Progetto SIGEOL

Definizione di prodotto

v1.0.1

Redazione: Beggiato Andrea, Grosselle Alessandro, Barbiero Mattia

6 marzo 2009

quixoft.sol@gmail.com

Verifica:	Alberti Andrea
Approvazione:	Freo Matteo
Stato:	Formale
Uso:	Esterno
Distribuzione:	QuiXoft
	Rossi Francesca
	Vardanega Tullio
	Conte Renato

Sommario

Descrizione dettagliata delle caratteristiche tecniche ed architetturali del prodotto Sigeol

Indice

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` ed ogni istanza di tale classe viene anche chiamata oggetto di tipo Active Record, dove gli attributi non vengono specificati direttamente, ma dedotti dalla tabella associata.

Un'istanza di una classe model corrisponde ad una riga della tabella associata, quindi se viene cancellato verrà eliminata anche la riga corrispondente.

Da `ActiveRecord: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 effettuata automaticamente prima della creazione del record o del suo aggiornamento nel database. Se la convalida fallisce, la tupla del database resterà invariata mentre all'istanza verrà aggiunto l'errore di validazione verificatosi. Inoltre è possibile specificare differenti validazioni per nuovi record, per aggiornamenti di tuple esistenti o per ogni operazione di salvataggio.

`Active Record` infine, mette ha disposizione un set di metodi ausiliari che implementano validazioni comuni a molti models come ad esempio il controllo della presenza di un attributo. Se il controllo fallisce, verrà aggiunto all'oggetto un messaggio di errore per lo specifico attributo. Ad esempio 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 :mail,
      :message=>"La mail non deve essere vuota",
      :on  => :save
```

dove `:message` contiene il messaggio d'errore e `:on` specifica in che fase effettuare la validazione(`:create`, `:update` o `:save`); se `:on` è omesso questi assumera il valore di default `save`.

Per creare, invece, una validazione non presente in quelle fornite dalle librerie di Rails è necessario aggiungere alle chiamate di `validate`, `validate_on_update`,`validate_on_create`, la lista dei metodi che si vuole aggiungere, passati come simboli.

- `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 solamente alla creazione di una nuova tupla

- `validate_on_update`: la validazione viene eseguita solamente alla modifica di un record 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 quindi verrà aggiunto come simbolo alla chiamata di `validate` nel seguente modo:

