Progetto SIGEOL

Definizione di prodotto

v0.2.0

Redazione:

5 marzo 2009

quixoft.sol@gmail.com

Verifica:	Verificatore
Approvazione:	Responsabile
Stato:	Preliminare — Formale
Uso:	Interno — Esterno
Distribuzione:	QuiXoft

Sommario

Descrizione dettagliata delle caratteristiche tecniche ed architetturali del prodotto Sigeol

Indice

1 Introduzione **1**
1.1 Scopo del documento . . . 1
1.2 Scopo del prodotto . . . 1
1.3 Glossario . . . 1
1.4 Riferimenti normativi . . . 1

2 Standard di progetto **1**
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

3 Specifica delle componenti **6**
3.1 Componente Model . . . 6
3.1.1 Introduzione . . . 6
3.2 Descrizione dei models . . . 10
3.2.1 AcademicOrganization . . . 10
3.2.2 Address . . . 11
3.2.3 Belong . . . 11
3.2.4 Building . . . 12
3.2.5 Capability . . . 12
3.2.6 Classroom . . . 13
3.2.7 Curriculum . . . 13
3.2.8 ExpiryDate . . . 14
3.2.9 GraduateCourse . . . 14
3.2.10 Period . . . 15
3.2.11 Teacher . . . 16
3.2.12 Teaching . . . 16
3.2.13 TimeTable . . . 17
3.2.14 TimetableEntry . . . 18
3.2.15 User . . . 19
3.2.16 DidacticOffice . . . 20
3.2.17 TemporalConstraint . . . 20
3.2.18 QuantityConstraint . . . 21
3.2.19 BooleanConstraint . . . 21
3.2.20 ConstraintsOwner . . . 22
3.3 Database . . . 22

3.4 Componente Controller 22
3.4.1 Azioni comuni a più controller 23
3.4.2 ApplicationController 25
3.4.3 GraduateCoursesController 25
3.4.4 CurriculumsController 26
3.4.5 UsersController 26
3.4.6 TeachersController 26
3.4.7 SessionsController 27
3.4.8 TeachingsController 27
3.4.9 BuildingsController 28
3.4.10 ClassroomsController 28
3.4.11 TimetablesController 29
3.5 Componente View 29
3.5.1 Layouts 30
3.5.2 Templates 31
3.5.3 Partials 33
3.6 Componente Helper 33
3.6.1 ApplicationHelper 33
3.6.2 BuildingsHelper 34
3.6.3 ClassroomsHelper 34
3.6.4 CurriculumsHelper 34
3.6.5 SessionsHelper 34
3.6.6 GraduateCoursesHelper 35
3.6.7 TeachersHelper 35
3.6.8 TeachingsHelper 35
3.6.9 TimetablesHelper 35
3.6.10 UsersHelper 36
3.7 Componente MiddleMan 36
3.7.1 SchedulerServlet 36
3.7.2 SchedulerJobListener 37
3.8 Componente Algorithm 37
3.8.1 AlgorithmJob 37
3.8.2 ItcSolver 37

4 Organizzazione delle directories **38**

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 complementa il presente ed ogni documento ufficiale.

2 Standard di progetto

2.1 Standard di progettazione architetturale

La definizione dell'intero sistema oggetto di studio è stata effetuata 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 architetturale 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

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. Verrano 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, nonchè 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'instanza

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`

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 parola, 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 utilizzarà i seguenti strumenti:

- 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/
- LaTeX
  http://www.latex-project.org/

3 Specifica delle componenti

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

3.1 Componente Model

3.1.1 Introduzione

Ogni classe appartente al model deriva direttamente da `ActiveRecord::Base`.

Quando una classe deriva da `Base`, il suo oggetto istanziato viene anche chiamato oggetto di tipo Active Record. Gli attributi non vengono specificati direttamente, ma vengono dedotti dalla tabella associata.

Per associare una classe Active Record ad una tabella è necessario seguire determinate convenzioni:

- la tabella deve essere chiamata con un nome plurale e tutto in minuscolo
- la classe deve essere chiamata con lo stesso nome della tabella, ma al singolare e con la prima lettera maiuscola

Un'istanza di una classe model corrisponde ad una riga della tabella associata. Se viene cancellato un oggetto model, verrà cancellata anche la riga corrispondente.

Da `base` si ereditano diversi metodi di pubblica utilità tra i quali:

- `save`: salva l'oggetto model. Se il model è nuovo viene creato un record nel database, altrimenti il record esistente viene aggiornato.
- `destroy`: elimina il record associato. Ovviamente dopo questa operazione l'oggetto model corrispondente alla riga cancellata, non potrà più modificare gli attributi.

Validazioni Grazie alla libreria Active Record è possibile validare gli attributi di un oggetto model. La validazione viene eseguita automaticamente prima di creare il record nel database. Se la convalida fallisce, l'istanza non verrà scritta nel db e verrà lasciata in memoria.

Inoltre è possibile specificare quando effettuare le validazioni (prima di aggiornare il record, prima di crearlo od ogni volta che lo si salva).

Si ha disposizione un set di helpers methods che implementano validazioni comuni a molti models; per esempio il controllo che il contenuto dell'attributo non sia nullo. Se il controllo fallisce, verrà aggiunto all'attributo un messaggio di errore. Nel model `User`, per controllare la presenza dell'attributo `mail`, si potrà utilizzare l'helper method `validates_presence_of:` in questo modo:

```
validates_presence_of:
:message=>"La mail non deve essere vuota"
:on  => :update
```

dove `:message` contiene il messaggio d'errore e `:on` specifica quando effettuare la validazione(`:create`, `:update`).

Se `:on` è omesso la validazione verrà fatta prima di ogni operazione di scrittura nel database.

