VIETNAM NATIONAL UNIVERSITY, HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Hua Viet Ngoc

TEACHING AND LEARNING SCHEDULER SYSTEM

Major: Computer Science

HA NOI - 2015

VIETNAM NATIONAL UNIVERSITY, HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Hua Viet Ngoc

TEACHING AND LEARNING SCHEDULER SYSTEM

Major: Computer Science

Supervisor: PhD Truong Anh Hoang

HA NOI - 2015

AUTHORSHIP

"I hereby declare that the work contained in this thesis is of my own and has not been previously submitted for a degree or diploma at this or any other higher education institution. To the best of my knowledge and belief, the thesis contains no materials previously published or written by another person except where due reference or acknowledgement is made."

Signature:……

SUPERVISOR'S APPROVAL

"I hereby approve that the thesis in its current form is ready for committee examination as a requirement for the Bachelor of Computer Science degree at the University of Engineering and Technology."

Signature:……

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to PhD Truong Anh Hoang, who helped me a lot in the process of implementing this thesis.

I would like to also thank some my friends in K56CA - University of Engineering and Technology and Framgia Company. They also support me to complete my thesis.

I greatly appreciate the following organizations: Information Technology Department, the University of Engineering and Technology

ABSTRACT

Currently, the volume of teaching is increasing, the complexity of the process of assigning teaching schedule so that rational and efficient as well as satisfying many binding conditions has prompted raises a system can solve the problems mentioned above. My proposal is to build a friendly system, easy to use for everyone but may allow custom attributes for building complex an optimal timetable.

My system is divided into 2 parts: the client (front-end) is a website with a friendly GUI, which allows CRUD data and builds constraints with the timetable; the server (back-end) will be the data processing center and find an optimal timetable.

In my system, the server use a scheduling system is FET (an open source free timetabling software), so most of the results of this system depend on the FET. After a period of testing, I found this system can solve a complex timetable in an acceptable time even with a university scale.

Overall, this system can solve these problems in practice, however to be able to apply immediately, it needs to be further improved because there are still some limitations, as well as to the involvement of multiple parties to conduct tests more systematically.

TABLE OF CONTENTS

List of Figures

List of Tables

ABBREVIATIONS

CRUD	Create, read, update and delete
CSV	Comma-separated values
FET	(name of free timetable software)
GUI	Graphics user interface
MVC	Model-View-Controller
ORM	Object-relational mapping
REST	Representational state transfer
ROR	Ruby on Rails framework

INTRODUCTION

1.1 Motivation

Nowadays, teaching activities at the university has become more complex because many different reasons: the number of classes, courses, lecturers, classrooms, along with a variety of different binding conditions on time for other work of the lecturers, bound in the courses of the same class are non-overlap,.... It makes building a timetable becomes extremely difficult.

Unfortunately, this issue has been resolved manually in many schools and universities. It cannot thoroughly solve the above-mentioned constraints, as well as the control, statistics, management becomes difficult. This motivated me to build a system to solve the problem on, create a better solution now.

In fact, in some places I know, they still do manually, but not encounter too many obstacles. Simply because of their classrooms so much, so arranging a timetable not good, still can satisfy the basic conditions. But with more complex constraints, or the facilities are not redundant, the timetable construction is still very difficult to manually resolve. Even with the large number of classrooms, there will be many vacant classrooms in many periods, this is wasteful. If we can control, we can use for other useful purposes. So it is clear that we need a system that can solve this issue.

1.2 Contributions and thesis overview

The purpose of this thesis is to present how to build a system as described above and my main contribution was the development of this system. We will present detail in later

The rest of this thesis is organized to as follows:

Chapter 2 presents some work related to the construction of this system include: using an open source software for scheduling (FET), Ruby on Rails (ROR) framework and some knowledge related to website development

Chapter 3 will present full of design, architecture, the how it work of the system.

Chapter 4 and Chapter 5 will be the results and conclusions

RELATED WORK

2.1 Some scheduling software

Today, scheduling problem is not a new issue. So sure there was a lot of scheduling software has been developed around the world. Before building the system, we should find out through some current software:

- *UniTime* (http://www.unitime.org/)
- *FET - Free Timetabling Software* (http://lalescu.ro/liviu/fet/)
- *Mimosa Scheduling Software* (http://www.mimosasoftware.com/)
- *Lantiv Scheduling Studio 7* (http://schedulingstudio.com/)

Above is some current scheduling software, however, *Mimosa Scheduling Software* and *Lantiv Scheduling Studio 7* software are not free software, as they are inclined to create and manage manually rather than with the automatic scheduling with conditions. *UniTime* and *FET* are open source software, automatic scheduling with constraints. In addition, there also are many other similar software only are most of them are offline and commercial software. Through the process of research, I think that they still do not really resemble my system. I am building a system with user-friendly interface, easy to use; work through the internet environment (although the schedule may belong to some individuals, but constraints can come from many sides, as well as other information should be shared). So I decided to build a new system.

However I noticed FET was designed for the purpose to expand, FET can solve pretty good core issue. FET is also open source software, so I decided to use the FET as part of the system.

2.2 FET

In this system, one of the most important pieces is the task scheduling. This is a difficult task. I decided do not build from scratch. And right there is a lot of software has solved this problem, but I am particularly attentive to FET.

FET is open source free software for automatically scheduling the timetable of a school, high-school or university. It uses a fast and efficient timetabling algorithm. It is licensed under the GNU Affero General Public License version 3 or later. [1]

Usually, FET is able to solve a complicated timetable in maximum 5-20 minutes. For simpler timetables, it may take a shorter time, under 5 minutes (in some cases, a matter of seconds). For extremely difficult timetables, it may take a longer time, a matter of hours. [1]

2.2.1 Brief history for FET

- On 31 October 2002 - **Liviu Lalescu** started the project. The algorithm was genetic, very inefficient.
- On 19 March 2006 - **Volker Dirr** joined the project. He reported that his difficult data file could not be solved by FET, but could be solved by other software, and insisted to continue the search for a better algorithm for FET.
- On 24 June 2007 - The current efficient heuristic FET-5 algorithm was found - marked a leap in the effectiveness of the algorithm
- FET is still being developed and supported until today

2.2.2 FET Features

- FET is written in C ++, so it can ensure the performance, rapid implementation of complex algorithms, build on the Qt framework, install and compile via Cmake. Platform independent implementation, allowing running on GNU/Linux, Windows, Mac and any system that Qt supports.
- Flexible modular XML format for the input file, allowing editing with an XML editor or by hand (besides FET interface)
- The resulted timetables are exported into XML formats
- Each constraint has a weight percentage, from 0.0% to 100.0% (but some special constraints are allowed to have only 100% weight percentage)

- A large and flexible palette of time constraints:
    - Break periods
    - For teacher(s):
        - Not available periods
        - Max/min days per week
        - Max hours daily/continuously
    - For students (sets):
        - Not available periods
        - Max days per week
        - Max hours daily/continuously
        - Min hours daily
    - For an activity or a set of activities/sub-activities:
        - A set of preferred starting times
        - Min/max days between them
        - Consecutive, ordered, grouped (for 2 or 3 (sub)activities)
        - Not overlapping
- A large and flexible palette of space constraints:
    - Room not available periods
    - For teacher(s):
        - Home room(s)
    - For students (sets):
        - Home room(s)
    - Preferred room(s):
        - For a subject
        - For an activity tag
- In addition, FET can solve some other constraints, but I think that no actually need should not put in here

2.2.3 FET Algorithms [1]

FET uses algorithms developed through several stages as well as uses some other software too, so it is very complex. So I just summarize the main idea of the algorithm through references

FET uses a heuristic algorithm, placing the activities in turn, starting with the most difficult ones. If it cannot find a solution, the algorithm swaps activities recursively if

that is possible in order to make space for a new activity or, in extreme cases, backtracks and switches order of evaluation.

The algorithm is heuristic (**Lalescu** named it "recursive swapping").

Input: a set of activities A_1...A_n and the constraints.

Output: a set of times TA_1...TA_n (the time slot of each activity. Rooms are excluded here, for simplicity. We solve with time and rooms similarly, and then combine them into final result). The algorithm must put each activity at a time slot, respecting constraints. Each TA_i is between 0 (T_1) and max_time_slots-1 (T_m).

