.params.username = 'root'
resources.db.params.password = 'root'
resources.db.params.dbname = 'authentication'
// connessione al database
$config = new Zend_Config_Ini(ROOT_DIR.'/application/configs/application.ini', 'staging');
$dbAdapter = Zend_Db::factory($config->resources->db);
```

Per poter effettuare la connessione al database in PHP è necessario utilizzare i driver PDO (*PHP Data Object*), ovviamente dopo aver configurato opportunamente i parametri di connessione com' è stato riportato al punto 1. I PDO, disponibili dalla versione 5.1 di PHP, sono stati sviluppati per sopperire all' esigenza di avere un' interfaccia generica che permettesse l' accesso a delle basi di dati. Infatti PHP anche prima della loro introduzione permetteva chiaramente il funzionamento di moltissimi databases, ma il problema, però, è sempre stato che le funzioni utilizzate per tale scopo erano diverse per ciascun database quindi, nel caso di una migrazione, sarebbe stato necessario riscrivere totalmente il codice. I PDO invece ci permettono dunque di accedere a varie basi di dati (MySQL, SQLserver, SQLite, PostgreSQL o Oracle) cambiando solamente un paio di stringhe (quelle della connessione appunto). Anche se è stato fatto una grande lavoro di standardizzazione, non c' è da dimenticare che i PDO non sono altro che una classe con delle relative sottoclassi, per cui in caso di cambiamento del DBMS sarebbe necessario andare a modificare il codice sorgente; inoltrenel caso in cui si dovesse presentare la possibilità di gestire contemporaneamente più database potrebbe generarsi il problema della confusione nella dichiarazione e nell' utilizzo dei parametri di connessione. Ferme restando anche queste problematiche i PDO introdotti da PHP rimangono comunque uno dei migliori strumenti per la connessione ai database.

Zend Framework essendo sviluppato interamente in PHP5 impiega anch' esso i PDO per comunicare con i database ma, com' era prevedibile, aggiunge ne migliora le funzionalità e ne completa la standardizzazione. Infatti, come mostra il codice al punto 2, nelle istruzioni di connessione non è presente nessun riferimento al tipo di DBMS utilizzato perciò anche se dovessimo sostituirlo con un' altro non sarebbe necessaria alcuna modifica del codice da parte del programmatore. Inoltre, avendo separato ed incapsulato, all' interno di una specifica sezione (staging), la definizione dei parametri di connessione dal resto del programma inserendoli nel file application.ini, non c' è il rischio che si generi confusione in quanto può esserci la configurazione dei parametri di connessione per un solo database in ogni sezione. Sicuramente un lettore attento se ne sarà già accorto, ma in questo modo risulta estremamente semplice migrare ad un altro DBMS semplicemente cambiando i valori di alcuni di questi parmetri appena discussi. In aggiunta, se seguiamo il discorso fatto, con un piccolissimo sforzo si può gestire facilmente più database contemporaneamente creando una nuova sezione del file di configurazione.

Gestione della visualizzazione

Ultimo, ma non per importanza, rimane da confrontare il modo in cui viene gestita tutta la parte della visualizzazione sperando di far risaltare l' efficacia del pattern MVC di Zend Framework. Purtroppo in questo caso il lettore dovrà usare di più la fantasia, rispetto a quanto fatto prima, sforzandosi di adattare le nostre considerazioni ad una caso reale e di conseguenza più complesso ed articolato perchè data la semplicità della nostra applicazione anche se, per via del codice che verrà presentato, questo punto potrebbe sembrare superfluo non

lo è affatto, ma invece rappresenta il fulcro di tutta la relazione presentata.

1. Realizzazione con PHP

Algorithm 91 Visualizzazione con PHP

```
// Acquisizione dati
if($count=='0'){
        // Elaborazione dati
        if(!"$result"){ print("Errore nell' inserimento dei dati nel database");
        }else{
                mysql_close($db);
                ?><h1 align = "center">Registrazione effettuata con successo</h1>
                <p class = "text">Esegui il <a href="login_form.html">LOGIN</a> opprure ritorna
alla <a href="index.html">HOME PAGE</a>.</p><?php
        }
}else{
        ?><h1><span>Errore.</span></h1><?php
        if($count == 1){
                ?><h3>E' stato trovato <span><?php echo $count?></span> errore</h3><?php
        }else{
                ?><h3>Sono stati trovati <span><?php echo $count?></span> errori</h3><?php
        }
        ?><br><?php for ($i = 1; $i <= 5; $i++) {
                        if($message["$i"] != ""){
                                echo $message[$i];
                        }
                }
        ?><p class = "error">Ritorna alla fase di </label><a
href="newuser_form.html">REGISTRAZIONE</a><br>opprure ritorna alla <a href="index.html">HOME
PAGE</a>.</p><?php
}?>
```

2. Realizzazione con ZEND

Algorithm 92 Visualizzazione con ZEND

```
// Inizializzazioni
$this->view->pageTitle = "Registration' s Form";
$this->view->bodyCorp = "Please fill out this form.";
$this->view->note = "Parameters with * are required.";
$this->view->form = $form;
if ($this->_request->isPost()) {
        // Controllo input ed acquisizione
                if($users->checkUnique($data['username'])){
                        $this->view->errorMessage = "Username already taken. Please choose
another one. ====>";
                        return;
                }
                // Elaborazione dati
        } else {
                $form->populate($formData);
        }
}
// Visualizzazione nel file .phtml
<h1><?php echo $this->pageTitle ;?></h1>
<h3><?php echo $this->bodyCorp ;?></h3>
        // ...
                <tr><h4><? echo $this->escape($this->note); ?></h4></tr>
                <tr><h4><? echo $this->escape($this->errorMessage); ?></h4></tr>
        // ...
                <td><?php echo $this->form ;?></td>
```

Come si è potuto osservare dal codice presentato nella parte di PHP, e per quanto già accenato in precedenza, il codice che si occupa dell' elaborazione dei dati è presente insieme a quello che riguarda la visualizzazone per l' utente. L' utilizzo di questra strategia porta con se due aspetti negativi fondamentali. Il primo è che dato il miscuglio delle due tipologie di codice diventa difficile e poco funzionale, oltre che scrivere il programma in se stesso, eseguire dei controlli in caso si verifichino delle anomalie nella fase di testing. Il secondo invece interessa l' operazione di modifica, infatti se si volesse andare a modificare la parte di codice preposta alla visualizzazione, ad esempio per rendere più accattivante la presentazione del sito, oppure la parte operativa, magari per aggiungere delle nuove funzionalità, ci troveremmo a dover destreggiarci tra un insieme caotico di codice che dovrebbe o non dovrebbe essere modificato, a seconda del tipo di modifica vogliamo apportare, rendendo estremamente arduo il compito del programmatore.

Contrariamente in ZF, grazie soprattutto alla perfetta implementazione del pattern MVC, ed in particolare modo grazie alla simbiosi tra il CONTROLLER e la VIEW, questi problemi sono stati risolti. Infatti, per quanto è stato presentato nel codice al punto 2, è stata eseguita la separazione delle due tipologie di codice, quindi quando stiamo programmando all' interno di un Controller realizzando i servizi e le funzionalita del sito non dobbiamo, per nessun motivo, preoccuparci del rendering dei risultati, ma semplicemente l' oggetto che deve essere visualizzato viene passato alla View senza nessun artificio. Perciò in un Controller non troveremo mai degli script di style, che ovviamente si utilizzeranno nella view per realizzare il rendering, ma solamente codice di elaborazione dati. In questo modo risulta molto facile esaminare, controllare ed eventualmente modificare qualsiasi aspetto del progetto che si sta realizzando.

Infine, anche se si è un pò trascurato visto la semplicità della nostra applicazione, si vuole porre l' attenzione anche sul MODEL. Infatti per architetture complesse questo componente risulta di estrema importanza, soprattutto nel dialogo con i database come viene esposto qui di seguito:

Algorithm 93 Utilizzo del MODEL