Infine, per creare una validazione, è necessario aggiungere alle chiamate di `validate`, `validate_on_update`,`validate_on_create`, la lista dei metodi(passati come simboli) che si vuole eseguire.

- `validate`: la validazione viene eseguita prima di ogni operazione di scrittura nel database; quindi sia in fase di creazione che di modifica

- `validate_on_create`:la validazione viene eseguita solo quando un nuovo model dev'essere salvato nel database
- `validate_on_update`:la validazione viene eseguita solamente quando si modifica una riga già esistente

Ad esempio, nel model `ExpiryDate`, per controllare che la data immessa sia maggiore di quella di oggi si implementerà il
metodo `is_correct_date?` Il nome poi verrà aggiunto come simbolo alla chiamata di `validate`:
`validate :is_correct_date?`
Ora, un qualsiasi oggetto di tipo `ExpiryDate` è valido solo se contiene una data maggiore della data di oggi.

Associazioni Oltre alle validazioni, nelle classi del model sono definite le relazione tra le tabelle. `Active record` supporta tre tipi di associazioni: one-to-one, one-to-many, many-to-many. Per dichiarare una relazione, basta inserire opportunamente nelle classi del model: `has_one`, `has_many`, `belongs_to` e `has_and_belongs_to_many`. Anche qui è necessario seguire delle convenzioni:

- la dichiarazione di `belongs_to` deve essere aggiunta nel model, associato alla tabella che contiene la chiave esterna
- dopo `has_many` e `has_and_belongs_to_many` deve essere aggiunto(come simbolo) il nome della tabella
- dopo `belongs_to` e `has_one` deve essere aggiunto(come simbolo) il nome al singolare della tabella

L'opzione `:dependent` indica ad `Active record` come comportarsi con la child table in caso di cancellazione di una riga del parent table. Ad esempio nel model GraduateCourse la relazione con la tabella curriculums dovrà essere definita in questo modo:
`has_many :curriculums, :dependent => :destroy`
Tradotta significa che cancellando una riga di `graduate_courses`, verranno cancellate tutte le righe associate di `curriculums`.

Associazioni polimorfe Nel progetto *Sigeol* sono presenti due associazioni polimorfe.

Uno user ha uno tipo specifico; ad esempio può essere o un `Teacher` o un `DidacticOffice`
Per implementare quest'associazione, si deve aggiungere alla tabella `users` due attributi: `specified_id` e `specified_type`. Al model associato invece si dovrà specificare la relazione polimorfa; e lo si farà in questo modo:
`belongs_to :specified, :polymorphic => true`.

Nelle classi associate invece si inserirà questa definizione:
`has_one :user, :as => :specified`
L'opzione `:as=>specified`, specifica che la relazione tra `User` e il model è polimorfa, usando l'attributo `specified`.

Ad esempio, sia u un oggetto di tipo `User` e t un oggetto di tipo `Teacher`, per specificare lo user, semplicemente basta fare:
`u.specified=t`

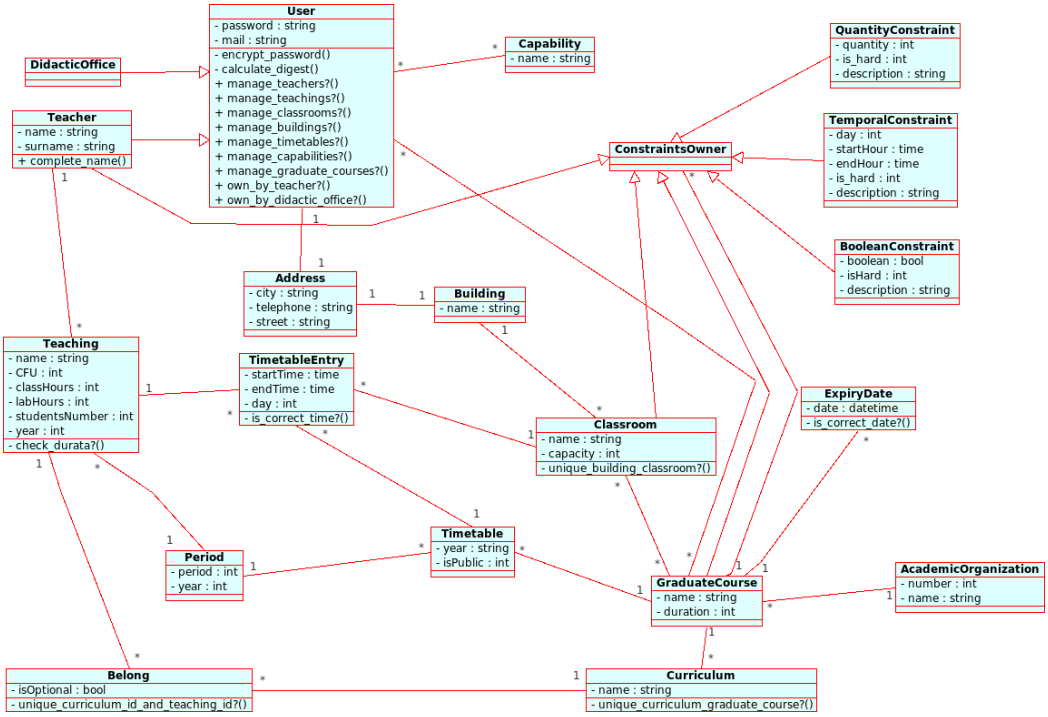
Il secondo tipo di associazione è chiamato *associazione polimorfa doppia* A parole semplice, il problema si può spiegare in questo modo: una corso di laurea o una classe, possono avere più vincoli di diverso tipo. A sua volta un vincolo può avere più tipi di proprietari: una classe, un insegnante o un corso di laurea. Insomma, un vincolo può avere più tipi di proprietari e un proprietario può avere più tipi di vincoli. Per implementare questa associazione si farà uso di un plugin chiamato :`has_many_polymorphs`.

