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 course

This Software engineering course has an important background goal, to learn how to work in teams to create software using an agile software development process.

Seeing as the development uses the web application framework Ruby on Rails, the teams will also have to learn to use this framework, discovering its potential and utilities. This includes using multiple programming languages like Ruby, HTML and CSS.

1.2 Aim of the project

One of the most difficult things in a students life is to properly plan meetings, lectures, sports, and other events, especially when these activities are shared between multiple people. For this reason the aim of this project is to implement an online scheduler. A scheduler is an instrument that organizes or maintains schedules, meaning lists of planned activities, keeping track of what needs to be done and when.

The initial idea was to provide an online service in order to help students with daily time organization issues, but it could also be used in other contexts, such as team training sessions, meetings between friends, and public events.

The implementation of this project will try to reach the following goals:

- Let people be able to create their own calendar and fill it with events.
- Allow people to share their own calendar with others.
- Let people be able to view and see changes in other people's calendar.

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?

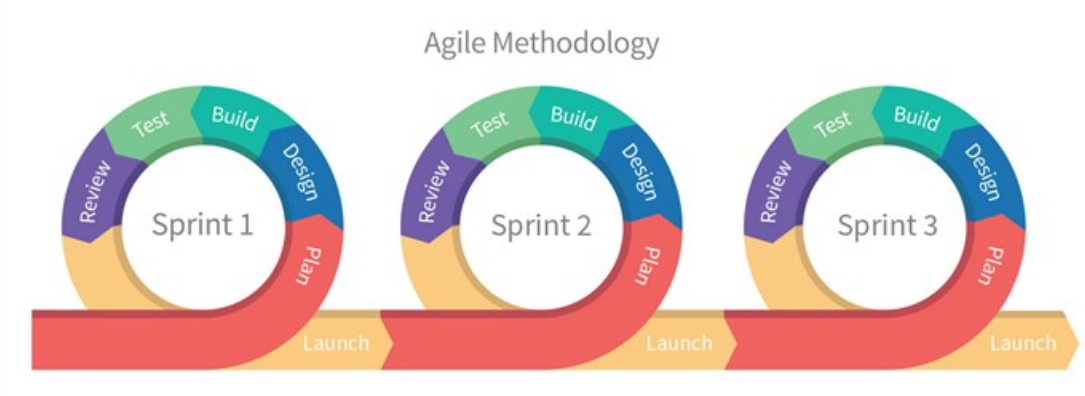

FIGURE 2.1: Agile diagram.
Source:https://www.linkedin.com/pulse/agile-lean-ux-anthony-miller

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]

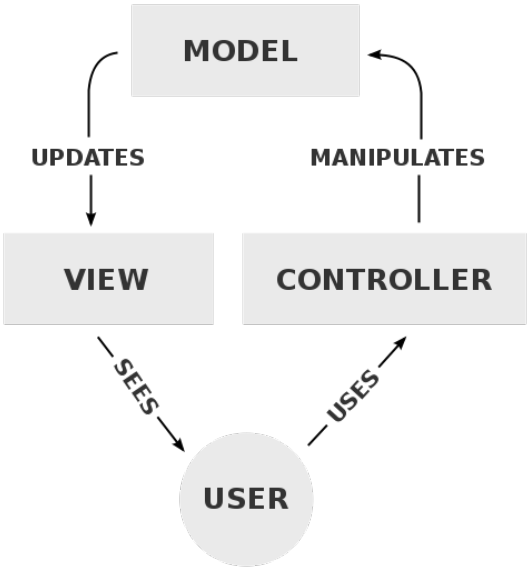

FIGURE 2.2: Diagram of interactions within the MVC pattern.
Source: https://en.wikipedia.org/wiki/Model-view-controller

