wheelsoft

Definizione di Prodotto

22 febbraio 2007

Sommario

Documento contenente la Definizione di Prodotto per il capitolato SIAGAS da parte del gruppo di Ingegneria del Software "Wheelsoft". Documento redatto secondo le Norme di Progetto v1.9.

Informazioni documento

Produzione	WheelSoft - wheelsoft@gmail.com
Redazione	Matteo Borgato - mborgato@studenti.math.unipd.it
	Alessio Rambaldi - arambald@studenti.math.unipd.it
Approvazione	Stefano Ceschi Berrini - sceschib@studenti.math.unipd.it
	Silvio Daminato - sdaminat@studenti.math.unipd.it
	Alberto De Bortoli - adeborto@studenti.math.unipd.it
	Giulio Favotto - gfavotto@studenti.math.unipd.it
	Michele Volpato - mvolpato@studenti.math.unipd.it
File	Definizione_di_Prodotto_v1.6.pdf
Versione	1.6
Stato	Formale
Uso	Esterno
Distribuzione	Wheelsoft
	prof. Alessandro Sperduti
	prof. Renato Conte
	prof. Tullio Vardanega

Diario delle modifiche

1.6	22/02/07 - Aggiunta sezione del raffinamento a livello di progettazione del *Logic Layer*(A.Rambaldi, M.Borgato)
1.5	21/02/07 - Aggiunta sezione tracciamento *Metodi/Requisiti*(M.Volpato), completata la sezione inerente al *Presentation Layer*,aggiunte immagini del Layout grafico, aggiunti gli alberi delle directories.(S.Ceschi Berrini)
1.4	20/02/07 - Aggiunta sezione *Raffinamento a livello di progettazione del Presentation Layer* e dettagliati i relativi componenti *admin*, *azienda*, *studente*, *super* (A. De Bortoli)
1.3	20/02/07 - Sviluppata la sezione inerente al Data Layer (S.Daminato)
1.2	18/02/07 - Ampliata la prima stesura del documento, aggiunte sezioni. (S.Daminato)
1.1	12/02/07 - Stesura del documento. (A.Rambaldi, M.Borgato)

Indice

1 Descrizione generale

1.1 Introduzione

1.1.1 Scopo del documento

Lo scopo del documento è definire nel dettaglio l'architettura del sistema già descritta nel documento Specifica Tecnica. Di ogni singolo componente saranno analizzati

- tipo, obiettivo e funzione;
- relazioni d'uso di altre componenti;
- interfacce e relazioni d'uso da altre componenti;
- attività svolte e dati trattati.

Il documento Specifica Tecnica è la base di partenza per la costituzione del presente documento. Perciò scopo di questo documento è anche guidare i programmatori durante la codifica, poiché non si debbano occupare di trovare soluzioni e implementazioni improvvisate. Infatti per ogni classe sono state definite le liste dei metodi (con relativi prototipi e comportamento) e dei campi dati, alle quali i programmatori si devono attenere.

Dato che è stato adottato un modello di ciclo di vita "incrementale", questa versione del documento è da considerarsi preliminare in quanto risultato di 3 iterazioni contro le 5 previste. Questo è dovuto alle modifiche apportate al piano di progetto in conseguenza del cambiamento delle date di consegna effettuato.

1.1.2 Scopo del prodotto

Il progetto da sviluppare, denominato *SIAGAS*, ha lo scopo di automatizzare il servizio di stage offerto dal corso di laurea in Informatica.

Attualmente questo servizio è gestito interamente "a mano" dal docente responsabile per le attività di stage (prof. A. Sperduti). Gli studenti non hanno un supporto automatizzato che li aiuti nella scelta dell'azienda dove svolgere lo stage.

Wheelsoft si propone di implementare un servizio web portabile, accessibile e manutenibile in modo che tutti gli studenti possano fruire di questo servizio nel migliore dei modi.

SIAGAS permetterà inoltre di facilitare la comunicazione tra aziende, studenti, responsabile e tutor; sono previste quindi funzionalità che permettano di gestirla appropriatamente.

Altro punto importante è l'organizzazione di eventi atti a favorire l'incontro tra azienda e studente. Il responsabile per le attività di stage dovrà gestire questi eventi in maniera semplice ed immediata. Disporrà quindi di un'interfaccia user-friendly, tramite la quale potrà governare quest'attività ed il resto del sistema che dovrà poter essere interrogato dal responsabile per ricerche di vario tipo.

Un ulteriore scopo del prodotto è seguire e guidare lo studente dal momento in cui lo stage viene approvato fino alla presentazione della tesi di laurea.

L'azienda che accoglie lo stagista deve poter inserire nel sistema il resoconto del lavoro svolto. Lo studente invece potrà inserire la tesi.

Per i dettagli si veda il documento "Analisi dei Requisiti".

1.1.3 Glossario

Per il glossario si faccia riferimento al documento "Glossario_v1.8.pdf".

1.1.4 Riferimenti

1. Normativi
   - Pagina web esistente della normativa sugli stage
     http://slash.math.unipd.it/laureainformatica/triennale/stage_laurea/stage.htm
   - Legge sulla privacy (Codice in materia di protezione dei dati personali)
   - Pagina web del capitolato d'appalto
     http://www.math.unipd.it/~tullio/IS-1/2006/Progetti/SIAGAS.html
   - Incontri documentati con il committente (prof. Sperduti)
2. Informativi
   - SWEBOK (www.swebok.org)
   - http://www.html.it/ sito generico di guide e tutorial per linguaggi HTML, PHP e CSS.
   - http://www.php.net/ sito dedicato a PHP5.
   - http://pear.php.net/ sito dedicato alla libreria PEAR per PHP5.

1.2 Standard di progetto

1.2.1 Standard di progettazione architetturale

L'architettura del sistema SIAGAS si basa sul pattern architetturale denominato 3-layer. Le sue tre componenti principali sono logicamente raggruppate in package nel seguente modo:

- Presentation Layer: rappresenta l'interfaccia utente e contiene il codice XHTML per la rappresentazione grafica e l'interazione con l'utente finale. Può contenere degli aspetti logici, ma solo al fine di visualizzare le informazioni in modalità differenti e più adeguate.
- Logic Layer: contiene la logica di controllo dell'applicativo gestionale. In particolare ne implementa le funzionalità vere e proprie. Consiste di codice PHP contente controlli di logica, gestisce le richieste del presentation layer e interroga il database al layer sottostante.
- Data Layer: Questo strato comprende il database e il software che ne gestisce l'accesso e memorizza i dati più a basso livello. Si appoggia sul database *SIAGASDB*.

In questo modo è possibile disaccoppiare la logica dai dati, replicando la prima, che è molto meno soggetta a cambiamenti ed evoluzione e non soffre di problemi di sincronizzazione, centralizzando opportunamente i secondi. Quindi si può distribuire il carico ed accedere in maniera efficiente ai dati.

Nella progettazione sono poi stati utilizzati i seguenti **design pattern**

- **Singleton**
  Il Singleton è un pattern creazionale e si può applicare ogni qualvolta nell'applicazione che si sta sviluppando ci si trovi di fronte ad una classe che modella un oggetto, il cui ruolo è unico ed ha delle responsabilità speciali. In questo caso si fa in modo che la classe abbia una sola istanza e l'applicazione fornisca un punto di accesso globale ad essa.

- **Factory**
  Il design pattern creazionale Factory Method definisce un'interfaccia (*Creator*) per ottenere una nuova istanza di un oggetto (*Product*) delegando ad una classe derivata (*ConcreteCreator*) la scelta di quale classe istanziare (*ConcreteProduct*).

- **Façade**
  È di tipo strutturale e tratta l'argomento del disaccoppiamento tra interfacce ed implementazione delle classi, degli oggetti e della loro composizione.

Per una visione più completa dei pattern utilizzati si faccia riferimento al documento "Specifica Tecnica v1.7".

1.2.2 Standard di documentazione del codice

Sono state rispettate le "Norme di codifica" riportate nel documento "Norme di progetto".

1.2.3 Standard di denominazione di entità e relazioni

Sono state rispettate le "Norme di codifica" riportate nel documento "Norme di progetto".

1.2.4 Standard di programmazione

Sono state rispettate le "Norme di codifica" riportate nel documento "Norme di progetto".

1.2.5 Strumenti di lavoro

Gli strumenti utilizzati sono riportati nel capitolo "Tecniche e strumenti" del documento "Norme di progetto".

2 Specifica delle componenti

2.1 Struttura dell'applicazione

L'applicazione si presenta all'utente finale come un applicativo web che sfrutta moduli scritti nel linguaggio PHP 5.2.0 i quali generano pagine dinamiche usando il linguaggio XHTML 1.1.

La struttura del sistema *SIAGAS* è rappresentata nei diagrammi presentati in seguito, divisi per livello.

2.1.1 Implementazione del Presentation Layer

Il Presentation Layer è sviluppato con pagine web create utilizzando il linguaggio XHTML 1.1, rese dinamiche da script PHP al loro interno. La grafica viene totalmente gestita da fogli di stile CSS. Questa parte è stata progettata e realizzata seguendo gli standard definiti dal W3C.

La grafica inoltre è stata studiata per integrarsi nell'attuale sito del corso di laurea in Informatica, nel quale *SIAGAS* presumibilmente sarà inserito una volta collaudato ed accettato.

La seguente immagine raffigura il layout della pagina che si presenta all'utente "Utente non autenticato" quando si collega al sistema. Il layout sarà pressoché identico per ogni utente, cambieranno solamente i contenuti e le azioni disponibili.

Nella figura si vede come sia stata sfruttata l'intera pagina del browser. Questa è stata divisa in quattro sezioni principali:

- **Header** la quale contiene il logo (FAC-SIMILE) dell'università di Padova, il nome del progetto ed il menù diverso per ogni utente;
- **Navigation path** nella quale è possibile visionare la pagina che si sta visitando;
- **Content** la quale contiene le informazioni riferite alla pagina;
- **Right Column** la quale contiene le varie sezioni per il login, per la guida contestuale e per le informazioni riguardanti l'accessibilità.