```
class NomeClasse extends Zend_Db_Table_Abstract{
        protected $_name = 'nome_tabella';

        public function one(){
                // ...
        }

        public function two(){
                // ...
        }

        // ...
}
```

Come si può osservare ad ogni model viene associata una specifica tabella del database al quale ci si è connessi, così viene resa più semplice l' interrogazione e quindi l' estrapolazione di informazioni dai record presenti. Infatti, grazie

al parametro *$_ name,* che identifica il nome della tabella, non solo non è più necessario inserirla in ogni query che si esegue, ma in questo modo vengono raggruppate anche tutte le operazioni riguardanti ogni singola tabella rendendo più agevole l' aggiornamento o l' aggiunta di nuove funzioni. Questo componente, anche se viene sempre messo in ombra dalle sue controparti data la sua funzione, è comunque molto importante all' interno del pattern MVC e, come se ne renderà conto in seguito nell' appendice B, facilita enormemente la costruzione degli algorimi.

A questo punto possiamo dire di aver esaurito tutti i confronti più rilevanti che riguardavano l' applicazione che abbiamo eseguito nel corso di questo capitolo 4 cercando di dare delle spiegazioni, quantomeno giustificate, che dimostrino i possibili vantaggi nell' utilizzare il framework Zend rispetto alla classica programmazione procedurale o ad oggetti. Anche se ci sarebbe ancora tanto da dire sui miglioramenti che uno sviluppatore di siti web riscontrerebbe utilizzando questo framework, come ad esempio la gestione dei pdf solamente per citarne uno tra i più comuni, per esigenze di trattazione passiamo a presentare le nostre conclusioni convinti che quanto abbiamo fatto sia stato sufficiente per invogliare il lettore a muovere i primi passi nel mondo di ZEND.

5 CONCLUSIONI

Inizialmente i siti internet sviluppati con PHP erano composti da pagine dove il codice HTML risiedeva nello stesso file del codice php, come abbiamo mostrato nella realizzazione della prima versione della nostra applicazione nel paragrafo 4.2, e questo modo di operare sembrava tutto quello che si potesse desiderare. Con l' andare del tempo e il crescere dei progetti sia in dimensione che in complessità iniziarono a manifestarsi i primi problemi nell' utilizzo di questa logica, questi principalmente riguardavano due aspetti fondamentali quali:

- la mantenibilità
- l' estensibilità.

Iniziarono quindi a nascere delle strutture con lo scopo di risolvere tali problematiche, una di queste è proprio il Framework Zend.

Infatti *Zend Framework* offre un gran numero di componenti che danno la possibilità al programmatore di creare applicazioni mantenibili ed estensibili a lungo termine ed inoltre implementa diversi *pattern* tra cui l' ormai famoso **Model - View - Controller** per la suddivisione della logica del recupero dei dati dal database e il successivo *render* su schermo. Non è un caso quindi che anche altri framework come ad esempio Joomla e ASP, solo per ricordarne due tra i più conosciuti, si stiano avvicinando ad utilizzare un' architettura basata sull' MVC.

Comunque, tornando all' oggetto dalla nostra analisi, ZF è sicuramenteuno dei più importanti strumentiche hanno permesso a PHP di raggiungere livelli di stabilità, sicurezza e manutenibilità difficili da raggiungere altrimenti. Grazie al contributo fornito da questo potente framework, l' utilizzo di PHP per applicativi *web enterprise* è diventato realtà. Purtroppo dato il carattere di questa relazione il cui scopo è solamente quello di introdurre l' utilizzo del Framework Zend, non c' è stato permesso di addentrarci in tanti altri argomenti interessanti, nè di trattare l' impiego e la configurazione di tanti altri servizi di cui dispone questo particolare strumento, il cui mondo è davvero vasto e tutto da scoprire.

Durante la fase di realizzazione ci si è accorti che l' implementazione dei principali design pattern e il supporto dei diversi tool per lo sviluppo di applicazioni con ZF rendono l'apprendimento di questo prodotto più rapido e semplice di quanto ci si potesse mai aspettare, infatti con l' impiego di poche righe di codice possiamo strutturare un applicativo perfettamente funzionante, che con il tempo e l'esperienza possiamo imparare a personalizzare ed estendere praticamente in ogni suo aspetto.

5.1 DISPONIBILITA' DI HOSTING

Da quello che è emerso durante le conclusioni non c' è nessun particolare motivo per il quale uno sviluppatore non dovrebbe scegliere di utilizzare *Zend Framework* anzichè sviluppare la sua applicazione con il linguaggio PHP. Perciò

possiamo affermare di aver esaurito completamente l 'espozione del nostro argomento, avendolo già trattato in ogni suo aspetto, salvo forse per una piccola ma fondamentale eccezione: **la disponibilità di hosting**.

> **HOSTING:**
>
> In informatica si definisce *hosting* (dall' inglese to host, ospitare) un servizio di rete che consiste nell' allocare su un *server web* le pagine web di un sito web, rendendolo così accessibile dalla rete Internet e ai suoi utenti. Tale "server web", definito "host", è connesso ad Internet in modalità idonea a garantire l' accesso alle pagine del sito mediante il web browser dell' host client dell' utente, con identificazione dei contenuti tramite dominio ed indirizzo IP.

Siccome lo scopo di progettare una *web application* è quella di renderla visibile dalla Rete ne consegue che il successo o il fallimento di tutto il nostro studio dipende dalla risposta alla seguente domanda:

> ***Esistono provider che consentano il funzionamento di web application sviluppate con Zend Framework ?***

Infatti per quanto valido possa essere lo strumento che abbiamo presentato è del tutto inutile utilizzarlo se non sono presenti dei *provider* che lo supportano.

Per nostra fortuna, il problema che abbiamo sollevato non sussiste in quanto esistono numerosi *provaider* che sono in grado di soddisfare le richiete di uno sviluppatore di applicazioni Zend, sia esteri che italiani dei quali se ne riporta una lista:

- servergroove
- a2hosting
- NTLab Italia
- Aruba
- Cloud

Quindi data la semplicità di realizzazione, la potenza del dispositivo stesso grazie anche all' MVC e alle numerose soluzioni per l' hosting il Framework Zend ha tutte le carte in regola per diventare uno dei migliori e dei più utilizzati framework di sempre e con questo concludiamo tutta la nostra trattazione soddisfatti di aver presentato uno strumento che avrà una notevole ripercussione nel prossimo futuro.

6 BIBLIOGRAFIA

- RAMEZ A. ELMASRI & SHAMKANT B. NAVATHE (2007) a cura di SILVANA CASTANO "Sistemi di basi di dati Fondamenti", 5th edition, Pearson Addison Wesley
- PAOLO ATZENI et al. (2007) "Basi di dati Architetture e linee di evoluzione", 2nd edition, McGraw-Hill
- ZEND TECHNOLOGIES LTD. (2012) "Zend Framework 2 documentation", release 2.0.3dev, Zend Technologies Ltd.
- PHPEveryDay PHP and Web Development Tutorial "Zend Framework Basic Tutorial" <http://www.phpeveryday.com/articles/Zend-Framework-Basic-Tutorial-P840.html> 2010
- ENRICO ZIMUEL PHP goes mobile "Sviluppo di applicazioni web mobile con Zend Framework" <http://www.slideshare.net/e.zimuel/php-goes-mobile> 2012
- MEHDI ACHOUR et al. "Manuale PHP" <http://php.net/manual/it/index.php> 2013
- ZEND TECHNOLOGIES Zend Framework 2 "The most popular framework for modern, hight-performing PHP application"<http://framework.zend.com/> 2006 - 2013
- ENRICO ZIMUEL "Programmare con PHP, best practices" & "Applicazioni web mobile con Zend Framework" <http://www.zimuel.it/> 2012
- MARCO LECCE "Guida Zend Framework" <http://www.html.it/guide/guida-zend-framework/> 2011
- FREECONTACTFORMS "How to make an HTML form, in simple steps" <http://www.freecontactform.com/html_form.php> 2005 - 2013
- MORRIS "PHP e il database" <http://morris-world.it/art5.php> 2012
- ALLWEB FREE "PHP" <http://www.allwebfree.it/php_php.php> 2012
- W3SCHOOLS.COM "CSS Examples" <http://www.w3schools.com/css/css_examples.asp> 1999 - 2013
- CLAYTON MCILRATH "Installing Zend Framework on MAMP" <http://thinkclay. com/technology/installing-zend-framework-on-mamp> 2011
- FAHEEM ABBAS "Zend Framework sign up and authentication" <http://zendgeek. blogspot.it/2009/07/zend-framework-sign-up-and.html> 2009
- RICHARD KNOP "User login and authentication with Zend_Auth and Zend_Acl" <http://blog.richardknop.com/2009/06/user-login-and-authentication-with-zend_auth-and-zend_acl/> 2009

- LUIS FREITAS "How to build a complete Zend Framework application" <http://blog.luisfreitas.pt/2010/11/06/how-to-build-a-complete-zend-framework-application/> 2010
- ROB ALLEN "Getting Started with Zend_Auth" <http://akrabat.com/zend-auth-tutorial/>
- JOSE DA SILVA "Starting with Zend Framework – Configuration File and Database Connection" <http://blog.josedasilva.net/starting-with-zend-framework-config-registry-db/> 2008
- SERGIO RINAUDO "Costruire un'applicazione reale con Zend Framework" <http://www.sergiorinaudo.com/costruire-unapplicazione-reale-con-zend-framework-parte-1/> 2008
- STAKOVERFLOW "Zend Db connection and configuration" <http://stackoverflow.com/questions/5998122/zend-db-connection-and-configuration> 2011
- ENRICO ZIMUEL "Build a secure login with Zend Framework" <http://www.zimuel.it/en/build-a-secure-login-with-zend-framework/> 2009
- STAKOVERFLOW "Zend: Redirect to Action with parameters" <http://stackoverflow.com/questions/1913509/zend-redirect-to-action-with-parameters> 2009
- SERGIO RINAUDO "PHP Warning: Cannot modify header information – headers already sent" <http://www.sergiorinaudo.com/php-warning-cannot-modify-header-information-headers-already-sent/> 2009
- CLAUDIO GARAU "PHP/OOP: Creare un sistema di registrazione e autenticazione per gli utenti" <http://www.mrwebmaster.it/php/php-oop-creare-sistema-registrazione-autenticazione-utenti_7706.html> 2011
- GÜAYOYO BIRONCHO "Guida allo Zend Framework" <http://www.rhadrix.net/?/Php/Zend/Guida-allo-Zend-Framework3> 2011
- MARCO LECCE "Zend Framework – Disabilitare il layout e la view" <http://www.marcolecce.com/blog/2010/07/22/zend-framework-disabilitare-il-layout-e-la-view/> 2010
- ANDREA ROMAGNOLI "ZF – Connessione simultanea a più database" <http://www.andrearomagnoli.name/2011/05/zend-framework-connessione-simultanea-a-piu-database/> 2011

7 APPENDICE A

Algorithm 94 index.html