Constraints:

C1) Basic: a list of pairs of activities which cannot be simultaneous (for instance, A_1 and A_2, because they have the same teacher or the same students);

C2) Lots of other constraints (excluded here, for simplicity).

The timetabling algorithm ("recursive swapping"), though it might be related to the algorithm known as "ejection chain"):

- 1) Sort activities, most difficult first. Not critical step, but speeds up the algorithm maybe 10 times or more. (It means we choose activities have most constraints – or least selections)

- 2) Try to place each activity (A_i) in an allowed time slot, following the above order, one at a time. Search for an available slot (T_j) for A_i, in which this activity can be placed respecting the constraints. If more slots are available, choose a random one. If none is available, do recursive swapping:

    - 2a) For each time slot T_j, consider what happens if you put A_i into T_j. There will be a list of other activities which don't agree with this move (for instance, activity A_k is on the same slot T_j and has the same teacher or same students as A_i). Keep a list of conflicting activities for each time slot T_j.

    - 2b) Choose a slot (T_j) with lowest number of conflicting activities. Say the list of activities in this slot contains 3 activities: A_p, A_q, A_r.

- 2c) Place A_i at T_j and make A_p, A_q, A_r unallocated.
- 2d) Recursively try to place A_p, A_q, A_r (if the level of recursion is not too large, say 14, and if the total number of recursive calls counted since step (2) on A_i began is not too large, say 2*n), as in step (2).
- 2e) If successfully placed A_p, A_q, A_r, return with success, otherwise try other time slots (go to step (2 b) and choose the next best time slot).
- 2f) If all (or a reasonable number of) time slots were tried unsuccessfully, return without success.
- 2g) If we are at level 0, and we had no success in placing A_i, place it like in steps (2 b) and (2 c), but without recursion. We have now 3 - 1 = 2 more activities to place. Go to step (2) (some methods to avoid cycling are used here).

2.2.4 FET's role in this system

As described above, FET can solve partial workload in this system but not all. FET is not easy for everyone, not entirely consistent with the teaching at university (I find it suitable for high school schedules). So we need to change, rebuild several mechanisms. And even when using FET, the workload must perform for building systems remains big.

2.3 Ruby on Rails framework (ROR)

This system is actually a website, so I need to choose a framework to build. There are many frameworks (on many languages: PHP, Python, Ruby, Java....) to choose from but I chose ROR (Ruby languages) because some advantages:

- ROR is an open-source web framework that’s optimized for programmer happiness and sustainable productivity. It lets you write beautiful code by favoring convention over configuration. [2]
- ROR can build a website quickly because it was staged for basic tasks.
- ROR is designed with many advantages
    - Ruby language:
        - Object-oriented programming

        - Functional programming
        - Meta-programming
    - Rails framework:
        - MVC
        - REST
        - ORM
    - Having asset pipeline
    - There are many good security mechanism

Although the system design can be completely independent of the tools for building it but my thesis is not only the design that presents the process of building up, so selection of platform or other technologies is very important.

Using framework, designs, patterns or techniques will reduce a lot of effort while ensuring the necessary requirements. I am pleased to present several characteristics of the rails.

2.3.1 Ruby language

Ruby is a dynamic, open source programming language with a focus on simplicity and productivity. It has an elegant syntax that is natural to read and easy to write.

Rails were built on the Ruby programming language. This is a modern language with syntax style is quite natural and understandable. There is a great community support and development lib (called gem).

Ruby is full of functions necessary for the development of an application: Object-oriented programming, Functional programming, Meta-programming...

2.3.2 MVC – Model-View-Controller

Model–view–controller (MVC) is a software architectural pattern for implementing user interfaces. It divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user. [4]

- The Controller is what connects the views to the model. A controller can send commands to CRUD the model. It can also send commands to its associated view to change the view's presentation of the model.
- The Model is where you should keep your data model, the algorithms.
- The View requests information from the model that it uses to generate an output representation to the user. View visualize the data (the UI)

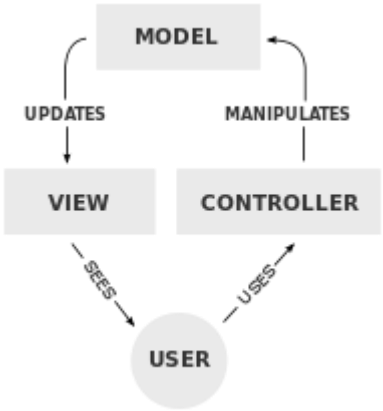

Figure 2.1: A typical collaboration of the MVC components [4]

MVC is widely used in website development. The clear dividing functions, tasks and methods of interact as above figures makes the architecture of the website coherently, clear, easy to control and manage.

2.3.3 REST – REpresentational State Transfer

REST is an architectural style for developing distributed, networked systems and software applications such as the World Wide Web and web applications. Although REST theory is rather abstract, in the context of Rails applications REST means that most application components are modeled as resources that can be created, read, updated, and deleted - operations that correspond both to the CRUD operations of relational databases and to the four fundamental HTTP request methods: POST, GET, PATCH, and DELETE.

As a Rails application developer, the RESTful style of development helps you make choices about which controllers and actions to write: you simply structure the application using resources that get created, read, updated, and deleted.

2.3.4 ORM – Object-relational mapping

Object-relational mapping, commonly referred to as its abbreviation ORM, is a technique that connects the rich objects of an application to tables in a relational database management system. Using ORM, the properties and relationships of the objects in an application can be easily stored and retrieved from a database without writing SQL statements directly and with less overall database access code.

In Rails, there is an implementation of the Active Record pattern which itself is a description of an Object Relational Mapping system. Active Record gives us several mechanisms, the most important being the ability to:

- Represent models and their data.
- Represent associations between these models.
- Represent inheritance hierarchies through related models.
- Validate models before they get persisted to the database.
- Perform database operations in an object-oriented fashion.

2.3.5 Asset pipeline

In Rails, there are three principal features to understand: asset directories, manifest files, and preprocessor engines:

- *Asset directories*: there are three canonical directories for static assets, each with its own purpose:
    - **app/assets**: assets specific to the present application
    - **lib/assets**: assets for libraries written by our development team
    - **vendor/assets**: assets from third-party vendors
- *Manifest files*: This is a mechanism of Rails can combine all javascript files into one, all css into one, increasing efficiency when loading a web page
- *Preprocessor engines*: This is a Rails mechanism to help process a file in a series engine. For example: file ‘foobar.js.erb.coffee’ gets run through both CoffeeScript (a language same javascript) and ERb (embedded ruby) (with the code running from right to left, i.e., CoffeeScript first).

With asset pipeline, the results are optimized to be efficient in a production application but we can work with multiple nicely formatted files in development. The result is the best of both worlds: convenience in development and efficiency in production.

2.4 Some other technologies

Outside of FET and ROR, I have studied and used some of technologies, techniques and protocols later (because they are also quite popular today, so I just mentioned, but will not say much about it):

- HTTP protocol: is an application protocol for distributed, collaborative, hypermedia information systems. HTTP is used popularly in World Wide Web
- HTML: is a revision of the Hypertext Markup Language, the standard programming language for describing the contents and appearance of Web pages.
- CSS: (Cascading Style Sheet) is a plain text file format used for formatting content on web pages.
- JavaScript: is an interpreted programming or script language (often used in the client side to manipulate with DOM - Document Object Model)
- Git: is a distributed revision control system with an emphasis on speed, data integrity, and support for distributed, non-linear workflows.
- Heroku: a cloud application platform that deploy web apps.