```
validate :is_correct_date?
```

Associazioni

Oltre alle validazioni, nel model vengono definite le relazione tra le tabelle, e quindi delle stesse classi. `Active record` supporta tre tipi di associazioni: one-to-one, one-to-many, many-to-many. Per dichiarare una relazione, è necessario rchiamare opportunamente i metodi `has_one`, `has_many`, `belongs_to` e `has_and_belongs_to_many` nelle classi del model. Anche qui è necessario seguire delle convenizioni:

- 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

E' possibile modificare le associazioni aggiungendo delle opzioni; in particolare nel progetto Sigeol si è utilizzata l'opzione `:dependent` che indica ad `Active record` come comportarsi quando si elimina una riga di una tabella che è associata ad un altra tramite chiave esterna (rapporto tra parent table e child table) Ad esempio nel model `GraduateCourse` la relazione con la classe `Curriculum` dovrà essere definita come `has_many :curriculums`, `:dependent => :destroy`, per indicare che eliminando un record di `graduate_courses`, verranno cancellate tutte le tuple di `curriculums` associate ad esso.

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`. In `User` si specificherà la relazione polimorfa:

`belongs_to :specified, :polymorphic => true`.
Nei model che rappresentano i tipi che possono possedere uno `User`, si inserirà la seguente associazione:
`has_one :user, :as => :specified`
L'opzione `:as=>specified`, specifica che la relazione tra `User` e il model è polimorfa, usando gli attributi `specified_id` e `specified_type`.

Il secondo tipo di associazione è chiamato *associazione polimorfa doppia*; un corso di laurea o un'aula, possono avere più vincoli di diverso tipo tra temporali, di quantità o booleani. A sua volta un vincolo può avere differenti proprietari: un'aula, un docente o un corso di laurea. 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 prende il nome di join model.

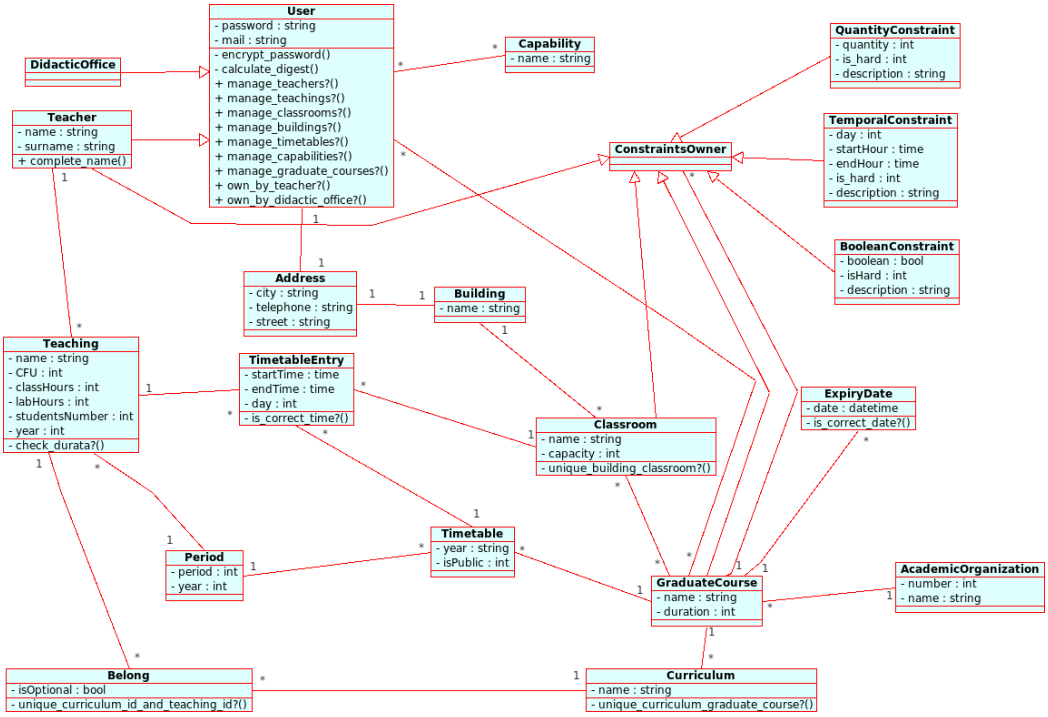
Funzioni di callback

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

Ad esempio, per eseguire determinate istruzioni prima della validazione dell'oggetto, basterà ridefinire la funzione di callback `before_validation`. Oltre a `before_validation` sono presenti altre funzioni tra cui `before_create`, `before_destroy`. `after_create`.

3.2 Descrizione dei models

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`.



3.2.1 AcademicOrganization

Descrizione degli 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 tupla con lo stesso valore dell'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 tupla 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, verrà reso maiscuolo il primo carattere del contenuto dell'attributo `name` e minuscoli i rimanenti.

3.2.2 Address

Descrizione degli 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, verrà reso maiuscolo il primo carattere del contenuto degli attributi `city` e `street` e minuscoli i rimanenti.

3.2.3 Belong

Descrizione degli attributi

- `curriculum_id`: chiave esterna di curriculums
- `teaching_id`: chiave esterna di teachings
- `isOptional`: attributo booleano che indica se 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 utilizzato per la validazione precedente.

3.2.4 Building

Descrizione degli attributi

- `name`: contiene il nome dell'edificio
- `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 effettuare le operazioni di validazione, verrà reso maiscuolo il primo carattere del contenuto dell'attributo `name` e minuscoli i rimanenti.

3.2.5 Capability

Descrizione degli attributi

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

Validazioni

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

3.2.6 Classroom

Descrizione degli attributi

- `name`: contiene il nome dell' aula
- `capacity`: contiene la capienza massima dell'aula
- `building_id`: chiave esterna di `buildings`

Validazioni

Le validazioni di `name` sono identiche all'attributo `name` del model `AcademicOrganization` senza 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 utilizzato nella validazione precedente

Funzioni di callback

before_validation

Prima di effettuare le operazioni di validazione, verrà reso maiscuolo il primo carattere del contenuto dell'attributo `name` e minuscoli i rimanenti.

3.2.7 Curriculum

Descrizione degli 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 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 utilizzato nella validazione precedente.

Funzioni di callback

before_validation

Prima di effettuare le operazioni di validazione, verrà reso maiscuolo il primo carattere del contenuto dell'attributo `name` e minuscoli i rimanenti.

3.2.8 ExpiryDate

Descrizione degli 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 utilizzato nella validazione precedente

3.2.9 GraduateCourse

