

---

# PROGETTO SIGEOL

*Definizione di prodotto*

*v0.1.0*

Redazione:

4 marzo 2009



*quixoft.sol@gmail.com*

<b>Verifica:</b>	Verificatore
<b>Approvazione:</b>	Responsabile
<b>Stato:</b>	Preliminare — Formale
<b>Uso:</b>	Interno — Esterno
<b>Distribuzione:</b>	QuiXoft

## Sommario

Descrizione dettagliata delle caratteristiche tecniche ed architetture del prodotto SIGEOL

---



## Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	Scopo del documento . . . . .	1
1.2	Scopo del prodotto . . . . .	1
1.3	Glossario . . . . .	1
1.4	Riferimenti normativi . . . . .	1
<b>2</b>	<b>Standard di progetto</b>	<b>1</b>
2.1	Standard di progettazione architetturale . . . . .	1
2.1.1	UML . . . . .	1
2.2	Pattern . . . . .	2
2.3	Standard di documentazione del codice . . . . .	3
2.4	Standard di denominazione di entità e relazioni . . . . .	3
2.5	Standard di programmazione . . . . .	3
2.5.1	Ruby e framework Rails . . . . .	3
2.5.2	Java . . . . .	5
2.6	Strumenti di lavoro . . . . .	5
<b>3</b>	<b>Specifica delle componenti</b>	<b>6</b>
3.1	Componente Model . . . . .	6
3.2	Componente Controller . . . . .	6
3.2.1	Azioni comuni a più controller . . . . .	7
3.2.2	ApplicationController . . . . .	9
3.2.3	GraduateCoursesController . . . . .	10
3.2.4	CurriculumsController . . . . .	10
3.2.5	UsersController . . . . .	10
3.2.6	TeachersController . . . . .	11
3.2.7	SessionsController . . . . .	11
3.2.8	TeachingsController . . . . .	12
3.2.9	BuildingsController . . . . .	12
3.2.10	ClassroomsController . . . . .	12
3.2.11	TimetablesController . . . . .	13
3.3	Componente View . . . . .	13
3.3.1	Layouts . . . . .	14
3.3.2	Templates . . . . .	15
3.3.3	Partials . . . . .	16
3.4	Componente Helper . . . . .	16
3.4.1	ApplicationHelper . . . . .	16
3.4.2	BuildingsHelper . . . . .	17
3.4.3	ClassroomsHelper . . . . .	17
3.4.4	CurriculumsHelper . . . . .	17
3.4.5	SessionsHelper . . . . .	17
3.4.6	GraduateCoursesHelper . . . . .	18



## INDICE

---

3.4.7	TeachersHelper . . . . .	18
3.4.8	TeachingsHelper . . . . .	18
3.4.9	TimetablesHelper . . . . .	18
3.4.10	UsersHelper . . . . .	19



# 1 Introduzione

## 1.1 Scopo del documento

Il presente documento denominato DESCRIZIONE DI PRODOTTO si prefigge di illustrare ed analizzare con maggior dettaglio i metodi ed i formalismi adottati nella definizione del prodotto SIGEOL

## 1.2 Scopo del prodotto

Il progetto sotto analisi, denominato SIGEOL, si prefigge di automatizzare la generazione, la gestione, l'ottimizzazione e la consultazione degli orari di lezione. Per maggiori dettagli consultare il documento denominato ANALISI DEI REQUISITI alla sua ultima versione.

## 1.3 Glossario

Le definizioni dei termini specialistici usati nella stesura di questo e di tutti gli altri documenti possono essere trovate nel documento denominato GLOSSARIO al fine di eliminare ogni ambiguità e di facilitare la comprensione dei temi trattati. Ogni termine la cui definizione è disponibile all'interno del glossario verrà marcato con una sottolineatura.

## 1.4 Riferimenti normativi

Il documento denominato NORME DI PROGETTO accompagna e completa il presente ed ogni documento ufficiale.

# 2 Standard di progetto

## 2.1 Standard di progettazione architettuale

La definizione dell'intero sistema oggetto di studio è stata effettuata attraverso l'uso di diagrammi UML e l'applicazione di pattern consolidati ed in uso in molti prodotti software.

### 2.1.1 UML

Il linguaggio UML è utilizzato per la modellazione architettuale di un sistema in quanto grazie alla sua capacità e chiarezza espressiva risulta di facile comprensione anche a persone esterne al progetto stesso. Il team QuiXoft ha utilizzato UML 2.0 per:

- Diagrammi use-case nel documento ANALISI DEI REQUISITI
- Diagrammi delle classi, delle componenti, di attività e delle sequenze nei documenti SPECIFICA TECNICA e DEFINIZIONE DI PRODOTTO

## 2.2 Pattern

All'interno dell'architettura del sistema sono stati utilizzati i seguenti pattern, presenti nel framework Ruby on Rails il quale è alla base di tutto il prodotto:

**MVC** Il pattern MVC (Model View Controller) si basa sulla separazione tra i componenti software del sistema, che gestiscono il modo in cui presentare i dati, e i componenti che gestiscono i dati stessi.

**Façade** Permette, attraverso un'interfaccia più semplice, l'accesso a sottosistemi aventi interfacce complesse e molto diverse tra loro, nonché a blocchi di codice complessi.