OUR SYSTEM

3.1 Requirements

The system should be developed to ensure the following requirements:

About user and authorities:

- There are 2 user types: Manager and Lecturer
- Lecturer:
    - Can edit/update profile
    - Can see useful data (lecturers, subjects, courses, timetable ...)
    - Can add, edit, remove personal constraints (busy time, max periods continuous/daily, ...)
- Manager:
    - Can edit/update profile
    - Can add, remove Lecturer model
    - Can CRUD other data (time (periods), space (rooms), subjects, courses, ...)
    - Can create constraints (break times, unavailable rooms in some periods, non-overlap courses ...)
    - Can control generating timetable

About GUI:

- Friendly, easy to use
- Intuitive, good interaction
- Responsive web (good view in mobile)

About functions:

- Manipulation of data to work correctly, ensure data integrity when changes

- Data can import from external file (CSV, Excel ...). This makes moving data from another place to the system conveniently
- Solving timetable has to be an acceptable time (or find a good solution in a determined interval). The configuration can adjustment flexibility in the process generating timetable
- Can search, change data easily

3.2 Interactive with FET

We will start building the basic architecture for the system starting from the processing of requests from users, and how to communicate with FET:

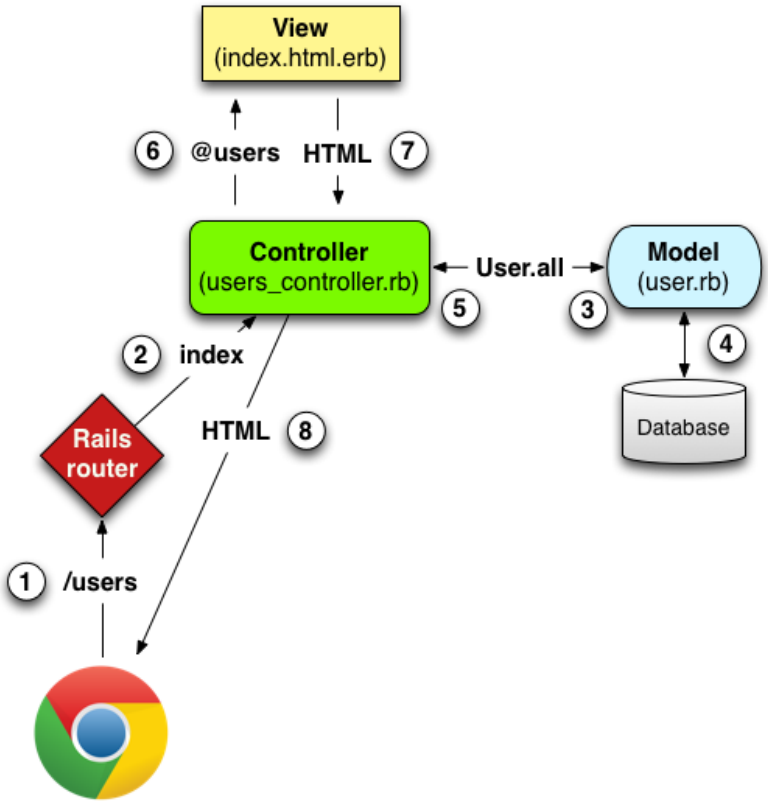

Figure 3.1: A detailed diagram of MVC in Rails. [3]

The above figure is the basic process of processing the request is sent from the client according to the MVC pattern. To communicate with the FET, we will build a Controller to create FET input (XML format) from data in the model. Then we active FET (C++ program) via command line to generate timetable. FET output is also XML

files. We will parse output XML files to transform into the model, make a complete timetable. The figure below describe more clearly:

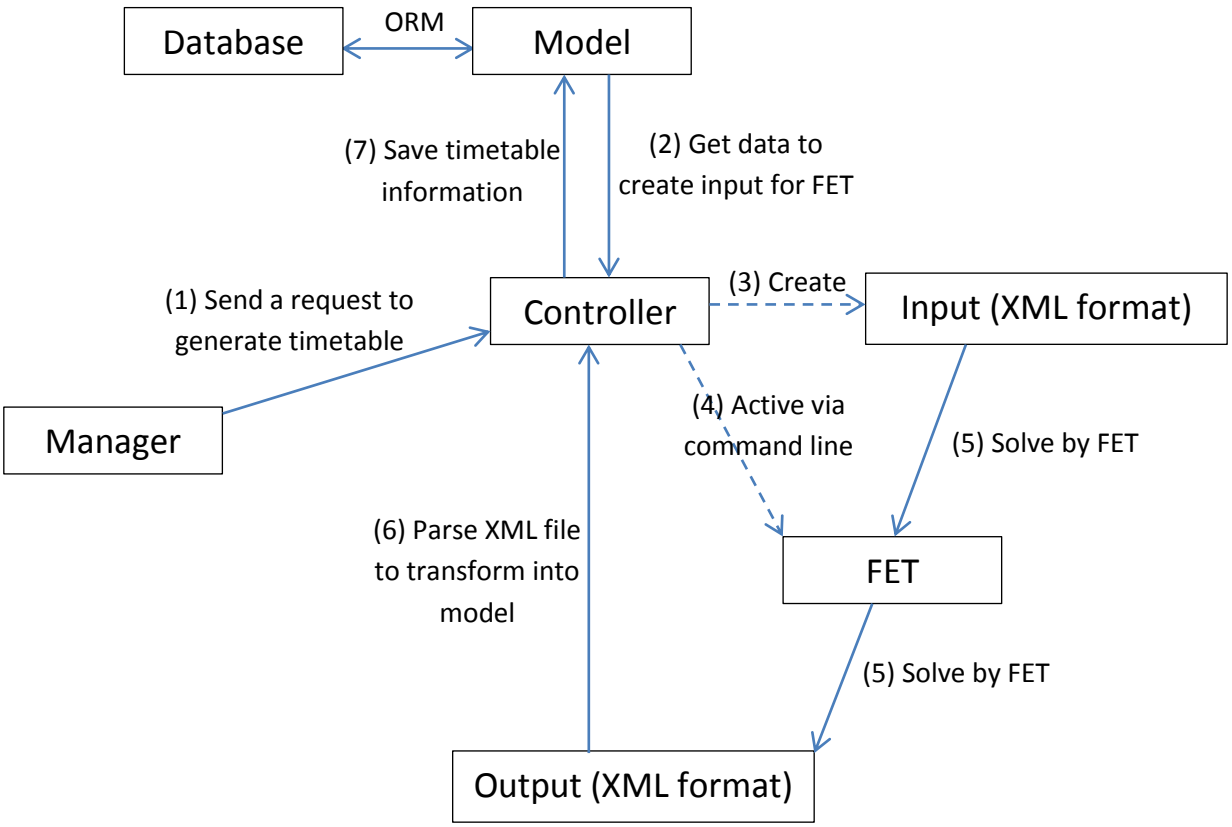

Figure 3.2: Interactive with FET

Also, when working with the FET, this requested requires time. So we should use multithreading techniques to other requests still work when FET is running.

3.3 Use-case diagram

First, let's consider the overall design for use-case diagrams generally.

We has 2 types of user: lecturer and manager, both 2 and inherited from the user, the user has manipulate the basic login / logout and to use sessions and cookies to remember user (we should care to security issues, Rails has available security mechanisms pretty good against common hacking type, including passwords or tokens remember are hashed before writing to the database, data encryption values in cookie, use SSH, ...). Let see more clearly illustrated in below:

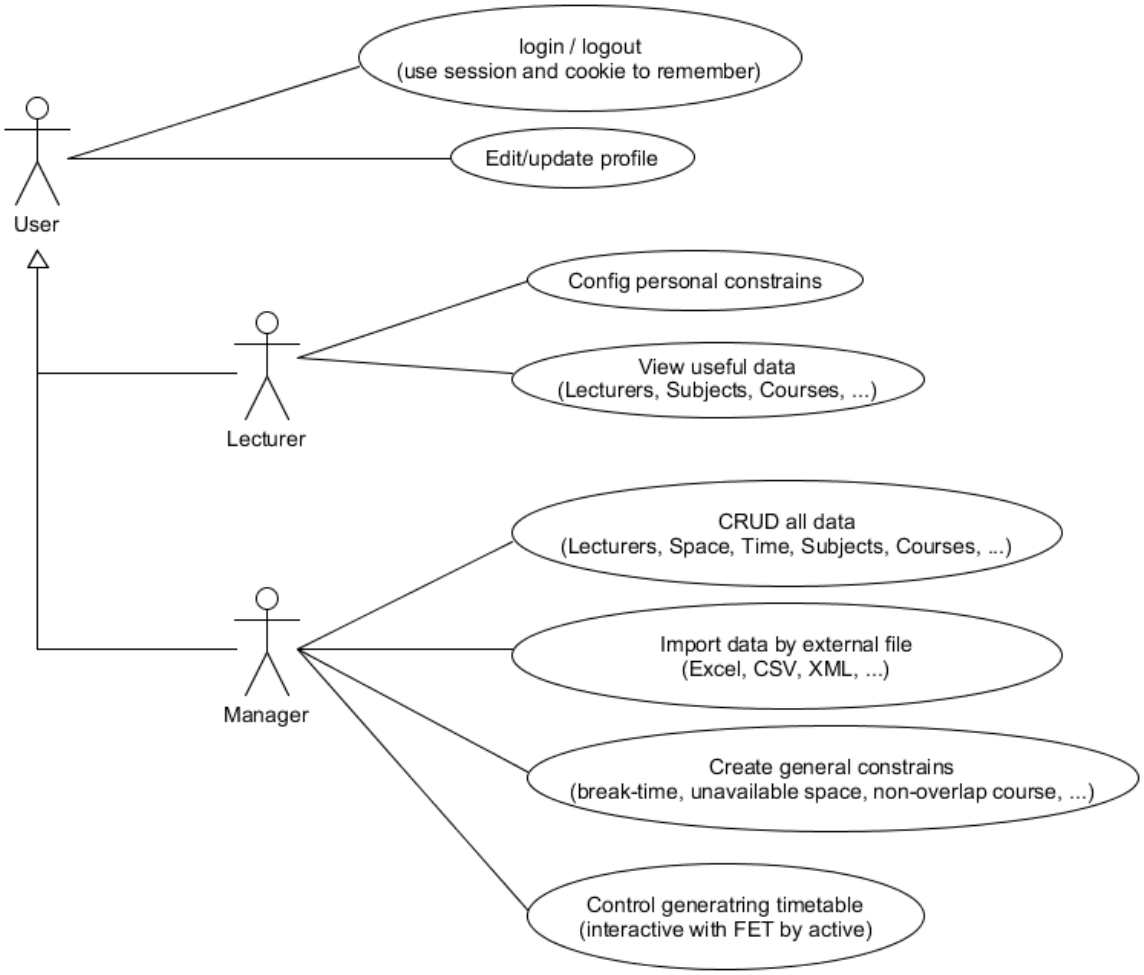

Figure 3.3: Use-case diagram (overview)

In Lecturer role, there are two main activities: view data (Data in this system include Lecturers, Subjects, Courses and several other data that will present later) and customization personal constraints (busy-time, max periods daily/continuous). In the function "view data", we need to build for convenience, to the appropriate view for each different model types (e.g. display timetable, rooms) and has other useful functions for searching and paging. Personal constraints lecturer in our system currently only include busy-time (for periods) and max continuous periods / daily. These constraints are necessary for realistic and can solve by FET. Actually FET has more constraints, but for simplicity, we have retained the necessary constraints to create the system, if over time, the binding is becoming increasingly more complex, we will continue to design accordingly, or try to convert them into the basic constraints.

In the manager role, there are four main actions: the right to work with all the data, input from external files (used for transferring data from other place quickly and conveniently), set the necessary constraints (break-time, unavailable space, non-overlap courses, courses space), controls the timetable creation with FET. Similar as with lecturer constrains, FET have more constrains but for simplicity, we have omitted and only keep what is important. We would like to explain briefly about the type of constraints. *Break-time*: this is the periods without teaching activities (break - E.g. periods in Sunday), we will not arrange courses happening at these periods. *Unavailable space*: not all rooms can be used in all the time, so we need to make constraints some classrooms that will not be used in specific periods. *Non-overlap courses*: because each class can study many subjects in one semester, to ensure these courses are not overlap is extremely necessary. *Space courses*: Some courses may happen in some particular room (e.g. laboratory).

Here, the complex activities mainly happening in the manager role. CRUD data have to ensure data integrity and data validation when changing (we will talk in more detail in the next section). Set constraints and how to communicate with FET mentioned above, we are pleased to talk about processing the import data from external file.

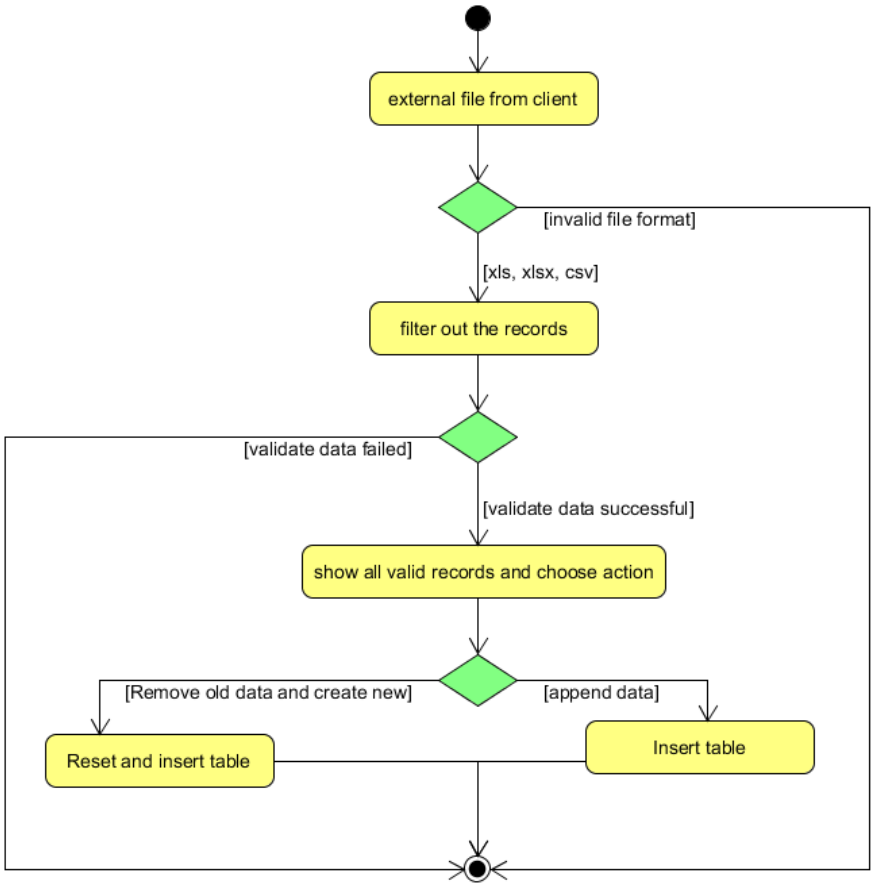

Figure 3.4: Activity diagram for import data from external files

Rails supports for uploading file through the HTML, the process of excel or csv file format is also supported, so just follow the above process, we can solve well this action.

3.4 Model and database

Model in this system divides into 3 main groups: *User*, *Data*, and *Constraints*. First let us consider group *User*.

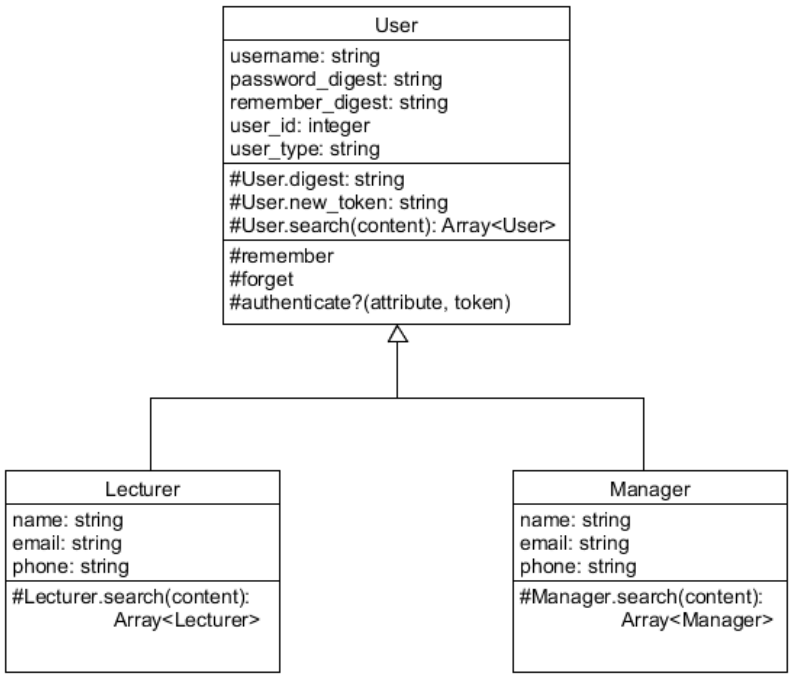

Figure 3.5: Model design for group User

A basic model in Rails is a class inherits from *ActiveRecord::Base* class, this class will make our class as a model in MVC (communicating with the database, validate the data before saving, associate with other models to ensure data integrity...). One important thing: in ROR, all class is stored in database which always exist column *id* (primary key and auto-increment) and 2 other columns *created_at* and *updated_at*. The addition an *id* to ensure uniqueness of the object and the integrity data (we only refer to another object via its *id*).