Descrizione degli 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` eccezion fatta per il numero di caratteri di 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, verrà reso maiscuolo il primo carattere del contenuto dell'attributo `name` e minuscoli i rimanenti.

3.2.10 Period

Descrizione degli attributi

- `subperiod`: individua un determinato periodo dell'anno accademico. Ad esempio in un corso di laurea con un'organizazzione a trimestri, `subperiod` con valore 1 identifica il primo trimestre
- `year`: individua l'anno, insteso come primo, secondo, terzo, ecc.

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.
- `unique_subperiod_year?` è un metodo privato utilizzato nella validazione precedente

3.2.11 Teacher

Descrizione degli 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, verrà reso maiscuolo il primo carattere del contenuto degli attributi `name` e `surname` e minuscoli i rimanenti.

before_destroy

Prima di eliminare l'oggetto verrà eliminato lo user ad esso associato.

3.2.12 Teaching

Descrizione degli 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`. Inoltre è permesso l'inserimento nella stringa di 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 0 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.

- `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 utilizzato nella precedente validazione.

Funzioni di callback

before_validation

Prima di effettuare le operazioni di validazione, verrà reso maiscuolo il primo carattere del contenuto dell'attributo `name`.

3.2.13 TimeTable

Descrizione degli 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 degli 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.
- `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 l'aula che verrà utilizzata.

Validazioni

- `startTime, endTime, day  timetable_id, classroom_id`: Nessun attributo dell'oggetto deve essere nullo.
- `: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 usato le validazione precedente

3.2.15 User

Descrizione degli 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.
- `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 `password` il controllo della presenza di contenuto non nullo, verrà eseguito solo nelle operazioni di aggiornamento e non nelle operazioni di creazione. 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\.]*$/`.
- `mail`: nella tabella `users` non deve essere presente una tupla con lo stesso valore dell'attributo `mail` dell'oggetto istanziato. Non possono esistere due user con lo stesso indirizzo e-mail. Inoltre il contenuto deve rispettare l'espressione regolare:
  `^([a-zA-Z0-9_\.\-\+]){4,20}\@`
  `(([a-zA-Z0-9\-])+\.)+([a-zA-Z0-9]{2,4})$/`.
  Si accettano tutti i tipi di indirizzo e-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. utilizzato nella precedente funzione.

- `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 utilizzato nella precedente funzione.

Metodi pubblici Nel model `User` sono presenti diversi metodi pubblici che ritornano true se l'oggetto possiede uno specifico privilegio. Ogni metodo inizierà con la radice `manage_`:

- `manage_teachers?`: ritorna true se l'utente possiede il privilegio di amministrazione dei docenti
- `manage_teachings?`: ritorna true se l'utente possiede il privilegio di amministrazione degli insegnamenti
- `manage_classrooms?`: ritorna true se l'utente possiede il privilegio di amministrazione delle aule

Non è necessario riportare tutti i metodi, dato che il loro scopo è facilmente intuibile dalla denominazione;

I metodi `own_by_teacher?` e `own_by_didactic_office?`, indicano se l'oggetto appartiene ad un `Teacher` o ad un `DidacticOffice`.

3.2.16 DidacticOffice

Funzioni di callback

- `before_destroy`: prima dell'eliminazione di un oggetto di tipo `DidacticOffice`, viene eliminato l'oggetto di tipo `User` ad esso associato.

3.2.17 TemporalConstraint

Descrizione degli 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 6

validate

- `validate :is_correct_time?`: l'oggetto è valido solo se `starthour` è minore di `endHour`.
- `is_correct_time?` è un metodo privato utilizzato nella precedente validazione.

3.2.18 QuantityConstraint

Descrizione degli attributi

- `quantity`: contiene un valore intero rappresentante la quantità del vincolo o preferenza

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 rappresentante la presenza o meno del vincolo o preferenza

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 rappresenta un vincolo, mentre un valore maggiore o uguale ad 1 indica una preferenza con importanza inversamente proporzionale al numero.

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`) o 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.

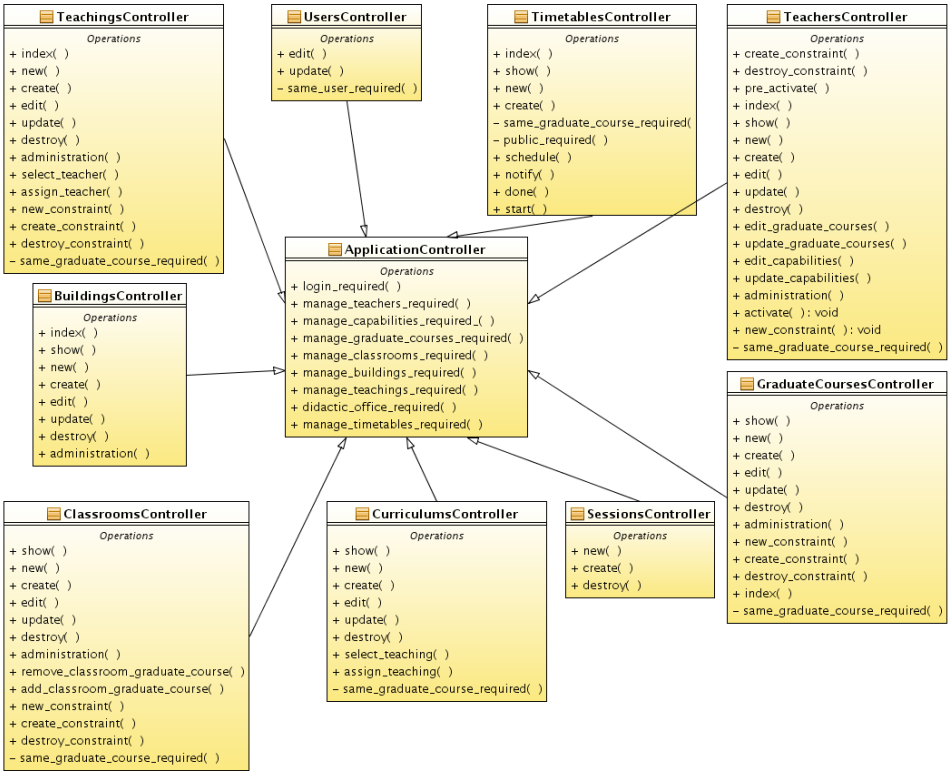
La sintassi per aggiungere un vincolo d'integrità alle chiave esterne è il seguente:

```
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 richista 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 MiddleMan (attraverso richiesta HTTP) i dati necessari ad impostare una nuova schedulazione.
- `notify`: metodo richiamato (attraverso richiesta HTTP) dal componente MiddleMan 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 (attraverso richiesta HTTP) dal componente MiddleMan utilizzato per segnalare la fine del calcolo.
- `start`: metodo che segnala al componente MiddleMan (attraverso richiesta HTTP) di avviare la generazione dell'orario per un determinato corso.

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

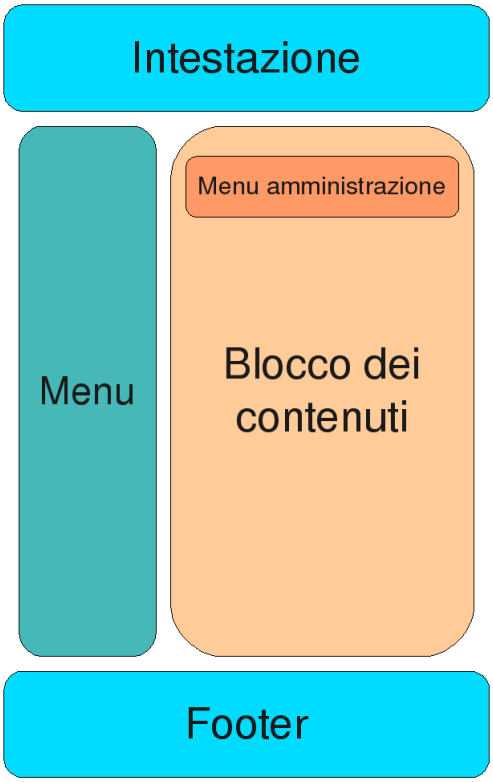
Il componente View si occupa di presentare le informazioni presenti nel sistema a chiunque le richieda, sia esso un utente finale o un altro sistema software. Per far ciò sono presenti diversi elementi in questa componente, tra i quali:

- 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 immagine 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