```
<style>
        body{ background-image:url('php.jpeg'); background-repeat:no-repeat;
              background-position: 100% 100%; }
        h1{ color:navy; text-align:center; font-family:"Times New Roman"; font-size:40px; }
        h3{ color:black; text-align:center; font-family:"Times New Roman"; font-size:25px; }
        span{ color: red; }
        p.text{ text-align:justify; font-family:"Times New Roman"; font-size:20px;
                position:absolute; left:34%; top:40%; }
</style>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"> <html>
<head>
        <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
        <title>HOME_PAGE</title>
</head>
<body>

        <h1>HOME PAGE</h1>
        <h3>Benvenuti sul mio sito. Questo sito e' realizzato con <span>PHP! </span>Buon
divertimento</h3>

        <br>

        <p class = "text">Perfavore esegui il <a href="login_form.html">login</a> o <a href =
"newuser_form.html">registrati</a> sul nostro sito.</p>

</body>
</html>
```

Algorithm 95 newuser_form.html

```
<style>
        h1{ color:navy; text-align:center; font-family:"Times New Roman"; font-size:40px; }
        h3{ color:black; text-align:center; font-family:"Times New Roman"; font-size:25px; }
        span{ color: red; }
        table.cover{ border-style:solid; border-left-style:none; border-right-style:none;
                     border-color:#ff0000; width:100%; background-color:#EAF2D3; }
</style>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"> <html>
<head>
        <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
        <title>NEWUSER_FORM</title>
</head>
<body>
        <h1>Effettua la registrazione</h1>
        <h3>Perfavore compila i campi richiesti</h3>