Questa suddivisione è stata gestita totalmente nel foglio di stile *CSS*, mantenendo la denominazione qui specificata.

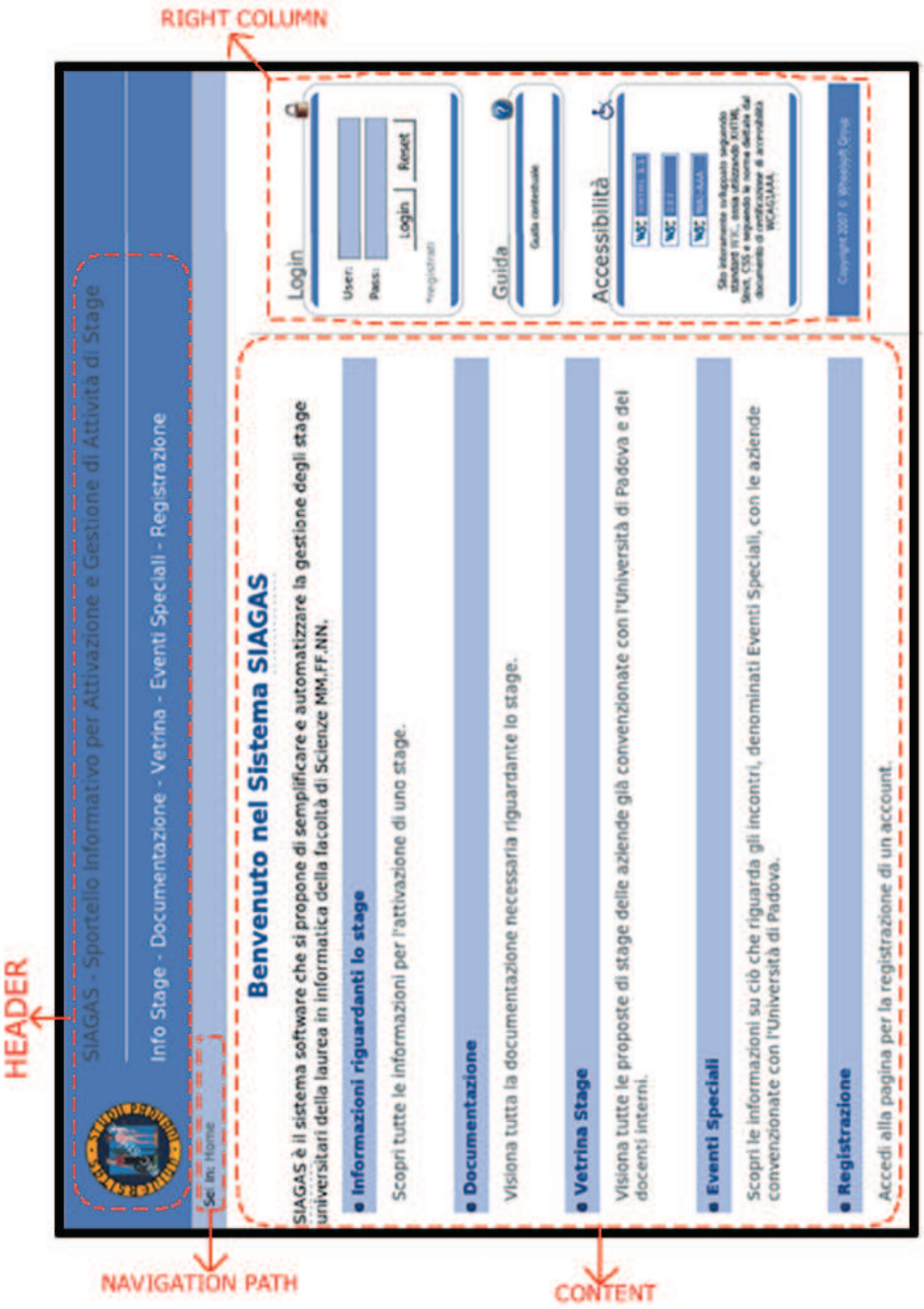

Figura 1: *Layout grafico*

L'immagine che segue rappresenta la mappa dei files per il Presentation Layer.

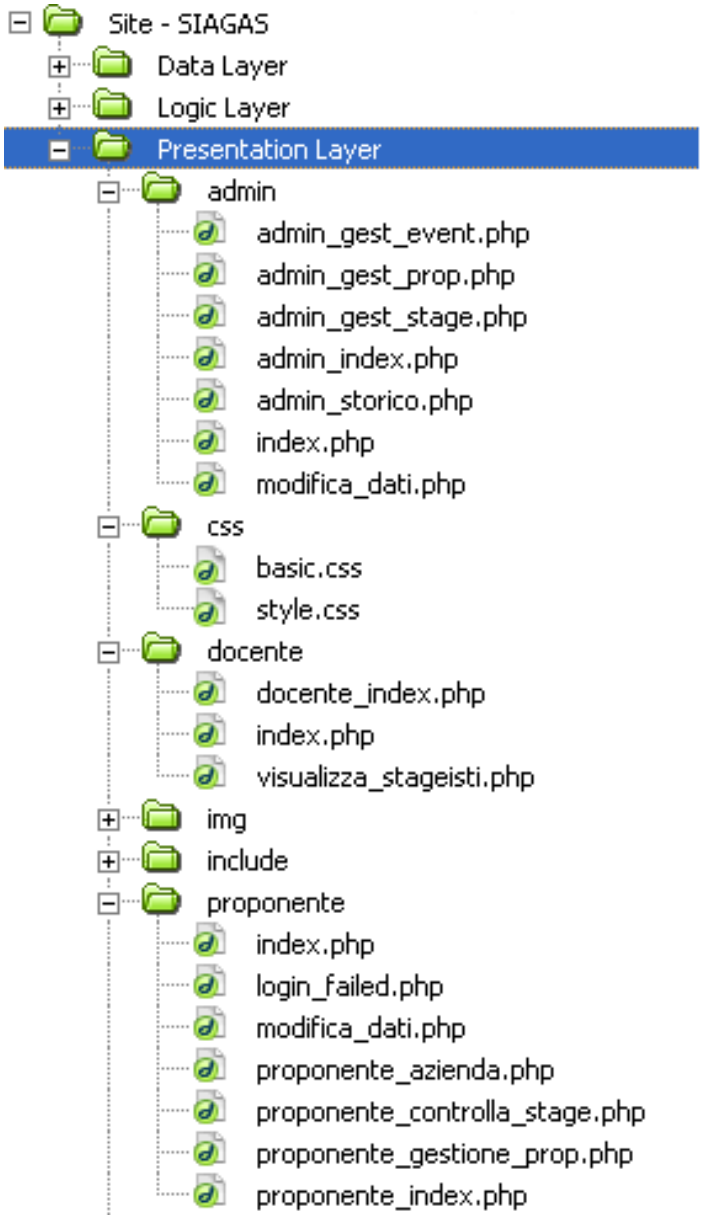

Figura 2: *Struttura del Presentation Layer(Parte 1)*

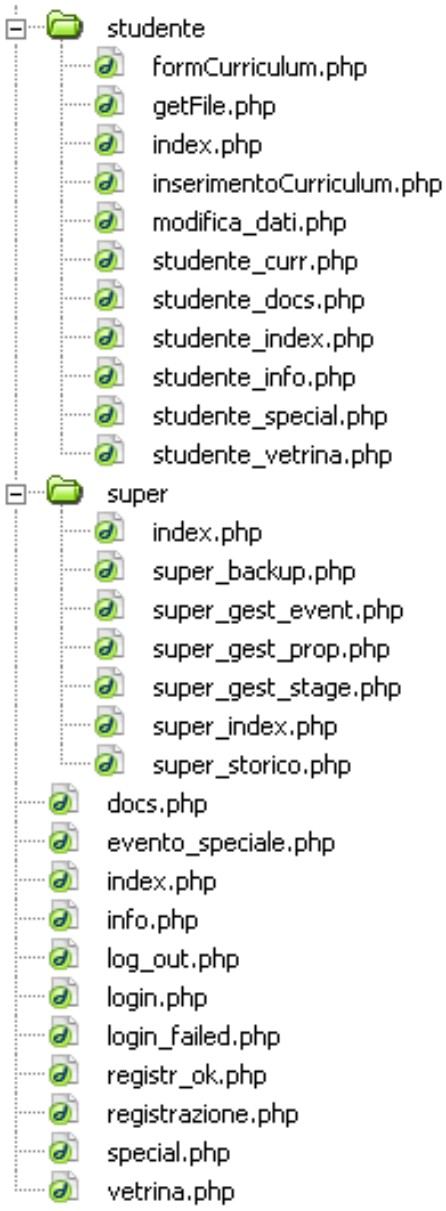

Figura 3: *Struttura del Presentation Layer(Parte 2)*

2.1.2 Implementazione del Logic Layer

Il Logic Layer è sviluppato con pagine scritte nel linguaggio PHP 5.0, e interfaccia l'utente con il *SIAGASDB*.

L'immagine seguente rappresenta l'organizzazione strutturale del Logic Layer.

