UNIVERSITY OF TROMSØ

INF-2900 Group Project - ScheduleIT

by

Andrea Spreafico - asp005@post.uit.no

Stephan Abrahamsen - sab004@post.uit.no

Eva García Domingo - edo015@post.uit.no

Emil Nysted Hagen - eha092@post.uit.no

Siarhei Kulakou - sku019@post.uit.no

A thesis submitted in partial fulfillment for the degree of Computer Science
Computer Science

in the

Faculty of Computer Science

Department of Computer Science

May 2017

Contents

Chapter 1

Introduction

1.1 Aim of the project

Thinking of actual students' problems, we tried to find a solution to some of them. We have developed a whole online platform for any group of people able to share a calendar. Our aim was to provide them with a tool that could organize different schedules, being able to show each schedule separately or as a whole calendar that includes all the events. Although our first target was students, the platform can be used by everyone who need an organizer. The only requirement a person needs is creating an account and he or she will be able to create a timetable and share it with people to join it.
People who join a timetable will be able to see the whole calendar but, obviously, they will not be able to edit it. by doing this, we have ensured the safety of a timetable. In addition, in order to maintain the confidentiality of our users, the content of the website cannot be seen by someone without an account.
With the aim of helping new users to create accounts and to learn how to use the application, we have completely filled the Help and About pages. Furthermore, in case some question could not be answered through these pages, an email account has been created for contacting us.

1.2 Aim of the course

One of the purposes of this project was to learn how to work together in a cooperative environment. Some of the challenges we have successfully faced to are: sharing code, dealing with different schedules or finding a common idea of what to do and how to do it. We have learnt that having a meeting once per week is unavoidable for sharing ideas,

problems and struggles, asking for help and being updated.
Finally, another important objective consisted on learning to use the framework Ruby on Rails. We found that it is such a powerful web developer, hard to use for the first time, but very sensitive to any change. That is, with so few code lines you can make big changes in your application. In addition, compared with others frameworks, it is quite easy to put together the logic with the graphics: Ruby helps you as much as a software can.

Chapter 2

Background Theory

2.1 Agile Software Development

Agile is a philosophy in software development, based on the idea of "iteration". That is, working incrementally, instead of doing everything at once. Each iteration consists on:

- Plan: What do the app have to do?
- Design: How we want to it to behave and show?
- Build: Code it
- Test: Try it
- Review: Is there something which does not work properly?

FIGURE 2.1: Agile diagram.
Source:https://www.linkedin.com/pulse/agile-lean-ux-anthony-miller

The *Plan* will be done by **User Stories**. A user story is a very simple way to write a thing which the software (the application, in this case), has to do. This story has to include the smallest problem possible. That is, is a problem can be divided into smaller problems, each of these lasts will be a story by itself. In order to write a correct user story, it has to answer to the following statement:

As a (role) I want (something) so that (benefit).

For example: *As a public user, I want to create a new account so that I will have got a profile*

A story also will have a priority (i.e. should, mandatory) and a estimated workload: in this project, a Fibonacci sequence from 1 to 13 has been used for this purpose (the easier story will de labelled with 1 and the most difficult, with 13). There are so many ways of doing user stories; this group have been using the web platform Pivotal Tracker. [1] [2]

2.2 Ruby on Rails

Ruby on Rails, or simply Rails, is a server-side web application framework written in Ruby under the MIT License. Rails is a model–view–controller (MVC) framework, providing default structures for a database, a web service, and web pages. [3]

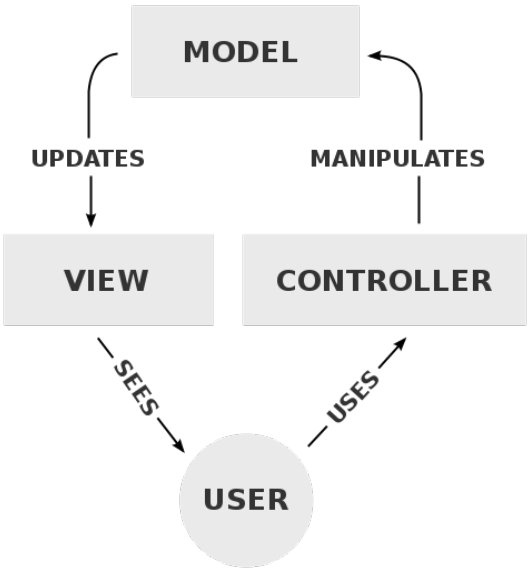

FIGURE 2.2: Diagram of interactions within the MVC pattern.
Source: https://en.wikipedia.org/wiki/Model-view-controller

The Model:

- Contains data for the application (often linked to a database)
- Contains state of the application (e.g. what orders a customer has)
- Contains all business logic
- Notifies the View of state changes (** not true of ROR, see below)
- No knowledge of user interfaces, so it can be reused

The View:

- Generates the user interface which presents data to the user
- Passive, i.e. doesn't do any processing
- Views work is done once the data is displayed to the user.
- Many views can access the same model for different reasons

The Controller:

- Receive events from the outside world (usually through views)
- Interact with the model
- Displays the appropriate view to the user

https://stackoverflow.com/questions/1931335/what-is-mvc-in-ruby-on-rails

Chapter 3

Analysis and Design

Concentrate on explaining the decisions made and the reasons for them.