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

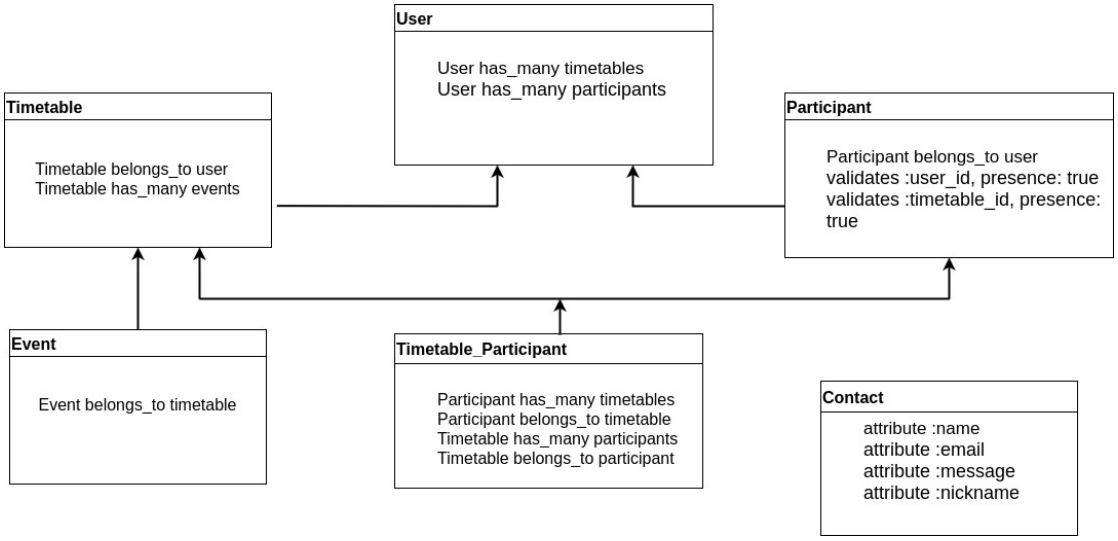

FIGURE 3.1: Implemented models.

[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 Timetable

The timetable model provide to the users the possibility to set up the general information about a specific schedule. The main utility of the timetable records is to group up the events of the same schedule, allowing users to find and check out the events which they are interested in.

3.2.3 Event

The event model allows users to create and record data about a new event of a specific timetable that they own. When ever a owner of a timetable wants to create an event, he has the possibility to do it using this model that provide several active record fields such as name of the events, date of the event, starting time, ending time..
Is important to remember that each single event belongs to a timetable.

3.2.4 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).

3.2.5 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 first 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 Team cooperation and communication

Promote cooperation and communication between team members is probably one of the most important thing during a group project development process. Most of the time it's hard to figure out a good way for cooperate and communicate between each other, and also during this project it has been an relative issue, in particular in the first part of the project. It was mainly caused by different schedules, lectures and available time

of each member of the team. Even that, every single team member agreed about the importance of frequent meetings, so we almost always planned a group meeting once per week. The main communication way between the team member has been a chat group: it was used for decide team meeting and also to update the other teams about ideas or news about the code. We basically spent the meeting's time:

- Helping each other with any kind of problems about the project
- Discussing new ideas
- Planning what to do in the following period
- Group-coding

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 usage

4.3.1 Git

During the whole project development **Git** has been an indispensable instrument.
Is possible to define Git as a version control system, which means that the principal purpose is to help a software team manage changes to source code over time. In particular, it keeps track of every modification to the code in a special kind of database and, if a mistake is made, developers have the possibility to turn back and compare earlier version of the code in order to fix the mistake.[5]
During this project, Git usage has been improved a lot between the first iteration and the last one. For example, during the first part of the project there were several problems about merging and push/pull of code, mainly caused by a lack of knowledge about how to use this system in a proper way. Then, once learnt from our mistakes, during the following parts of the project Git usage has been improved a lot and it provided a significant help to the team work. Probably the most appreciated Git function during this project was the possibility to always check out the latest version of the code, since all the team members were committing code quite often.

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. Furthermore, it let you add different tasks to each story, which makes very easy to schedule your work and to see

others' progress. Also, a person can "choose" a task, and marked it as *started*, for others know which tasks have already a "owner".