In the above figure, each rectangle corresponds to a class, the upper part of the rectangle are the attributes (attributes will be created in the database respectively by Rails - ORM). The bottom part is the list of operations.

Class *user* with basic attributes: *username*, *password_digest* (password will be hashed before they are written to the database - a security feature when information database is leaked, they cannot know the real password), *remember_digest*: similar *password_digest*, it save hash value of a token that is generated randomly and stored in cookies - remember login function). The operations: *remember*, *forget* and *authenticate?* used to authenticate when login, remember login and logout. There are 3 static methods: *User.digest*: compute the hash value of a string. *User.new_token*: generate a random string (token stored in cookies to remember login) and *User.search* (returns an array of objects user for searching).

Similarly, *Lecturer* and *Manager* also have some basic attributes and search functions, however *Lecturer* and *Manager* inherit from *User*. As we know, in a relational database management system, there is no concept for inheriting object directly, so we must find ways to save the inherited objects in the database. We will add the User class 2 attributes: *user_id*, and *user_type*, then associate *Lecturer* and *Manager* with the *User* class, Rails know how to solve the problem as a class are inherited themselves.

In the above models also contains the necessary data validation (e.g. *username* only contain alphanumeric, *password* length at least 6 characters, format *email*...). Rails have quite simple mechanism to solve data validation.

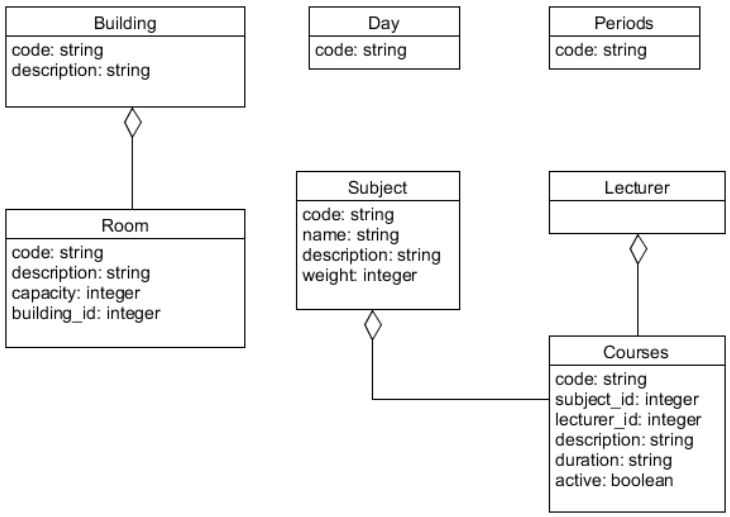

Figure 3.6: Model design for group Data

Group data design is simple (just include most important things). It includes space (*Building* and *Room*), time (*Day* and *Period*, by default, we have 7 days a week and 10

periods per day). *Subject* is presented in the above figure. Note: this design for university which uses credit system, *Course* information includes subject, lecturer and several other attributes. Especially I would say more about the *duration* and *active* attributes. *Duration* is number of continuous periods in a course per week. By default, the course will cover number of continuous periods same number of credits subject. But in many situations, that is not right (e.g. a 4-credits courses will be divided into 2 sessions per week, each session occupy 2 continuous periods. So information stored in the database corresponding is "2-2" or "2_2"). If accordance with design database standards, the value stored in the database must be pure, however our system is designed that way for simplicity. And *active* attribute, it set a course that has been arranged in the timetable or not.

Basically, the data in reality may actually be more complex, however, the current system is only designed as above.

Next go along to the group constraints:

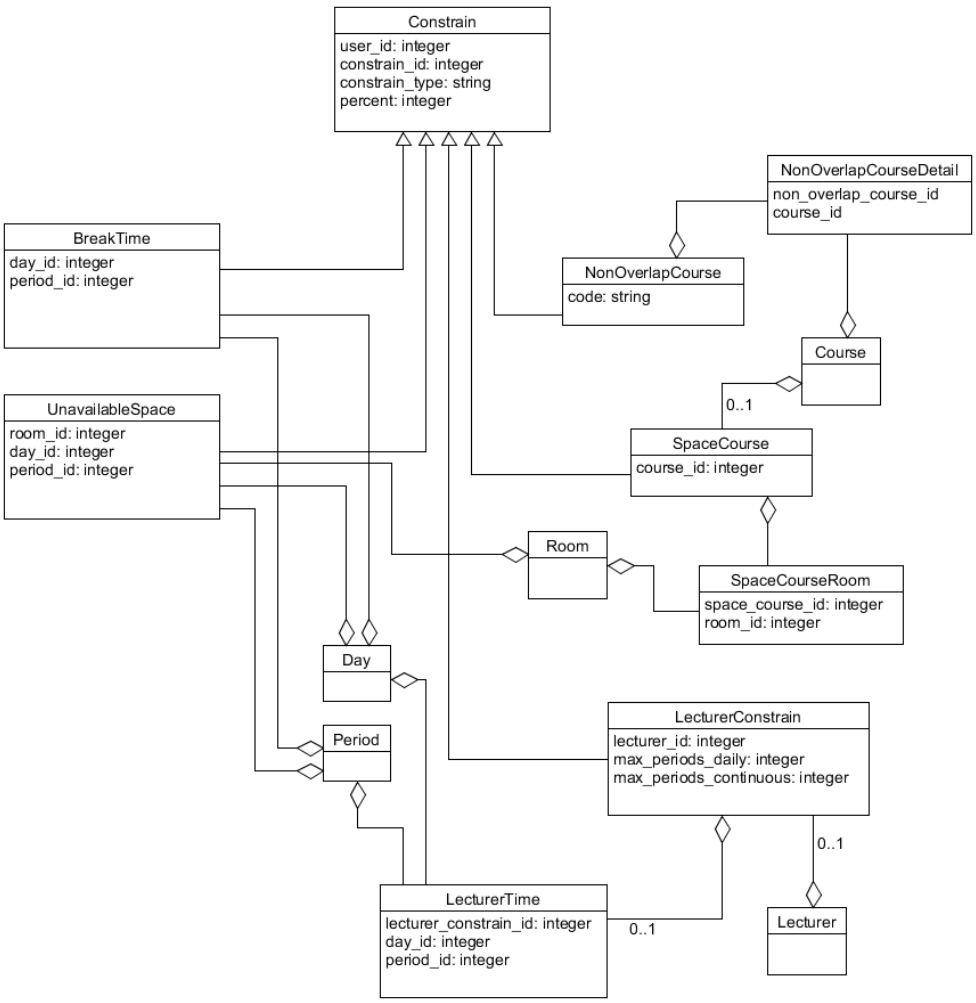

Figure 3.7: Model design for group Constraint

All constraints in the system are described above. We have two basic constraints with *BreakTime* (several periods will not take place for teaching activities, such as the periods on Saturday or Sunday), and *UnavailableSpace* (a certain rooms cannot use in some periods). *NonOverlapCourse*: a set of courses not take place simultaneously (e.g. the courses of same class). *SpaceCourse*: Some courses will take place in several certain rooms. *LecturerConstraint*: Each lecturer has personal constraint with different busy-time.

In above design, we add several extra classes for resolving many-to-many relationships between models. Although we do not really need to add these classes (just add in the database), but Rails using ORM, mapping each model (corresponding to a class) corresponding to a table in the database. This enables the CRUD models simpler. In addition, there is some relationship with the model in the group Data to ensure data integrity.

We do not need to worry too much into a database management system (DBMS). Rails using ORM, and so we can design database independent with an DBMS, we can use MySQL, Postgresql... or any other DBMS, even a DBMS does not support foreign keys. The association and data integrity can be defined at the logical level in the Rails model.

3.5 Controller

For each model, they are a resource in Rails, which means they have all the CRUD operations. The operations will be processed in the Controller. Let's take an example with the *Subjects* resource.