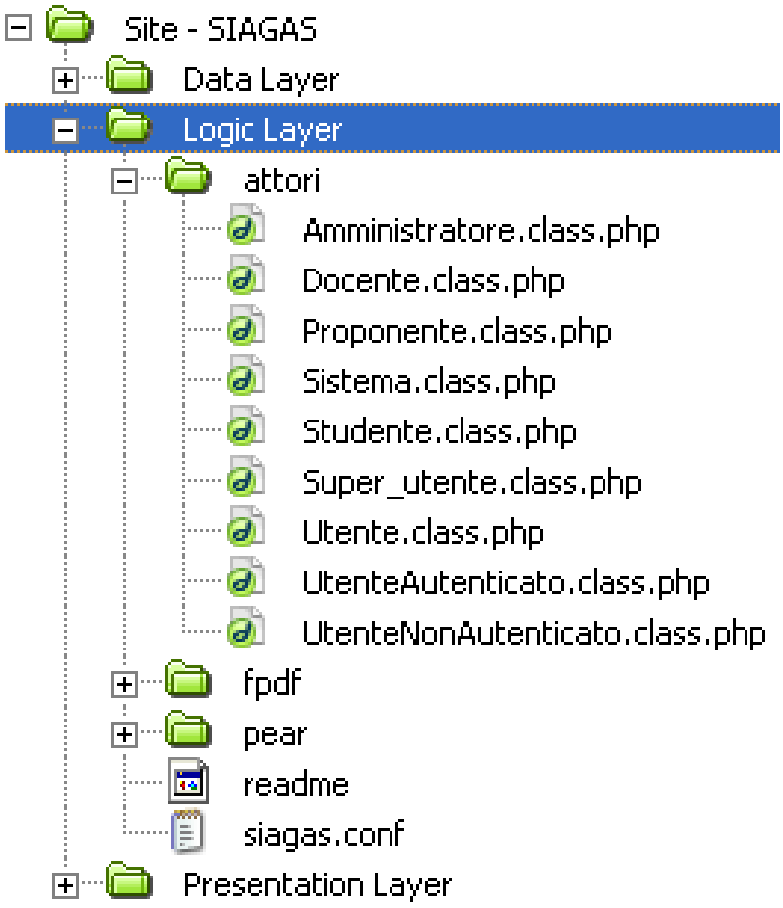

Figura 4: *Struttura del Logic Layer*

2.1.3 Implementazione del Data Layer

Il Data Layer è sviluppato in MySql ed è composto unicamente dal *SIAGASDB*. L'immagine seguente rappresenta l'organizzazione strutturale del Data Layer (contiene solo il file SQL con la creazione ed il popolamento delle tabelle).

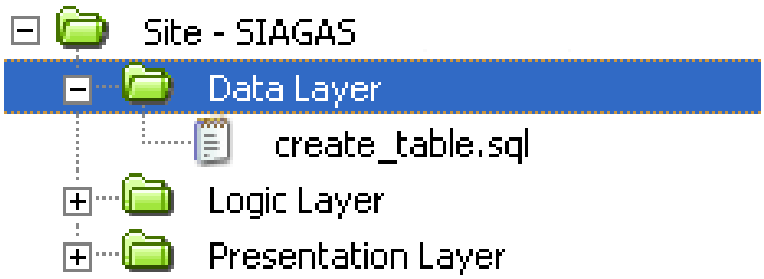

Figura 5: *Struttura del Data Layer*

2.2 Raffinamento a livello di progettazione del Presentation Layer

Qui di seguito verranno spiegate in dettaglio le componenti di questo Layer:

- le pagine con estensione *php*;
- i fogli di stile *CSS*;
- i files contenenti codice in comune a più pagine;

Verranno quindi descritte le pagine e le cartelle contenute nella ${WEB-ROOT} e la gestione dei permessi (sessioni).

Si descriverà inoltre come è stata gestita la sicurezza.

2.2.1 Sessioni

Per quanto riguarda i permessi di accesso alle pagine, questi sono stati gestiti utilizzando le sessioni messe a disposizione dal linguaggio PHP. Ogni tipologia di "Utente autenticato" avrà una propria sessione attiva fino ad un "log-out" oppure fino alla chiusura del browser. Un "Utente non autenticato" non avrà una sessione attiva finchè non si sarà autenticato nel sistema. All'inizio di ogni pagina PHP è stato gestito il controllo delle sessioni, questo per far in modo che ogni tipo di utente, nel caso in cui inserisca un URL errato oppure tenti di accedere ad una pagina protetta, venga rediretto nella propria homepage.

- L'"Utente non autenticato" potrà accedere solo ad alcune pagine nella ${WEB-ROOT}, quindi se tenta di accedere a pagine riservate, sarà rediretto in **${WEB-ROOT}/index.php**.

- Gli "Utenti autenticati" potranno accedere alle pagine delle proprie cartelle, quindi se tenteranno di accedere a pagine non proprie, saranno rediretti in **${WEB-ROOT}/cartella_utente/utente_index.php**. Con "cartella_utente" si intende il nome della cartella di un determinato utente e con "utente_index.php" si intende la home page di un "Utente autenticato".

Qui di seguito verrà riportato lo script PHP, comune ad ogni pagina, per il controllo delle sessioni:

```
<?php
// controllo accesso
session_start();
if (isset($_SESSION['Id_utente'])) {
    switch ($_SESSION['Usergroup']) {
        case "studente":
            header('location:studente/studente_index.php');
            break;
        case "amministratore":
            header('location:admin/admin_index.php');
            break;
        case "superutente":
            header('location:super/super_index.php');
            break;
        case "proponente":
            header('location:proponente/
            proponente_index.php');
            break;
        case "docente":
            header('location:docente/docente_index.php');
            break;
    }
}else {
    header('location:./index.php');
}
?>
```

2.2.2 Sicurezza

Per quanto riguarda la sicurezza, sono stati seguiti i seguenti punti:

- Protezione delle sessioni;
- Per evitare il download diretto di files personali, questi verranno salvati in una cartella superiore alla root del webserver, in quanto al browser è impedito l'accesso; per permettere all'utente di scaricare un file, verrà richiamato uno script PHP il quale provvede ad inviare al client il file in HTTP;
- Controllo delle parentesi angolari nei form, per evitare possibili inserimenti di codice nel database;

- Per evitare attacchi di tipo SQL injection sono state utilizzate direttive del PHP chiamate *magic_quote_gpc*;
- il passaggio delle variabili tra le pagine tramite metodo HTML POST;
- l'uso di algoritmi di hashing (come MD5) per il salvataggio delle password su database.

inoltre è in fase di studio l'utilizzo della tecnologia SSL, grazie al programma freeware OpenSSL. È possibile, per un utente non autenticato, decidere se utilizzare o meno questo tipo di sicurezza, al momento dell'entrata nel sistema. Se SSL è attivo, lo rimarrà per tutta la durata della navigazione nel sistema.

2.2.3 index.php

- **Path**
  ${WEB-ROOT}/index.php
- **Tipo, obiettivo e funzione**
  È la pagina principale del sistema, scritta in XHTML e codice PHP. Ha l'obiettivo di permettere ad un utente non autenticato di capire il funzionamento del sistema e di accedere alle varie pagine.
- **Relazione d'uso di altre componenti**
  index.php utilizza particolari metodi del linguaggio PHP per il controllo della sessione (spiegata in precedenza). Contiene inoltre i vari links alle pagine accessibili dagli utenti non autenticati.
- **Interfaccia e relazioni da altre componenti**
  Il link a questa pagina è contenuto nelle pagine raggiungibili dagli utenti non autenticati.
- **Attività svolte e dati trattati**
  Questo indice fa da interfaccia per tutte le funzioni principali, quali la visualizzazione della vetrina di stage, della documentazione, degli eventi speciali. Inoltre dà la possibilità di accedere ad una pagina di informazioni e di registrazione nel sistema. La pagina "index.php" svolge l'attività di reindirizzamento nel caso in cui un utente sia autenticato nel sistema e voglia accedere a questa pagina. Inoltre ha lo scopo di far visualizzare le funzionalità del sistema.

2.2.4 docs.php

- **Path**
  ${WEB-ROOT}/docs.php
- **Tipo, obiettivo e funzione** Questa pagina contiene la documentazione sugli stage. È scritta in XHTML e codice PHP.
- **Relazione d'uso di altre componenti**
  Vengono utilizzati metodi del linguaggio PHP per il controllo della sessione.

- **Interfaccia e relazioni da altre componenti**
  Pagina accedibile da tutte le altre visitate da un utente non autenticato.

- **Attività svolte e dati trattati**
  Questa pagina dà la possibilità all'utente non autenticato di visionare la documentazione riguardante gli stage. Se qualche documento è disponibile in formato PDF, in *docs.php* verranno resi noti i relativi links.

2.2.5 info.php

- **Path**
  ${WEB-ROOT}/info.php

- **Tipo, obiettivo e funzione**
  Questa pagina è scritta in XHTML e codice PHP e contiene le informazioni riguardanti gli stage.

- **Relazione d'uso di altre componenti**
  Vengono utilizzati metodi del linguaggio PHP per il controllo della sessione.

- **Interfaccia e relazioni da altre componenti**
  Pagina accedibile da tutte le altre visitate da un Utente non autenticato.

- **Attività svolte e dati trattati**
  Questa pagina visualizza le informazioni riguardanti l'inizio e la prosecuzione di un'attività di stage.

2.2.6 login.php

- **Path**
  ${WEB-ROOT}/login.php

- **Tipo, obiettivo e funzione**
  La pagina "login.php" fa in modo che, una volta richiamata, venga creata una sessione. E' scritta solamente in codice php perchè non deve visualizzare nessuna informazione.

- **Relazione d'uso di altre componenti**
  Questa pagina utilizza la classe "UtenteNonAutenticato.class" ed alcuni metodi di PHP.

- **Interfaccia e relazioni da altre componenti**
  La pagina "login.php" viene chiamata ogniqualvolta un utente non autenticato, che possiede un account nel sitema, voglia autenticarsi e poter quindi accedere alla propria homepage.