        <form method="GET" action="inserimento.php" align="center">
        <table class = "cover">
                <caption><span>I campi con * sono obbligatori</span></caption>
                <tr>
                        <td width = 36%></td>
                        <td><table>

                                <tr>
                                        <td><label>username<span> *</span> :</label></td>
                                        <td><input name="username" type="text" size="40"
maxlength="50"/></td>
                                </tr>
                                <tr>
                                        <td><label>password<span> *</span> :</label></td>
                                        <td><input type="password" maxlength="50" size="40"
name="password" /></td>
                                </tr>
                                <tr>
                                        <td><label>full name :</label></td>
                                        <td><input name="fullname" type="text" size="40"
maxlength="150" /></td>
                                </tr>
                                <tr>
                                        <td><label>e-mail<span> *</span> :</label></td>
                                        <td><input name="email" type="text" size="40"
maxlength="200" /></td>
                                </tr>
                                <tr>
                                        <td><label>data di nascita :</label></td>
                                        <td><input name="birthday" type="text" size="40"
maxlength="10" value = "aaaa/mm/gg"/></td>
                                </tr>
                                <tr>
                                        <td></td>
                                        <td><button type="submit">Vai</button> <button
type="reset">Cancella</button></td>
                                </tr>
                        </table></td>
                </tr>
        </table class ="cover">
        </form>

</body>
</html>
```

Algorithm 96 login_form.html

```
<style>
        h1{ color:navy; text-align:center; font-family:"Times New Roman"; font-size:40px; }
        h3{ color:black; text-align:center; font-family:"Times New Roman"; font-size:25px; }
        span{ color: red; }
        table.cover{ border-style:solid; border-left-style:none; border-right-style:none;
                     border-color:#ff0000; width:100%; background-color:#EAF2D3; }
</style>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"> <html>
<head>
        <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
        <title>LOGIN_FORM</title>
</head>
<body>
        <h1>Effettua il login</h1>
        <h3>Inserisci i tuoi dati</h3>
        <form method="GET" action="autenticazione.php" align="center">
        <table class = "cover">
                <caption><span>I campi con * sono obbligatori</span></caption>
                <tr>
                        <td width = 41%></td>
                        <td><table>

                                <tr>
                                        <td><label>username<span> *</span> :</label></td>
                                        <td><input name="username" type="text" size="20"
maxlength="50" /></td>
                                </tr>
                                <tr>
                                        <td><label>password<span> *</span> :</label></td>
                                        <td><input type="password" maxlength="50" size="20"
name="password" /></td>
                                </tr>
                                <tr>
                                        <td></td>
                                        <td><button type="submit">Accedi</button></td>
                                </tr>
                        </table></td>
                </tr>
        </table class ="cover">
        </form>
</body>
</html>
```

Algorithm 97 inserimento.php parte 1

```
<style>
        h1{ color:darkgreen; font-family:"Times New Roman"; font-size:40px; }
        h3{ position:absolute; left:12%; top:6%; }
        p.text{ text-align:justify; font-family:"Times New Roman"; font-size:20px;
                position:absolute; left:32%; top:20%; }
        p.error { text-align:justify; font-family:"Times New Roman"; font-size:20px;
                position:absolute; left:38%; top:40%; }
        span{ color: red; }
</style>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"> <html>
<head>
        <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body>
<?php
        $my_host='localhost';
        $my_username='root';
        $my_password='root';
        $db_name='authentication';
        $table_name='users';

        $count=0;
        $message = array("","","","","");

        if($_GET['username'] == ''){
                $count=$count+1;
                $message["1"] ="Il campo username e' vuoto. Devi inserire un username<br>";
        }elseif(strpbrk($_GET['username'], ' ,£,$,%,&,/,(,),=,?,^,>,<,|,\,ç,@,#,+,*,[,],§,°')){
                $count=$count+1;
                $message["1"] ="Devi inserire una username corretta<br>";
        }else{
                $username = $_GET['username']; }

        if($_GET['password'] == ''){
                $count=$count+1;
                $message["2"] ="Il campo password e' vuoto. Devi inserire una password<br>";
        }elseif(strpbrk($_GET['password'], ' ,£,$,%,&,/,(,),=,?,^,>,<,|,\,ç,@,#,+,*,[,],§,°')){
                $count=$count+1;
                $message["2"] ="Devi inserire una password valida<br>";
        }else{
                $password = $_GET['password']; }

        if($_GET['fullname']!=''){
                $fullname=$_GET['fullname'];
                if(!preg_match('/^[a-zA-Z]/', $fullname)){
                        $count=$count+1;
                        $message["3"] ="Devi inserire un nome valido<br>"; }
        }

        if($_GET['email']==''){
                $count=$count+1;
                $message["4"] ="Il campo e-mail e' vuoto. Devi inserire una e-mail<br>";
        }elseif(strpbrk($_GET['email'], ' ') || !strpbrk($_GET['email'], '@') || !strpbrk($_GET
['email'], '.') || strrpos($_GET['email'], ".") < strrpos($_GET['email'], "@")){
                $count=$count+1;
                $message["4"] ="Devi inserire un' e-mail valida<br>";
        }else{
                $email=$_GET['email']; }
```

Algorithm 98 inserimento.php parte 2

```
        if($_GET['birthday']!=''){
                $date=$_GET['birthday'];
                if(strlen($date)==10){
                        if(preg_match('/[0-9]{4}\/[0-9]{2}\/[0-9]{2}/',$date) || preg_match('/[0-9]
{4}-[0-9]{2}-[0-9]{2}/',$date)){
                                $date=str_replace('/','-',$date);
                                $anno=substr($date,0,4);
                                $mese=substr($date,5,2);
                                $giorno=substr($date,8,2);
                                if(!checkdate($mese, $giorno, $anno)){
                                        $count=$count+1;
                                        $message["5"] ="La data inserita non e' valida<br>"; }
                        }else{
                                $count=$count+1;
                                $message["5"] ="Formato data non valido<br>"; }
                }else{
                        $count=$count+1;
                        $message["5"] ="Devi inserire una data valida<br>"; }
        }

        if($count=='0'){

                $db = mysql_connect($my_host, $my_username, $my_password) or die ("Sever error
connection");
                mysql_select_db($db_name,$db) or die ("Database error connection");

                $pwd = md5($password);