3.1 Web application architecture

3.2 Models design

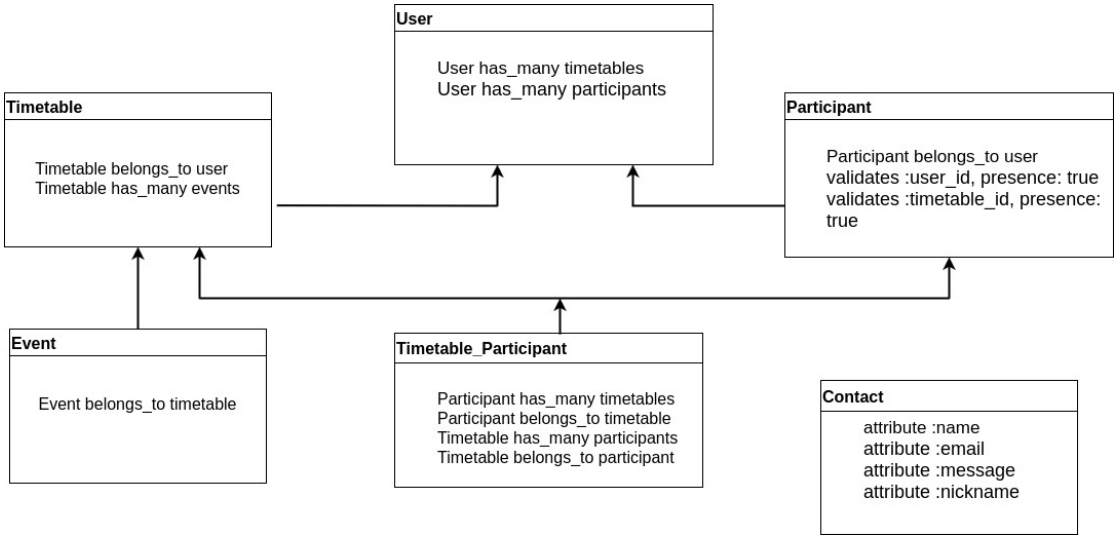

FIGURE 3.1: Implemented models.

[4]

3.2.1 User

The user model is responsible for keeping track of the application's user information. It is directly related to the timetable and participant models, because each user must be able to own several timetables, as well as participating.
The rest of the model is set up to ensure that the user information is correct. It validates the user e-mail, uses a Bcrypt function to ensure that all users have passwords, and in turn, hashes the password string so that no plaintext passwords are stored in the database. An added feature also allows the user to upload an avatar (profile picture) for their profile. [3]

3.2.2 Session

3.2.3 Timetable

The timetable

3.2.4 Event

3.2.5 Participant

The participant model was the last one on implementation, and the most difficult, due to the fact it has relations with two other models:

- Users: a participant is a user. That is, one-to-many relationship.
- Timetables: many-to-many relationship
  - A participant "belongs" to a timetable, but can participate in many timetables.
  - A timetable can have many participants.

Obviously, the fields for the foreign keys to user and timetable can not be empty.
The many-to-many relation was done separating the "belongs_to/has_many" instead using the formula "has_and_belongs_to_many" because this last one caused us problems in the admin page and when trying to use the foreign keys. These problems were caused because the use of the foreign key as *user.participant* were not different from *user.participants* (there were not two different relationships).

3.2.6 Contact

The contact model is a small and simple one, but it contains the necessary parts for a proper e-mail header. Whenever a user submits a new contact form, the model validates the fields of the form, as well as one "invisible" field, used for catching spam bots. This "spam catcher" works by refusing to submit forms where the field *nickname* contains any characters. Since the field being hidden by some CSS code, we know that a human can't see this field, unless they have access to the code, in which case it's most likely a robot of some kind.

Chapter 4

Team Work

4.1 How we apply Agile process

As Agile is based on iterations, this project has been organized in three of them. During each iteration, the *plan-design-built-try* process has been developed in the best way possible, learning from errors and tried to improve the process from one iteration to another.