- **Attività svolte e dati trattati**
  La pagina riceve *username* e *password* dal form per il login tramite il metodo POST fornito dal linguaggio XHTML. Successivamente crea un nuovo "UtenteNonAutenticato" e, tramite un suo metodo chiamato *signIn(username,password)*, permette di autenticarsi o meno e quindi iniziare una nuova sessione. Qui di seguito viene riportata la parte di codice relativa al login.

```
<?php
    ob_start();
    set_include_path('../includes/' . PATH_SEPARATOR .
    get_include_path());
    require_once 'attori/UtenteNonAutenticato.class.php';

    $username = $_POST[user];
    $password = md5($username . $_POST[pass]);
    $utente = new UtenteNonAutenticato();
    $auth   = $utente->signIn($username, $password);
    if ($auth) {
        session_start();
        $_SESSION['Id_utente'] = $auth[0]->getId();
        $_SESSION['Usergroup'] = $auth[1];
        header('location:index.php');
    }
    else {
        header('location:login_failed.php');
    }

    ob_end_flush();
?>
```

2.2.7 log_out.php

- **Path**
  ${WEB-ROOT}/log_out.php
- **Tipo, obiettivo e funzione** La pagina "log_out.php" fa in modo che, una volta richiamata, venga cancellata una sessione. È scritta solamente in codice PHP perché non deve visualizzare nessuna informazione.
- **Relazione d'uso di altre componenti**
  Questa pagina utilizza dei metodi PHP riguardanti le sessioni.
- **Interfaccia e relazioni da altre componenti**
  La pagina di log_out viene richiamata ogniqualvolta si voglia terminare una sessione. Quindi il link a questa pagina sarà presente nelle pagine relative agli utenti autenticati.
- **Attività svolte e dati trattati**
  Non fa nessun controllo di autenticazione, in quanto deve solo terminare una sessione. Una volta terminata, reindirizza alla pagina "index.php". Qui di seguito verrà fornito un estratto di codice per il log_out:

```
<?php
// Inizializza la sessione.
session_start();
// Resetta tutte le variabili di sessione.
$_SESSION = array();
```

```
// distrugge la sessione.
session_destroy();
// Infine Reindirizza alla homepage
header('location:./');
?>
```

2.2.8 login_failed.php

- **Path**
  ${WEB-ROOT}/login_failed.php
- **Tipo, obiettivo e funzione**
  Questa pagina è scritta in XHTML e PHP e viene richiamata nel caso in cui un utente sbagli l'autenticazione.
- **Relazione d'uso di altre componenti**
  La pagina utilizza codice PHP solamente per includere header e footer.
- **Interfaccia e relazioni da altre componenti**
  È possibile accedervi solo quando si sbaglia il login, quindi non è contenuta in altre pagine.
- **Attività svolte e dati trattati**
  Visualizza un messaggio di login errato e mette a disposizione un link che redirige alla homepage del sistema.

2.2.9 registrazione.php

- **Path**
  ${WEB-ROOT}/registrazione.php
- **Tipo, obiettivo e funzione**
  Questa pagina è scritta in XHTML e PHP e serve per registrare un nuovo account nel sistema.
- **Relazione d'uso di altre componenti**
  Vengono utilizzati vari metodi del linguaggio PHP ed un form XHTML per l'invio dei dati.
- **Interfaccia e relazioni da altre componenti**
  Il link alla pagina "registrazione.php" è disponibile in tutte le pagine accessibili da un utente non autenticato.
- **Attività svolte e dati trattati**
  Compilando il form contenuto in questa pagina, l'utente non autenticato potrà creare un nuovo account nel sistema ed in seguito potrà autenticarsi come *Studente*. Inoltre nella pagina è presente una parte di codice PHP che permette di controllare i valori inseriti nel form. Se i valuri inseriti sono consistenti, viene richiamata un'altra pagina chiamata *registr_ok*. Se i valori non sono validi viene visualizzato un messaggio d'errore. Di seguito viene riportata la schermata di registrazione:

Figura 6: *Layout grafico della pagina per la registrazione*

2.2.10 registr_ok.php

- **Path**
  ${WEB-ROOT}/registr_ok.php
- **Tipo, obiettivo e funzione**
  Questa pagina è scritta in XHTML e PHP e serve a visualizzare un messaggio di avvenuta registrazione, nonché di avviamento di una nuova sessione per un utente autenticato di tipo*Studente*.
- **Relazione d'uso di altre componenti**
  Vengono utilizzati dei metodi del linguaggio PHP per controllare le sessioni.
- **Interfaccia e relazioni da altre componenti**
  La pagina viene visualizzata solo quando avviene una corretta registrazione, quindi non c'è nessun link che permetta di aprirla.
- **Attività svolte e dati trattati**
  La pagina oltre a confermare l'avvenuta registrazione inizia una sessione per lo *Studente* e contiene inoltre un link alla sua homepage.

2.2.11 special.php

- **Path**
  ${WEB-ROOT}/special.php
- **Tipo, obiettivo e funzione**
  Questa pagina è scritta in XHTML e PHP ed ha l'obiettivo di far visualizzare tutti gli eventi speciali(ad esempio Stage-IT).
- **Relazione d'uso di altre componenti**
  In questa pagina sono utilizzati vari metodi scritti in PHP, inoltre viene

creato un oggetto di tipo *UtenteNonAutenticato* per chiamare un proprio metodo *getEvento()*.

- **Interfaccia e relazioni da altre componenti**
  Il link alla pagina "special.php" è disponibile in tutte le pagine accessibili da un utente non autenticato.

- **Attività svolte e dati trattati**
  L'attività concerne il mostrare i link agli eventi speciali disponibili e dare la possibilità agli utenti non autenticati di accedervi per visionarli.

2.2.12 evento_speciale.php

- **Path**
  ${WEB-ROOT}/evento_speciale.php

- **Tipo, obiettivo e funzione**
  Questa pagina è scritta in XHTML e PHP ed ha l'obiettivo di far visualizzare tutte le proposte e le informazioni riguardanti un particolare evento speciale(ad esempio Stage-IT).

- **Relazione d'uso di altre componenti**
  In questa pagina sono utilizzati vari metodi del linguaggio PHP ed inoltre qui viene creato un oggetto di tipo utente non autenticato per chiamare un proprio metodo *getProposteEvento()*, il quale ritorna tutte le proposte relative ad un evento.

- **Interfaccia e relazioni da altre componenti**
  Il link a "evento_speciale.php" è disponibile solo nella pagina "special.php".

- **Attività svolte e dati trattati**
  L'attività concerne il mostrare le proposte inerenti ad un determinato evento speciale.

2.2.13 vetrina.php

- **Path**
  ${WEB-ROOT}/vetrina.php

- **Tipo, obiettivo e funzione**
  Pagina scritta in XHTML e PHP che ha l'obiettivo di mostrare agli utenti non autenticati tutti gli stage disponibili.

- **Relazione d'uso di altre componenti**
  In questa pagina sono utilizzati vari metodi del PHP ed inoltre viene creato un oggetto di tipo utente non autenticato per chiamare un proprio metodo *getProposteEvento()*, il quale ritorna tutti gli stage.

- **Interfaccia e relazioni da altre componenti**
  Il link a *vetrina.php* è raggiungibile da tutte le pagine accessibili da un utente non autenticato.

- **Attività svolte e dati trattati**
  La pagina visualizza tutti gli stage disponibili in un determinato momento.

2.2.14 css/

- **Path**
  ${WEB-ROOT}/css/
- **Tipo, obiettivo e funzione**
  Cartella contenente i fogli di stile CSS. Qui viene gestita totalmente la grafica della parte presentazione.
- **Relazione d'uso di altre componenti**
  Non ha relazioni con altre componenti
- **Interfaccia e relazioni da altre componenti**
  I CSS sono importati da tutti i file del Presentation Layer.
- **Attività svolte e dati trattati**
  Svolge l'attività di gestire il layout grafico delle pagine XHTML. Si può vedere in 1 come si presenta il layout utilizzando i fogli di stile.

2.2.15 img/

- **Path**
  ${WEB-ROOT}/img/
- **Tipo, obiettivo e funzione**
  Cartella contenente le immagini.

2.2.16 include/

- **Path**
  ${WEB-ROOT}/include/
- **Tipo, obiettivo e funzione**
  Cartella contenente le parti in comune a più files del Presentation Layer.
- **Relazione d'uso di altre componenti**
  Vengono utilizzati vari metodi del PHP.
- **Interfaccia e relazioni da altre componenti**
  I files di questa cartella vengono inclusi dai files del Presentation Layer tramite il metodo *require_once()* messo a disposizione dal PHP.

2.2.17 admin/

- **Path**
  ${WEB-ROOT}/admin/
- **Tipo, obiettivo e funzione**
  Cartella contenente le pagine in formato php relative alle sessioni di tipo "amministratore". Le pagine contenute in questa cartella sono visitabili soltanto da un utente autenticato come amministratore e ne permettono le azioni lecite per il suddetto tipo d'utente. La pagina *azienda_index.php* si presenta simile alla pagina mostrata in figura 7.

- **Relazione d'uso di altre componenti**
  Le pagine contenute utilizzano i file php del logic layer quali: *Amministratore.class.php*, *Sistema.class.php*.

- **Interfaccia e relazioni da altre componenti**
  La pagina *admin_index.php* è raggiungibile dalla pagina *index.php* situata gerarchicamente al livello superiore ${WEB-ROOT}/ soltanto tramite la procedura standard di login.

- **Attività svolte e dati trattati**
  Dalla pagina *admin_index.php* è possibile accedere dunque alle restanti pagine relative all'amministratore che consentono di effettuare le azioni lecite per il suddetto tipo d'utente, le quali sono specificate nel documento "Analisi dei Requisiti". In caso l'utente autenticato come amministratore cerchi di accedere ad altre zone private del sistema verrà rediretto alla pagina *admin_index.php* presente in ${WEB-ROOT}/admin/. L'amministratore può gestire i dati memorizzati nel SIAGASDB relativi a se stesso, alle aziende, agli studenti e agli eventi speciali.

2.2.18 proponente/

- **Path**
  ${WEB-ROOT}/proponente/

- **Tipo, obiettivo e funzione**
  Cartella contenente le pagine in formato php relative all'utente proponente di una proposta di stage. Le pagine contenute in questa cartella sono visitabili soltanto tramite un collegamento dedicato per accedere alla pagina *proponente_index.php* e ne permettono le azioni lecite per il suddetto tipo d'utente. La pagina *proponente_index.php* si presenta simile alla pagina mostrata in figura 7.