Chapter 5

Discussion

Is possible to say that the obtaiend results are achieving most of the initil goals of this project. Like reported in the introduction part [1], the two main goals of this work were:

- Implement a web application that provides a scheduler.
- Apply the development agile process.s

The resulting web application actually provides a online scheduler that contains several utilities and functions. Almost all the user stories that were supposed to be realized have been correctly implemented during this work. In particular, the results provide an answer to the initial goals of the scheduler implementation.

Further more, during the whole development process the agile approach had a high consideration and priorization from all the members of the team. Has been actually possible to see a big improvement about the agile process application between the beginning and the end of this work.

More details about evaluation and limitations of the implemented project can be find in the next section.

5.1 Evaluation and Limitations

The following list reports the main limitations that have been found during the implementation process.

- Time limitation has probably been the biggest issue. Working on a project with several people means that you have to somehow find a way to combine different schedules, that is not always that easy. Also a quite short project deadline implies a limitated extra features implementation.

- Limitated amount of previous knowledges and experiences about both Ruby on Rails and agile process imply that each single member of the team, mainly during the first part of the work, had to document himself about it. It means a significant investment of time and effort focused about get knowledge and less about web application's extra features.

Despite of the limitations that have been reported above here, this work provided to all the team member:

- New knowledges and experiences about agile process, in particular how to handle a group project, communication and collaboration with other people.
- Improved abilities about gitlab/github usage, since it has been an indispensable instrument during the whole work.
- Fist but really complete approach with framework Ruby on Rails.

5.2 Team efforts and Time lists

One of the most important thing during the whole development process was splitting the jobs between the team members in a efficient way and at the same time let every single team member put the same effort into the project.
Was actually asked to every single team member to constantly contribute at the work as much as possible. How it's possible to see in the following graphic, there has been a quite constant effort and work on the project.

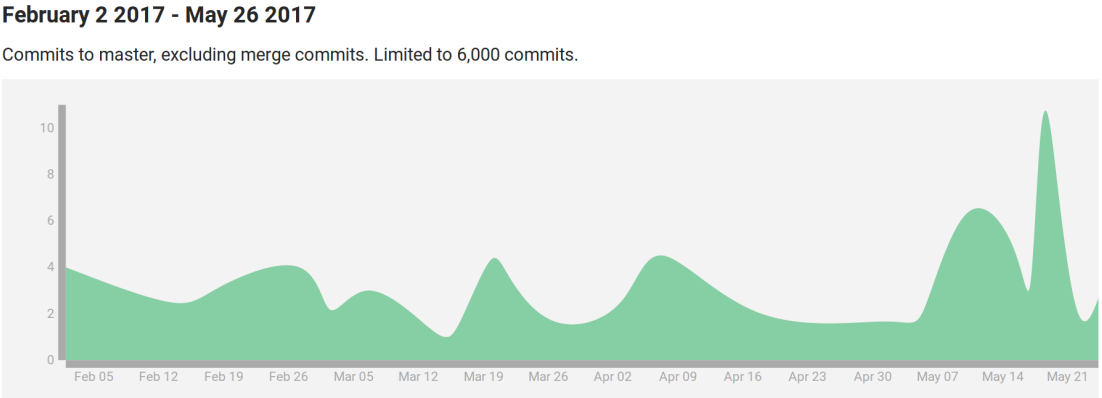

FIGURE 5.1: Commits to master, excluding merge commits.
Source: https://inf2900v17.cs.uit.no/team1/coffee-overflow/graphs/master

Chapter 6

Conclusions

6.1 Summary

What have you achieved?

6.2 Further Works on the project

- **Develop the web application like a smartphone application**
- **Public/Private schedules**

  - Send participation requests.

  - Ask to join
- **Possibility of comment events**
- **Possibility to follow a single event instead of the whole timetable**

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/wiki/Ruby_on_Rails, 2017. [Online; accessed 25-May-2017].

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

[5] Atlassian. What is a version control. https://www.atlassian.com/git/tutorials/what-is-version-control.