**REST** Representational state transfer (REST) è un tipo di architettura software per i sistemi di ipertesto distribuiti come il World Wide Web. REST si riferisce ad un insieme di principi di architetture di rete, i quali delineano come le risorse sono definite e indirizzate.

**Convention Over Configuration** Convention Over Configuration è un paradigma di programmazione che prevede configurazione minima (o addirittura assente) per il programmatore che utilizza un framework che lo rispetti, obbligandolo a configurare solo gli aspetti che si differenziano dalle implementazioni standard o che non rispettano particolari convenzioni di denominazione o simili. Significa che Rails prevede delle impostazioni di default per qualsiasi aspetto dell'applicazione. Utilizzando queste convenzioni sarà possibile velocizzare i tempi di sviluppo evitando di realizzare scomodi file di configurazione. L'esempio più chiaro del COC si può notare a livello di modelli: rispettando le convenzioni previste dal framework è possibile realizzare strutture di dati complesse con molte relazioni tra oggetti in pochissimo tempo, in maniera quasi meccanica e soprattutto senza definire nessuna configurazione. Questo concetto differenzia Rails dai framework che prevedono molte righe di configurazione per ogni aspetto dell'applicazione. Con il COC tutto diventa più snello e più dinamico. Ovviamente per situazioni in cui le convenzioni non possano essere rispettate, Rails permette di utilizzare schemi funzionali diversi da quelli previsti.

**DRY** Questo concetto, fortemente filosofico, prevede che ciascun elemento di un'applicazione debba essere implementato solamente una volta e niente debba essere ripetuto. Questo significa che, mediante Rails, è possibile gestire funzionalità ripetitive con una estrema fattorizzazione del codice ("scrivo una volta e uso più volte") che facilita sia lo sviluppo iniziale che eventuali modifiche successive del prodotto.

**View Helper** Questo pattern disaccoppia il Business Logic dallo strato



## 2 STANDARD DI PROGETTO

---

View, il che facilita la manutenibilità. Aiuta a separare, in fase di sviluppo, la responsabilità del web designer e dello sviluppatore.

**Active Record** Secondo il pattern Active Record esiste una relazione molto stretta fra tabella e classe, colonne e attributi della classe.

- una tabella di un database relazionale è gestita attraverso una classe
- una singola istanza della classe corrisponde ad una riga della tabella
- alla creazione di una nuova istanza viene creata una nuova riga all'interno della tabella, e modificando l'istanza la riga viene aggiornata

### 2.3 Standard di documentazione del codice

Il team QuiXoft si avvalerà dello strumento RDoc, specifico per il linguaggio Ruby. Questo strumento estrapola dal codice sorgente i commenti al codice stesso, organizzandoli e rendendoli disponibili alla consultazione tramite pagine HTML. Per questo motivo ogni membro del team alla stesura di qualsiasi classe o metodo dovrà documentarlo tramite la sintassi specifica di RDoc.

### 2.4 Standard di denominazione di entità e relazioni

Lo schema di denominazione deve essere determinante per la comprensione del flusso logico dell'applicazione. Verranno quindi utilizzati nomi significativi che identifichino la funzione e lo scopo dell'elemento. Inoltre saranno seguite le convenzioni generali dello specifico linguaggio di programmazione utilizzato per realizzare l'elemento, nonché ulteriori convenzioni dettate dal framework utilizzato. Per maggiori informazioni consultare la sezione 2.5

### 2.5 Standard di programmazione

Ogni file deve contenere esattamente una classe od un modulo, eccezion fatta per i template di file HTML e JavaScript. Inoltre è necessaria ai fini di una migliore leggibilità, l'uso di una corretta indentazione, fornita dall'ambiente di sviluppo.

#### 2.5.1 Ruby e framework Rails

Di seguito sono elencate le convenzioni utilizzate negli elementi sviluppati con il linguaggio Ruby.



### **Variabili locali**

Prima lettera minuscola, seguita da altri caratteri minuscoli. Se la variabile comprende più parole, queste andranno separate con un `_` (underscore). Esempio: `variabile_locale`

### **Variabili d'istanza**

Si utilizza la stessa convenzione adottata nelle variabili locali, con l'aggiunta di un `@` (at) prima del nome. Esempio: `@variabile_istanza`

### **Variabili di classe**

Si utilizza la stessa convenzione adottata nelle variabili locali, con l'aggiunta di una doppia `@` (at) prima del nome. Esempio: `@@variabile_classe`

### **Variabili globali**

Si utilizza la stessa convenzione adottata nella variabili locali, con l'aggiunta di un `$` (dollar) prima del nome. Esempio: `$variabile_globale`

### **Costanti**

Prima lettera maiuscola, seguita da altri caratteri maiuscoli. Se la variabile comprende più parole, queste andranno separate con un `_` (underscore). Esempio: `UNA_COSTANTE`

### **Metodi d'istanza**

Prima lettera minuscola, seguita da altri caratteri minuscoli. Se il nome comprende più parole, queste andranno separate con un `_` (underscore). Esempio: `metodo_istanza`

### **Classi e moduli**

Prima lettera maiuscola, seguita da altri caratteri minuscoli. Se il nome comprende più parole, la prima lettera di ogni parola deve essere maiuscola. Esempio: `UnaClasse`