- **Relazione d'uso di altre componenti**
  Le pagine contenute utilizzano i file php del logic layer quali: *Proponente.class.php*, *Sistema.class.php*.

- **Interfaccia e relazioni da altre componenti**
  La pagina *proponente_index.php* è raggiungibile dalla pagina *index.php* situata gerarchicamente al livello superiore ${WEB-ROOT}/ soltanto tramite la procedura standard di login.

- **Attività svolte e dati trattati**
  Dalla pagina *proponente_index.php* è possibile accedere dunque alle restanti pagine relative al proponente che consentono di effettuare le azioni lecite per il suddetto tipo d'utente, le quali sono specificate nel documento "Analisi dei Requisiti". In caso l'utente cerchi di accedere ad altre zone private del sistema verrà rediretto alla pagina *proponente_index.php* presente in ${WEB-ROOT}/proponente/. L'utente proponente può gestire i dati memorizzati nel SIAGASDB relativi a se stesso (o alla sua azienda) e agli studenti stagisti presso di lui (o presso la sua azienda).

2.2.19 docente/

- **Path**
  ${WEB-ROOT}/docente/
- **Tipo, obiettivo e funzione**
  Cartella contenente le pagine in formato php relative all'utente docente che funge da tutor. Le pagine contenute in questa cartella sono visitabili soltanto tramite un collegamento dedicato per accedere alla pagina *docente_index.php* e ne permettono le azioni lecite per il suddetto tipo d'utente. La pagina *docente_index.php* si presenta simile alla pagina mostrata in figura 7.
- **Relazione d'uso di altre componenti**
  Le pagine contenute utilizzano i file php del logic layer quali: *Docente.class.php*, *Sistema.class.php*.
- **Interfaccia e relazioni da altre componenti**
  La pagina *docente_index.php* è raggiungibile dalla pagina *index.php* situata gerarchicamente al livello superiore ${WEB-ROOT}/ soltanto tramite la procedura standard di login.
- **Attività svolte e dati trattati**
  Dalla pagina *docente_index.php* è possibile accedere dunque alle restanti pagine relative al docente che consentono di effettuare le azioni lecite per il suddetto tipo d'utente, le quali sono specificate nel documento "Analisi dei Requisiti". In caso l'utente cerchi di accedere ad altre zone private del sistema verrà rediretto alla pagina *docente_index.php* presente in ${WEB-ROOT}/docente/. L'utente docente si limita a visualizzare i dati memorizzati nel SIAGASDB relativi a se stesso e agli studenti stagisti presso di lui.

2.2.20 studente/

- **Path**
  ${WEB-ROOT}/studente/
- **Tipo, obiettivo e funzione**
  Cartella contenente le pagine in formato php relative alle sessioni di tipo "studente". Le pagine contenute in questa cartella sono visitabili soltanto da un utente autenticato come studente e ne permettono le azioni lecite per il suddetto tipo d'utente. Le seguenti 2 stampe a schermo mostrano le pagine *studente_index.php* e *studente_dati.php* come esempio.
- **Relazione d'uso di altre componenti**
  Le pagine contenute utilizzano i file php del logic layer quali: *Sistema.class.php*, *Studente.class.php*.
- **Interfaccia e relazioni da altre componenti**
  La pagina *studente_index.php* è raggiungibile dalla pagina *index.php* situata gerarchicamente al livello superiore ${WEB-ROOT}/ soltanto tramite la procedura standard di login.

Figura 7: *Layout grafico della home page dello studente*

Figura 8: *Layout grafico della pagina dedicata ai dati dello studente*

- **Attività svolte e dati trattati**
  Dalla pagina *studente_index.php* è possibile accedere dunque alle restanti pagine relative all'amministratore che consentono di effettuare le azioni lecite per il suddetto tipo d'utente, le quali sono specificate nel documento "Analisi dei Requisiti". In caso l'utente autenticato come studente cerchi di accedere ad altre zone private del sistema verrà rediretto alla pagina *studente_index.php* presente in ${WEB-ROOT}/admin/. Lo studente può gestire i dati memorizzati nel SIAGASDB relativi a se stesso.

2.2.21 super/

- **Path**
  ${WEB-ROOT}/super/

- **Tipo, obiettivo e funzione**
  Cartella contenente le pagine in formato php relative alle sessioni di tipo "superutente". Le pagine contenute in questa cartella sono visitabili soltanto da un utente autenticato come superutente e ne permettono le azioni lecite per il suddetto tipo d'utente. La pagina *azienda_index.php* si presenta simile alla pagina mostrata in figura 7.

- **Relazione d'uso di altre componenti**
  Le pagine contenute utilizzano i file php del logic layer quali: *Super_utente.class.php*, *Sistema.class.php*.

- **Interfaccia e relazioni da altre componenti**
  La pagina *super_index.php* è raggiungibile dalla pagina *index.php* situata gerarchicamente al livello superiore ${WEB-ROOT}/ soltanto tramite la procedura standard di login.

- **Attività svolte e dati trattati**
  Dalla pagina *admin_index.php* è possibile accedere dunque alle restanti pagine relative al superutente che consentono di effettuare le azioni lecite per il suddetto tipo d'utente, le quali sono specificate nel documento "Analisi dei Requisiti". In caso l'utente autenticato come superutente cerchi di accedere ad altre zone private del sistema verrà rediretto alla pagina *super_index.php* presente in ${WEB-ROOT}/super/. Il super-utente ha il controllo totale del sistema: può gestire i dati memorizzati nel SIAGASDB relativi a se stesso, alle aziende, agli studenti e agli eventi speciali. Può inoltre effettuare il backup del sistema.

2.3 Raffinamento a livello di progettazione del Logic Layer

Lo strato Logic si divide in attori, entità, la libreria Pear e la libreria FPDF. In seguito vengono descritte ed analizzate in dettaglio, descrivendo gli obiettivi, le varie funzionalità e le eventuali relazioni con le altre classi.

2.3.1 attori.Amministratore

- **Path dei file**
  ${WEB-ROOT}/../logic/attori/Amministratore.class

- **Tipo, obiettivo e funzione del componente**
  Classe Php rappresentante l'amministratore di SIAGAS. È una specializzazione di *UtenteAutenticato* quindi solamente l'utente autenticato, in questo caso come amministratore, ha il permesso e i diritti di operare.

- **Relazioni d'uso di altre componenti**
  Per la gestione dei vari metodi viene richiesto l'utilizzo della classe *Sistema* che viene implementata attraverso il pattern creazionale Singleton. Un particolare metodo denominato getInfo() di questa componente utilizza il seguente comando SQL per recuperare le informazioni relative all'amministratore dall'omonima tabella.

```
public function getInfo() {
    $sys = Sistema::getSistema();
```

```
        $res = $sys->query("SELECT nome, mail,
         telefono, username, cognome,
         data_creazione FROM amministratore
         WHERE id_utente = '$this->_id_utente'");
        return $res->fetchRow();
}
```

Come si vede viene anche chiamato il metodo getSistema(), appartenente alla classe Sistema.

- **Interfacce e relazioni di uso da altre componenti**
  Questa classe viene istanziata dalla classe *UtenteAutenticato*. La classe *Super_utente* eredita tutti i metodi di Amministratore.

- **Attività svolte e dati trattati**
  Le principali attività che può svolgere l'amministratore sono le seguenti:

  1. Gestione stage scelto da uno studente;
  2. Gestione proponente e relative proposte;
  3. Gestione eventi speciali e storico aziende;

  Può quindi gestire i dati relativi alle aziende e agli studenti memorizzati nel SIAGASDB.

- **Elenco dei metodi**
  Costruttore pubblico, il parametro passato *arg0* è l'id dell'amministratore:

```
public function __construct($arg0)
```

Metodo pubblico che ritorna le informazioni relative all'amministratore:

```
public function getInfo()
```

Metodo pubblico che conferma lo stage scelto da uno studente, ritorna un booleano che indica se lo stage è confermato e attivo:

```
public function confermaStage($proposta)
```

Metodo pubblico che conferma una proposta di un proponente:

```
public function confermaProposta($proposta)
```

Metodo pubblico che crea un nuovo proponente per gli stage:

```
public function registraProponente()
```

Gestione generale di Eventi speciali. Ritorna un booleano per indicare che la gestione è stata fatta senza errori:

```
public function gestisciEventiSpeciali()
```

Visualizza lo storico delle aziende che hanno già proposto stage:

```
public function visualizzaStorico()
```

Metodo pubblico per il quale lo studente si iscrive a un evento speciale. Il parametro passato *docente* è l'id del docente da associare come tutor interno all'id dello *studente* a cui associare il professore:

```
public function associaTutor($docente, $studente)
```

Metodo pubblico che permette di creare una mailing list:

```
public function creaMailingList()
```

Metodo pubblico che crea un'allerta per amministrare un evento particolare:

```
public function creaAllerta()
```

Metodo pubblico che modifica i dati personali dello studente:

```
public function modificaDati($dati)
```

2.3.2 attori.Docente

- **Path dei file**
  ${WEB-ROOT}/../logic/attori/Docente.class
- **Tipo, obiettivo e funzione del componente**
  Classe Php che rappresentante un docente. È una specializzazione di *UtenteAutenticato* e quindi solamente l'utente autenticato, in questo caso come docente, ha il permesso e i diritti di operare per visualizzare e gestire gli stagisti che sta seguendo.
- **Relazioni d'uso di altre componenti**
  Questa componente utilizza la classe *Sistema* per la visualizzazione e la valutazione degli stage e degli stagisti seguiti.
- **Interfacce e relazioni di uso da altre componenti**
  Questa classe viene istanziata dalla classe *UtenteAutenticato*.
- **Attività svolte e dati trattati**
  L'utente autenticato specializzato come *docente* può interrogare il *Sistema* per richiedere la lista degli studenti stagisti che gli sono associati e valutarne i progressi.
- **Elenco dei metodi**
  Costruttore pubblico che riceve come parametro l'id del docente:

```
public function __construct($arg0)
```