Firstly, let see MVC architecture in Rails again, we will start from routings. Following table represents the implementation of the REST architecture in Rails:

HTTP request	URL	Action	Purpose
GET	/subjects/	index	page to list all subjects
GET	/subjects/1	show	page to show subject with id 1
GET	/subjects/new	new	page to make a new subject
POST	/subjects/	create	create a new subject
GET	/subjects/1/edit	edit	page to edit subject with id 1
PATCH	/subjects/1	update	update subject with id 1
DELETE	/subjects/1	destroy	delete user with id 1

Table 3.1: RESTful routes provided by the Subjects resource

One should note that most current browsers only support 2 methods for form of http: GET and POST. REST in Rails also uses some other methods. However, we do not worry because Rails automatically added to the hidden input in the form to set really method when html code is created by Rails.

We will have a *subjects_controller* that have 7 functions corresponding to 7 above routes Each function will work with *Subject* model and call to the corresponding view to create UI (HTML).

For each controller, we can control authority for each action. For example, when there is a request to *delete* a subject, we can check whether the current user is the manager. In Rails, this is quite simply, before implement the action *delete* in the *subjects_controller*, we will declare with Rails that will run via another function to check current user.

From the controller, we can call the Model, the Relation, and Association of Model. For example, if we want to get all *Course*, we simply call `Course.all`. Each Course have a Subject, we also simply called `course.subject`. The data association and working with the database is supported by Rails.

In addition to the standard operations with the model, we also completely free of routing problems, handles the other functions, free to define other functions in the controller. Example to handle an AJAX request, we can define a function in controller handling a defined route for this request. Rails support to generate JavaScript code with Preprocessor engine (embedded ruby), and automatically sent to the client and execute this code to simplify some common AJAX request without writing a specific JavaScript code to process the user side.

We also can handle more complex tasks (e.g. interaction with FET) through the controller by external modules and they will be included such as an apart in controller. We should not write directly to the controller, which helps controllers be true to the meaning of it, not break the features of MVC architecture.

3.6 View

With each Model manipulation, we have a distinct View. The design view is not simple. To have a beautiful design, simple, display well on different types of screens (mobile, PC ...) requires us to build a good system CSS. Here I have used the bootstrap (is a free and open-source collection of tools for creating in websites and web applications) and SCSS (an expanded version of CSS to work more convenient).

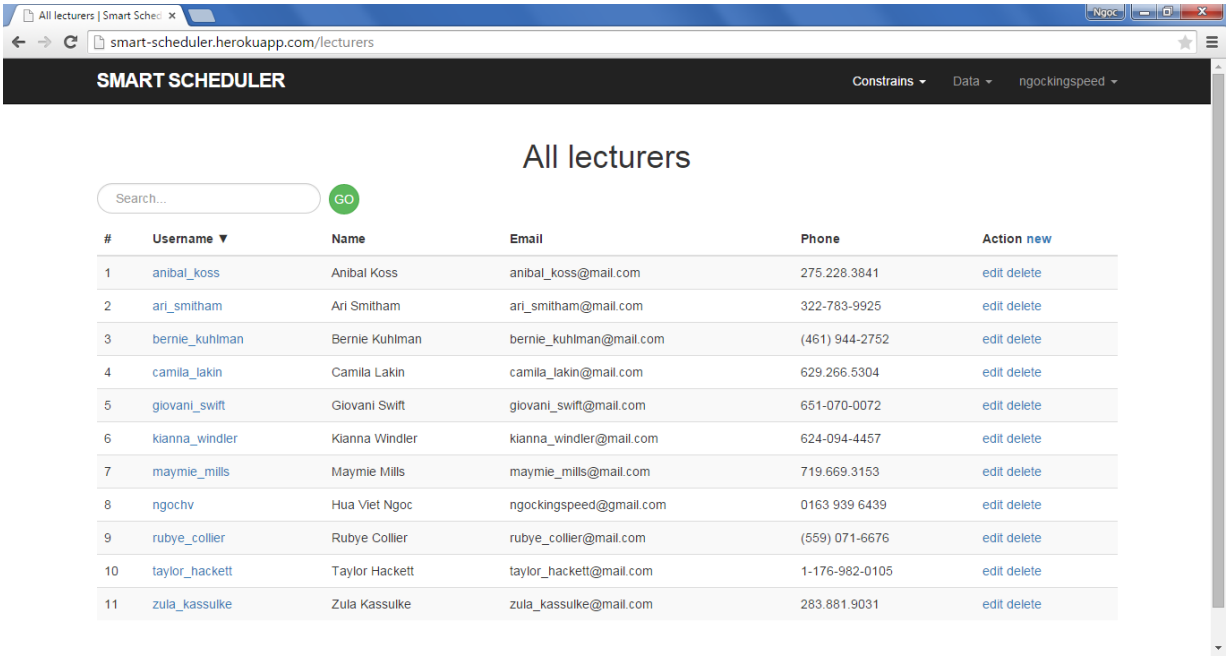

Figure 3.8: Listing Lecturers

Above figure describes all objects of a model (here is *Lecturer*). It correspond *index* in controller. Here I also have built search and paging features here. Most listing all objects of a Model is shown above. However, there are also some other models have a different View.

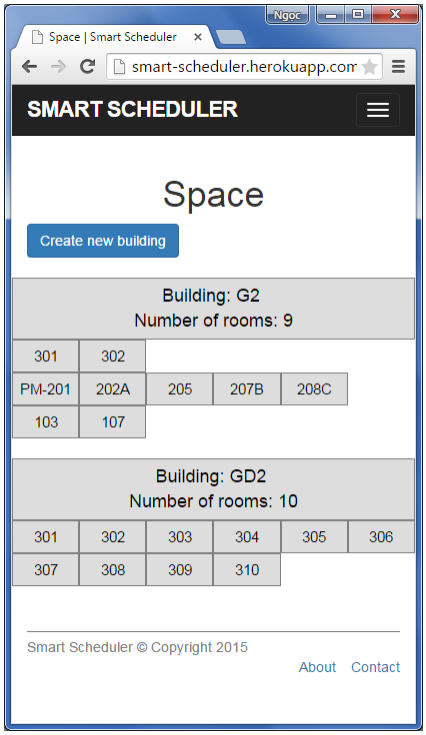

Figure 3.9: Listing Room (mobile resolution)

The above figure described space (*buildings* and *rooms*). It describes the friendly and intuitive. Rooms also arranged by their floor (I detect floor by regex).

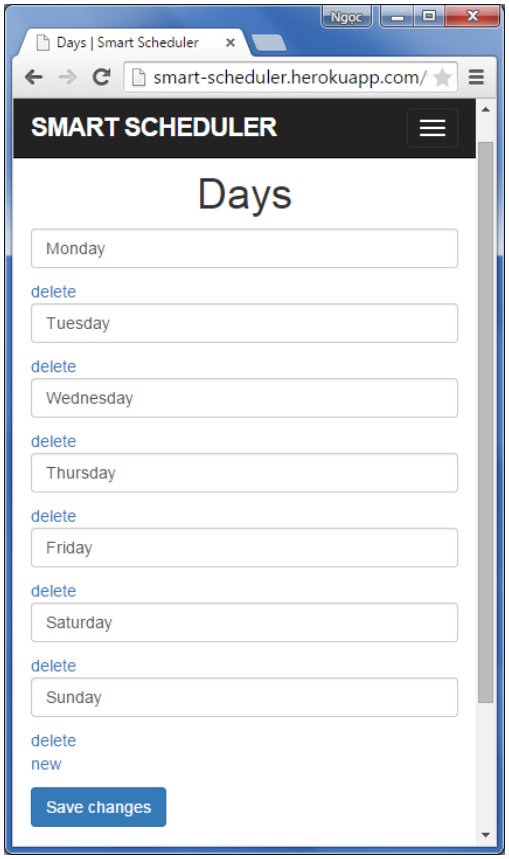

Figure 3.10: Edit day (mobile resolution)

The figure above described edit an object (here is *Day*). I have used a bit of technique with JavaScript to add / remove elements without reloading.