                $query="INSERT INTO $table_name (username, password, fullname, email, birthday)
VALUES('".$_GET['username']."','$pwd','".$_GET['fullname']."','".$_GET['email']."','".$_GET
['birthday']."')";
                $result=mysql_query($query,$db);
                if(!"$result"){
                        print("Errore nell' inserimento dei dati nel database");
                }else{
                        mysql_close($db);
                        ?>
                        <h1 align = "center">Registrazione effettuata con successo</h1>
                        <p class = "text">Esegui il <a href="login_form.html">LOGIN</a> opprure
ritorna alla <a href="index.html">HOME PAGE</a>.</p>
                        <?php
                }
        }else{
                ?>
                <h1><span>Errore.</span></h1>
                <?php
                if($count == 1){
                        ?>
                        <h3>E' stato trovato <span><?php echo $count?></span> errore</h3>
                        <?php
                }else{
                        ?>
                        <h3>Sono stati trovati <span><?php echo $count?></span> errori</h3>
                        <?php
                }?>
                <br>
                <?php
                for ($i = 1; $i <= 5; $i++) {
                        if($message["$i"] != ""){
                                echo $message[$i];
                        }
                }?>
                <p class = "error">Ritorna alla fase di </label><a
href="newuser_form.html">REGISTRAZIONE</a><br>opprure ritorna alla <a href="index.html">HOME PAGE</
a>.</p>
                <?php
        }?>
</body>
</html>
```

Algorithm 99 autenticazione.php parte 1

```
<style>
        h1{ color:darkorange; font-family:"Times New Roman"; font-size:40px; }
        h3{ position:absolute; left:12%; top:6%; }
        p.text{ text-align:justify; font-family:"Times New Roman"; font-size:20px;
                position:absolute; left:33%; top:20%; }
        p.error { text-align:justify; font-family:"Times New Roman"; font-size:20px;
                  position:absolute; left:38%; top:40%; }
        span{ color: red; }
</style>
<!DOCTYPE HTML PUBLIC Ò-//W3C//DTD HTML 4.01 Transitional//ENÓ><html>
<head>
        <meta http-equiv=ÔContent-TypeÔ content=Ótext/html; charset=iso-8859-1?>
</head>
<body>
<?php
        session_start();
        $my_host='localhost';
        $my_username='root';
        $my_password='root';
        $db_name='authentication';
        $table_name1='users';
        $table_name2='loggedusers';

        $error=0;
        $message = array("","");

        if($_GET['username'] == ''){
                $error=$error+1;
                $message["1"] ="Il campo username e' vuoto. Devi inserire un username<br>";
        }elseif(strpbrk($_GET['username'], ' ,£,$,%,&,/,(,),=,?,^,>,<,!,\,?,@,#,+,*,[,],Ù,Á')){
                $error=$error+1;
                $message["1"] ="Devi inserire una username corretta<br>";
        }else{
                $username = $_GET['username']; }

        if($_GET['password'] == ''){
                $error=$error+1;
                $message["2"] ="Il campo password e' vuoto. Devi inserire una password<br>";
        }elseif(strpbrk($_GET['password'], ' ,£,$,%,&,/,(,),=,?,^,>,<,!,\,?,@,#,+,*,[,],Ù,Á')){
                $error=$error+1;
                $message["2"] ="Devi inserire una password valida<br>";
        }else{
                $password = $_GET['password'];
                $pwd = md5($password); }

        if($error==0){

                $db = mysql_connect($my_host, $my_username, $my_password) or die ("Sever error
connection");
                mysql_select_db($db_name,$db) or die ("Database error connection");
                $query="SELECT * FROM $table_name1 WHERE username='$username' AND password='$pwd'";
                $result=mysql_query($query);
                $count=mysql_num_rows($result);
                $array_result=mysql_fetch_row($result);
                $id=$array_result[0];
                mysql_close($db);
```

Algorithm 100 autenticazione.php parte 2

```
                if ($count==1){

                        $_SESSION["username"] = $username;
                        $_SESSION["pwd"] = $pwd;

                        $db = mysql_connect($my_host, $my_username, $my_password) or die ("Sever
error connection");
                        mysql_select_db($db_name,$db) or die ("Database error connection");
                        $query1="SELECT * FROM $table_name2 WHERE userid='$id'";
                        $result1=mysql_query($query1);
                        $count1=mysql_num_rows($result1);

                        $date = date("y-m-d H:m:s");

                        if($count1==1){
                                $query2="UPDATE $table_name2 SET lastupdate='$date' WHERE
userid='$id'";
                                $result2=mysql_query($query2);
                        }else{
                                $query3="INSERT INTO $table_name2 (userid,sessionid,lastupdate)
VALUES('$id', '$username','$date')";
                                $result3=mysql_query($query3);
                        }
                        mysql_close($db);

                        session_write_close();
                        header ("location: pagina.php");
                }else{?>
                        <h1 align = "center">Utente non trovato<br></h1>
                        <p class = "text">Riesegui il <a href="login_form.html">LOGIN</a> opprure
ritorna alla <a href="index.html">HOME PAGE</a>.</p>
                        <?php
                }
        }else {?>
                <h1><span>Errore.</span></h1>
                <?php
                if($error == 1){?>
                        <h3>E' stato trovato <span><?php echo $error?></span> errore</h3>
                        <?php
                }else{?>
                        <h3>Sono stati trovati <span><?php echo $error?></span> errori</h3>
                        <?php
                }?>
                <br>
                <?php
                for ($i = 1; $i <= 5; $i++) {
                        if($message["$i"] != ""){
                                echo $message[$i];
                        }
                }?>
                <p class = "error">Ritorna alla fase di </label><a href="login_form.html">LOGIN</
a><br>opprure ritorna alla <a href="index.html">HOME PAGE</a>.</p>
                <?php
        }?>
</body>
</html>
```

Algorithm 101 pagina.php

```
<style type="text/css">
        h1{ color:darkgreen; font-family:"Times New Roman"; font-size:40px; text-align:center; }
        h3{ font-family:"Times New Roman"; font-size:20px; text-align:center; }
        span{ color: red; }
        p{ text-align:justify; font-family:"Times New Roman"; font-size:20px;
           font-weight:bold; position:absolute; left:75%; top:90%; }
        #submit{ background-color: #FF8800; border: 2px solid #FCA800; color: #fff;
                 font-weight: bold; padding: 0; height: 25px; width: 80px; }
</style>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"> <html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <title>AREA RISERVATA</title>
</head>
<body>
<?php
    session_start();
    if (!isset($_SESSION['username']) && !isset($_SESSION['pwd'])){
        header("location:index.html");
        exit; } ?>

    <h1>Login effettuato con successo<h1>
    <h3>Ciao <span><?php echo $user =$_SESSION['username']; ?></span>.Spero ti sia divertito e che
ti sia servito utilizzare questo sito sviluppato con <span>PHP</span></h3>

    <form action="logout.php" align="center">
                <table width="100%" align="center"border="0" cellspacing="0">
                    <tr bgcolor="Gray">
                            <td width ="50%"><font color="white">Sei connesso come: <?php
                                                $user =$_SESSION['username'];
                                                echo "$user"; ?></font></td>
                            <td width="50%" align="right"><input type="submit" id="submit"
name="submit" value="Logout"/></td>
                    </tr>
                </table>
    </form>