Metodo pubblico che permette al tutor interno di valutare lo stage che ha seguito. Ritorna un booleano che indica se l'esecuzione non ha riscontrato errori:

```
public function valutaStage($stage)
```

Metodo pubblico che permette di visualizzare gli stagisti seguiti dal docente. Ritorna una stringa degli studenti associati:

```
public function visualizzaStagisti()
```

Metodo pubblico che modifica i dati personali dello studente:

```
public function modificaDati($dati)
```

2.3.3 attori.Proponente

- **Path dei file**
  ${WEB-ROOT}/../logic/attori/Proponente.class
- **Tipo, obiettivo e funzione del componente**
  Classe Php rappresentante un proponente di stage. È una specializzazione di *UtenteAutenticato* e quindi solamente l'utente autenticato, in questo caso come proponente, ha il permesso e i diritti di operare per inserire e visualizzare le proprie proposte e valutare i curricula degli studenti.
- **Relazioni d'uso di altre componenti**
  Per la gestione dei vari metodi viene richiesto l'utilizzo della classe *Sistema* che viene implementata attraverso il pattern creazionale Singleton.
- **Interfacce e relazioni di uso da altre componenti**
  Questa classe viene istanziata dalla classe *UtenteAutenticato*.
- **Attività svolte e dati trattati**
  Un utente autenticato come proponente può eseguire principalmente le seguenti operazioni:
  - Inserire e visualizzare e valutare gli stage;
  - Inserirsi ad un evento speciale;
  - Visualizzare e valutare i curricula degli studenti;

  Altre funzionalità che vengono interrogate dalla classe *Sistema* permettono di ritornare i dati del proponente, gli stage attivi di un dato proponente e se aderisce alla convenzione Quadro.
- **Elenco dei metodi**
  Costruttore pubblico che riceve come parametro l'id del proponente:

```
public function __construct($arg0)
```

Metodo pubblico che ritorna le informazioni relative al proponente in formato stringa:

```
public function getInfo()
```

Metodo pubblico che ritorna l'elenco degli stage attivi del proponente in formato string:

```
public function getStageAttivi()
```

Metodo pubblico che inserisce una proposta di stage, ritorna un booleano che indica l'inserimento corretto della proposta:

```
public function inserisciProposta($proposta)
```

Metodo pubblico utile al tutor esterno per la valutazione dello stage che ha seguito, il parametro passato è l'id dello stage da valutare. Ritorna un booleano per indicare l'esecuzione corretta del metodo:

```
public function valutaStage($stage)
```

Metodo pubblico utilie all'azienda per l'iscrizione a un dato evento speciale:

```
public function iscrizioneEvento($evento_speciale)
```

Metodo pubblico che verifica che l'azienda abbia aderito all'assicurazione Quadro:

```
public function verificaAdesioneQuadro()
```

Metodo pubblico che ritorna il link ai curricula degli studenti che hanno acconsentito al trattamento dei dati personali:

```
public function getCurricula()
```

Metodo pubblico che modifica i dati personali dello studente:

```
public function modificaDati($dati)
```

2.3.4 attori.Sistema

- **Path dei file** ${WEB-ROOT}/../logic/attori/Sistema.class
- **Tipo, obiettivo e funzione del componente**
  Classe Php implementata con Design Pattern Singleton contenente metodi di utilità per le altre classi e per il Presentation layer. Contiene inoltre i dati per la connessione al database, ed è quindi l'unica classe che utilizza pear.MDB2.
- **Relazioni d'uso di altre componenti**
  Utilizza la libreria pear.MDB2 per interfacciarsi al data layer.

- **Attività svolte e dati trattati**
  Il sistema si occupa di:
  - Connessione, interrogazioni, e modifiche al database
  - Creazione di password casuali
- **Elenco dei metodi**

  Metodo pubblico che ritorna il riferimento all'oggetto Sistema:

```
public static function getSistema()
```

  Costruttore privato chiamato da getSistema in caso l'oggetto Sistema non sia già stato creato. Questo legge i parametri di connessione al database dal file di configurazione:

```
private function __construct()
```

  Metodo privato che controlla ad ogni inserimento nel database che i dati da inserire non contengano tag html, che risultano pericolosi per la sicurezza del sistema:

```
private function valida($stringa)
```

  Metodo pubblico che crea una notifica secondo parametri ricevuti:

```
public function creaNotifica($parametri)
```

  Metodo privato che connette l'oggetto al database mediante l'utilizzo della libreria pear:MDB2:

```
private function connectMe() {
    $dsn = $this->_parametri;
  $options = array(
  'debug' => 2,
  'result_buffering' => false,
  );

  $connessione =& MDB2::factory($dsn, $options);
  if (PEAR::isError($connessione)) {
    echo "Errore:connessione al database fallita";
    die($connessione->getMessage());
  } else {
    return $connessione;
  }
}
```

  Con la riga

```
$connessione =& MDB2::factory($dsn, $options);
```

si crea la vera e propria connessione, il parametro *dsn* identifica i parametri di connessione al database e *options* identifica le opzioni dettate dalla libreria.

Metodo pubblico che permette di effettuare interrogazioni la database, il parametro *query* è una interrogazione SQL:

```
public function query($query)
```

Al suo interno la riga:

```
$res =& $connessione->query($query);
```

effettua l'interrogazione tramite la libreria pear.MDB2.

Metodo pubblico che permette di effettuare modifiche alle righe del database. Il parametro *query* è una query SQL:

```
public function exec($query)
```

Al suo interno la riga:

```
$res =& $connessione->exec($query);
```

esegue la richiesta di modifica. Questa funzione dovrebbe essere utilizzata per le sole query che modificano il database (es. INSERT, UPDATE).

Metodo pubblico statico (non serve l'oggetto Sistema istanziato) per creare password casuali di *lunghezza* caratteri, ed è possibile scegliere i tipi di caratteri utilizzati tramite i parametri booleani *consonanti_minuscole_on*, *consonanti_maiuscole_on*, *vocali_minuscole_on*, *vocali_maiuscole_on*, *numeri_on*, *speciali_on*:

```
public static function creaPassword($lunghezza,
    $consonanti_minuscole_on, $consonanti_maiuscole_on,
    $vocali_minuscole_on, $vocali_maiuscole_on,
    $numeri_on, $speciali_on)
```

Utilizzata per creare i link per le aziende, da autenticare nel sistema, inoltre può essere utilizzata per creare password per gli studenti sbadati che dimenticano la propria. Ha 7 parametri; ciò è in contrasto con le norme di codifica ma è un'eccezione decisa in sede di progettazione.

2.3.5 attori.Studente

- **Path dei file**
  ${WEB-ROOT}/../logic/attori/Studente.class
- **Tipo, obiettivo e funzione del componente**
  Classe Php rappresentante uno studente. È una specializzazione di *UtenteAutenticato* e quindi solamente l'utente autenticato, in questo caso come studente, ha il permesso e i diritti di operare per inserire, visualizzare e modificare i documenti, valutare le proproste di stage presenti nel sistema e scegliere un tutor.

- **Relazioni d'uso di altre componenti**
  La classe *Sistema* viene utilizzata per ricavare i dati dello studente ed eventualmente modificarli attraverso l'opportuno metodo descritto nella sezione "elenco metodi".

- **Interfacce e relazioni di uso da altre componenti**
  Questa classe viene istanziata dalla classe *UtenteAutenticato*.

- **Attività svolte e dati trattati** Un utente autenticato come studente può eseguire principalmente le seguenti operazioni:

  – Visualizzare ed eventualmente modificare i dati personali;
  – Scegliere una o più proposte di stage;
  – Confermare o eliminare la scelta di uno degli stage scelti nel punto precedente;
  – Scrivere una valutazione finale dello stage svolto;

  Altre funzionalità che vengono interrogate dalla classe *Sistema* permettono di ritornare i dati dello studente.

- **Elenco dei metodi**
  Costruttore pubblico che riceve come parametro l'id dello studente:

```
public function __construct($arg0)
```

  Metodo pubblico che ritorna l'id univoco dello studente in questione:

```
public function getId()
```

  Metodo pubblico che ritorna le informazioni relative allo studente attraverso una stringa:

```
public function getInfo()
```

  Aggiunge un documento dello studente. Ritorna un booleano che indica se i dati sono stati modificati correttamente. Il parametro *dati* passato è una stringa contenete i vari dati personali dello studente:

```
public function modificaDati($dati)
```

  Metodo pubblico che permette l'aggiunta di un documento dello studente. Ritorna un booleano che indica la corretta aggiunta del documento:

```
public function aggiungiDocumento()
```

  Metodo pubblico che consente allo studente di scegliere una proposta di stage , attraverso il parametro *proposta*. (Non ne conferma l'inizio). Ritorna un booleano che indica il corretto inserimento del documento:

```
public function scegliProposta($proposta)
```

Metodo pubblico che permette di confermare la scelta di uno degli stage preselezionati, ritorna un booleano di controllo:

```
public function confermaProposta($proposta)
```

Metodo pubblico che permette di eliminare una *proposta* preselezionata, ritorna un booleano che conferma la corretta cancellazione:

```
public function eliminaProposta($proposta)
```

Metodo pubblico che consente di selezionare un *docente* passato come parametro, come tutor interno, ritorna un booleano di controllo:

```
public function selezionaTutor($docente)
```

Metodo pubblico che consente allo studente di iscriversi a un particolare *evento* speciale

```
public function iscrizioneEvento($evento)
```

Metodo pubblico denominato wizard stage che guida lo studente nella scelta di una *proposta* e successivamente di un *tutor*. In questo modo lo studente non si trova in difficoltà nella scelta di uno stage:

```
public function wizardStage($proposta,$docente)
```

Metodo pubblico che permette allo studente di dare una valutazione conclusiva dello stage, ritorna un booleano di controllo:

```
public function valutaStage($stage)
```

2.3.6 attori.Super_utente

- **Path dei file** ${WEB-ROOT}/../logic/attori/Super_utente.class
- **Tipo, obiettivo e funzione del componente** Classe Php rappresentante il super-utente. È una specializzazione di *Amministratore* e quindi solamente l'utente autenticato, in questo caso come super-utente, ha il permesso di operare e modificare più dati possibili nel sistema. Il super-utente è unico nel sistema, ed eredita tutte le funzionalità di Amministratore, inoltre può gestire e modificare alcuni dati di altri utenti iscritti nel sistema.
- **Relazioni d'uso di altre componenti** Per la gestione dei vari metodi viene richiesto l'utilizzo della classe *Sistema* che viene implementata attraverso il pattern creazionale Singleton.
- **Interfacce e relazioni di uso da altre componenti** Questa classe viene istanziata dalla classe *UtenteNonAutenticato*.