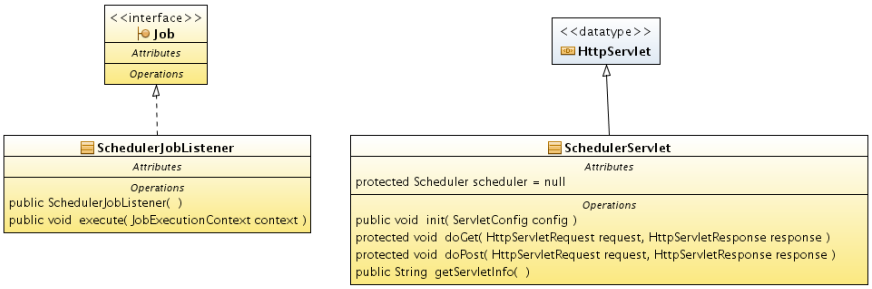
I metodi presenti all'interno del `UsersHelper` sono i seguenti:

- `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:

- aggiunta di nuove date per la generazione dell'orario relativo ad uno specifico corso di laurea
- delega alla componente `Algorithm` la generazione dell'orario relativo ad uno specifico corso di laurea
- segnala all'applicazione la fine del calcolo



3.7.1 SchedulerServlet

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

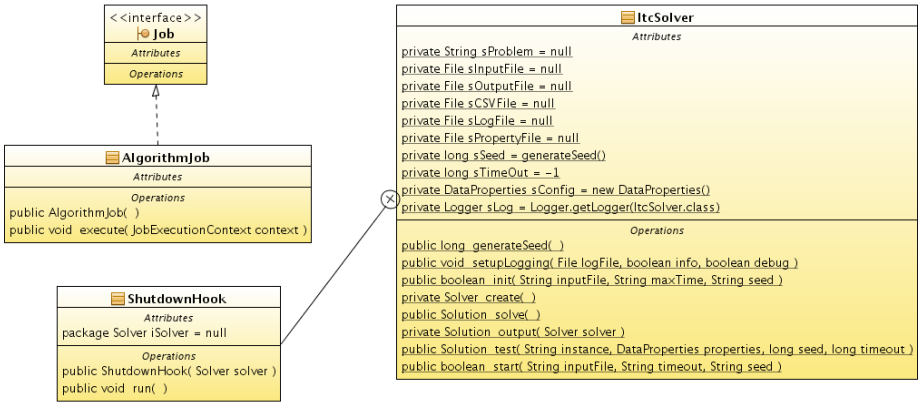
POST Parametri: course, date
crea una nuovo trigger che si attiverà alla data (date) per il corso di laurea (course) specificato

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.

3.8.3 Descrizione dell'algoritmo

L'algoritmo utilizzato per la generazione dell'orario si basa su `Constraint Solver Library` (CPSolver 1.1, copyright (C) 2007 Tomáš Müller): tale libreria permette la modellazione di una realtà universitaria al fine di elaborare diverse tipologie di orari, in base alle necessità.

Risultati internazionali

Il CPSolver 1.1 ha concorso nella `International Timetabling Competition 2007 (ITC 2007)`: tale competizione era strutturata in 3 diverse tipologie di problemi da risolvere:

1. Generazione di orari d'esame
2. Generazione di orari di lezione basati sulla struttura del corso di laurea
3. Generazione di orari di lezione basati sulle iscrizioni dei diversi studenti ai corsi

L'algoritmo è giunto tra i finalisti in tutti i diversi problemi, aggiudicandosi il primo posto nelle tipologie 1 e 2.
(sito web ufficiale: *http://www.cs.qub.ac.uk/itc2007/index.htm*)

Motivazioni della scelta

CPSolver 1.1 è stato scelto dal nostro team in seguito ad un'attenta analisi. Le motivazioni che ci hanno portato ad adottarlo sono riassumibili in:

- Ottimi risultati a livello internazionale
- Totalmente open source
- La libreria è scritta interamente in Java, e Ruby, tramite l'implementazione JRuby, supporta in maniera eccellente Java
- Il codice sorgente è strutturato in maniera ordinata e corretta
- La documentazione relativa è ampia e facilmente accessibile

Curriculum Course Timetabling

Il `Curriculum Corse Timetabling (CCT)` rappresenta la sezione del CPSolver 1.1 che risolve il problema della generazione di orari di lezione basati sulla struttura dei corso di laurea. Qui di seguito ne verranno analizzati gli aspetti di maggiore importanza.