    <p><span>Sviluppato da Pettenuzzo Giordano</span></p>

</body>
</html>
```

Algorithm 102 logout.php

```
<?php
        session_start();
        if (isset($_SESSION['username']) && isset($_SESSION['pwd'])){
            session_unset();
            session_destroy();
            session_write_close();
            session_regenerate_id(true);
            header ("location:index.html"); }
?>
```

8 APPENDICE B

Algorithm 103 public - index.php

```
<?php
define('ROOT_DIR', dirname(dirname(__FILE__)));

// Setup path to the Zend Framework files
set_include_path('.'
. PATH_SEPARATOR . ROOT_DIR.'/application/'
. PATH_SEPARATOR . ROOT_DIR.'/application/models'
. PATH_SEPARATOR . ROOT_DIR.'/library/'
. PATH_SEPARATOR . get_include_path()
);

// Register the autoloader
require_once "Zend/Loader/Autoloader.php";
$autoloader = Zend_Loader_Autoloader::getInstance();
$autoloader->setFallbackAutoloader(true);

// Initialise Zend_Layout's MVC helpers
Zend_Layout::startMvc(array('layoutPath' => ROOT_DIR.'/application/views/layouts'));

// set up database
require_once 'Zend/Registry.php';
$config = new Zend_Config_Ini(ROOT_DIR.'/application/configs/application.ini', 'staging');
$dbAdapter = Zend_Db::factory($config->resources->db);
$registry = Zend_Registry::getInstance();
Zend_Registry::set('configuration',$config);
Zend_Registry::set('dbAdapter',$dbAdapter);

// Run!
$frontController = Zend_Controller_Front::getInstance();
$frontController->addControllerDirectory(ROOT_DIR.'/application/controllers');
$frontController->throwExceptions(true);
try {
      $frontController->dispatch();
} catch(Exception $e) {
      echo nl2br($e->__toString());
}
?>
```

Algorithm 104 application.ini

```
[production]
phpSettings.display_startup_errors = 0
phpSettings.display_errors = 0
includePaths.library = APPLICATION_PATH "/../library"
bootstrap.path = APPLICATION_PATH "/Bootstrap.php"
bootstrap.class = "Bootstrap"
appnamespace = "Application"
resources.frontController.controllerDirectory = APPLICATION_PATH "/controllers"
resources.frontController.params.displayExceptions = 0

[staging : production]
resources.db.adapter = 'Pdo_Mysql'
resources.db.params.host = 'localhost'
resources.db.params.username = 'root'
resources.db.params.password = 'root'
resources.db.params.dbname = 'authentication'

[testing : production]
phpSettings.display_startup_errors = 1
phpSettings.display_errors = 1

[development : production]
phpSettings.display_startup_errors = 1
phpSettings.display_errors = 1
resources.frontController.params.displayExceptions = 1
```

Algorithm 105 BaseUrl.php

```
<?php
class Zend_View_Helper_BaseUrl
{
    function baseUrl()
    {
        $fc = Zend_Controller_Front::getInstance();
        $request = $fc->getRequest();
        return $request->getBaseUrl();
    }
}
```

Algorithm 106 LoginForm.php

```php
<?php
class forms_LoginForm extends Zend_Form
{
    public function __construct($options = null)
    {
        parent::__construct($options);
        $this->setName('login');
        $this->setMethod('post');

        $username = new Zend_Form_Element_Text('username');
        $username->setLabel('Username:*')
                 ->setRequired(true)
                 ->addFilter('StripTags')
                 ->addFilter('StringTrim')
                 ->addValidator('NotEmpty');

        $password = new Zend_Form_Element_Password('password');
        $password->setLabel('Password:*')
                 ->setRequired(true)
                 ->addValidator('NotEmpty');

        $submit = new Zend_Form_Element_Submit('submit');
        $submit->setLabel('Login');

        $this->addElements(array($username, $password, $submit));
    }
}
```

Algorithm 107 RegistrationForm.php

```
<?php
class forms_RegistrationForm extends Zend_Form
{
    public function __construct($options = null)
    {
        parent::__construct($options);
        $this->setName('registration');

        $fullname = new Zend_Form_Element_Text('fullname');
        $fullname->setLabel('Full name:')
                 ->addFilter('StripTags')
                 ->addFilter('StringTrim')
                 ->addValidator('regex', false, array('/^[a-zA-Z]/'))
                 ->addValidator('stringLength', false, array(6, 20))
                 ->addValidator('alnum', false, array(' '));

        $username = new Zend_Form_Element_Text('username');
        $username->setLabel('User name:*')
                  ->setRequired(true)
                  ->addFilter('StripTags')
                  ->addFilter('StringTrim')
                  ->addValidator('NotEmpty');

        $password = new Zend_Form_Element_Password('password');
        $password->setLabel('Password:*')
                  ->setRequired(true)
                  ->addValidator('NotEmpty');

        $email = new Zend_Form_Element_Text('email');
        $email->setLabel('Email address:*')
              ->addFilter('StringToLower')
              ->setRequired(true)
              ->addValidator('NotEmpty')
              ->addValidator('EmailAddress');

        $birthday = new Zend_Form_Element_Text('birthday');
        $birthday->setLabel('birthday:')
                 ->addValidator('Date'); ;

        $submit = new Zend_Form_Element_Submit('submit');
        $submit->setLabel('Registration');

        $cancel = new Zend_Form_Element_Reset('reset');
        $cancel->setLabel('Cancel');

        $this->addElements(array($fullname, $username, $password, $email, $birthday, $submit,
$cancel));
    }
}
```

Algorithm 108 IndexController.php

```
<?php
class IndexController extends Zend_Controller_Action
{
        public function indexAction()
        {
                $this->view->pageTitle = "My Home page";
                $this->view->bodyCorp = "Welcome to my site. This site is built using ";
                $this->view->zendName = "Zend Framework! ";
                $this->view->fun = "Enjoy it!";
                $this->view->separator = " | ";
                $this->view->footer ="Developed by Pettenuzzo Giordano";
        }
}
?>
```

Algorithm 109 RegistrationController.php

```
<?php
class RegistrationController extends Zend_Controller_Action
{
        function indexAction()
        {
                $this->view->pageTitle = "Registration' s Form";
                $this->view->bodyCorp = "Please fill out this form.";
                $this->view->note = "Parameters with * are required.";

                $form = new forms_RegistrationForm();
                $registry = Zend_Registry::getInstance();