- **Attività svolte e dati trattati** Oltre a tutto ciò che può fare l'amministratore, cioè gestire il sistema, senza poter modificare dati sensibili degli utenti, il super-user può modificare in caso di necessità i dati, oppure eliminare degli studenti.

- **Elenco dei metodi** Metodo pubblico per la modifica dei dati inseriti dagli utenti, il parametro *utente* è l'identificatore dell'utente da modificare:

```
public function modificaUtente($utente)
```

Metodo pubblico per l'eliminazione di un utente che ha espresso il desideriodi essere eliminato dal sistema, o che crea problemi di qualsiasi tipo al sistema. Il parametro indica l'identificatore dell'utente:

```
public function eliminaUtente($utente)
```

Metodo pubblico per l'eliminazione dal database degli studenti che hanno concluso lo stage prima di una certa data e che quindi non hanno più ragione di esistere. Il parametro *data* è la data entro il quale lo studente deve aver concluso per essere eliminato:

```
public function eliminaVecchiStudenti($data)
```

2.3.7 attori.Utente

- **Path dei file** ${WEB-ROOT}/../logic/attori/Utente.class

- **Tipo, obiettivo e funzione del componente**
Classe Php non istanziabile rappresentante l'utente generico del sistema SIAGAS. Da questa classe vengono derivati gli attori descritti in precedenza: *Amministratore, Docente, Amministratore, Proponente, Super_Utente.*

- **Relazioni d'uso di altre componenti**
Questa classe utilizza la classe *Sistema.*

- **Interfacce e relazioni di uso da altre componenti**
Le classi *Utente_Autenticato* e *Utente_Non_Autenticato* implementano l'interfaccia Utente.

- **Attività svolte e dati trattati**
Essendo una classe non istanziabile viene utilizzata per interfacciare i vari *utenti.* Viene utilizzata principalmente per gestire la documentazione, tutti gli eventi speciali e le proposte presenti in vetrina.

- **Elenco dei metodi**
Metodo pubblico che ritorna l'elenco delle proposte di stage presenti nel *Sistema*, ritorna un'array di stringhe:

```
public function getProposte()
```

Metodo pubblico che ritorna tutti gli eventi speciali:

```
public function getEventi() {
    $sys =  Sistema::getSistema();
    //Prendo gli eventi
  $evento =& $sys->query("SELECT * FROM evento_speciale
    ORDER BY data_inizio");
    $eventi = null;
    // Fetch delle righe
  while ($row = $evento->fetchRow()) {
    $eventi[] = $row;
  }
    return $eventi;
}
```

Metodo pubblico che ritorna tutte le proposte delle aziende iscritte ad un determinato *evento* speciale, ritorna un'array di stringhe:

```
public function getProposteEvento($evento)
```

Metodo che ritorna i link a tutte le tesi di laurea in un'array:

```
public function getAllTesi()
```

2.3.8 attori.UtenteAutenticato

- **Path dei file** ${WEB-ROOT}/../logic/attori/UtenteAutenticato.class
- **Tipo, obiettivo e funzione del componente**
  Classe Astratta Php rappresentante un utente autenticato nel sistema. È una specializzazione di *Utente* ed è l'interfaccia del Design Pattern Factory.
- **Relazioni d'uso di altre componenti**
  La classe eredita da *Utente*, ed è ereditata dagli utenti del sistema: *Studente*, *Amministratore*, *Docente*, *Proponente*.
- **Attività svolte e dati trattati**
  Un utente autenticato può eseguire principalmente le seguenti operazioni:
  - Modificare i dati personali;
  - Ritornare l'id dell'utente;
- **Elenco dei metodi**

  Metodo pubblico per ritornare l'id dell'utente autenticato

```
public function getId()
```

  Metodo pubblico astratto per la modifica dei dati dell'utente autenticato nel sistema. Questo metodo viene implementato nelle classi che ereditano la classe *Utente autenticato* e che rappresentano gli utenti.

```
public function getInfo()
```

2.3.9 attori.UtenteNonAutenticato

- **Path dei file** ${WEB-ROOT}/../logic/attori/UtenteNonAutenticato.class
- **Tipo, obiettivo e funzione del componente**
- **Tipo, obiettivo e funzione del componente**
  Classe Php rappresentante l'utente non ancora autenticato, quindi l'utente che dopo l'accesso si specializzerà in *Utente_Autenticato*. Il principale obiettivo di questa classe è la creazione di un nuovo account, oppure se l'utente possiede già un account, la relativa autenticazione a seconda delle varie specializzazioni tra *Studente, Amministratore, Super_utente, Proponente*.
- **Relazioni d'uso di altre componenti**
  Per la gestione dei vari metodi viene richiesto l'utilizzo della classe *Sistema* che viene implementata attraverso il pattern creazionale Singleton e delle varie classi di specializzazione di Utente: *Studente, Amministratore, Super_utente, Proponente*.
- **Interfacce e relazioni di uso da altre componenti** Questa classe viene utilizzata soltanto dal Presentation Layer.
- **Attività svolte e dati trattati** Come descritto nella precedente sezione "Tipo, obiettivo e funzione" la classe ha il compito di autenticare un utente non autenticato, oppure di creare un nuovo account nel caso l'utente non possegga un account.
- **Elenco dei metodi**

  Metodo pubblico che permette la creazione di un nuovo account. L'account che verrà creato avrà come nome e id un valore univoco per tutto il sistema. Ritorna UtenteAutenticato, oppure un valore nullo se la creazione non è andata a buon fine:

```
public function creaAccount($dati[])
```

  Metodo pubblico che permette l'autenticazione di un utente nel sistema. Ritorna un'array di tipo UtenteAutenticato se la username e la password corrispondono, null se l'autenticazione non è andata a buon fine. Nel database viene salvato l'hash MD5 dell'username concatenato con la password:

```
public function signIn($username, $password)
```

  Metodo pubblico che permette l'autenticazione di un proponente nel sistema. Ritorna il Proponente dell'azienda se il codice è valido, altrimenti un valore nullo:

```
public function signProp($codice)
```

2.3.10 pear.PEAR

- **Path dei file**
  ${WEB-ROOT}/../logic/pear/PEAR/
- **Tipo, obiettivo e funzione del componente**
  Libreria strutturata Php che fornisce pacchetti per lavorare e ricavare informazioni dal nucleo di pear.
- **Interfacce e relazioni di uso da altre componenti**
  La libreria PEAR viene utilizzata da pear.MDB2 la quale accede al database SIAGASDB.
- **Elenco metodi utilizzati**
  Metodo pubblico che esamina una variabile se è un oggetto PEAR_Error. Il parametro passato *connessione* è un oggetto di tipo MDB2.

```
public isError($connessione)
```

2.3.11 pear.MDB2

- **Tipo, obiettivo e funzione del componente** API per l'accesso al database, utile per costruire comandi in SQL in modo portabile, quindi indipendente dal database.
- **Relazioni d'uso di altre componenti** La libreria pear.MDB2 è utilizzata dalla classe *Sistema* che si occupa di connessione e interrogazione al database.
- **Elenco dei metodi utlizzati** Metodo pubblico della libreria pear.MDB2 per la connessione a un database, il parametro *dsn* identifica il "data source name", cioè l'indirizzo del database, compreso di username e password; il parametro *options* è un array che contiene opzioni di connessione (false di default).

```
MDB2::factory($dsn, $options)
```

Metodo pubblico per interrogare il sistema, il parametro *query* è una stringa con la query SQL.

```
function &query($query)
```

Metodo pubblico per modificare le righe del database, il parametro *query* è una stringa con la query SQL.

```
public function exec($query)
```

2.3.12 fpdf

- **Tipo, obiettivo e funzione del componente** Libreria per la creazione di file di tipo *PDF* dinamici.
- **Relazioni d'uso di altre componenti** La libreria *fpdf* è utilizzata da pagine dinamiche con codice php nella sezione studente. Inoltre la classe viene estesa per migliorare la compilazione dei file dinamici.
- **Elenco dei metodi utlizzati** Si procede all'overloading dei metodi sottoelencati per poter inserire un header e un footer in ogni pagina:

```
function header()
```

```
function footer()
```

Inoltre i metodi pubblici più importanti che si sono utilizzati sono: Metodo per scrivere un testo *txt* come se fosse in una cella di larghezza *w* e di altezza *h*, un altro parametro importante è *ln* che determina la posizione corrente dopo la chiamata del metodo.

```
Cell(float w [,float h[,string txt[,mixed border[,
  int ln[,string align[,int fill[,mixed link]]]]]]])
```

Metodo pubblico per impostare il tipo di font con il parametro *family*, lo stile con il parametro *style* e l'ampiezza del carattere con il parametro *size*

```
SetFont(string family [, string style [, float size]])
```

Metodo per impostare il valore dell'ordinata nel documento con il parametro *y*.

```
SetY(float y)
```

2.4 Raffinamento a livello di progettazione del Data Layer

Il Data Layer è composto unicamente dal *SIAGASDB*. Per il suo funzionamento si consiglia l'utilizzo di MySql, tuttavia è progettato per essere portabile. L'installazione sarà agevolata con il file di configurazione "siagas.conf" presente nella directory superiore a webroot.