### **Model**

Si utilizza la stessa convenzione per le classi ed inoltre il nome dovrà essere il singolare (in lingua inglese) del nome della tabella del database a cui si riferisce. Esempio: `Order`



## 2 STANDARD DI PROGETTO

---

### **Controller**

Si utilizza la stessa convenzione per le classi ed inoltre il nome dovrà essere il plurale (in lingua inglese) del nome del Model a cui si riferisce, seguito dalla parola *Controller*. Esempio: **OrdersController**

### **Tabelle del database**

Prima lettera minuscola, seguita da altri caratteri minuscoli. Se il nome comprende più parole, queste andranno separate con un `_` (underscore). Inoltre il nome deve essere il plurale del model (in lingua inglese) a cui la tabella si riferisce. Esempio: **orders**

### **Chiave primaria**

Il nome della chiave primaria dovrà essere **id**

### **Chiavi esterne**

Il nome della chiave esterna dovrà essere il singolare (in lingua inglese) della tabella di riferimento, seguito da un `_` (underscore) e dalla parola *id*, con ogni carattere minuscolo. Esempio: **order\_id**

### **Tabelle per le relazioni molti a molti**

Concatenazione tramite `_` (underscore) dei nomi al plurale (in lingua inglese) dei model coinvolti in ordine alfabetico, con ogni carattere minuscolo. Esempio **items\_orders**

### **File**

Ogni nome di file è caratterizzato dalla presenza di soli caratteri minuscoli e la concatenazione di più parole è effettuata tramite `_` (underscore).

### **2.5.2 Java**

Per quanto riguarda i file sorgenti scritti utilizzando il linguaggio Java fanno fede le norme e convenzioni acquisite da ogni membro del team durante il corso di Programmazione 3 o Programmazione concorrente e distribuita, a seconda dell'ordinamento a cui il componente appartiene.

## **2.6 Strumenti di lavoro**

Durante tutto lo svolgimento del progetto, il team QuiXoft utilizzerà i seguenti strumenti:





### 3 SPECIFICA DELLE COMPONENTI

---

- IDE NetBeans 6.5  
<http://www.netbeans.org/>
- JDK 6 Update 12  
<http://java.sun.com/>
- JRuby 1.1.5  
<http://jruby.codehaus.org/>
- GlassFishV3  
<https://glassfish.dev.java.net/>
- MySql 5.0  
<http://www.mysql.com/>
- Rails framework  
<http://rubyonrails.org/>
- RDoc  
<http://rdoc.sourceforge.net/>
- rcov  
<http://rubyforge.org/projects/rcov/>
- W3C validator  
<http://validator.w3.org/>
- Prototype 1.6  
<http://www.prototypejs.org/>
- L<sup>A</sup>T<sub>E</sub>X  
<http://www.latex-project.org/>

## 3 Specifica delle componenti

Il sistema Sigel è strutturato seguendo il paradigma MVC, con l'aggiunta di ulteriori: Helper, MiddleMan e Algorithm.

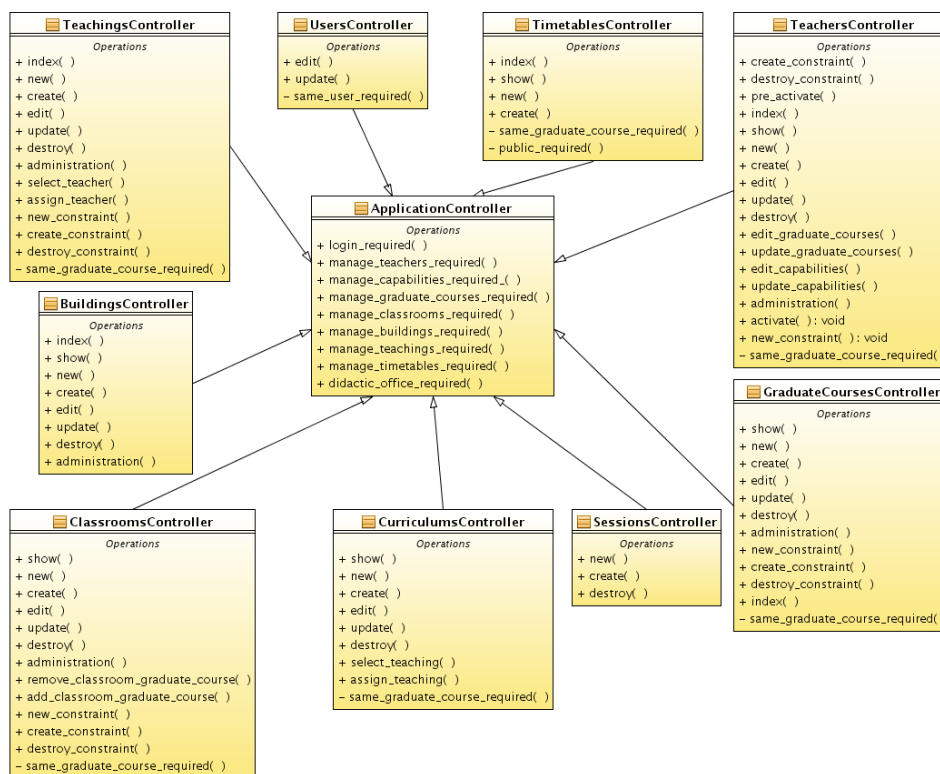
### 3.1 Componente Model

### 3.2 Componente Controller

La componente Controller si occupa di gestire le azioni che l'utente effettua, solitamente attraverso una view. Ogni metodo pubblico rappresenta quindi una specifica azione, eccezion fatta per l'Application Controller. Di seguito è riportato il diagramma delle classi che illustra questa componente. Sono stati omessi volutamente i tipi di ritorno per i metodi che implementano un'azione, in quanto queste operazioni non sono utilizzate per

### 3 SPECIFICA DELLE COMPONENTI

restituire un valore, bensì inizializzano alcune variabili d'istanza che saranno successivamente disponibili nella view specifica per quella azione. Inoltre per aumentare la leggibilità è stata omessa la derivazione della classe ApplicationController da ActionController::Base.