Entità coinvolte Le entità coinvolte nella generazione dell'orario sono:

- i corsi di laurea su cui si desidera elaborare l'orario delle lezioni
- gli insegnamenti appartenenti ai corsi di laurea interessati
- il numero dei giorni di lezione (solitamente 5)
- il numero di fasce orarie giornaliere

Vincoli

I vincoli che l'algoritmo deve rispettare sono:

- tutte le lezioni di un insegnamento devono essere presenti all'interno dell'orario, ed assegnate a giorni e/o fasce orarie differenti
- due lezioni non possono tenersi contemporaneamente nella stessa aula
- due lezioni non possono tenersi contemporaneamente nello stesso giorno e nella stessa fascia oraria
- tutte le lezioni dello stesso anno devono essere tenute in giorni e/o fasce orarie differenti, vietando sovrapposizioni
- tutte le lezioni tenute dallo stesso docente devono essere tenute in giorni e/o fasce orarie differenti
- tutti i giorni e/o fasce orarie caratterizzati da indisponibilità devono essere rispettati/e

Fattori di scelta

La soluzione migliore viene trovata in base ai seguenti fattori:

- la capacità dell'aula preposta ad ospitare la lezione di un determinato insegnamento deve cercare di garantire il posto a sedere agli studenti previsti dall'insegnamento stesso
- le lezioni di un determinato insegnamento devono essere tenute almeno in un numero minimo prefissato di giorni diversi
- le lezioni dello stesso anno devono essere il più possibile adiacenti le une con le altre, riducendo le eventuali ore buche
- le lezioni dello stesso insegnamento devono essere tenute il più possibile nella stessa aula
- le preferenze sui singoli insegnamenti devono essere il più possibile rispettate

Componenti interne

Tutte le informazioni relative a CPSolver 1.1, compresa la relativa documentazione, sono reperibili all'indirizzo: *http://cpsolver.sourceforge.net/api/index.html*.

La versione originale dell'algoritmo permette la sola gestione dei vincoli. Al fine di garantire un prodotto il più possibile adattabile alle diverse realtà universitarie, il team QuiXoft ha innestato all'interno della libreria la gestione delle preferenze sui singoli insegnamenti. Tali modifiche sono state apportate nel modo meno invasivo possibile, lasciando il più incontaminato possibile il codice sorgente originale.

Prima di continuare la trattazione delle componenti occorre rendere nota la seguente convenzione in uso da parte del CPSolver 1.1 e che è stata mantenuta durante il nostro sviluppo:

- i giorni interessati nella generazione dell'orario vengono individuati tramite valore numerico: ove N°Giorni rappresenta il numero dei giorni di lezione, 0,1, .. N°Giorni-1 rappresentano in formato numerico i giorni di interesse. Ipotizzando, ad esempio, 5 giorni a disposizione: 0 = LUN, 1 = MAR, .. , 4 = VEN
- le fasce orarie vengono individuate tramite valore numerico: ove N°FasceOrarie rappresenta il numero di fasce orarie giornaliere, 0,1 .. N°FasceOrarie rappresentano in formato numerico le fasce orarie. Ipotizzando, ad esempio, 6 fasce orarie: 0 = 1° fascia, 1 = 2°fascia, .. , 5 = 6°fascia

I nomi delle classi e dei metodi rispecchiano le convenzioni standard di Java. La lingua di riferimento adottata, come nel CPSolver 1.1, è la lingua inglese, commenti compresi.

La classe MyPreferences_DaySlot

La classe `MyPreferences_DaySlot` rappresenta il gestore delle preferenze. Le preferenze che gestisce sono relative ad un insegnamento, in particolare rappresentano i giorni e/o le fasce orarie in cui tale insegnamento non è desiderabile venga tenuto. L'insieme delle preferenze è organizzato sotto forma di `Vector` (*http://java.sun.com/j2se/1.4.2/docs/api/java/util/Vector.html*), il cui tipo base è `DaySlot_PreferenceInfo`.

La classe interna DaySlot_PreferenceInfo

La classe `DaySlot_PreferenceInfo` rappresenta una singola preferenza. Ogni preferenza è caratterizzata da diverse informazioni quali:

- il codice identificativo dell'insegnamento a cui essa fa riferimento
- il giorno e la fascia oraria in cui è desiderabile non venga tenuto l'insegnamento

- il peso della preferenza stessa

Il peso rappresenta un valore numerico attraverso il quale si assegna un valore di importanza ad una preferenza. Tale peso permette all'algoritmo di tenere in maggiore considerazione le preferenze "molto desiderabili", rispetto, ad esempio, a preferenze minori.