Per la traduzione delle gerarchie ISA della definizione data nel documento "Specifica Tecnica" si è utilizzato l'accorpamento del padre nelle figlie, aggiungendo gli attributi del padre nelle entità figlie.

Il database è definito in dettaglio[1] in seguito, tabella per tabella.

2.4.1 amministratore

Nome campo	Tipo	Descrizione
superutente	tinyint (1) NOT NULL	=1 se superutente =0 se solo amministratore
id_utente	int (11) NOT NULL	
nome	varchar (25) NOT NULL	
mail	varchar (50) NOT NULL	
telefono	varchar (14)	
username	varchar (16) NOT NULL	
password	varchar (32) NOT NULL	hash *MD5* del username concatenato alla password
cognome	varchar (25) NOT NULL	
data_creazione	date NOT NULL	

2.4.2 docente

Nome campo	Tipo	Descrizione
id_utente	int (11) NOT NULL	
nome	varchar (25) NOT NULL	
mail	varchar (50) NOT NULL	
telefono	varchar (14)	
username	varchar (16) NOT NULL	
password	varchar (32) NOT NULL	hash *MD5* del username concatenato alla password
cognome	varchar (25) NOT NULL	
data_creazione	date NOT NULL	

[1] Sottolineate le chiavi primarie, in corsivo le chiavi esterne; la descrizione è omessa per i campi dati dal significato ovvio

2.4.3 studente

Nome campo	Tipo	Descrizione
id_utente	int (11) NOT NULL	
nome	varchar (25) NOT NULL	
mail	varchar (50) NOT NULL	
telefono	varchar (14)	
username	varchar (16) NOT NULL	
password	varchar (32) NOT NULL	hash *MD5* del username concatenato alla password
cognome	varchar (25) NOT NULL	
data_creazione	date NOT NULL	
matricola	varchar (9) NOT NULL	matricola dello studente all'interno dell'Università
curriculum	varchar (55)	URI del pdf con il curriculum
trattamento_dati	tinyint (1) NOT NULL	=1 se consente il trattamento =0 se non consente il trattamento
indirizzo	varchar (35)	
comune	varchar (35)	
provincia	varchar (16)	
cap	varchar (5)	

2.4.4 proponente

Nome campo	Tipo	Descrizione
id_utente	int (11) NOT NULL	
nome	varchar (25) NOT NULL	
mail	varchar (50) NOT NULL	
telefono	varchar (14)	
indirizzo	varchar (100)	
codLink	varchar (64)	parametro per l'autenticazione del proponente
dispStagisti	smallint (2)	#stagisti che si possono ospitare
cQuadro	tinyint (1) NOT NULL	Adesione alla convenzione "Quadro" =1 se l'azienda aderisce =0 se l'azienda non aderisce
settore	varchar (25)	settore di operatività del proponente
fax	varchar (14)	
referente	varchar (50)	recapito del referente
sito	varchar (70)	
data_inizio_link	date NOT NULL	data di invio del primo link
data_ultimo_link	date NOT NULL	data di invio dell'ultimo link
facolta	varchar (45)	

2.4.5 proposta

Nome campo	Tipo	Descrizione
id_proposta	int (11) NOT NULL	
n_stagisti	smallint (2) NOT NULL	#stagisti richiesti per lo stage
descrizione	text NOT NULL	
durata	smallint (2) NOT NULL	durata dello stage in mesi
destinatari	text	destinatari della proposta
fk_proponente	int (11) NOT NULL	riferimento al proponente

2.4.6 stage

Nome campo	Tipo	Descrizione
id_stage	int (11) NOT NULL	
data_inizio	date NOT NULL	
data_fine	date NOT NULL	
fk_studente	int (11) NOT NULL	riferimento allo studente
fk_docente	int (11)	riferimento al tutor interno
fk_proposta	int (11) NOT NULL	riferimento alla proposta
definitivo	tinyint (1) NOT NULL	=1 se la scelta è definitiva =0 se la scelta non è definitiva
valutazione_studente	text	
valutazione_proponente	text	
valutazione_tutor	text	

2.4.7 evento_speciale

Nome campo	Tipo	Descrizione
id_evento	int (11) NOT NULL	
nome	varchar (50) NOT NULL	
descrizione	text NOT NULL	
data_inizio	date NOT NULL	
data_fine	date NOT NULL	
luogo	varchar (50) NOT NULL	
responsabile	varchar (50)	recapito del responsabile

2.4.8 iscrizione_aziende

Nome campo	Tipo	Descrizione
data_iscrizione	date NOT NULL	
fk_esterno	int (11) NOT NULL	riferimento all'azienda
fk_evento	int (11) NOT NULL	riferimento all'evento

2.4.9 iscrizione_studente

Nome campo	Tipo	Descrizione
data_iscrizione	date NOT NULL	
fk_studente	int (11) NOT NULL	riferimento allo studente
fk_evento	int (11) NOT NULL	riferimento all'evento

2.4.10 log

Nome campo	Tipo	Descrizione
id_log	int (11) NOT NULL	
tipo	varchar (20) NOT NULL	
data	datetime NOT NULL	
descrizione	text NOT NULL	
fk_utente	int (11) NOT NULL	riferimento all'utente che provoca l'errore

3 Appendice

3.1 Codice sorgente

Il codice sorgente, come da ultimi accordi, non verrà allegato in quanto questa applicazione web non si presta ad una compilazione e/o installazione automatica. Viene comunque indicato sia in questo documento che nella lettera di presentazione un URL dove è disponibile visionare quanto è stato realizzato finora. L'applicazione disponibile all'URL indicato è soggetta a continue modifiche ed aggiornamenti in quanto è utilizzata da Wheelsoft per test e verifiche. Allo stato attuale sono stati realizzati i seguenti punti:

- Creazione del SIAGASDB popolato con dati d'esempio;
- Creazione di tutte le classi del Logic Layer;
- Implementazione di tutti i metodi delle classi *attori.Utente*, *attori.UtenteAutenticato*, *attori.UtenteNonAutenticato*;
- Implementazione dei metodi *Classe.getId(...)* per ogni classe e dei metodi *Amministratore::getInfo(...)*, *Proponente::getInfo(...)*, *Proponente::getStageAttivi(...)*, *Sistema::getSistema(...)*, *Sistema::valida(...)*, *Sistema::connectMe(...)*, *Sistema::query(...)*, *Sistema::exec(...)*, *Sistema::creaPassword(...)*, *Studente::getInfo(...)*, *Studente::modificaDati(...)*.
- Creazione di tutte le pagine XHTML, dei fogli di stile CSS e degli script PHP dedicati alla gestione dell'autenticazione ed alla creazione dinamica delle pagine.

Dati per l'accesso all'applicazione

- **URL**: http://www.wheelsoft.servehttp.com
- **Username per l'accesso**: prof
- **Password per l'accesso**: cqe4DH

Una volta visualizzata la schermata principale è possibile creare un nuovo utente cliccando su "Registrazione" per poter visualizzare le pagine dedicate agli studenti.

3.2 Tracciamento della relazione metodi - requisiti

La seguente tabella illustra i metodi che soddisfano i requisiti identificati dal codice univoco definito nell'analisi dei requisiti.

3.2.1 Requisiti funzionali

Requisito	Metodo
RFOb01	metodo *attori.Utente::getProposte()*
RFOb02	metodo *attori.Studente::aggiungiDocumento('curriculum')*
RFOb03	metodo *attori.Studente::iscrizioneEvento(evento)*
	metodo *attori.Proponente::iscrizioneEvento()*
RFOb04	metodo *attori.Studente::aggiungiDocumento(tipo_documento)*
RFOb05	pagine XHTML con guida contestuale
	metodo *attori.Studente::wizardStage()*
RFOb06	metodo *attori.Studente::inserisciProposta(proposta)*
RFOb07	metodo *attori.Amministratore::gestisciUtenti(id_utente)*
	metodo *attori.Amministratore::confermaProposta(proposta)*
RFOb08	metodo *attori.Amministratore::gestisciEventiSpeciali()*
RFOb09	metodo *attori.Amministratore::generaDocumento()*
RFOb10	tabella *data.proponente*
RFOb11	metodo *attori.Sistema::generaLink()*
	metodo *attori.Amministratore::registraProponente()*
RFOb12	metodo *associaTutor(docente, studente)*
RFOb13	classe *attori.Super_utente*
RFOb14	metodo *attori.Proponente::verificaAdesioneQuadro()*
RFDe01	metodo *attori.Utente::getAllTesi()*
RFDe02	metodo *attori.Proponente::getCurricula()*
RFDe03	metodo *attori.Sistema::generaNotifica(parametri)*
RFDe04	metodo *attori.Amministratore::creaMailingList()*
RFDe05	metodo in fase di progettazione
RFOp01	metodo *attori.Amministratore::valutaStage()*
	metodo *attori.Docente::valutaStage()*
	metodo *attori.Proponente::valutaStage()*
RFOp02	metodo *attori.Amministratore::visualizzaStorico()*
RFOp03	metodo *attori.Amministratore::associaTutor()*
RFOp04	metodo *attori.Docente::visualizzaStagisti()*
RFOp05	metodo *attori.Sistema::generaLink()*

3.2.2 Requisiti di qualità

Requisito	Soddisfacimento
RQOb01	validazione WAI-AAA del Presentation layer
RQOb02	utilizzo *PEAR::MDB2* e struttura OO
RQOb03	pagine XHTML con guida contestuale nel Presentation layer
RQOb04	utilizzo di XHTML, CSS per il Presentation layer
	utilizzo di PHP 5
	portabilità rispetto al Data layer garantita dall'uso di *PEAR::MDB2*