Per il corretto uso del plugin, si dovrà aggiungere una nuova classe model che associerà tutti i tipi di vincoli con tutti i tipi di proprietari. Questo tipo di model è chiamato join model.

Funzioni di callback `ActiveRecord` mette a disposizione dei metodi che monitorano lo stato dell'oggetto. E' possibile quindi definire delle operazioni da eseguire prima o dopo l'alterazione dello stato dell'istanza.

Ad esempio, per eseguire determinate istruzioni prima di validare, basterà ridefinire la funzione di callback `before_validation`. Oltre a `before_validation` sono presenti altre funzioni come `before_create`, `before_destroy`, `after_create`.

Diagramma delle classi Di seguito è riportato il diagramma delle classi che illustra questa componente. Per aumentare la leggibilità è stata omessa la derivazione delle classi da `ActiveRecord::Base`.



3.2 Descrizione dei models

3.2.1 AcademicOrganization

Descrizione attributi

- name: contiene il nome del tipo di organizzazione accademica; ad esempio Trimestre.
- number: contiene il numero di periodi dell'organizzazione accademica

Validazioni

- `name`:
  nella tabella `academic_organizations` non deve essere presente una riga con lo stesso valore contenuto nell'attributo. La stringa non dovrà essere nulla e dovrà essere al massimo di 30 caratteri. Infine dovrà rispettare la seguente espressione regolare: `/^[a-zA-Zàòèéùì\s]*$/`
- `number`:
  nella tabella `academic_organizations` non deve essere presente una riga con lo stesso valore contenuto nell'attributo. Inoltre potrà assumere solo valori interi compresi tra 1 e 6

Funzioni di callback

before_validation Prima di effettuare le operazioni di validazione, il contenuto dell'attributo `name` verrà messo tutto in minuscolo, tranne la prima lettera che verrà messa in maiuscolo.

3.2.2 Address

Descrizione attributi

- street: contiene la via
- city: contiene il nome della città
- telephone: contiene un numero di telefono

Validazioni Le validazioni di `city` e di `street` sono identiche all'attributo `name` del model `AcademicOrganization`

- `telephone`: il numero di telefono dovrà essere al massimo di 13 caratteri e dovrà rispettare l'espressione regolare `/^[0-9]{2,4}-[0-9]{6,8}$/`.
  In questo modo il contenuto di `telephone` avrà un numero di cifre compreso tra due e quattro(prefisso), seguite dal simbolo - ed infine un secondo numero di cifre compreso tra sei e otto(numero di telefono). Non è necessario inserire e controllare prima del numero di telefono il prefisso internazionale, perchè si ipotizza che tutti i numeri inseriti siano italiani.

Funzioni di callback

before_save Prima di salvare l'oggetto nel database, il contenuto degli attributi `city` e `street` vengono messi in minuscolo, tranne la prima lettera che viene messa in maiuscolo.

3.2.3 Belong

Descrizione attributi

- `curriculum_id`: chiave esterna di curriculums
- `teaching_id`: chiave esterna di teachings
- `isOptional`: è un attributo booleano. Se settato a true, indica che l'insegnamento individuato dalla chiave esterna `teaching_id`, è opzionale rispetto al curriculum individuato della chiave esterna `curriculum_id`

Validazioni

- `curriculum_id`: il contenuto della chiave esterna `curriculum_id` non deve essere nullo
- `teaching_id`: il contenuto della chiave esterna `teaching_id` non deve essere nullo

validate_on_create

- `validate_on_create :unique_curriculum_id_and_teaching_id?`: E' possibile salvare nel database l'oggetto istanziato, solo se nessuna riga della tabella associata `belongs`, ha gli stessi valori di `teaching_id` e di `curriculum_id` dell'istanza.
  `unique_curriculum_id_and_teaching_id?` è un metodo privato.

Funzioni di callback

before_destroy Dopo aver eliminato l'oggetto, se nessun curriculum utilizza l'insegnamento(associato all'istanza appena cancellata), allora anch'esso verrà cancellato.

3.2.4 Building

Descrizione attributi

- `name`: contiene il nome del palazzo
- `address_id`: chiave esterna di addresses

Validazioni Le validazioni di `name` sono identiche all'attributo `name` del model `AcademicOrganization`

- `address_id`: il contenuto della chiave esterna non deve essere nullo

Funzioni di callback

before_validation Prima di validare l'oggetto, il contenuto di `name` viene messo in minuscolo, tranne la prima lettera che viene messa in maiuscolo.

3.2.5 Capability

Descrizione attributi

- `name`: contiene il nome del servizio, disponibile per gli utenti del sistema

Validazioni Le validazioni di `name` sono identiche all'attributo `name` del model `AcademicOrganization`

3.2.6 Classroom

Descrizione attributi

- `name`: contiene il nome della classe
- `capacity`: contiene la capienza massima della classe
- `building_id`: chiave esterna di `buildings`

validazioni Le validazioni di `name` sono identiche all'attributo `name` del model `AcademicOrganization` senza però il controllo dell'unicità

- `capacity`: il contenuto deve essere un intero compreso tra 0 e 1000. Sarà possibile lasciare questo attributo nullo.
- `building_id`: il contenuto della chiave esterna non deve essere vuoto

validate

- `validate :unique_building_classroom?`: per salvare l'istanza nel database, è necessario che l'oggetto di tipo `Building`, con chiave primaria uguale a `building_id`, non abbia associato un oggetto di tipo `Classroom` con `name` uguale a quello definito nell'istanza.