#### 3.2.1 Azioni comuni a più controller

Per evitare fastidiose ripetizioni in questa sezione verranno descritti i metodi che figurano con lo stesso nome in diversi controller. La scelta dello stesso nome non è casuale, in quanto rispecchia la funzione dell'azione.

- **index:** rende disponibile alla view specifica un insieme d'istanze ed è raggiungibile eseguendo una richiesta GET all'indirizzo /nomecontroller. Ad esempio l'azione **index** di **GraduateCourseController** fornisce alla view un insieme di corsi di laurea ed è raggiungibile attraverso una richiesta GET all'indirizzo /graduate.courses.
- **show:** rende disponibile alla view specifica un'istanza ed è raggiungibile eseguendo una richiesta GET all'indirizzo /nomecontroller/id. Usando sempre l'esempio dei corsi di laurea, eseguendo una richiesta GET all'indirizzo /graduate.courses/1 verranno visualizzate le informazioni relative al corso di laurea con id 1.



### 3 SPECIFICA DELLE COMPONENTI

---

- **new**: rende disponibile alla view specifica un'istanza vuota per permettere l'inserimento di un nuovo oggetto. E' raggiungibile eseguendo una richiesta GET `/nomecontroller/new`
- **create**: acquisisce i dati da una richiesta POST per salvare l'oggetto nel database attraverso il model. Solitamente i dati provengono da una form con metodo POST presente nella view per l'azione **new**. E' possibile comunque invocare questa azione mediante una richiesta POST all'indirizzo `/nomecontroller`
- **edit**: rende disponibile alla view specifica un'istanza esistente per permetterne la modifica. e' raggiungibile attraverso una richiesta GET all'indirizzo `/nomecontroller/id/edit`
- **update**: acquisisce i dati da una richiesta PUT per aggiornare lo stato dell'oggetto nel database attraverso il model. Solitamente i dati provengono da una form con metodo PUT presente nella view per l'azione **edit**. E' possibile comunque invocare questa azione mediante una richiesta PUT all'indirizzo `/nomecontroller/id`
- **destroy**: distrugge l'oggetto attraverso il model. Questa azione viene invocata tramite una richiesta DELETE all'indirizzo `/nomecontroller/id`
- **administration**: rende disponibile alla view specifica un insieme d'istanze per effettuarne l'amministrazione. Questa azione è raggiungibile attraverso una richiesta GET all'indirizzo `/nomecontroller/administration`.
- **new\_constraint**: rende disponibile alla view specifica un'istanza per un vincolo od una preferenza per permetterne l'inserimento. E' raggiungibile eseguendo una richiesta GET all'indirizzo `/nomecontroller/id/new_constraint`
- **create\_constraint**: acquisisce i dati da una richiesta POST per salvare il vincolo o la preferenza nel database attraverso il model. Solitamente i dati provengono da una form con metodo POST presente nella view per l'azione **new\_constraint**. E' possibile comunque invocare questa azione mediante una richiesta POST all'indirizzo `/nomecontroller/id/create_constraint`
- **destroy\_constraint**: distrugge l'oggetto attraverso il model. Questa azione viene invocata tramite una richiesta DELETE all'indirizzo `/nomecontroller/id/destroy_constraint/id`

Solitamente non è necessaria l'autenticazione o il possesso di alcuni privilegi per eseguire le azioni **index** e **show**. Per quanto riguarda gli altri metodi,

invece, può ritenersi necessaria l'autenticazione od il possesso di alcuni privilegi. Per ogni controller sarà descritto questo aspetto. Differente invece è il caso del metodo `same_graduate_course_required`, dichiarato privato nei controllers che lo implementano. Questo metodo non rispecchia un'azione, ma è utilizzato come filtro, ovvero è chiamato prima o dopo una determinata azione, per impedire la modifica o la cancellazione di un oggetto appartenente ad un corso di laurea diverso da quello dell'utente autenticato.

### 3.2.2 ApplicationController

Questa classe deriva direttamente da `ActionController::Base`, ed è estesa da ogni controller. Prevede metodi di pubblica utilità per gli altri controller, ma nessuna azione. L' `ApplicationController` del sistema Sigel prevede i seguenti metodi pubblici, utilizzati come filtri dagli altri controller.