4.1.1 First iteration

We started the project deciding **what** we wanted to do; the aim of the application. We also decided that the fist iteration should be for the general structure of the platform: main page, basic static pages and basic functionality. For designing, we draw the different pages and how they should look, behave and connect between them. Also, we decided and did a schema of the structure of the models (see Figure 3.1).
After design the basic structure, we did the user stories for that structure in a shared document (via Google Drive). For example:

- As a public user, I'll be able to reach the web application site. 1 point
- As a public user, I'll need to be able to create a private user. 5 points
- As an admin, I'll be able to see the list of users. 5 points

For coding, this iteration was the most difficult one, as far as we had to learn Ruby on Rails from the very beginning. For the first steps, we used a Ruby tutorial which help us to learn how to start a project in Rails.
Due to the hard start, we ended the first iteration doing around 80% of the user stories

we had at the beginning. Thus, we re-scheduled the work plan, checking which worked and which did not.

4.1.2 Second iteration

We started this with the basic structure implemented; we planned the amount of work we could face in the time we had and did the corresponding user stories. This time, we tried to be more effective and organized and started to use *Pivotal Tracker* for user stories. The improvement of using this platform instead of *Google Drive* is explained in Section 4.3.2.
The objective of this iteration was to do the real implementation of the application. That is, all the functionality of the platform, as we had thought it in the beginning. We re-planned (and draw it again) the pages, but this time with the non-basic behaviour we wanted. For example, this was the time for implement the ability of an user of joining a timetable, or for letting a timetable to have several events.
In terms of coding, that meant ending the database and relationships between the models, and being used to the capacities of Ruby on Rails. The most powerful tool of Ruby we found is the use of foreign keys; the way you can pass though all the models in one variable, and the capacity of mixing html with Ruby's code.
Even though we couldn't achieve all the implementation scheduled, we did almost everything and had to re-plan just a few things for the third and last iteration. In order to plan it properly, we did a document with all the errors or bad behaviours we could find. This document was not unique, but was for re-writing it while we used more and more the application, during the whole third iteration.

4.1.3 Third iteration

The work planned for this one was separated in three parts: 1) to end with the re-scheduled work from iteration two (e.g. the contact form); 2) to do the graphics of the website and 3) going through the app and fix all the small "problems" (e.g. a problem with the admin page, or a missing *Help* link). As before, we used Pivotal Tracker for the user stories, for being conscious of the remaining work. During this iteration, we did "small" agile iterations, due to the fact we were fixing problems and redoing the code.
We ended the iteration with the application working properly (except a graphic issue in the about page) and all the expected functionality implemented.

4.2 Communication between the team

The communication and meetings had been an important issue, because the different schedules, lectures and availability or each member of the group. Even though, we considered important to meet once per week and tried hard for achieve it. We used a chat group for arranging the best time for meetings; this group also was used for being updated about the work (the user stories) others members were doing (in addition to Pivotal Tracker, as we will see later).
During the meetings, we helped each others with code problems that we found and could not face with no help. Also, we used the time of meetings for self-coding in group, and update the amount of pending tasks. The group-coding time resulted very productive, and we found that during this time we did many tasks. Because of that, along iterations, we have spent more time working in group (although most of the time was for self-coding).

4.3 Instruments and applications used

4.3.1 Git

The main tool for sharing code (and not only for sharing) in software development is **Git**, a version control system which principal purpose is to maintain the control over different versions of the same project; that is, being able to get back "in time" if something went wrong. Also, it is a powerful tool for update every small change done in the code, and everybody could have the last version of the project.
During the three iterations, Git have been used for the control of the project. First iteration was very useless in respect of updating the changes, and it resulted in problems with merging each one's version. We learnt from our mistakes, and tried to improve in pushing every small change in the code, in order of not having merging problems and working with the last version. We used also the team times for merging, with the aim of being able of ask about other's code.

4.3.2 Pivotal Tracker

As we have explained before, we did user stories in a shared document in the first iteration; it resulted not very productive. Thus, we looked for some tool to improve the usefulness of user stories. We found Pivotal Tracker, a website that let you start a project and add user stories to it. They have the format before explained (see Section 2.1), and also let you give each story Fibonacci points.

Chapter 5

Discussion

Chapter 6

Conclusions

What have you achieved? Give a critical appraisal (evaluation) of your own work - how could the work be taken further (perhaps by another student next year)?

6.1 Evaluation

6.2 Further Works on the project

6.3 Further Works on the team work process

Bibliography

[1] Agile modeling. User stories: an agile introduction. http://www.agilemodeling.com/artifacts/userStory.htm.

[2] Jonathan Rasmusson. Agile in a nutshell. http://www.agilenutshell.com/.

[3] Wikipedia. Ruby on rails — wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Ruby_on_Rails&oldid=782105184, 2017. [Online; accessed 25-May-2017].

[4] draw:io. Draw io. https://www.draw.io/.