  `unique_building_classroom?` è un metodo privato

Funzioni di callback

before_validation Prima di effettuare le operazioni di validazione, il contenuto dell'attributo `name` viene messo tutto in minuscolo, tranne la prima lettera che viene messa in maiuscolo.

3.2.7 Curriculum

Descrizione Attributi

- `name`: contiene il nome del curriculum
- `graduate_course_id`:
  contiene la chiave esterna di `graduate_courses`

Validazioni Le validazioni di `name` sono identiche all'attributo `name` del model `AcademicOrganization`, senza però il controllo dell'unicità

- `graduate_course_id`: la chiave esterna di `graduate_courses` non deve essere nulla

validate

- `validate :unique_curriculum_graduate_course?`:
  per salvare l'oggetto nel database, è necessario che il graduate course, con id uguale a `graduate_course_id`, non abbia associato un curriculum con il nome uguale a quello definito nell'oggetto

  `unique_curriculum_graduate_course?` è un metodo privato.

Funzioni di callback

before_validation Prima di effettuare le operazioni di validazione, il contenuto dell'attributo `name` viene messo tutto in minuscolo, tranne la prima lettera che viene messa in maiuscolo.

3.2.8 ExpiryDate

Descrizione attributi

- `date`: successivamente a questa data, il docente non potrà più inserire le proprie indisponibilità

Validazioni

- `date` : il contenuto di `date` non deve essere vuoto
- `graduate_course_id` : il contenuto della chiave esterna non deve essere vuoto

validate

- `validate :is_correct_date?`: la data inserita deve essere maggiore della data di oggi. `is_correct_date?` è un metodo privato

3.2.9 GraduateCourse

Descrizione Attributi

- `name`: contiene il nome del corso di laurea
- `duration`: indica la durata, in anni, del corso di laurea

validazioni Le validazioni di `name` sono identiche all'attributo `name` del model `AcademicOrganization` tranne per il numero di caratteri che da 30 son passati a 50.

- `:duration`: deve essere un intero compreso tra 1 e 6
- `academic_organization_id`:
  la chiave esterna non deve essere nulla. Quindi un oggetto di tipo `GraduateCourse` per essere valido, deve avere associato un oggetto di tipo `AcademicOrganization` anch'esso valido.

Funzioni di callback

before_validation Prima di effettuare le operazioni di validazione, il contenuto dell'attributo `name` viene messo tutto in minuscolo, tranne la prima lettera che viene messa in maiuscolo.

3.2.10 Period

Descrizione attributi

- `subperiod`: individua un determinato periodo dell'anno accademico. Ad esempio in un corso di laurea con un'organizzazione a trimestri, il subperiod 1 significa primo trimestre
- `year`: individua l'anno d'appartenenza di un corso d'insegnamento. Ad esempio Analisi Matematica è un corso d'insegnamento del primo anno accademico.

La coppia `subperiod` ed `year` individuano in che periodo e in che anno, viene svolto un determinato insegnamento.

validazioni

- `subperiod`: dovrà contenere un valore intero compreso tra 1 e 4
- `year`: dovrà contenere un valore intero compreso tra 1 e 6

validate_on_create

- `validate_con_create :unique_subperiod_year?`: l'oggetto è valido solo se nella tabella `periods` non è presente nessuna riga con gli stessi valori di `period` e di `year` dell'istanza. Se non è valido, l'oggetto non verrà salvato nel database. `unique_subperiod_year?` è un metodo privato

3.2.11 Teacher

Descrizione attributi

- `name`: contiene il nome del docente
- `surname`: contiene il cognome del docente

Validazioni Le validazioni di `name` e di `surname` sono identiche all'attributo `name` del model `AcademicOrganization`. Non è presente però la validazione di unicità. In questo modo sono ammessi i casi di omonimia.

Funzioni di callback

before_validation Prima di effettuare le operazioni di validazione, il contenuto dell'attributo `name` e `surname` vengono messi in minuscolo, tranne la prima lettera che viene messa in maiuscolo.

before_destroy Prima di eliminare l'oggetto d'istanza(e la sua riga associata), elimina il suo user associato presente nella tabella `users`.

3.2.12 Teaching

Descrizione attributi

- `name`: contiene il nome dell'insegnamento
- `CFU`: contiene il numero di crediti formativi dell'insegnamento
- `classHours`: contiene il numero di ore settimanali di lezione in aula
- `labHours`: contiene il numero di ore settimanali di lezione in laboratorio
- `studentsNumber`: contiene la stima del numero di studenti che frequenteranno l'insegnamento

Validazioni Le validazioni di `name` sono identiche all'attributo `name` del model `AcademicOrganization`. In più però ci sarà la possibilità di inserire nella stringa anche valori numerici.

- `CFU`: per essere valido deve contenere un valore intero compreso tra 1 e 20
- `labHours, classHours`: gli attributi per essere validi devono contenere un valore intero compreso tra 1 e 50

- `studentsNumber`:per essere valido deve contenere un valore intero compreso tra 1 e 1000

Tutti questi attributi possono essere nulli. Questo perchè non si può sempre sapere all'atto della creazione dell'insegnamento, il numero di ore di lezione o la stima del numero di studenti che frequenteranno il corso d'insegnamento. Può anche capitare che non sia chiaro nemmeno il numero di crediti formativi.

- `period_id`: la chiave esterna non deve essere nulla. Questo significa che l'oggetto istanziato deve essere associato ad un periodo valido.

validate

- `validate :check_durata?`:
  Un insegnamento può appartenere a più curriculum; due diversi curriculum possono avere un insegnamento in comune, ma appartenere a due differenti corsi di laurea con diversa organizzazione accademica. Per questo motivo un `Teaching` è valido solo se l'oggetto `Period` associato, ha un valore di `year` ,minore o uguale, al minimo valore dell'attributo `duration` scelto fra tutti i `GraduateCourse`, che hanno almeno un `Curriculum` associato a quel `Teaching`. Ovviamente dovrà essere confrontato anche l'attributo `subperiod`(appartenente al model `Period` associato) con l'attributo `number`. `check_durata?` è un metodo privato.