                $DB = Zend_Registry::get('dbAdapter');
                $users = new Users(array('db' => $DB));
                $this->view->form = $form;

                if ($this->_request->isPost()) {
                        $formData = $this->_request->getPost();
                        if ($form->isValid($formData)) {
                                $data = $form->getValues();
                                if($users->checkUnique($data['username'])){
                                        $this->view->errorMessage = "Username already taken.
Please choose another one. ====>";
                                        return;
                                }
                                $users->create($data);
                                $this->_helper->redirector('success');
                        } else {
                                $form->populate($formData);
                        }
                }
        }

        public function successAction()
        {
                $this->view->pageTitle = "Registration Process";
                $this->view->bodyCorp = "Registration Success.";
                $this->view->footer ="Developed by Pettenuzzo Giordano";
        }
}
?>
```

Algorithm 110 LoginController.php parte 1

```
<?php
class LoginController extends Zend_Controller_Action
{
        function indexAction()
        {
                $auth = Zend_Auth::getInstance();

                $this->view->pageTitle = "Login' s Form";
                $this->view->bodyCorp = "Insert data.";
                $this->view->note = "Parameters with * are required.";

                $form = new forms_LoginForm();
                $registry = Zend_Registry::getInstance();

                $DB = Zend_Registry::get('dbAdapter');
                $loggedUsers = new LoggedUsers(array('db' => $DB));
                $users = new Users(array('db' => $DB));
                $this->view->form = $form;

                if ($this->_request->isPost()) {
                        $formData = $this->_request->getPost();
                        if ($form->isValid($formData)) {
                                $data = $form->getValues();

                                $authAdapter = new Zend_Auth_Adapter_DbTable($users->getAdapter(),
$users->getName());
                                $authAdapter->setTableName('users')
                                            ->setIdentityColumn('username')
                                            ->setCredentialColumn('password');
                                $authAdapter->setIdentity($data['username'])
                                            ->setCredential(md5($data['password']));
                                $result = $auth->authenticate($authAdapter);
                                switch ($result->getCode()) {
                                        case Zend_Auth_Result::FAILURE_IDENTITY_NOT_FOUND:
                                                $this->view->errorUser = 'Identity not found.
====>';
                                        break;

                                        case Zend_Auth_Result::FAILURE_CREDENTIAL_INVALID:
                                                $this->view->errorPwd = 'Invalid credential.
====>';
                                        break;

                                        case Zend_Auth_Result::SUCCESS:
                                                $user = $authAdapter->getResultRowObject();
                                                $userid = $user->id;
                                                $sessionid = $user->username;
                                                $date = new Zend_Date();
                                                $lastupdate = $date->get('YYYY-MM-dd HH:MM:ss');
                                                $param = array('sessionid' => $sessionid,'userid'
=> $userid, 'lastupdate' => $lastupdate);
                                                if($loggedUsers->findUser($data['username'])== 1){
                                                        $loggedUsers->updateUser($lastupdate,
$sessionid);
                                                }
                                                else{
                                                        $loggedUsers->create($param);
                                                }
```

Algorithm 111 Login controller.php parte 2

```
                                        /*------ costruzione sessione ------*/
                                        if($result->isValid()){
                                                $storage = new Zend_Auth_Storage_Session();
                                                $storage->write($authAdapter-
>getResultRowObject());
                                        } else {
                                                $this->view->errorMessage = "Something was
wrong with your authentication.";
                                        }
                                        //reindirizzamento
                                        $this->_helper->redirector('loggedin');
                                break;

                                default:
                                        $this->view->errorMessage = 'Sorry but we can not
find your user. ====>';
                                break;
                        }
                }
                else {
                        $form->populate($formData);
                }
        }

    }

    public function loggedinAction()
    {
            $storage = new Zend_Auth_Storage_Session();
            if ($storage->IsEmpty()){
                    $this->_helper->redirector('index', 'Index');
                    exit;
            }
            $user = $storage->read();
            $this->view->pageTitle = "Login Process";
            $this->view->bodyCorp = "<c>Login Success</c>. Hi <span>$user->username</span> i
hope it been useful use this site made with <span>Zend</span>.";
            $this->view->footer ="Developed by Pettenuzzo Giordano";
            $this->view->id = $user->id;
            $this->view->username = $user->username;
            $this->view->fullname = $user->fullname;
            $this->view->email = $user->email;
            $this->view->birthday = $user->birthday;
    }

    public function logoutAction()
    {
            $this->_helper->layout->disableLayout();
            $this->_helper->viewRenderer->setNoRender(true);
            /*----- per esegiure il logout -----*/
            $storage = new Zend_Auth_Storage_Session();
            $storage->clear();
            Zend_Auth::getInstance()->clearIdentity();
            $this->_helper->redirector('index', 'Index');
    }
}
?>
```

Algorithm 112 Users.php

```php
<?php
class Users extends Zend_Db_Table_Abstract
{
        protected $_name = 'users';

        public function getName()
        {
                return $this->_name;
        }

        public function create(array $data)
        {
                $data['password'] = md5($data['password']);
                return $this->insert($data);
        }

        function checkUnique($username)
        {
                $select = $this->_db->select()
                            ->from($this->_name,array('username'))
                            ->where('username=?',$username);
                $result = $this->getAdapter()->fetchOne($select);
                if($result){
                        return true;
                }
                return false;
        }
}
?>
```

Algorithm 113 LoggedUsers.php

```
<?php
class LoggedUsers extends Zend_Db_Table_Abstract
{
        protected $_name = 'loggedusers';

        public function create(array $data)
        {
                return $this->insert($data);
        }

        function findUser($sessionid)
        {
                $select = $this->_db->select()
                            ->from($this->_name,array('sessionid'))
                            ->where('sessionid=?',$sessionid);
                $result = $this->getAdapter()->fetchOne($select);
                if($result){
                        return true;
                }
                return false;
        }

        public function updateUser($date, $where)
        {
                $table = $this->_name;
                // Start a transaction explicitly.
                $this->getAdapter()->beginTransaction();
                try {
                        $sql = "UPDATE $table SET lastupdate='$date' WHERE sessionid='$where'";
                        $this->getAdapter()->query($sql);
                        $this->getAdapter()->commit();
                } catch (Exception $e) {
                        $this->getAdapter()->rollBack();
                        echo $e->getMessage();
                }
        }
}
?>
```

Algorithm 114 layout.phtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title><?php echo $this->escape($this->pageTitle); ?></title>
    <link rel="stylesheet" href="<?php echo $this->baseUrl(); ?>/main.css" type="text/css">
</head>
<body>
    <div id="content">
        <?php echo $this->layout()->content ?>
    </div>
</body>
</html>
```

Algorithm 115 main.css

```
body {background-image:url('zf.jpg'); background-repeat:no-repeat;
      background-position: 100% 1400%;}

h1{color:navy; text-align:center; font-family:"Times New Roman";
   font-size:40px; margin-bottom:5px;}

h3{color:black; text-align:center; font-family:"Times New Roman";
   font-size:25px;}

h4{color:red; text-align:center; font-family:"Times New Roman";}

p.text{color:black; text-align:justify; font-family:"Times New Roman";
       font-size:20px; position:absolute; left:43%; top:40%;}

span{color: red;}

caption {caption-side:top;}
```