Pesi e priorità delle preferenze L'assegnazione dei pesi viene gestita internamente alla classe in base ad una scala di priorità. I valori dei pesi sono stati scelti dal team QuiXoft in modo da garantire una corretta ed efficiente elaborazione. La relazione tra il valore di priorità ed il peso che tale priorità associa alla preferenza è riassumibile con la tabella seguente:

Priorità	Peso
<0	0
1	20
2	16
3	13
4	10
5	8
6	6
7	5
8	4
9	3
10	2
>10	1

I metodi

I metodi privati presenti all'interno della classe `MyPrefereces_DaySlot` sono:

- `int getWeightFromPriority (int)` : in base al valore di priorità restituisce il peso corrispondente secondo la tabella precedente.
- `DaySlot_PreferenceInfo getPreference (String, int, int)` : restituisce un riferimento alla preferenza di uno specifico insegnamento, in base al giorno ed alla fascia oraria di interesse. Nel caso in cui tale preferenza non esistesse il valore restituito è `null`.
- `bool existsCoursePreference (String)` : in base al codice identificativo di un insegnamento, restituisce TRUE se esiste almeno una preferenza relativa allo stesso, FALSE nel caso in cui non ne esista alcuna.

I metodi pubblici messi a disposizione sono:

- `addPreference(String, int, int, int)` : aggiunge una preferenza all'insieme delle preferenze. I parametri formali rappresentano l'identificativo dell'insegnamento, il giorno e la fascia oraria in cui è desiderabile non venga tenuta alcuna lezione dell'insegnamento specificato, ed il valore di priorità che la preferenza assumerà all'interno del sistema di calcolo.

- `getPreferencesPenality(String)` : restituisce il valore di penalità associato ad un determinato insegnamento. I valore di penalità viene calcolato in base alle preferenze non soddisfatte.

Dati di input

Per quanto riguarda l'input, l'algoritmo si basa su un file con estensione `.cct` (Curriculum Course Timetabling) il quale contiene i dati necessari all'elaborazione dell'orario delle lezioni.

La struttura interna del file di input è definibile attraverso le diverse sezioni che la compongono; più precisamente esse sono:

- Sezione `MODEL`: informazioni generali
- Sezione `COURSES`: elenco dei vari insegnamenti
- Sezione `ROOMS`: elenco delle aule
- Sezione `CURRICULA`: elenco dei relativa corsi di laurea
- Sezione `UNAVAIBILITY_CONSTRAINTS`: elenco delle indisponibilità
- Sezione `PREFERENCES`: elenco delle preferenze

La sezione `MODEL` rappresenta il preambolo del file. Tratta le informazioni di carattere generale riguardanti la realtà universitaria sulla quale si genererà l'orario delle lezioni. Si suddivide nei seguenti campi:

- `Name`: indica il nome della realtà di riferimento
- `Courses`: indica il numero complessivo dei corsi
- `Rooms`: indica il numero delle aule a disposizione
- `Days`: indica il numero di giorni di lezione
- `Periods_per_day`: indica il numero di fasce orarie in cui è suddiviso ogni giorno
- `Curricula`: indica il numero dei corsi di laurea e relativi curricula interessati nella generazione dell'orario
- `Constraints`: indica il numero di vincoli di indisponibilità

Per la trattazione delle sezioni seguenti occorre avvalersi della seguente convenzione:

- il carattere — indica la divisione dal campo precedente al successivo, e corrisponde al carattere `blank` nel file reale
- i campi preceduti dal carattere # indicano campi a cui è assegnata un'informazione di tipo numerico

La sezione `COURSES` identifica la sezione nella quale vengono elencati tutti gli insegnamenti e relative caratteristiche. Ogni riga di questa sezione rappresenta un singolo corso, e si attiene alla struttura seguente:

```
CourseID | Teacher | #Lectures | MinWorkingDays | #Students
```

La sezione `ROOMS` identifica la sezione nella quale vengono elencate tutte le aule a disposizione e relativa capacità. Ogni riga di questa sezione rappresenta una singola aula, e si attiene alla struttura seguente:

```
RoomID | #Capacity
```

La sezione `CURRICULA` identifica la sezione nella quale vengono elencati tutti i corsi di laurea con i relativi curricula ed i relativi insegnamenti che ne fanno parte. Ogni riga di questa sezione rappresenta un singolo corso di laurea, e si attiene alla struttura seguente:

```
CurriculumID | #Courses | CourseID ... CourseID
```

La sezione `UNAVAILABILITY_CONSTRAINTS` identifica la sezione nella quale vengono elencati tutti i vincoli di indisponibilità degli insegnamenti. Ogni riga di questa sezione rappresenta un singolo vincolo, e si attiene alla struttura seguente:

```
CourseID | #Day | #TimeSlot
```

La sezione `PREFERENCES` identifica la sezione nella quale vengono elencate tutte le preferenze degli insegnamenti: rappresentano quando non è desiderabile venga tenuta una lezione di un determinato insegnamento. Ogni riga di questa sezione rappresenta una singola preferenza, e si attiene alla struttura seguente:

```
CourseID | #Day | #TimeSlot
```

Dati di output

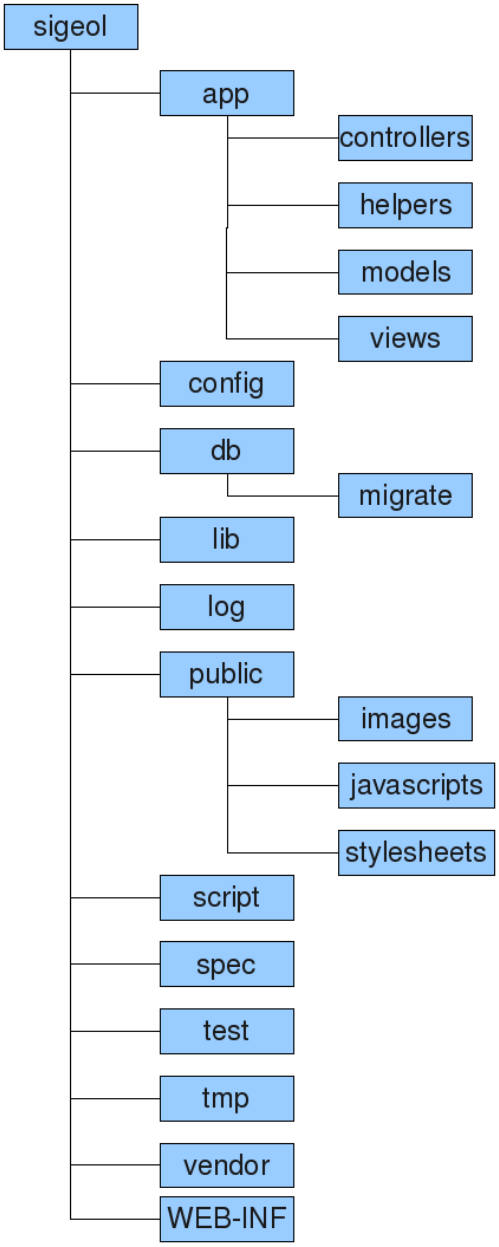
Per quanto riguarda l'output l'algoritmo produce un file che rappresenta l'intero orario delle lezioni. Ogni riga rappresenta una singola lezione, e si attiene alla struttura seguente:

```
CourseID | RoomID | #Day | #TimeSlot
```

I valori identificativi dell'insegnamento (`CourseID`) e dell'aula (`RoomID`) derivano dai valori delle sezioni `COURSES` e `ROOMS` trattate nella sezione 3.8.3

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
6 marzo 2009	1.0.1	Correzione dell'impostazione delle pagine
1 marzo 2009	1.0.0	Approvazione del responsabile e passaggio di stato a Formale
24 febbraio 2009	0.10.1	Verifica della versione 0.10.0
23 febbraio 2009	0.10.0	Stesura sezione Organizzazione delle directories e Tracciamento requisiti
23 febbraio 2009	0.9.0	Stesura sezione Organizzazione delle directories
21 febbraio 2009	0.8.0	Stesura sezione Componente Middle-Man e Algorithm
21 febbraio 2009	0.7.0	Completata stesura sezione Componente Controller e redatta la sezione Componente Helper
20 febbraio 2009	0.6.0	Completata stesura sezione Componente View ed inserimento immagine layout
20 febbraio 2009	0.5.2	Inseriti i diagrammi delle classi della Componente Controller e Model
20 febbraio 2009	0.5.1	Correzione di errori grammaticali della versione 0.5.0
19 febbraio 2009	0.5.0	Stesura della sezione Componente Model
19 febbraio 2009	0.4.1	Verifica e correzione errori di sintassi della versione 0.4.0
18 febbraio 2009	0.4.0	Prima stesura della sezione Componente View
18 febbraio 2009	0.3.0	Prima stesura della sezione Componente Controller
17 febbraio 2009	0.2.0	Stesura della sezione Standard di progetto
15 febbraio 2009	0.1.0	Stesura dell'indice