Funzioni di callback

before_validation Prima di effettuare le operazioni di validazione, il contenuto dell'attributo `name` viene messo tutto in minuscolo, tranne la prima lettera che viene messa in maiuscolo.

3.2.13 TimeTable

Descrizione attributi

- `year`: contiene l'anno accademico. Ad esempio 2008-09
- `period_id`: chiave esterna di `periods`
- `graduate_course_id`: chiave esterna di `graduate_courses`

Validazioni

Attributi year,period_id,graduate_course_id

- `year, period_id, graduate_course_id`:
  il contenuto degli attributi devono essere non nulli.
- `year` : deve rispettare l'espressione regolare
  `/^[0-9]{4,4}-[0-9]{2,2}$/`;
  dovrà avere nei primi quattro caratteri solo cifre, il quinto dovrà essere riservato al carattere - ed infine negli ultimi due caratteri dovranno esserci altri due numeri

3.2.14 TimetableEntry

Descrizione Attributi

- `startTime`: contiene un oggetto di tipo Time e indica l'ora di inizio
- `endTime`:contiene un oggetto di tipo Time e indica l'ora di fine
- `day`: contiene un intero che indica un determinato giorno della settimana. Ad esempio 1 significa Lunedì.
- `teaching_id`: chiave esterna di `teachings`
- `classroom_id`: chiave esterna di `classrooms`
- `timetable_id`: chiave esterna di `timetables`

`startTime`,`endTime` e `day`
individuano quando verrà svolta la lezione del corso d'insegnamento, individuato dalla chiave esterna `teaching_id`. `classroom_id` individuerà la classe che verrà utilizzata.

Validazioni

- `startTime, endTime, day`
  `timetable_id, classroom_id`:
  Nessun attributo dell'oggetto devono essere nulli.
- `:day`: deve essere un intero compreso tra 1 e 6

validate

- `validate :is_correct_time?`: L'oggetto d'istanza è valido solo se l'attributo `startTime` ha un oggetto di tipo tempo minore dell'oggetto contenuto in `endTime`. `:is_correct_time?` è un metodo privato

3.2.15 User

Descrizione attributi

- mail: contiene una stringa che rappresenta la mail dello user
- password: contiene la password(crittografata tramite SHA1) dello user
- random: contiene un valore casuale, che verrà utilizzato per calcolare il digest del contenuto dell'attributo `mail`
- digest: Il contenuto dell'attributo `mail` concatenato al valore di `random`, viene crittografato tramite SHA1.

  Il risultato dell'operazione verrà salvato in questo attributo.

Validazioni

- `password, mail, random, digest`:
  tutti gli attributi devono essere non nulli. Per la `password`,il controllo della presenza di contenuto non nullo, verrà fatto solo nelle operazioni di update e non nelle operazioni di create. Questo perchè un utente verrà inizialmente inserito nel sistema e successivamente verrà invitato tramite mail a completare la registrazione inserendo la password.
- `password`: dovrà contenere come minimo 6 caratteri e dovrà rispettare l'espressione regolare: `/^[a-zA-Z0-9\.]*$/`.
- `mail`: nella tabella `users` non deve essere presente una riga con lo stesso valore dell'attributo `mail` dell'oggetto istanziato.

  Non possono esistere due user con la stessa password. Inoltre il contenuto deve rispettare l'espressione regolare:
  `^([a-zA-Z0-9_\.\-\+]){4,20}\@`
  `(([a-zA-Z0-9\-])+\.)+([a-zA-Z0-9]{2,4})$/`.
  Si accettano tutti i tipi di mail con un numero di caratteri compreso tra 4 e 20. E' possibile utilizzare lettere maiuscole, minuscole, numeri e i caratteri - e +. Sono valide anche le mail che hanno più domini, come ad esempio prova@math.unipd.it

Funzioni di callback

- `before_save :encrypt_password`: prima di salvare l'oggetto di tipo `User` nel database, la password viene crittografata attraverso l'algoritmo SHA1. `encrypt_password` è un metodo privato.
- `before_validation :calculate_digest`:
  prima di validare l'oggetto, viene chiamato il metodo

`calculate_digest`. Questo metodo calcola il digest, tramite l'algoritmo SHA1, della stringa contenuta in `mail`, concatenata ad un numero casuale(contenuto nell'attributo `random`). Il risultato infine viene salvato nell'attributo `digest`. `calculate_digest` è un metodo privato.

Metodi pubblici Nel model `User` sono presenti diversi metodi pubblici che ritornano true se l'oggetto d'istanza è associato a un particolare servizio. Se è associato significa che quel particolare utente è abilitato ad utilizzare quel servizio. Ogni metodo inizierà con la radice `manage_`:

- `def manage_teachers?`: se torna true, l'utente può modificare le informazioni dei docenti associati hai suoi stessi corsi di laurea
- `manage_teachings?`: se torna true, l'utente può modificare le informazioni dei corsi d'insegnamento associati hai suoi stessi corsi di laurea
- `manage_classrooms?`: se torna true, l'utente può modificare le informazioni delle classi associate ai suoi stessi corsi di laurea

Non è necessario riportare tutti i metodi, perchè il loro scopo è facilmente intuibile dalla seconda parte del nome; tutti i metodi sono riportati nel diagramma delle classi mostrato all'inizio della sottosezione intitolata Introduzione.