Algorithm 116 IndexController view -> index.phtml

```
<style>
        p.footer{ text-align:justify; font-family:"Times New Roman"; font-size:20px;
           font-weight:bold; position:absolute; left:68%; top:92%; }
</style>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <title><? echo $this->escape($this->pageTitle); ?></title>
</head>

<body>
        <h1><? echo $this->escape($this->pageTitle); ?></h1>

        <h3><? echo $this->escape($this->bodyCorp); ?><span><? echo $this->escape($this->zendName); ?
></span><? echo $this->escape($this->fun); ?></h3>

        <p class = "footer"><span><?php echo $this->footer ;?></span></p class = "footer">

        <p class = "text">Possible Actions:<br />
                <a href="<?php echo $this->url(array('controller'=>'Login','action'=>'index'));?
>">Login</a><span><? echo $this->escape($this->separator); ?></span>
                <a href="<?php echo $this->url(array
('controller'=>'Registration','action'=>'index'));?>">Registration</a>
        </p class = "text">

</body>
</html>
```

Algorithm 117 RegistrationController view -> index.phtml

```
<style>
        table.cover { border-style:solid; border-left-style:none; border-right-style:none;
                      border-color:#ff0000; width:100%; background-color:#EAF2D3;}

        table.cover, td { height:510px; text-align:left; vertical-align:top;}
        table.error{ text-align:left; vertical-align:top;}
</style>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <title><? echo $this->escape($this->pageTitle); ?></title>
</head>
<body>
        <h1><?php echo $this->pageTitle ;?></h1>
        <h3><?php echo $this->bodyCorp ;?></h3>

        <table class = "cover">
                <tr>
                        <td width = 43%>
                        <table class = "error">
                                <tr><h4><? echo $this->escape($this->note); ?></h4></tr>
                                <tr><h4><? echo $this->escape($this->errorMessage); ?></h4></tr>
                        </td>
                        </table class ="error">
                        <td>
                        <table>
                                <tr>
                                        <td><?php echo $this->form ;?></td>
                                </tr>
                        </table>
                        </td>
                </tr>
        </table class ="cover">

</body>
</html>
```

Algorithm 118 success.phtml

```
<style>
        p { color:darkgreen}
        p.success { color:black; text-align:justify; font-family:"Times New Roman";
                    font-size:20px; position:absolute; left:39%; top:40%; }
p.footer{ text-align:justify; font-family:"Times New Roman"; font-size:20px;
          font-weight:bold; position:absolute; left:68%; top:92%; }
</style>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <title><? echo $this->escape($this->pageTitle); ?></title>
</head>
<body>
        <h1><?php echo $this->pageTitle ;?></h1>
        <h3><p><?php echo $this->bodyCorp ;?></p></h3>

        <p class = "success">Now you can do: <a href="<?php echo $this->url(array
('controller'=>'Login','action'=>'index'));?>">Login</a> on our site</p class = "success">
        <p class = "footer"><span><?php echo $this->footer ;?></span></p class = "footer">
</body>
</html>
```

Algorithm 119 LoginController view -> index.phtml

```
<style>
        table.cover { border-style:solid; border-left-style:none; border-right-style:none;
                      border-color:#ff0000; width:100%; background-color:#EAF2D3; }
        table.cover, td { height:255px; text-align:left; vertical-align:top; }
        table.error { vertical-align:top; text-align:center; }
        p{ color:#660099; }
</style>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <title><? echo $this->escape($this->pageTitle); ?></title>
</head>
<body>
        <h1><?php echo $this->pageTitle ;?></h1>
        <h3><?php echo $this->bodyCorp ;?></h3>
        <table class = "cover">
                <caption><h4><? echo $this->escape($this->note); ?></h4></caption>
                <tr>
                        <td width = 43%>
                        <table class = "error">
                                <tr><h4><? echo $this->escape($this->errorUser); ?><p><? echo
$this->escape($this->errorMessage); ?><p></h4></tr><br>
                                <tr><h4><? echo $this->escape($this->errorPwd); ?></h4></tr>
                        </td>
                        </table class ="error">
                        </td>
                        <td>
                        <table>
                                <tr>
                                        <td><?php echo $this->form ;?></td>
                                </tr>
                        </table>
                        </td>
                </tr>
        </table class ="cover">
</body>
</html>
```

Algorithm 120 loggedin.phtml

```
<style>
        c { color:darkgreen}
        p.footer { text-align:justify; font-family:"Times New Roman"; font-size:20px;
                  font-weight:bold; position:absolute; left:68%; top:92%; }
        #user { position:absolute; top: 30%; left: 22%; z-index: 10; }
        #customers { font-family:"Trebuchet MS", Arial, Helvetica, sans-serif;
                     border-collapse:collapse; }
        #customers td, #customers th { font-size:1em; border:1px solid #99CCFF ;
                                       padding:3px 7px 2px 7px; }
        #customers th { font-size:1.1em; text-align:left; padding-top:5px; padding-bottom:4px;
                        background-color:#B0B0B0; color:#ffffff; }
        #customers tr.alt td { color:#000000; background-color:#99CCFF; }
        #submit{ background-color: #FF8800; border: 2px solid #FCA800; color: #fff;
                 font-weight: bold; padding: 0; height: 25px; width: 80px; }
</style>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <title><? echo $this->escape($this->pageTitle); ?></title>
</head>
<body>
        <h1><?php echo $this->pageTitle ;?></h1>
        <h3><?php echo $this->bodyCorp ;?></h3>
        <p class = "footer"><span><?php echo $this->footer ;?></span></p class = "footer">

<div id="user">
        <table id="customers">
                <tr>
                        <th colspan = 2>Sei connesso come : <?php echo $this->username ;?></th>
                </tr>
                <tr>
                        <td>Numero utente</td>
                        <td><?php echo $this->id ;?></td>
                </tr>
                <tr class="alt">
                        <td>Username</td>
                        <td><?php echo $this->username ;?></td>
                </tr>
                <tr>
                        <td>Password</td>
                        <td>Quella che hai inserito nel momento della registrazione</td>
                </tr>
                <tr class="alt">
                        <td>Nome completo</td>
                        <td><?php echo $this->fullname ;?></td>
                </tr>
                <tr>
                        <td>e-mail</td>
                        <td><?php echo $this->email ;?></td>
                </tr>
                <tr class="alt">
                        <td>Data di nascita</td>
                        <td><?php echo $this->birthday ;?></td>
                </tr>
                <tr>
                        <th>If you have finish you can do :</th>
                        <th><a href="<?php echo $this->url(array
('controller'=>'Login','action'=>'logout'));?>" style="text-decoration: none;"><input
value="Logout" class="Forminput" type="button" type="submit" id="submit" name="submit"></a></th>
                </tr>
        </table>
</div>

</body>
</html>
```