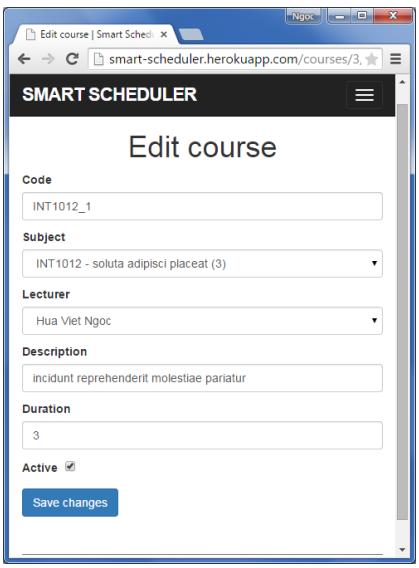

Figure 3.11: Edit course (mobile resolution)

The above figure describes edit a different model (*Course*). I use selection (combobox, dropdown...) is supported by HTML for association data (*Course* reference *Subject* and *Lecturer*)

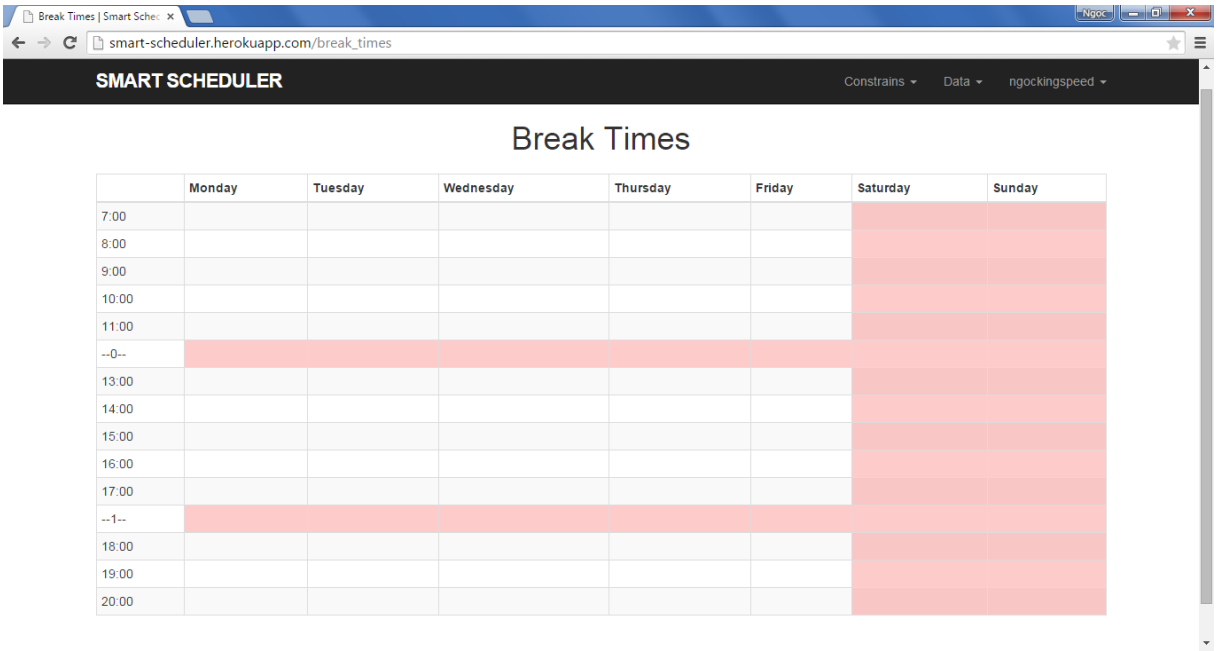

Figure 3.12: Break Times constraint

The figure above describes break-times (there will be no teaching activities take place on these periods). Change by clicking directly into the cell (AJAX).

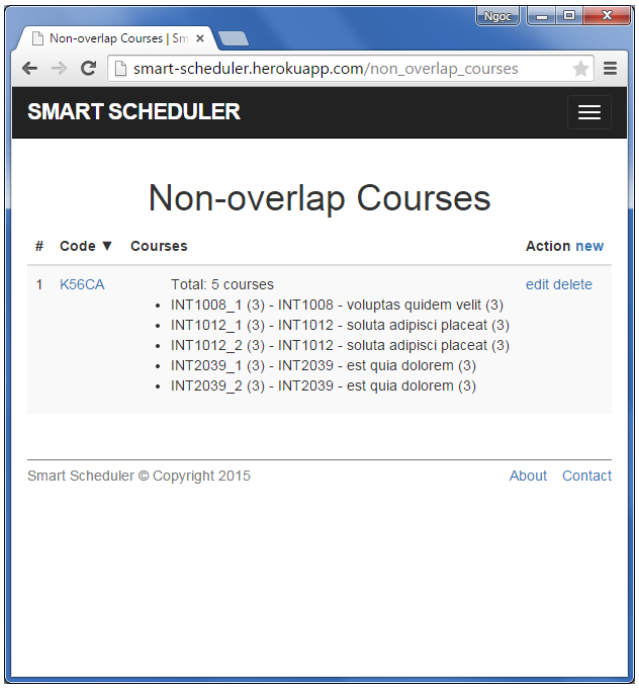

Figure 3.13: Non-overlap Courses constraint

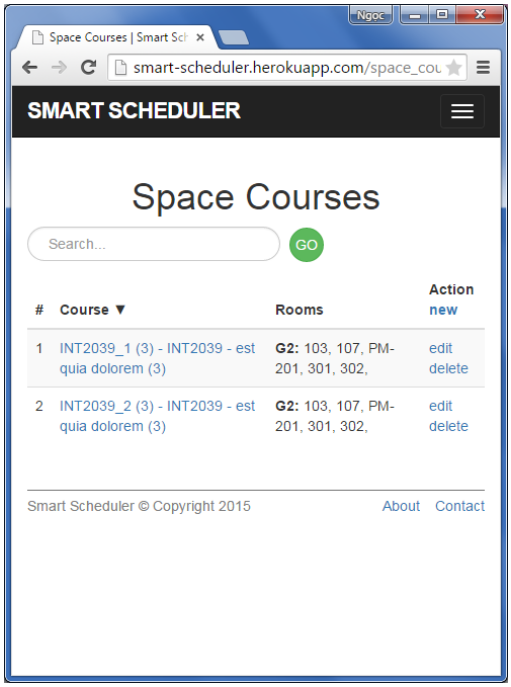

Figure 3.14: Space Courses constraint

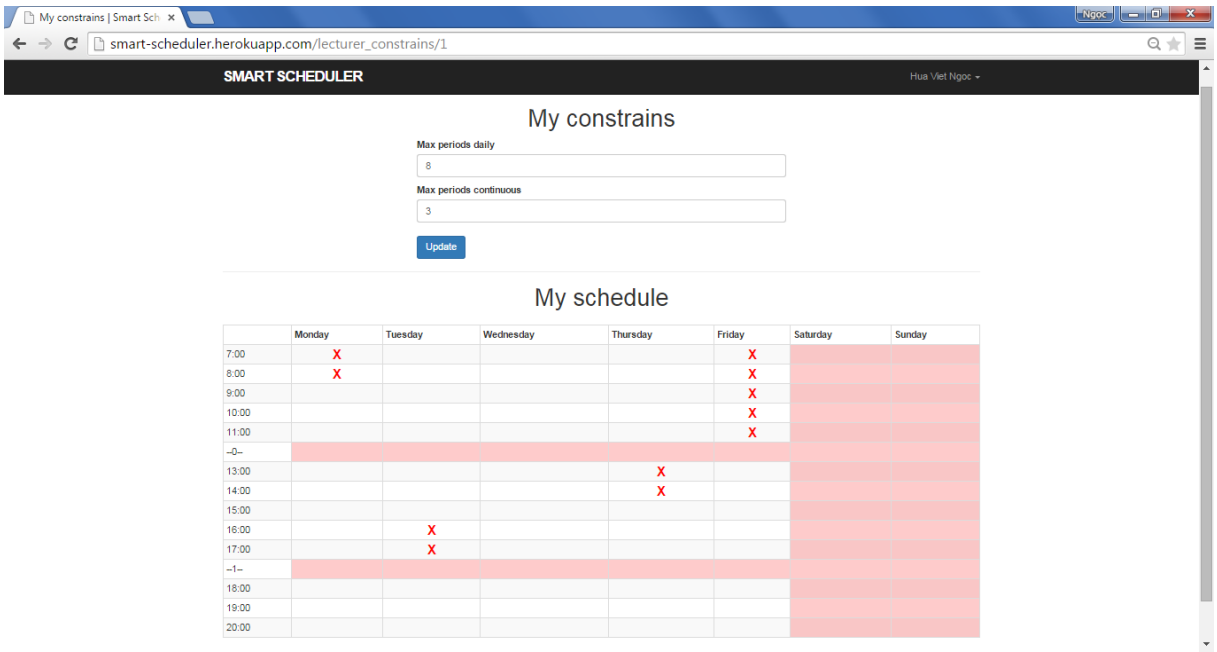

Figure 3.15: Lecturer constraint

Figure 3.13, 3.14, and 3.15 describe View of the several constraints (Non-overlap Constraint - A set of course not take place simultaneously; Space Courses - A course has a set of preferred Room; Lecturer constraint - The conditions teaching of lecturer, busy-time...)