I metodi `own_by_teacher?` e `own_by_didactic_office?`, indicano se l'oggetto appartiene ha un `Teacher` o ha un `DidacticOffice`. In altre parole specifica se lo user è un docente o una segreteria didattica.

3.2.16 DidacticOffice

Funzioni di callback

- `def before_destroy`: Se un oggetto di tipo `DidacticOffice` viene distrutto, deve essere cancellato anche l'oggetto `User` associato

3.2.17 TemporalConstraint

Descrizione attributi

- `startHour`:contiene un oggetto di tipo `Time` ed indica l'ora di inizio dell'indisponibilità
- `endHour`: contiene un oggetto di tipo `Time` ed indica l'ora di fine dell'indisponibilità
- `day`: contiene un intero che individua il giorno dell'indisponibilità

Validazioni

- `startHour,endHour,day`: tutti gli attributi devono essere non nulli
- `day`: deve contenere un valore intero compreso tra 1 e 5

validate

- `validate :is_correct_time?`: l'oggetto d'istanza è valido solo se `starthour` è minore di `endHour`. `is_correct_time?` è un metodo privato.

3.2.18 QuantityConstraint

Descrizione attributi

- `quantity`: contiene un valore intero

Validazioni

- `quantity`: il contenuto di `quantity` deve essere presente
- `validates_numericality_of :quantity`: il contenuto di `quantity` deve essere un intero compreso tra 1 e 1000

3.2.19 BooleanConstraint

Descrizione attributi

- `bool`: contiene un valore booleano

Validazioni

- `bool`: il contenuto di `bool` deve essere presente

I model
`BooleanConstraint`,`TemporalConstraint` e `QuantityConstraint`
hanno in comune i seguenti attributi:

- `description`: contiene una descrizione del vincolo
- `isHard`: contiene un valore di tipo intero. Se questo è uguale a 0, significa che il vincolo tassativamente dovrà essere soddisfatto. Per valori compresi tra 1 e 10, il vincolo è una preferenza con importanza inversamente proporzionale al numero(più grande è il numero meno importante è la preferenza)

3.2.20 ConstraintsOwner

All'interno di questa classe, si definiranno le relazioni tra i vincoli e i relativi proprietari. Un vincolo può avere come proprietario: un corso di laurea(`GraduateCourse`), una classe(`Classroom`) e un docente(`Teacher`).

Un proprietario, può avere tre tipi di vincoli, booleano, temporale e di quantità. Grazie a questo model join ed al plugin `has_many_polymorphs`, sarà possibile associare in modo semplice e veloce un proprietario con un qualsiasi tipo di vincolo

3.3 Database

L'integrità referenziale è un concetto molto importante da non dimenticare durante la progettazione di un database. Da poco MySql ha implementato un supporto per la chiavi esterne attraverso il nuovo table engine INNODB.

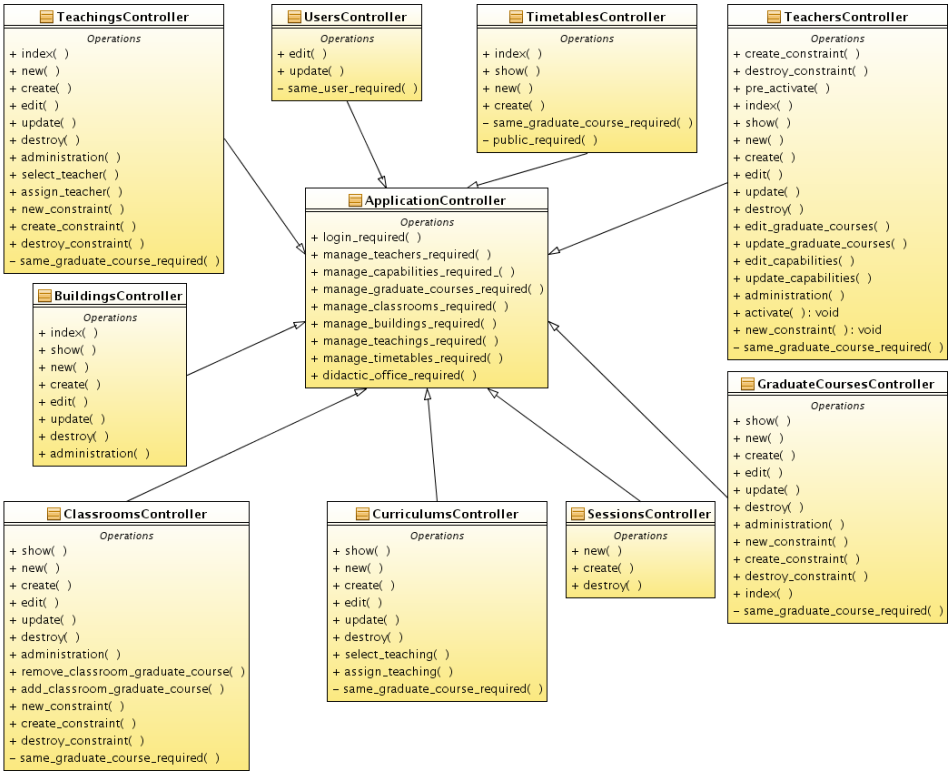
La sintassi per aggiungere un vincolo d'integrità alle chiave esterne è il seguente:

```
add constraint constraint_name
foreign key (from_column) references to_table(id)
```

Ovviamente ogni tabella del database dovrà utilizzare come engine, INNODB, altrimenti la clausula `foreign key` verrà ignorata.

3.4 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 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`.



3.4.1 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.

- `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_contraint/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.

3.4.2 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 Sigeol 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.

3.4.3 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`.

3.4.4 CurriculumsController

Il controller `CurriculumsController` permette all'utente con gli opportuni privilegi di creare, modificare ed eliminare le informazioni relative ai curricula dei propri corsi di laurea, nonchè di aggiungere o rimuove insegnamenti da quest'ultimi attreverso i metodi `select_teaching`, raggiungibile medianta 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`.

3.4.5 UsersController

Il controller `UsersController` 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.

3.4.6 TeachersController

Il controller `TeachersController` permette all'utente con gli opportuni privilegi di creare, modificare ed eliminare i docenti dei propri corsi da laurea, nonchè di attribuire ad essi nuovi privilegi e corsi di laurea. Attraverso l'azione `new` viene rischiesto 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 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 possibilile 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`.

3.4.7 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.

3.4.8 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, nonchè di aggiungere o rimuove il docente

che tiene l'insegnamento in questione attraverso le azioni `select_teacher`, raggiungibile medianta 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`.

3.4.9 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.

3.4.10 ClassroomsController

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

3.4.11 TimetablesController

Il controller `TimetablesController` permette all'utente con gli opportuni privilegi di creare ed eliminare gli orari. Implementa alcune azioni aggiuntive oltre a quelle di base:

- `schedule`: metodo usato nella creazione di una nuova istanza di schedulazione. Invia al Middle Man i dati necessari ad impostare una nuova schedulazione.
- `notify`: metodo richiamato dal Middle Man per notificare all'applicazione l'attivazione di un'evento precedentemente schedulato. Inoltre richiama il metodo `start` per avviare la generazione dell'orario
- `done`: metodo richiamato dal Middle Man utilizzato per segnalare la fine del calcolo
- `start`: metodo che segnala al Middle Man di avviare la generazione dell'orario

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 intente visualizzare sia stato dichiarato pubblico da un utente con gli opportuni privilegi.

3.5 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:

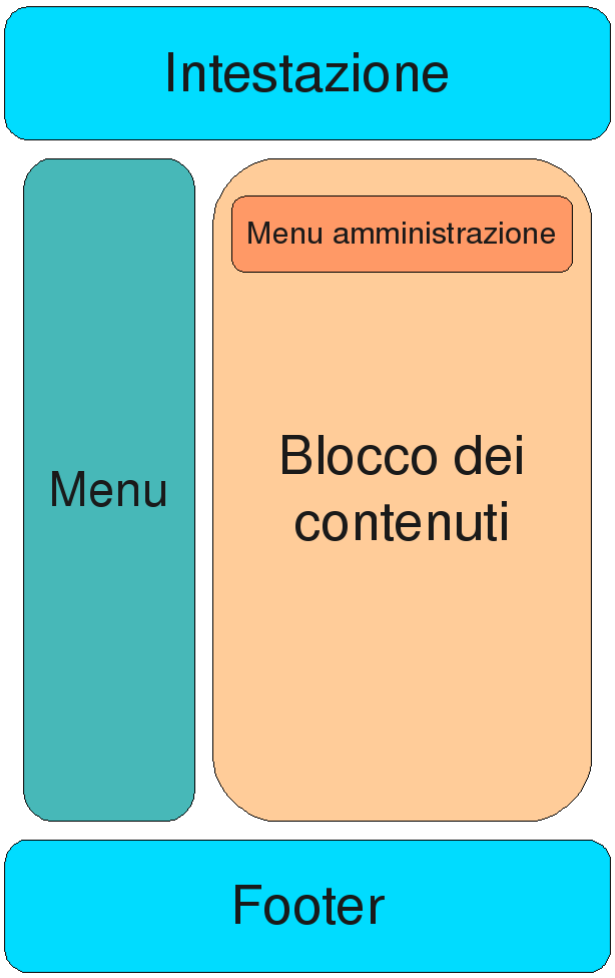
- Layouts: impostano l'impaginazione e la struttura dei template.

- 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.

3.5.1 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 immagina esplicativa della struttura del layout:



3.5.2 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 relativa 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`, nonchè 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 `CurriculumsController` 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>`.

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` dtenuto dal docente utilizzando metodi per la presentazione automatica della lista dei corsi di laurea come i tag XHTML `<select>` e `<option>`

3.5.3 Partials

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\views\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\views\shared`, come ad esempio `_user_sidebar.html.erb`.

3.6 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.

3.6.1 ApplicationHelper

L' `ApplicationHelper` contiene i metodi di utilità che non trovano una collocazione logica negli altri helper, nonchè 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 maiscolo 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`.

3.6.2 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`.

3.6.3 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`.

3.6.4 CurriculumsHelper

I metodi presenti all'interno del `CurriculumsHelper` 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`.

3.6.5 SessionsHelper

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

3.6.6 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`.

3.6.7 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`.

3.6.8 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`.

3.6.9 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`.

3.6.10 UsersHelper

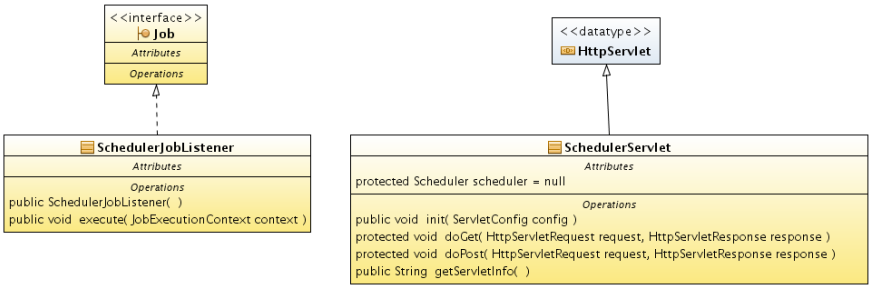
I metodi presenti all'interno del `UsersHelper` sono i seguenti:

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

3.7 Componente MiddleMan

Il componente `MiddleMan` viene utilizzato per effettuare esecuzioni (istantanee e/o schedulate) di calcolo dell'orario.
Esso implementa le seguenti operazioni:

- ricezione delle richieste, da parte dell'applicazione, di aggiunta di nuove date per la generazione dell'orario relativo ad uno specifico corso di laurea
- assegnazione alla componente `Algorithm` della generazione dell'orario relativo ad uno specifico corso di laurea
- segnalazione all'applicazione della fine del calcolo



3.7.1 SchedulerServlet

Servlet che si occupa di ricevere le richieste (di tipo HTTP POST e GET) dall'applicazione:

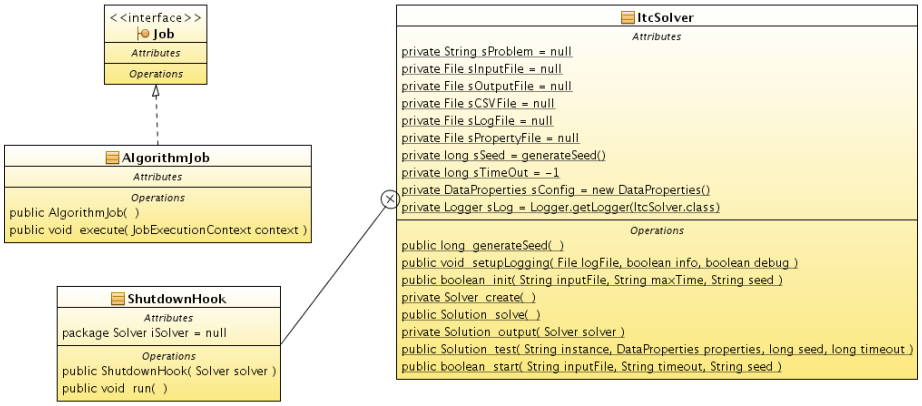
POST Parametri: course, date
crea una nuovo trigger che si attiverà alla data (date) per il corso di laurea (course) specificato

GET Parametri: course, inputfile, timeout
viene inizializzata ed eseguita la componente `Algorithm` con i parametri ricevuti

3.7.2 SchedulerJobListener

Classe che viene richiamata all'attivazione di un evento precedentemente schedulato (nel nostro caso la generazione dell'orario di un determinato corso di laurea ad una certa data). Si occupa di segnalare all'applicazione (attraverso il metodo **execute**) l'attivazione del calcolo dell'orario del corso di laurea specificato

3.8 Componente Algorithm



3.8.1 AlgorithmJob

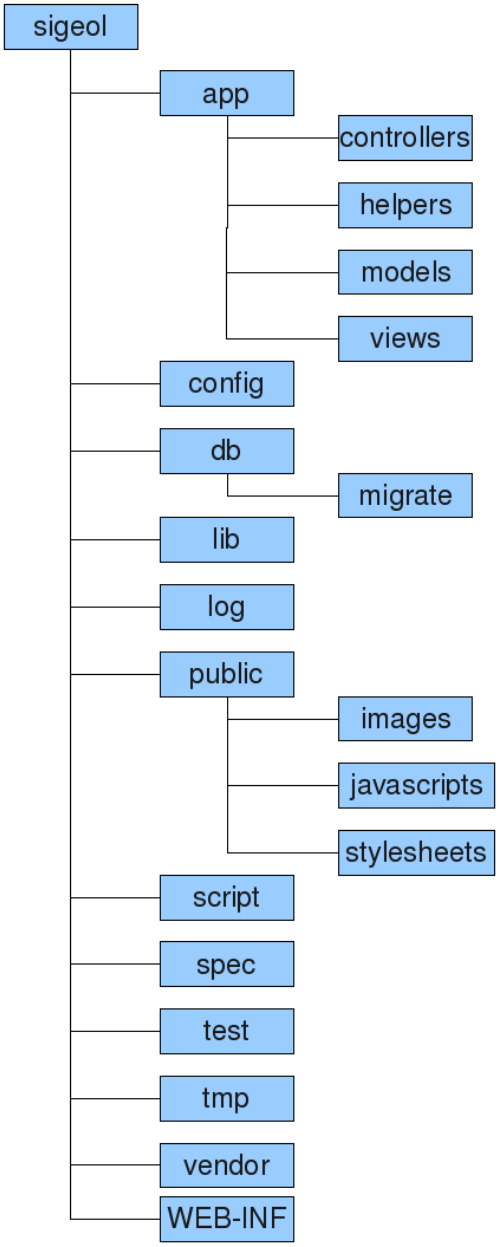
Ogni singola istanza della classe `AlgorithmJob` esegue il calcolo dell'orario relativo ad un determinato corso di laurea.

3.8.2 ItcSolver

Classe che si occupa di leggere il file di input (contenente tutte le informazioni richieste relative al calcolo e generazione dell'orario) ed eseguire effettivamente l'algoritmo.

4 Organizzazione delle directories

Per facilitare la comprensione dell'organizzazione del sistema Sigeol, viene presentata in questa sezione la struttura delle directories. Per un dettaglio maggiore dell'organizzazione si prega di far riferimento al sito ufficiale di Ruby on Rails. Nella figura che segue viene illustrata la gerarchia



A differenza delle consuete applicazioni sviluppate tramite il framework Rails,

il sistema Sigeol presenta la cartella `WEB-INF` che contiene tutte le classi e librerie utilizzate dalla servlet e il suo file di configurazione web.xml.
Per ulteriori informazioni sulla tecnologia servlet visitare il sito http://java.sun.com/products/servlet/
.

Diario delle modifiche

Data	Versione	Modifica
4 marzo 2009	0.2.0	Stesura parte sul componente Middle-Man
20 febbraio 2009	0.1.0	Stesura dell'indice