- `login_required`: se l'utente non è autenticato questo metodo reindirizza alla pagina di login
- `manage_teachers_required`: se l'utente non possiede i privilegi per gestire i docenti questo metodo reindirizza alla pagina principale mostrando un errore.
- `manage_capabilities_required`: se l'utente non possiede i privilegi per gestire i privilegi questo metodo reindirizza alla pagina principale mostrando un errore.
- `manage_graduate_courses_required`: se l'utente non possiede i privilegi per gestire i corsi di laurea questo metodo reindirizza alla pagina principale mostrando un errore.
- `manage_classrooms_required`: se l'utente non possiede i privilegi per gestire le aule questo metodo reindirizza alla pagina principale mostrando un errore.
- `manage_buildings_required`: se l'utente non possiede i privilegi per gestire gli edifici questo metodo reindirizza alla pagina principale mostrando un errore.
- `manage_teachings_required`: se l'utente non possiede i privilegi per gestire gli insegnamenti questo metodo reindirizza alla pagina principale mostrando un errore.
- `manage_timetables_required`: se l'utente non possiede i privilegi per gestire gli orari questo metodo reindirizza alla pagina principale mostrando un errore.
- `didactic_office_required`: se l'utente non appartiene ad una segreteria didattica questo metodo reindirizza alla pagina principale mostrando un errore.



## 3 SPECIFICA DELLE COMPONENTI

---

### 3.2.3 GraduateCoursesController

Il controller `GraduateCourseController` permette alla segreteria didattica di creare e eliminare i corsi di laurea. Inoltre permette agli utenti con gli opportuni privilegi di aggiornare le informazioni relative ai propri corsi di laurea. I filtri utilizzati in questo controller vengono tutti anteposti alle azioni e sono i seguenti:

- `login_required` non è utilizzato nelle azioni `index` e `show`
- `manage_graduate_courses_required` è utilizzato nelle azioni `edit`, `update`, `destroy`, `administration`
- `didactic_office_required` è utilizzato nelle azioni `new`, `create` e `destroy`
- `same_graduate_course_required` è utilizzato nelle azioni `edit`, `update`, e `destroy`.

### 3.2.4 CurriculumController

Il controller `CurriculumController` permette all'utente con gli opportuni privilegi di creare, modificare ed eliminare le informazioni relative ai curricula dei propri corsi di laurea, nonché di aggiungere o rimuove insegnamenti da quest'ultimi attraverso i metodi `select_teaching`, raggiungibile mediante una richiesta GET all'indirizzo `/curriculums/id/select_teaching`, e `assign_teaching`, raggiungibile mediante una richiesta POST all'indirizzo `/curriculums/id/assign_teaching`. Il primo fornisce alla view una lista degli insegnamenti presenti nel corso di laurea a cui il curriculum appartiene, mentre il secondo crea questa associazione tramite il model. I filtri utilizzati in questo controller vengono tutti anteposti alle azioni e sono i seguenti:

- `login_required` non è utilizzato solamente nell'azione `show`
- `manage_graduate_courses_required` non è utilizzato solamente nell'azione `show`
- `same_graduate_course_required` è utilizzato nelle azioni `edit`, `update`, `select_teaching` e `assign_teaching`.

### 3.2.5 UsersController

Il controller `UserController` dispone solamente delle azioni `edit` e `update` e del metodo privato `same_user_required` (utilizzato come filtro anteposto alle due azioni assieme al filtro `login_required`). Questa scelta progettuale deriva dalla relazione tra i model `Teacher`, `DidacticOffice` e `User`. Confrontare la sezione 3.1. Il metodo privato `same_user_required` garantisce che non si stia cercando di modificare un utente diverso dal proprio.



### 3.2.6 TeachersController

Il controller **TeachersController** permette all'utente con gli opportuni privilegi di creare, modificare ed eliminare i docenti dei propri corsi da laurea, nonché di attribuire ad essi nuovi privilegi e corsi di laurea. Attraverso l'azione **new** viene richiesto un indirizzo e-mail di un docente e l'azione **create** verificherà se questo è presente o meno nel database. Nel primo caso se il docente appartiene già a quel corso di laurea verrà segnalato un errore, altrimenti verrà aggiunto; nel secondo caso verrà inviata al nuovo docente una mail con le istruzioni per la registrazione al sistema. Il nuovo docente tramite l'azione **pre\_activate**, raggiungibile mediante una richiesta GET all'indirizzo `/teachers/id/pre_activate?digest=`, potrà completare la creazione dello proprio user, e del relativo indirizzo, che sarà delegata all'azione **activate**, raggiungibile mediante una richiesta POST all'indirizzo `/teachers/id/activate?digest=`. La certezza che solamente il docente invitato potrà creare il proprio account è resa possibile tramite il parametro **digest** che solamente chi ha ricevuto la mail di invito può conoscere.

Infine attraverso le azioni **edit\_graduate\_courses**, **edit\_capabilities**, **update\_graduate\_courses** e **update\_capabilities**, raggiungibili rispettivamente mediante una richiesta GET all'indirizzo `/teachers/id/edit_graduate_courses` o `/teachers/id/edit_capabilities` e mediante una richiesta POST all'indirizzo `/teachers/is/update_graduate_courses` o `/teachers/id/update_capabilities`, è possibile modificare o rimuovere corsi di laurea e privilegi ai docenti da user che ne hanno la facoltà. I filtri utilizzati in questo controller vengono tutti anteposti alle azioni e sono i seguenti:

- **login\_required** non è utilizzato nelle azioni **index**, **show** ed **activate**
- **manage\_teachers\_required** è utilizzato nelle azioni **new**, **create**, **administration**, **edit\_graduate\_courses**, **update\_graduate\_courses**
- **manage\_capabilities\_required** è utilizzato nelle azioni **edit\_capabilities** e **update\_capabilities**
- **same\_graduate\_course\_required** è utilizzato nelle azioni **edit\_graduate\_courses**, **edit\_capabilities**, **update\_graduate\_courses** e **update\_capabilities**.

### 3.2.7 SessionsController

Il controller **SessionsController** permette la creazione di una sessione al momento dell'autenticazione dello user. Utilizza quindi le azioni **new**, **create** e **destroy** senza alcun filtro, raggiungibili rispettivamente con una richiesta GET all'indirizzo `/session/new`, POST all'indirizzo `/session` e DELETE all'indirizzo `/session/id`.

### 3.2.8 TeachingsController

Il controller `TeachingsController` permette all'utente con gli opportuni privilegi di creare, modificare ed eliminare le informazioni relative agli insegnamenti dei propri corsi di laurea, nonché di aggiungere o rimuovere il docente che tiene l'insegnamento in questione attraverso le azioni `select_teacher`, raggiungibile mediante una richiesta GET all'indirizzo `/teachings/id/select_teacher`, e `assign_teacher`, raggiungibile mediante una richiesta POST all'indirizzo `/teachers/id/assign_teacher`. Il primo fornisce alla view una lista dei docenti presenti nel corso di laurea a cui il curriculum appartiene, mentre il secondo crea questa associazione tramite il model. I filtri utilizzati in questo controller vengono tutti anteposti alle azioni e sono i seguenti:

- `login_required` non è utilizzato nelle azioni `index` e `show`
- `manage_teachings_required` non è utilizzato nelle azioni `index` e `show`
- `same_graduate_course_required` è utilizzato nelle azioni `edit`, `update`, `destroy`, `select_teacher` e `assign_teacher`.

### 3.2.9 BuildingsController

Il controller `BuildingsController` permette all'utente con gli opportuni privilegi di creare, modificare ed eliminare le informazioni relative agli edifici ed ai relativi indirizzi. I filtri utilizzati in questo controller vengono tutti anteposti alle azioni e sono i seguenti:

- `login_required` non è utilizzato nelle azioni `index` e `show`
- `manage_buildings_required` non è utilizzato nelle azioni `index` e `show`

Non è presente il filtro `same_graduate_course_required` in quanto un edificio non appartiene ad uno o più corsi di laurea, bensì all'intero sistema universitario.

### 3.2.10 ClassroomsController

Il controller `ClassroomsController` permette all'utente con gli opportuni privilegi di creare, modificare ed eliminare le informazioni relative alle aule, nonché di aggiungere o rimuovere le aule ai propri corsi di laurea attraverso le azioni `add_classroom_graduate_course` e `remove_classroom_graduate_course` raggiungibili attraverso una richiesta POST agli indirizzi, rispettivamente, `/classrooms/id/add_classroom_graduate_course` e



### 3 SPECIFICA DELLE COMPONENTI

---

/classrooms/id/remove-classroom-graduate-course I filtri utilizzati in questo controller vengono tutti anteposti alle azioni e sono i seguenti:

- `login_required` non è utilizzato solamente nell'azione `show`
- `manage_classrooms_required` non è utilizzato solamente nell' azione `show`
- `same_graduate_course_required` non è utilizzato nelle azioni `show`, `new` e `create`

#### 3.2.11 TimetablesController

Il controller `TimetablesController` permette all'utente con gli opportuni privilegi di creare ed eliminare gli orari. I filtri utilizzati in questo controller vengono tutti anteposti alle azioni e sono i seguenti:

- `login_required` non è utilizzato nelle azioni `index` e `show`
- `manage_timetables_required` non è utilizzato nelle azione `index` e `show`
- `same_graduate_course_required` non è utilizzato nelle azioni `index`, `show`, `new`, `create`
- `public_required` è utilizzato nell'azione `show` ed assicura che l'orario che si intende visualizzare sia stato dichiarato pubblico da un utente con gli opportuni privilegi.

### 3.3 Componente View

Il componente View si occupa di presentare le informazioni presenti nel sistema a chiunque le richieda, sia esso un utente finale o un altro sistema software. Per far ciò sono presenti diversi elementi in questa componente, tra i quali:

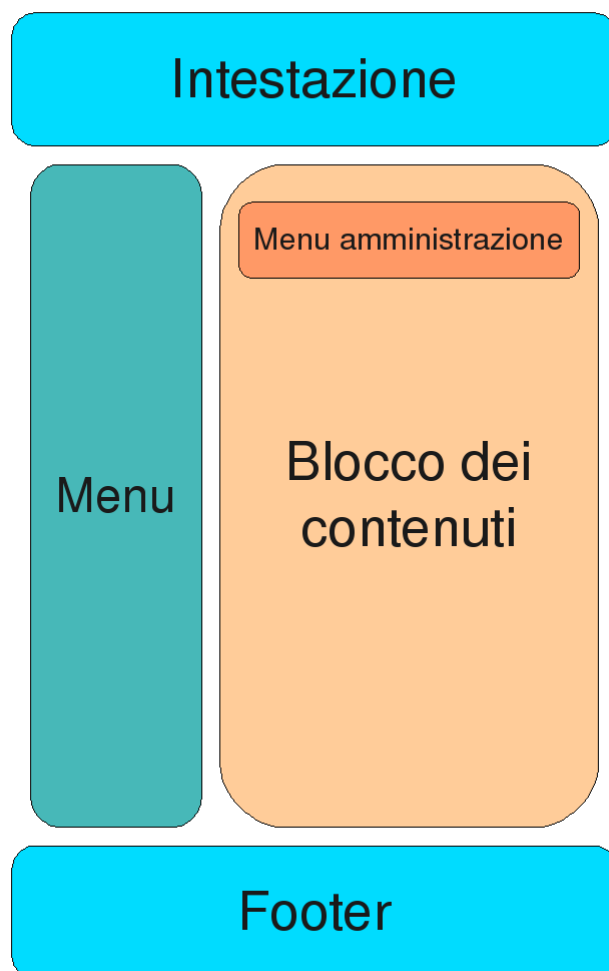
- **Layouts:** impostano l'impaginazione e la struttura dei template.
- **Templates:** vengono incorporati in un layout e si occupano della effettiva presentazione delle informazioni
- **Partials:** vengono incorporati in uno o più template o in uno o più layout. Sono frammenti di codice che presentano un sottoinsieme delle informazioni del sistema, riutilizzabili da più view.



#### 3.3.1 Layouts

Il sistema Sigeol prevede un unico layout XHTML per tutta l'applicazione in modo da garantire la stessa impaginazione per ogni template. Come nella maggior parte dei siti web moderni è prevista un'intestazione, un footer, un menu presente nella parte sinistra della pagina ed un blocco dei contenuti presenti nella parte centrale dove verranno renderizzati i template. In aggiunta sarà presente un menu di amministrazione ove richiesto che verrà collocato nella parte superiore del blocco dei contenuti.

Di seguito è riportata una immagine esplicativa della struttura del layout:





## 3 SPECIFICA DELLE COMPONENTI

---

### 3.3.2 Templates

Il sistema Sigeol prevede un diverso template per ogni azione raggiungibile attraverso una richiesta GET presente in ogni controller. Di seguito viene riportata una descrizione di ogni template:

#### Template comuni a più controller

Per evitare fastidiose ripetizione di seguito sono riportate le descrizioni dei template per le azioni che figurano in più controller:

- `new.html.erb`: presenta un form per l'inserimento di una nuova istanza.
- `edit.html.erb`: presenta un form per la modifica di un'istanza esistente.
- `index.html.erb`: presenta un elenco ordinato di un insieme di istanze.
- `show.html.erb`: presenta le informazioni relative ad una determinata istanza.
- `administration.html.erb`: presenta un elenco ordinato di un insieme di istanza e le relative funzioni per amministrarle.
- `new_constraint.html.erb`: presenta un form per l'inserimento di un nuovo vincolo o preferenza

Inoltre per garantire l'interoperabilità dei dati sono presenti i template XML per tutte le azioni `index` e `show`, nonché un template PDF per l'azione `show` di `TimetablesController`.

#### Template per `TeachingsController`

Il template XHTML `select_teacher.html.erb` è presente per il `TeachingsController` per l'azione `select_teacher`. Questo presenta un form per l'assegnamento di un docente ad un insegnamento utilizzando metodi per la presentazione automatica della lista dei docenti come i tag XHTML `<select>` e `<option>`.

#### Template per `CurriculumsController`

Il template XHTML `select_teaching.html.erb` è presente per il `TeachingsController` per l'azione `select_teaching`. Questo presenta un form per l'assegnamento di un insegnamento ad un curriculum utilizzando metodi per la presentazione automatica della lista degli insegnamenti come i tag XHTML `<select>` e `<option>`.



### 3 SPECIFICA DELLE COMPONENTI

---

#### Template per TeachersController

Il `TeachersController` prevede diverse azioni raggiungibili attraverso una richiesta GET ed i relativi template XHTML sono i seguenti:

- `pre_activate.html.erb`: presenta un form per l'attivazione dello `user` del docente invitato
- `edit_capabilities.html.erb`: presenta un form per l'assegnazione di nuovi privilegi allo `user` detenuto dal docente utilizzando metodi per la presentazione automatica della lista dei privilegi come li tag XHTML `<input type="checkbox">`
- `edit_graduate_courses.html.erb`: presenta un form per l'assegnazione di nuovi corsi di laurea allo `user` detenuto dal docente utilizzando metodi per la presentazione automatica della lista dei corsi di laurea come i tag XHTML `<select>` e `<option>`

#### 3.3.3 Partial

Il sistema Sigeol prevede l'uso di partials per la visualizzazione delle informazioni che sono riutilizzate in più view. Ognuno di essi è caratterizzato dalla presenza di un `_` (underscore) prima del nome del file che sono della forma `_show_nomemodel.html.erb` e `_show_nomemodel_admin.html.erb` e dovranno essere contenuti nella rispettiva directory che contiene i template del controller associato a `nomemodel` (ad esempio `_show_teaching.html.erb` è contenuto all'interno della directory `app\view\teachings`. I partial del primo tipo visualizzano le informazioni dell'oggetto di tipo `nomemodel` per le azioni di pubblico accesso, mentre i secondi per le azioni di amministrazione con le relative funzioni.

Per i controller che presentano l'azione `administration` e necessitano di un menu di amministrazione è presente un partial `_menu_admin.html.erb` per la visualizzazione del suddetto menu che risiederà nella stessa directory che contiene i template del controller in questione.

I partials, infine, che vengono utilizzati dal layout sono contenuti nella directory `app\view\shared`, come ad esempio `_user_sidebar.html.erb`.

#### 3.4 Componente Helper

Il componente Helper si occupa di raccogliere le funzionalità di utilità necessarie ai componenti del pattern MVC. Ogni file che rappresenta un Helper non conterrà una classe, bensì un modulo, ovvero un insieme di metodi di pubblica utilità. Per ogni controller è previsto lo specifico Helper, anche se non è necessario che siano presenti dei metodi in quanto non tutti i controller (e le rispettive view) necessitano di funzioni ausiliarie.



## 3 SPECIFICA DELLE COMPONENTI

---

### 3.4.1 ApplicationHelper

L' `ApplicationHelper` contiene i metodi di utilità che non trovano una collocazione logica negli altri helper, nonché tutte le funzionalità ausiliare richieste da più classi delle diverse componenti. I metodi presenti all'interno dell' `ApplicationHelper` sono i seguenti:

- `first_upper(name)`: metodo che restituisce la stringa `name` a cui viene reso maiuscolo il primo carattere e minuscoli i rimanenti.
- `login_form`: metodo che renderizza il partial per la form per il login.
- `standard_sidebar`: metodo che renderizza il partial per il menu pubblico.
- `user_sidebar`: metodo che renderizza il partial per il menu privato.
- `link(user)`: metodo che restituisce l'insieme dei link disponibili per `user`.

### 3.4.2 BuildingsHelper

I metodi presenti all'interno del `BuildingsHelper` sono i seguenti:

- `menu_admin`: metodo che renderizza il partial per il menu di amministrazione per gli edifici.
- `show_building_admin(building)`: metodo che renderizza il partial per l'amministrazione del `building`.
- `show_building(building)`: metodo che renderizza il partial per la visualizzazione del `building`.

### 3.4.3 ClassroomsHelper

I metodi presenti all'interno del `ClassroomsHelper` sono i seguenti:

- `menu_admin`: metodo che renderizza il partial per il menu di amministrazione per le aule.
- `show_classroom_admin(classroom)`: metodo che renderizza il partial per l'amministrazione della `classroom`.
- `show_classroom(classroom)`: metodo che renderizza il partial per la visualizzazione della `classroom`.



## 3 SPECIFICA DELLE COMPONENTI

---

### 3.4.4 CurriculumHelper

I metodi presenti all'interno del **CurriculumHelper** sono i seguenti:

- **show\_curriculum\_admin(curriculum)**: metodo che renderizza il partial per l'amministrazione del curriculum.
- **show\_curriculum(curriculum)**: metodo che renderizza il partial per la visualizzazione del curriculum.

### 3.4.5 SessionsHelper

Non è presente nessun metodo all'interno di **SessionsHelper**.

### 3.4.6 GraduateCoursesHelper

I metodi presenti all'interno del **GraduateCoursesHelper** sono i seguenti:

- **menu\_admin**: metodo che renderizza il partial per il menu di amministrazione per i corsi di laurea
- **show\_graduate\_course\_admin(graduate\_course)**: metodo che renderizza il partial per l'amministrazione del **graduate\_course**.
- **show\_graduate\_course(graduate\_course)**: metodo che renderizza il partial per la visualizzazione del **graduate\_course**.

### 3.4.7 TeachersHelper

I metodi presenti all'interno del **TeachersHelper** sono i seguenti:

- **menu\_admin**: metodo che renderizza il partial per il menu di amministrazione per i docenti.
- **show\_teacher\_admin(teacher)**: metodo che renderizza il partial per l'amministrazione del **teacher**.
- **show\_teacher(teacher)**: metodo che renderizza il partial per la visualizzazione del **teacher**.

### 3.4.8 TeachingsHelper

I metodi presenti all'interno del **TeachingsHelper** sono i seguenti:

- **menu\_admin**: metodo che renderizza il partial per il menu di amministrazione per gli insegnamenti.
- **show\_teaching\_admin(teaching)**: metodo che renderizza il partial per l'amministrazione del **teaching**.
- **show\_teaching(teaching)**: metodo che renderizza il partial per la visualizzazione del **teaching**.



## 3 SPECIFICA DELLE COMPONENTI

---

### 3.4.9 TimetablesHelper

I metodi presenti all'interno del `TimetablesHelper` sono i seguenti:

- `menu_admin`: metodo che renderizza il partial per il menu di amministrazione per gli orari.
- `show_timetable_admin(timetable)`: metodo che renderizza il partial per l'amministrazione del `timetable`.
- `show_timetable(timetable)`: metodo che renderizza il partial per la visualizzazione del `timetable`.

### 3.4.10 UsersHelper

I metodi presenti all'interno del `UsersHelper` sono i seguenti:

- `show_not_active_users(user)`: metodo che renderizza il partial per la visualizzazione dello `user` non ancora attivo.



### 3 SPECIFICA DELLE COMPONENTI

---

#### Diario delle modifiche

DATA	VERSIONE	MODIFICA
<i>20 febbraio 2009</i>	0.1.0	Stesura dell'indice