RESULTS

4.1 Results

The system has just been implemented about 80%, but it has made the core functionality (scheduling). The interface is quite simple but not really convenient, display well on mobile.

Results for scheduling functions depend on FET, no customization and improvement, making the design model is still not close to reality.

The following table shows the result of the ability to perform scheduling task.

Lecturers	Courses	Rooms	Time-slots	Time to process
30	150	15	50	< 1 minute
40	200	20	50	< 1 minute
50	250	25	50	< 1 minute
100	600	50	65	< 5 minute

Table 4.1: Time to process of scheduling task

The data in this test is random as well as its constraints is not complicated so almost no take a long time in the process of scheduling. The system need to test more and use data sets in practice to get the most accurate results.

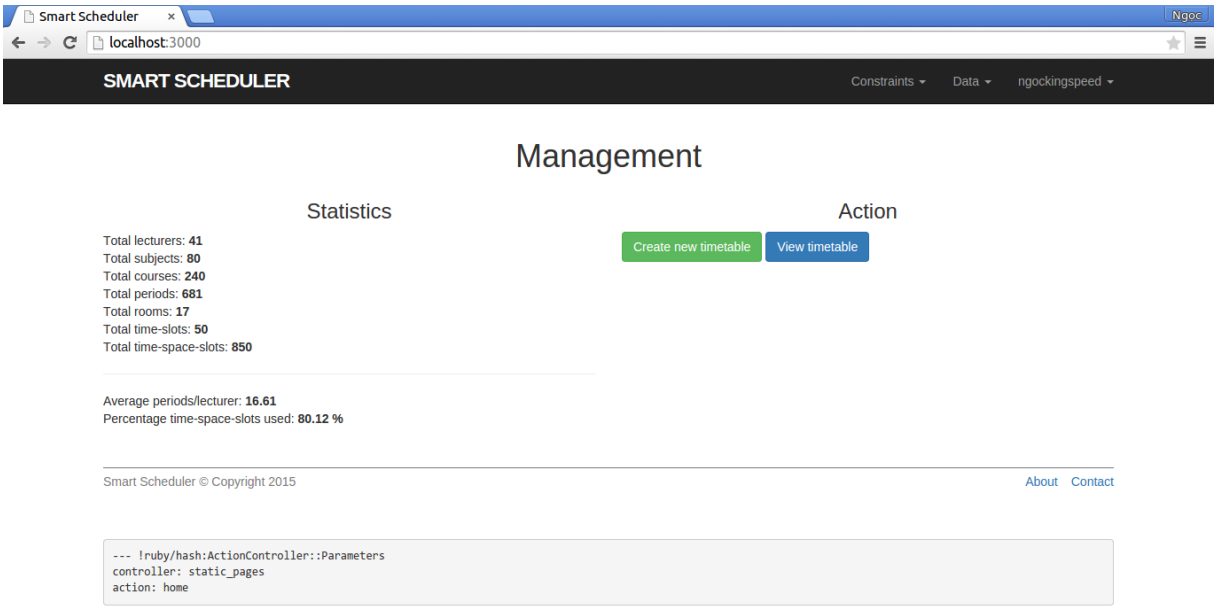

Figure 4.1: Screen for manager home

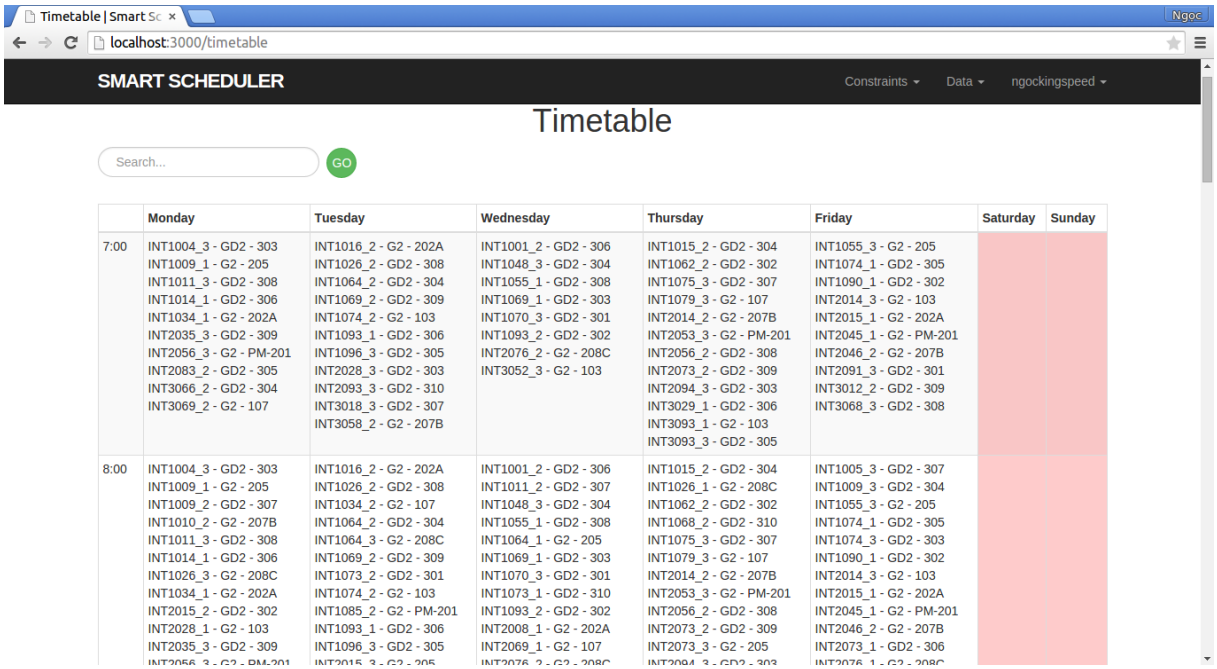

Figure 4.2: Screen for timetable

Figure 4.1 is manager home. There are a few useful statistical parameters and control to generate a Timetable. Figure 4.2 is describe a timetable, we can search timetable for a subject, a course, a lecturer or a room.

4.2 Comparisons

If compared with some current software, the system obviously lacking in both functionality and user interface. However, this system has several advantages such as: environmental activities through internet, which means it can run on many devices, the installation will also become easier for users, even not even need to install anything. Scalability and great development, scheduling function is built on an independent system effectively handle this task. In addition, the environment internet will connect multiple parties. Creating schedules are subject to a few individuals but the data can come from many sides, as well as the sharing of data between the parties together.

CONCLUSIONS

5.1 Conclusions

Generally, the current scheduling problem is not new. There are a lot of software on the market solve this. However the development of a system as a web service is necessary.

My system was first step built, but there are many shortcomings, but it shows the feasibility of putting into practice deployment.

Through this thesis, I had to learn a lot of things and also have built part of the system. Include MVC architecture, work with the ROR framework, external system (FET) and a lot of other technologies as well as programming techniques.

Although much remains to be done but initial results show that the ability to develop the system is available. Below section is some of the future wok will need to continue research, innovation and deployment.

5.2 Future works

- Continue to implement the missing functionality
- Learn more about FET and improve its algorithms
- Design more detail about models and constraints
- Redesigned user interface appropriate and more convenient
- Support for more languages (Vietnamese...)
- Build more function: create timetable for final examination
- Build timetable with date time in practice (it means we have a timetable attach with the calendar, and can adjust some specific periods on specific date)

REFERENCES

[1] FET Free Timetabling Software [Online] http://lalescu.ro/liviu/fet/

[2] Ruby On Rails framework [Online] http://rubyonrails.org/

[3] Rails tutorial [Online] https://www.railstutorial.org/book

[4] MVC [Online] http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller