POLITECNICO
MILANO 1863

Design Document

Travlendar+

Federico Betti - Tommaso Bianchi

Deliverable:	DD
Title:	Design Document
Authors:	Federico Betti - 899914
	Tommaso Bianchi - 894183
Version:	1.0
Date:	November 25, 2017
Download page:	Github
Copyright:	Copyright © 2017, Federico Betti - Tommaso Bianchi
	All rights reserved

Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose

This document is a Design Document (DD). The purpose of this document is to describe the architecture of the system for the Travlendar+ project, giving both its high level description and a more detailed representation of the various artifacts that will compose it; this will be done primarily by means of various types of UML diagrams. In the last part of the document it will be also presented an implementation plan, outlining the main steps to be followed in order to build the system.
This document is addressed mainly to the development team who will be in charge of the implementation of the system, but it can be a useful reading to anyone interested in understanding how Travlendar+ is working.

1.2 Scope

Travlendar+ is a calendar-based service that helps users in managing the scheduling of their meetings, whether for work or personal reasons. The system will be able to automatically compute and account for travel time between meetings to make sure that the user is never late and to support the user in its travels by identifying the best mobility option according to its preferences.
Users can create meetings, and when meetings are created at locations that are unreachable in the allotted time, a warning is created. Travlendar+ should support a multitude of travel means, including walking, biking, public transportation and driving. A user should be able to provide reasonable constraints on different travel means. Additionally a user will also be able to specify flexible breaks, so that the system would then be sure to reserve enough time for them each day.
Some complementary services, such as planners or maps, already exists on the market; however they do not offer both the possibility to schedule events and to have meaningful information on how to travel between them.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Since the domain of our system remains the same, the terminology that we will use throughout this document is still the one we defined in the RASD document.

1.3.2 Acronyms

- **Design Document (DD)**: the present document.
- **Unified Modelling Language (UML)**: a specification defining a graphical language for visualizing, specifying, constructing, and documenting the artifacts of distributed object systems.
- **Commercial Off-the-Shelf (COTS)**: software and services that are built and delivered usually from a third party vendor.
- **Model-View-Controller (MVC)**: a design pattern.
- **Uniform Resource Locator (URL)**: a reference to a web resource that specifies its location on a computer network and a mechanism for retrieving it.
- **Application Programming Interface (API)**: a set of subroutine definitions, protocols, and tools for building application software.

- **User Interface (UI)**: the space where interactions between humans and machines occur.
- **User Experience (UX)**: a person's perceptions of system aspects such as utility, ease of use and efficiency.
- **Global Positioning System (GPS)**: a global navigation satellite system that provides geolocation and time information to a GPS receiver anywhere on Earth.

1.4 Reference Documents

The following list contains the main documents we have taken as reference.

- Assignment: "Assignments.pdf" (can be found in our Github repository here).
- RASD: "RASDv1.2.pdf" (can be found in our Github repository here).

1.5 Document Structure

The DD is organized into 6 main sections:

1. **Introduction**: this section contains an overview on the purpose and the scope of this document, toghether with a glossary with the definitions of the main terms we will refer to in the followings.
2. **Architectural Design**: this section contains a description of the architecture proposed for the development of the Travlendar+ project, described at different levels of abstractions mainly by using UML diagrams. This should be the base of all the implementation efforts and serve as a model to which the implementation should aim at.
3. **User Interface Design**: this section contains a more detailed description of the user interface of the system than the one briefly presented in the RASD document. This is achieved by using UX diagrams that describe the entirety of the interaction between the system and a user, toghether with a selection of mockups that should cover the most important pages and sketch the style of what will be our complete UI implementation.
4. **Algorithm Design**: this section contains the description and the pseudocode of an algorithm proposed to solve the problem of scheduling new meetings, a core functionality in our system and one of the most delicate.
5. **Implementation, Integration and Test Plan**: this section contains a description of the plan we will use to coordinate the implementation efforts, toghether with some indications on how to test the various components of the system and their interaction.
6. **Appendix**: this section contains some information about the work behind the drafting of this document, such as the amount of time spent by each of the authors, the tools used to produce all the material and a revision history that keeps track of all the modifications.

2 Architectural Design

2.1 Overview

This section will describe, at different levels of abstraction, the architecture of Travlendar+ system. We will use: component diagrams to show the structure of the various software components and their interactions in the form of interfaces; deployment diagrams to show how the software artifacts will be arranged in the various physical devices; sequence diagrams to give a deeper representation of the information flow between the various components and subsystems. At the end we will also provide a table to show how every requirement listed in the RASD document will be mapped to one or more software components.
Travlendar+ architecture will be structured around two main divisions:

- **Client Side/Server Side**: the first one will be located on the users' physical device, such as computers and smartphones, while the second one will be deployed in a hosted server; the main logic of the system will be run server side. With our architecture different clients will be supported with the same server application.
- **Internal Components/External Components**: the first ones will be implemented by our development team or purchased on the marked as standalone components and will be the core of the system, while the second ones will be operated through interfaced over the internet protocol. We will use external components whenever they will have better features and lower costs than the ones that we could develop internally.

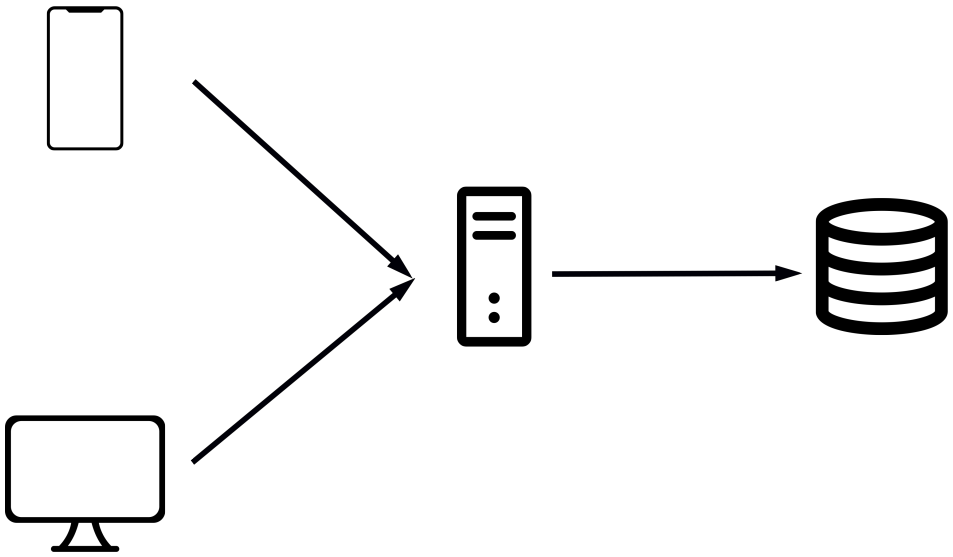

Figure 1: System Overview

2.2 Component View

The two component diagrams expose how the system architecture is build at two different level of abstraction: the High Level Component Diagram shows how the main subsystems are arranged and interconnected and what interfaces they use to communicate; the System Core Component Diagram, instead, contains a more detailed description of which software components have to be implemented to build the core of our system.
Under each diagram there are descriptions of the main functionalities performed by every component shown.

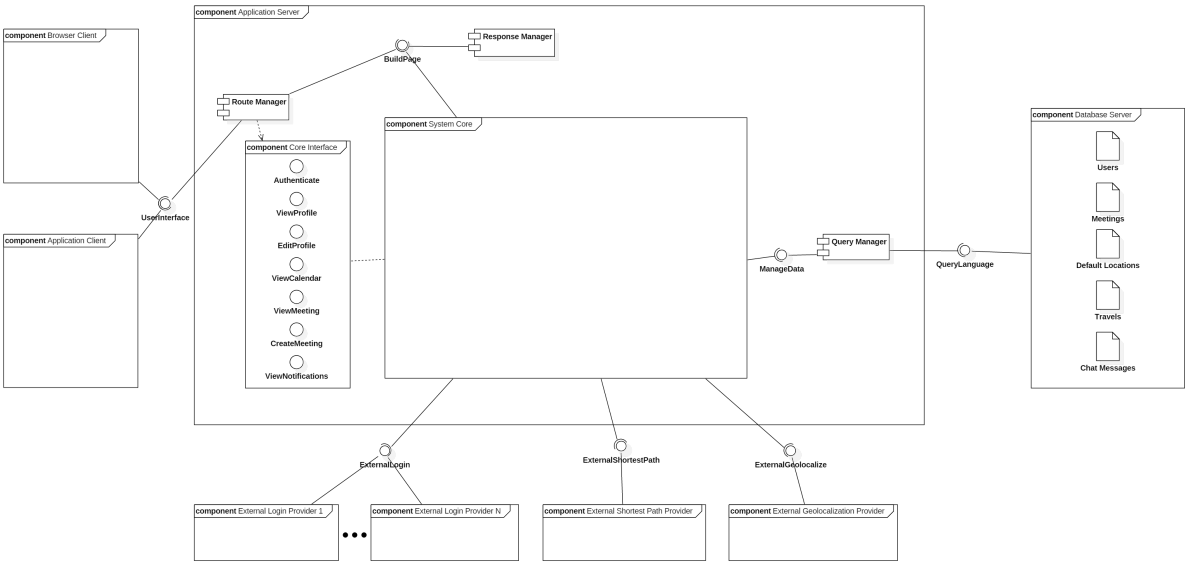

Figure 2: High Level Component Diagram

- **Route Manager**: manages the mapping between URLs and core system interfaces; it is the gateway for every request incoming from clients.
- **Response Manager**: builds the responses to send to clients, formatting them in the right way for both web browsers and client applications.
- **Query Manager**: manages the interface towards the database, allowing the System Core to interact with it in a more abstract way.

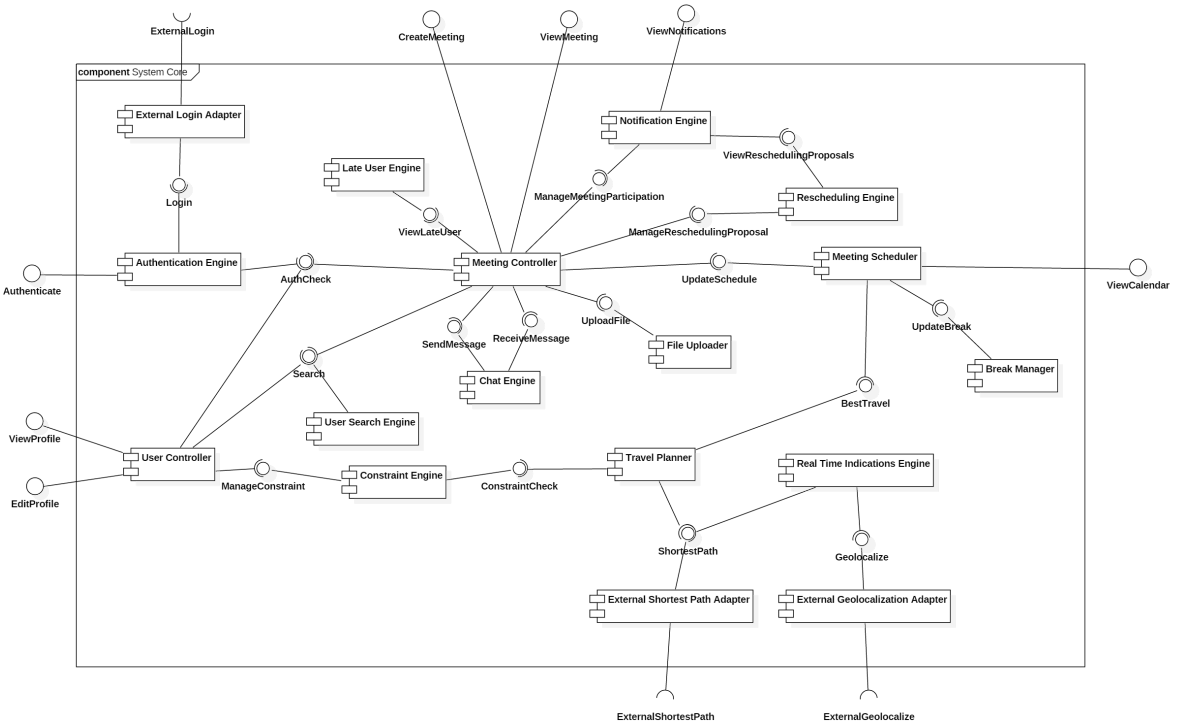

Figure 3: System Core Component Diagram

- **Authentication Engine**: manages everything related to registration and login and provides to the rest of the system an interface to check if a request comes from a user and, if so, from what user.
- **External Login Adapter**: manages the interface towards external login providers.
- **User Controller**: manages everything related to the users, such as their profile and their settings.
- **User Search Engine**: manages the search of a user in the system by some keywords, such as their email or nickname.
- **Constraint Engine**: manages all the constraints available in the system and provides an interface for the user to edit its own and an interface for checking if a specific constraint has to be enforced on a specific travel.
- **Meeting Controller**: manages everything related to meetings, from users' and administrators' perspective.
- **Chat Engine**: manages the chat system available in every meeting.
- **Late User Engine**: uses the data about the positions of the users to extrapolate who's late for a meeting and create a dashboard for administrators to visualize it.
- **File Uploader**: manages the upload of files for meetings.
- **Travel Planner**: computes the best travel between two given locations for a given user.
- **Real-Time Indications Engine**: manages the real-time indications given to a user to guide him to the next meeting.
- **External Shortest Path Adapter**: manages the interface towards external shortest path providers.
- **External Geolocalization Adapter**: manages the interface towards external geolocalization providers.
- **Meeting Scheduler**: manages the users' schedules and the insertion of new meetings into it.
- **Break Manager**: manages the flexible breaks updating their effective time whenever it's needed.
- **Rescheduling Engine**: manages rescheduling proposals and their responses.
- **Notification Engine**: manages all the notifications that are sent to users, such as warnings and meeting invitations.

2.3 Deployment View

The Deployment View diagram shows the hardware components and the software components that compose the Travlendar+ system. Our system is developed in three tiers that correspond exactly to the three layers that are illustrated in the diagram:

- The presentation layer on the client has two possible hardware components: PC Client and Mobile Client. The PC Client runs the Web Browser within which there will be deployed the Application Client, that has the aim to implement a light logic within the client. On the Mobile Client there are both the Web Browser, exactly as in the PC Client, and the Mobile Application of Travlendar+ , that aims to provide a user interface more precisely targeted to mobile devices.
- The application layer runs on an Application Server. It has the aim of managing the logic of the system, providing web pages or data for the application and communicating with the database. On the Application Server it must be deployed the Web Server within which the real System Backend runs.
- The data layer of the system operates on the Database Server within which the real Database of Travlendar+ runs. It has the role of storing all the data needed to run the logic of the system.

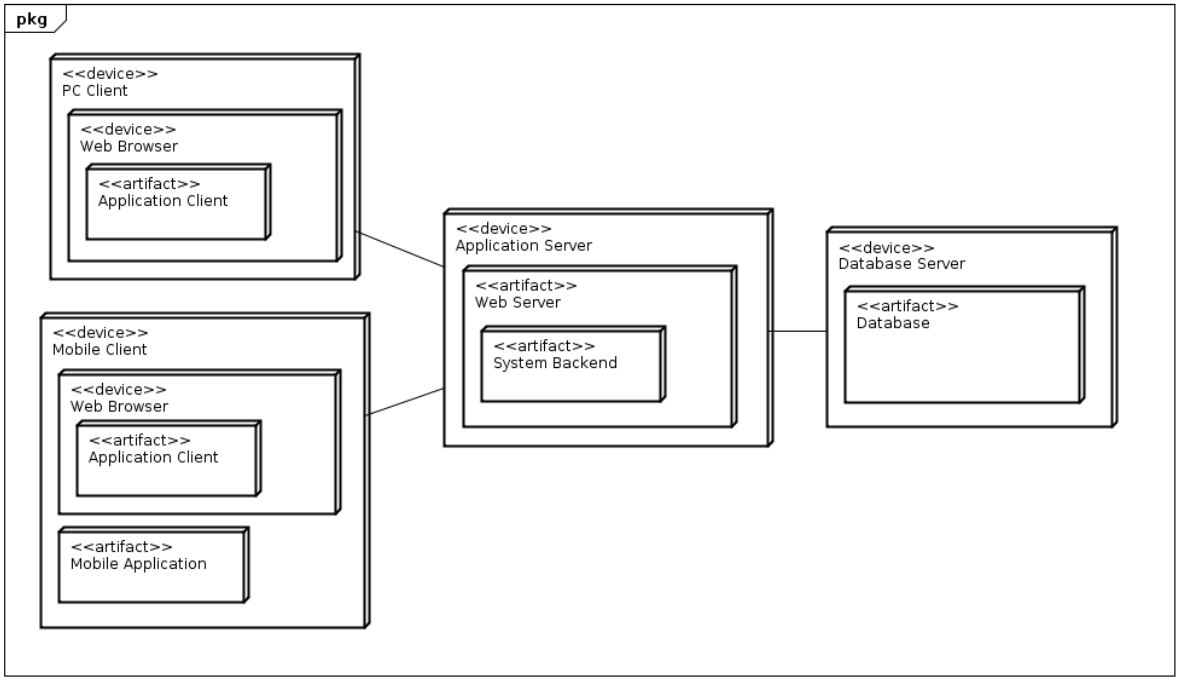

Figure 4: Deployment Diagram

2.4 Runtime View

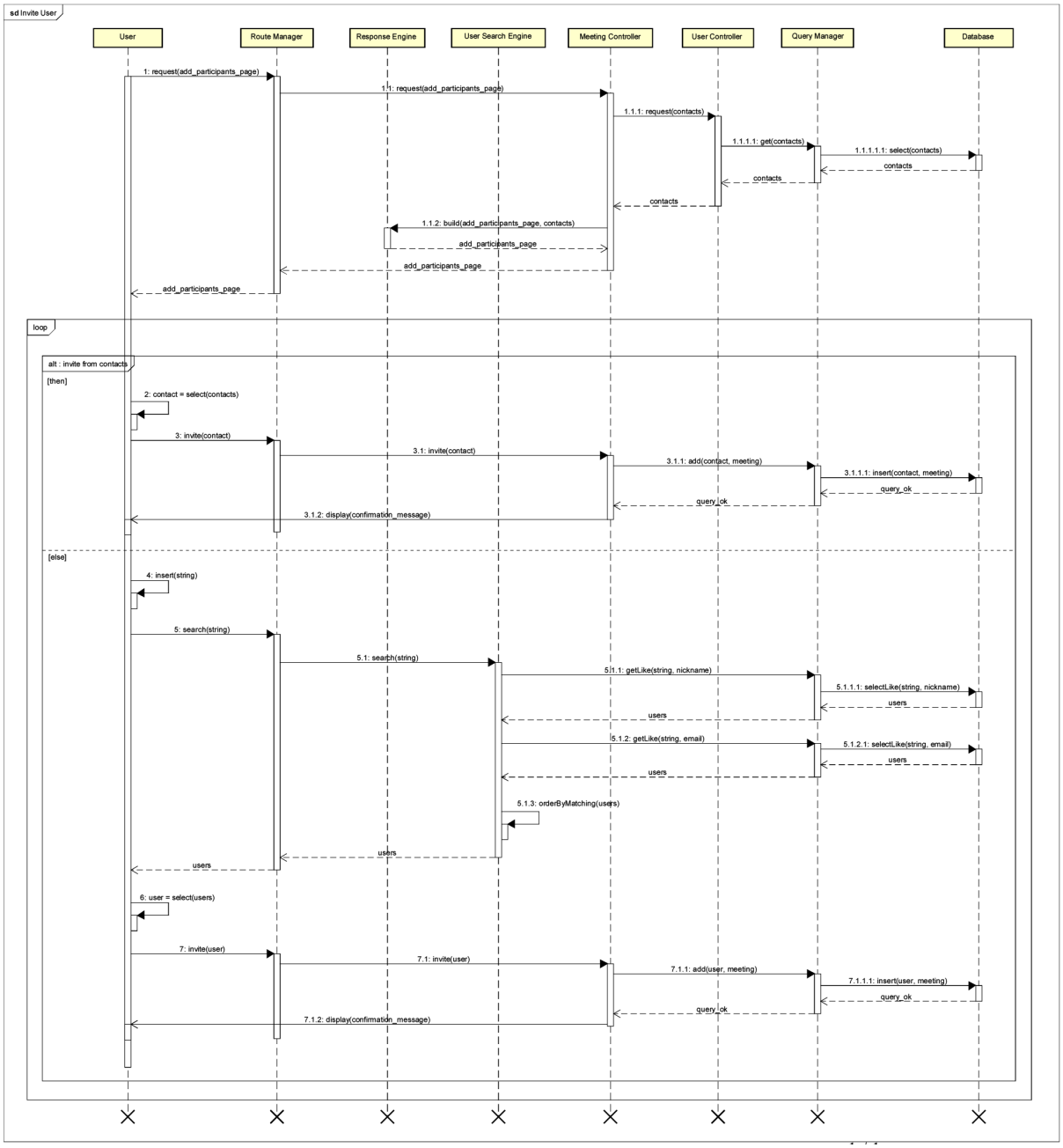

Figure 5: Invite User Runtime View

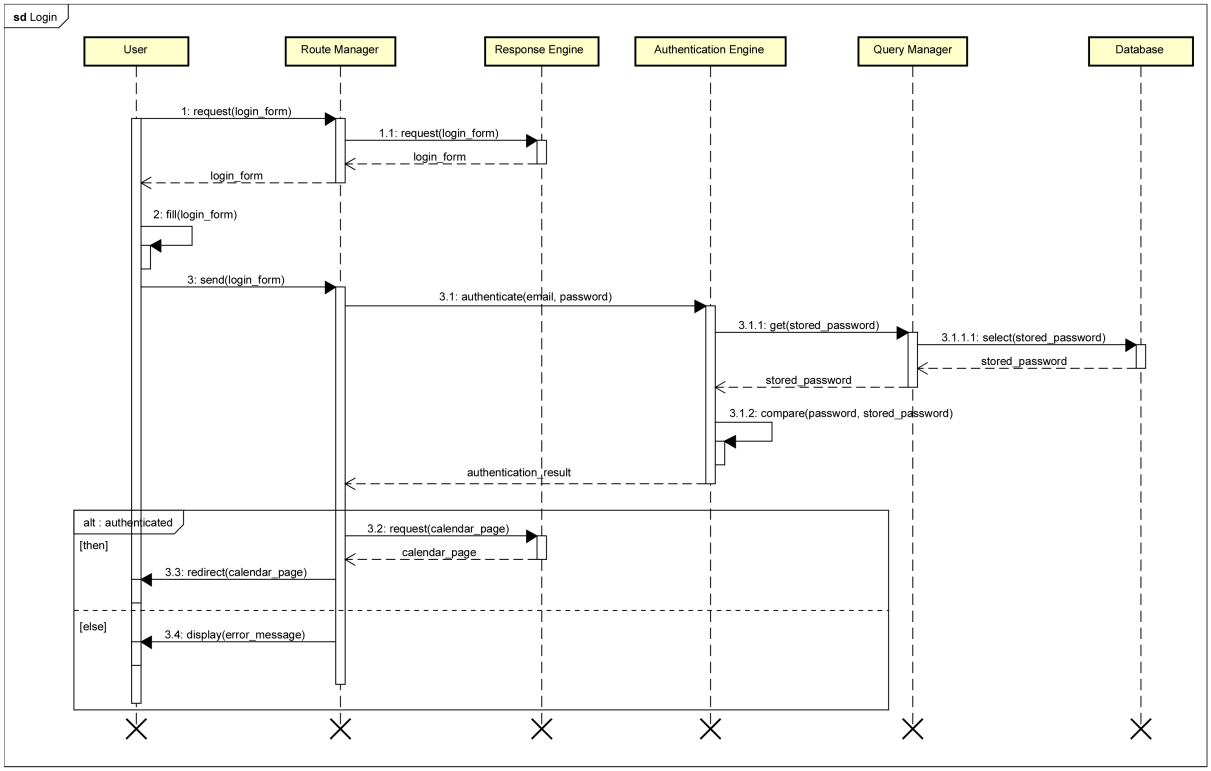

Figure 6: Login Runtime View

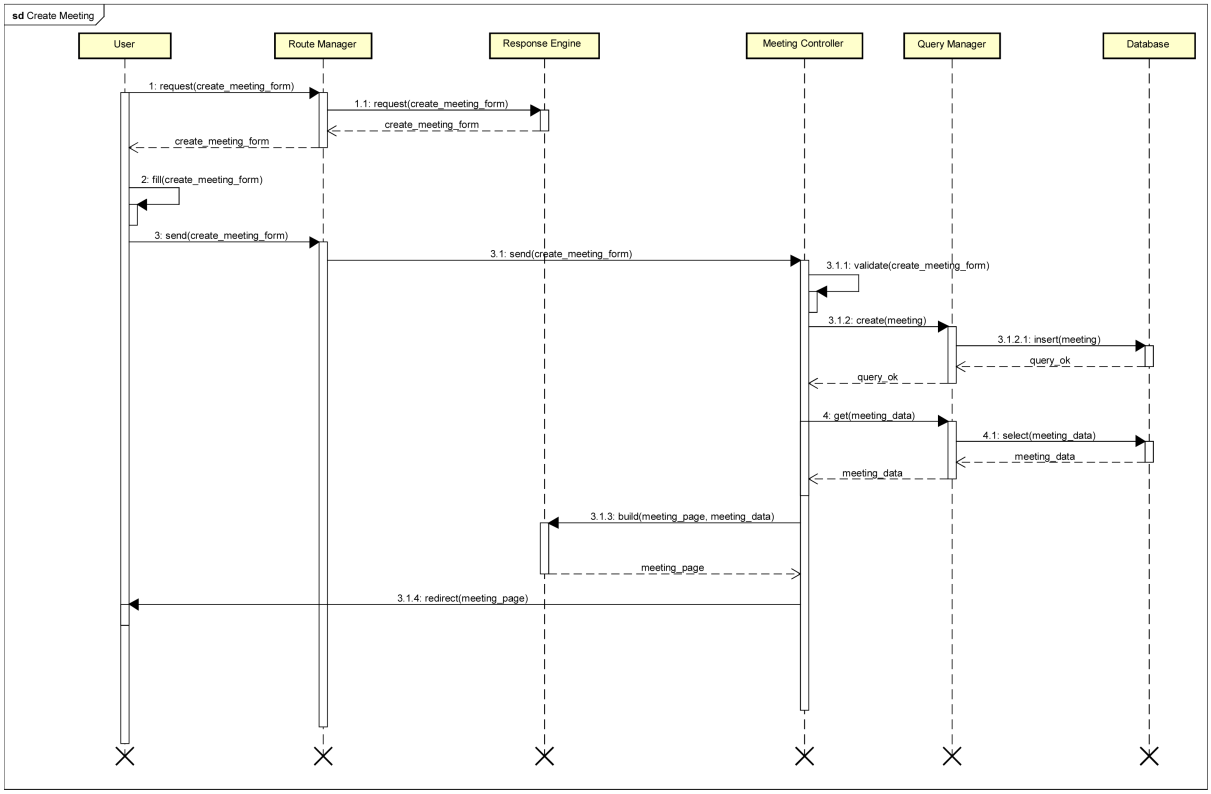

Figure 7: Create Meeting Runtime View

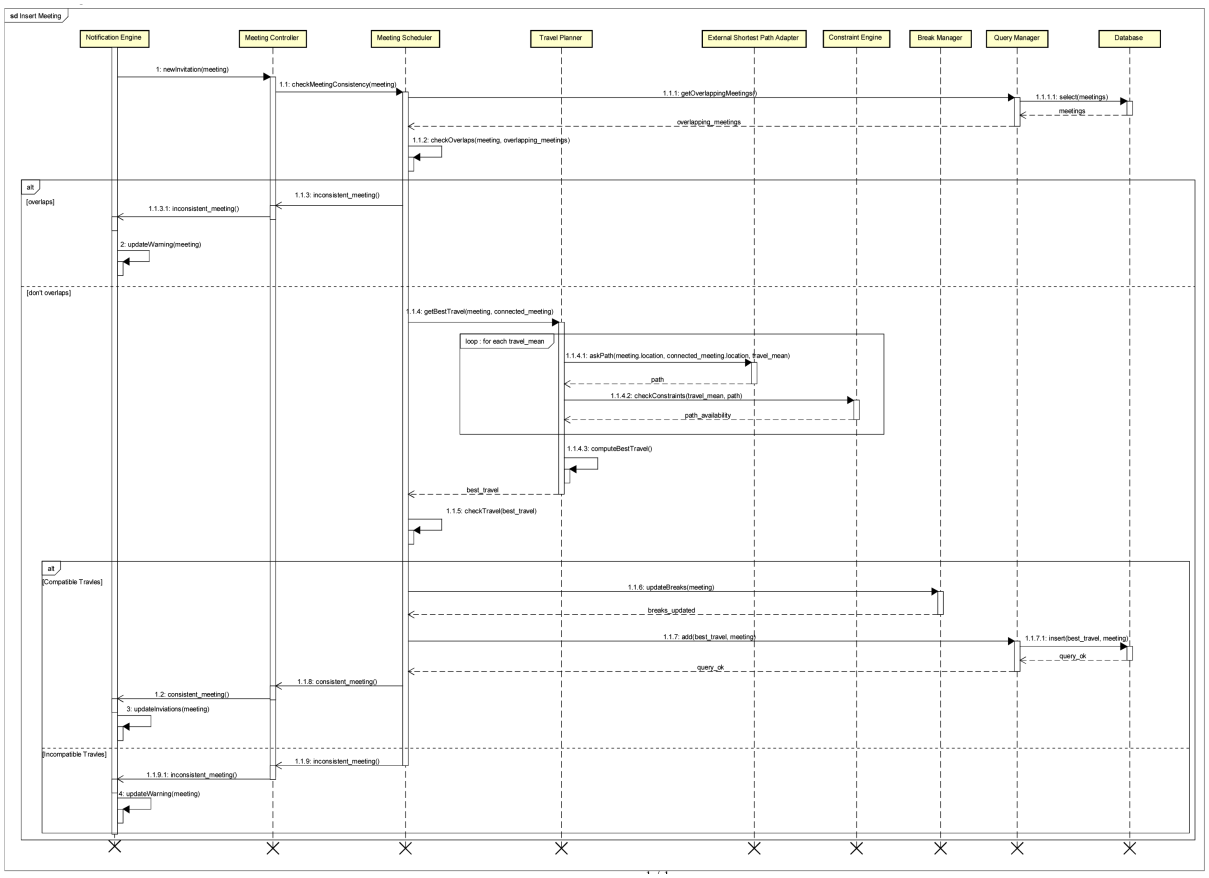

Figure 8: Insert Meeting Runtime View

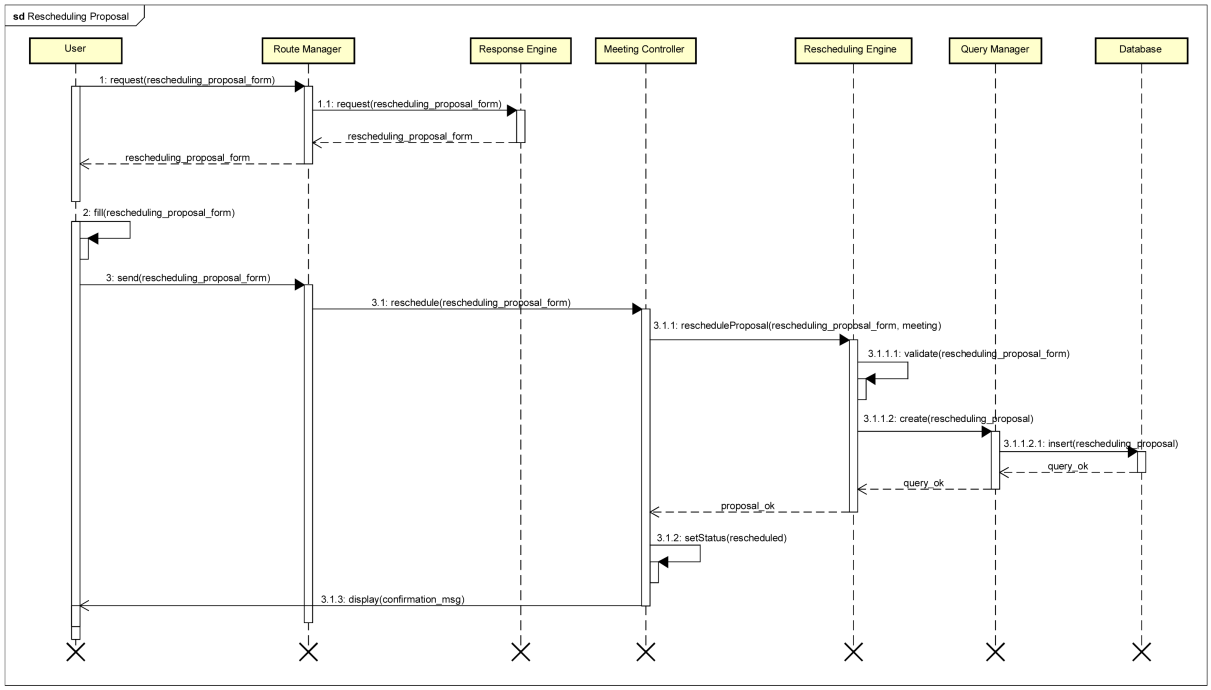

Figure 9: Rescheduling Proposal Runtime View

2.5 Selected Architectural Styles and Patterns

2.5.1 Architectural Styles

The main architectural style adopted for Travlendar+ system is the client-server one, the most well known and used architectural style for distributed applications. It will be adopted in the 3-tier variant, with the presentation layer on the client (the web browser and the mobile app), the application layer on the web server and the data layer on the database server. The main advantages of this choice are the clear decoupling between data and logic, the possibility to increase the portability reaching clients through their web browser and the availability of a lot of COTS components to develop the system in a very cost-effective way.
Since we will have to develop a chat component for the meetings and the real-time indications components, the client will have to be a little more than a thin client: in fact, it will be enhanced by some lightweight logic in order to support long polling or HTTP server push.
Those components have some characteristics similar to microservices, but we decided not to implement them in this way because they are not fully independent from the main system. Thus, the communication burden would have been too much for our performance target.

2.5.2 Design Patterns

The main design pattern that will support our client-server architecture is the MVC (Model-View-Controller), because it closely follows the division between data (the model), logic (the controller) and presentation (the view) present also in our 3-tier architecture.
The observer or publish-subscribe pattern will also be used, allowing the various components of the system to register themselves and react to event raised by other components. This will be useful in implementing the real-time indications components, that needs to activate itself only when the beginning of a travel is close.
Finally, the adapter pattern may be followed in implementing the interface with the various external providers, as they will all provide different interaction protocols that need to cooperate with our system in the same way; for example, we will have multiple External Login Providers, but our system will operate with one or another in a very transparent way.

2.6 Requirements Traceability

Requirements	**Components**
R1 Each user must provide an email that is not already present in the system	Authentication Engine, User Controller
R2 Each user must provide a nickname that is not already present in the system	Authentication Engine, User Controller
R3 Users have to be registered into the system before logging in	Authentication Engine
R4 Users have to provide an existing email or nickname and the associated password in order to log in; in alternative they can use a supported external login system	Authentication Engine, External Login Adapter

R5 A user must be logged into the system to perform any action except registering and logging in	Authentication Engine, Route Manager
R6 Users cannot have meetings while their status is set to auto-decline	User Controller, Meeting Controller
R7 A user cannot have different default locations sharing the start time	User Controller
R8 Time travel between subsequent default locations should be less than the difference between their start time	User Controller, Travel Planner
R9 Each meeting that is not an Instant Meeting has at least two participants	Meeting Controller
R10 Each meeting has at least one administrator	Meeting Controller
R11 Each meeting has a title, a date and a location	Meeting Controller
R12 Each participant in a meeting can access shared files and the chat	Meeting Controller, File Uploader, Chat Engine
R13 Users participate in a meeting if and only if they accept the invitation	Meeting Controller
R14 Users do not participate in a meeting if they decline the invitation	Meeting Controller
R15 Users can write in the chat of a meeting if and only if they have received and accepted an invitation to it	Meeting Controller, Chat Engine
R16 For each meeting there is a warning iff the meeting is inconsistent	Meeting Scheduler, Notification Engine
R17 The system suggests you a time, according to your settings, to have a break such that no meeting overlaps with it; if no time slot is valid, a warning is generated	Meeting Scheduler, Break Manager
R18 At least one travel mean is available in the preference list	User Controller
R19 The travel mean suggested by the system is always the first in the weighted preference list that satisfied all the constraints; if no travel mean satisfied all the constraints than the system suggests the fastest one	Travel Planner, Constraint Engine

Table 1: Requirement Traceability

3 User Interface Design

3.1 UX Diagrams

User Interface Design is an important aspect during the developing process of a system. It has the aim to model the part of the system that is related with users, hence it must be simple and efficient in terms of accomplishing users' goals. Since users' schedule is the crucial point in Travlendar+ , we thought to use the calendar page as home page of the application in order to facilitate the interactions that the user should do more often. Additionally UI of Travlendar+ should be concise, responsive, professional and good-looking for the purpose of satisfying all the possible type of users that could interact with the system.

In our charts, screens marked with * are referring to other ones that will be analysed more accurately in other diagrams.
We are going to present 5 different UX diagrams:

- **Registration/Login**: UX diagram of login and registration phase that can be done also using external login providers. During the registration process users have to complete an input form inserting default locations, a preference list and a nickname. This is a crucial phase in order to guarantee the correct behaviour of all other system functionalities.
- **Calendar Page**: UX diagram of the home page seen by a user after the login process. It shows meetings, travels and gives the possibility to the user to move inside the system.
- **Settings**: UX diagram of the settings page of the system. It shows all the customizable features such as the preference list, default locations, constraints, statuses and breaks.
- **Notification**: UX diagram that shows how notifications are managed by the system and how users can interact with them.
- **Manage Meeting**: UX diagram that shows all screens and available actions about a meeting, as seen from an administrator's point of view.

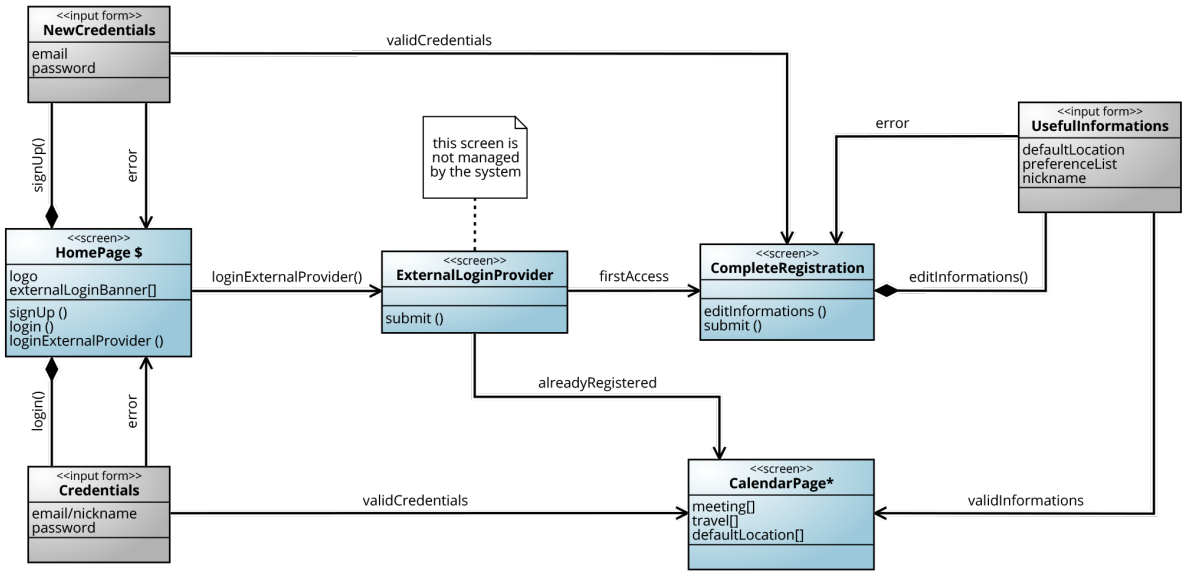

Figure 10: Registration and Login UX

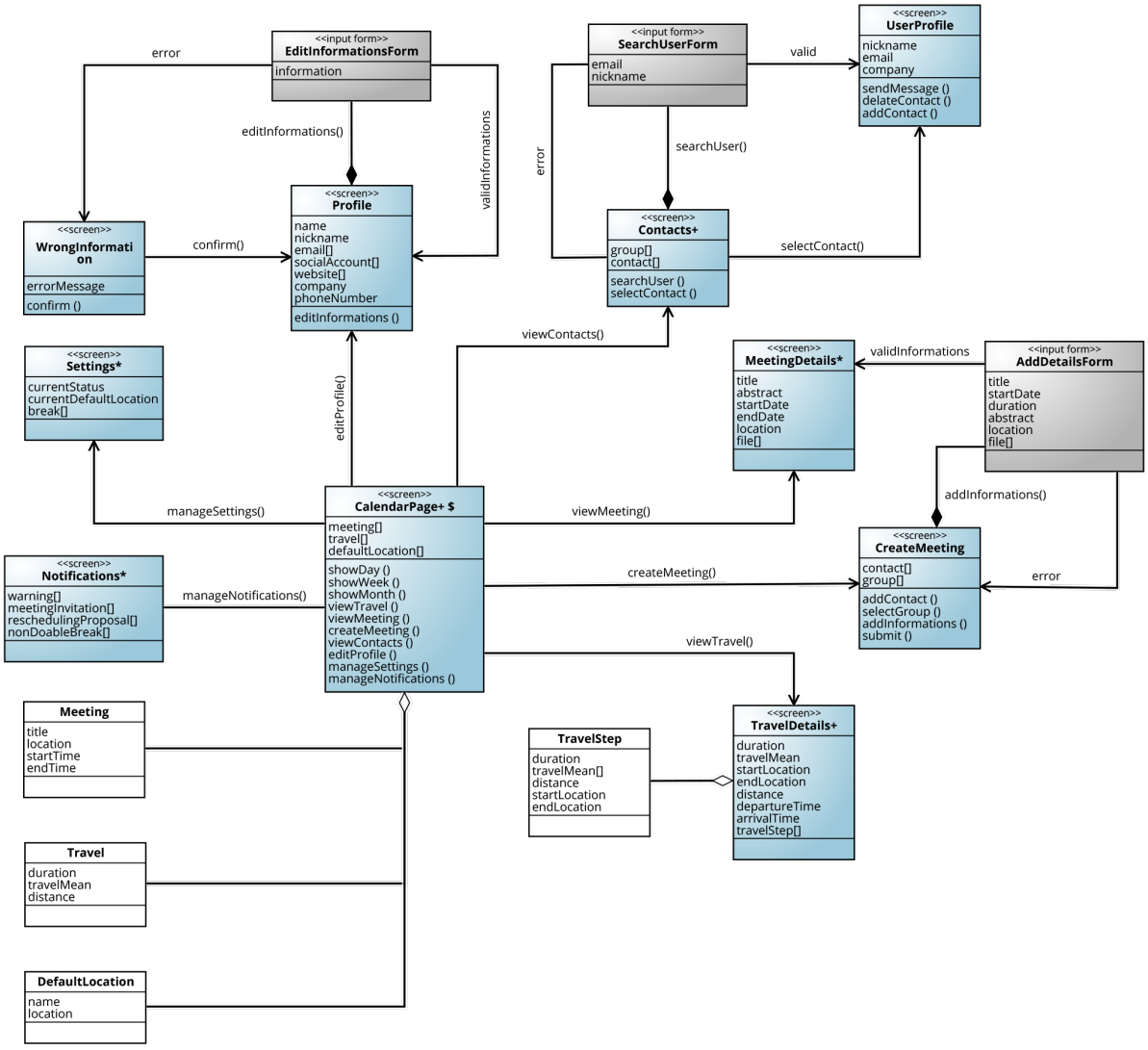

Figure 11: Calendar Page UX

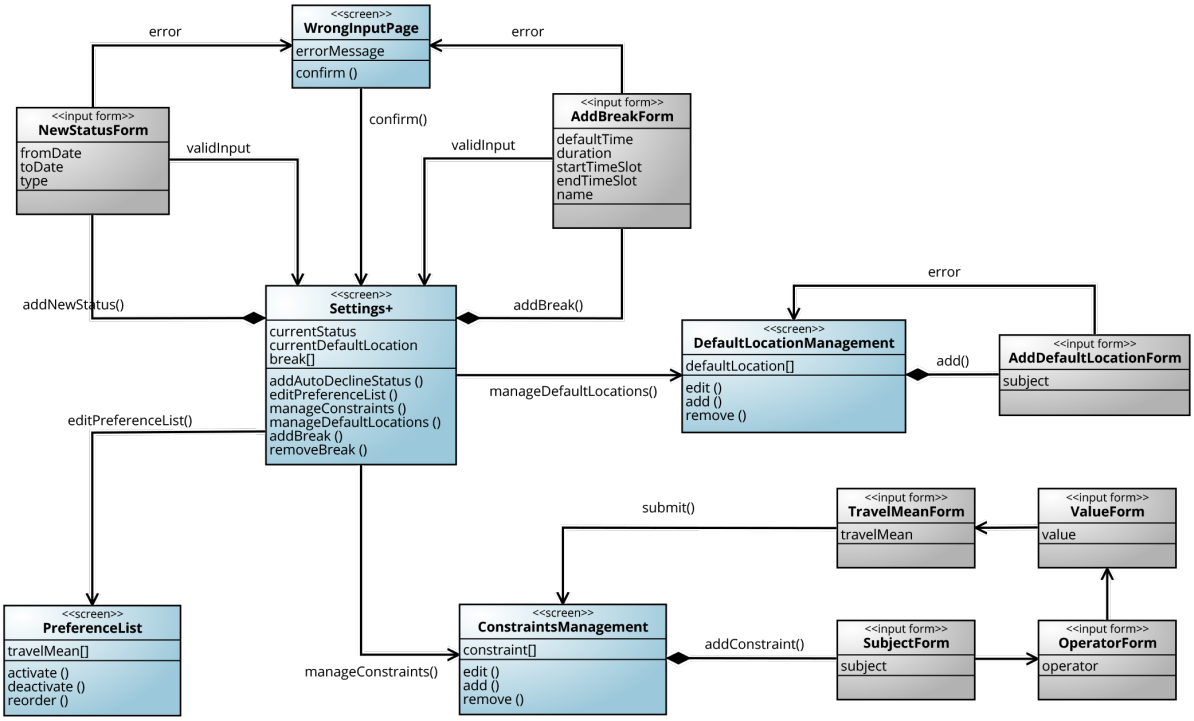

Figure 12: Settings UX

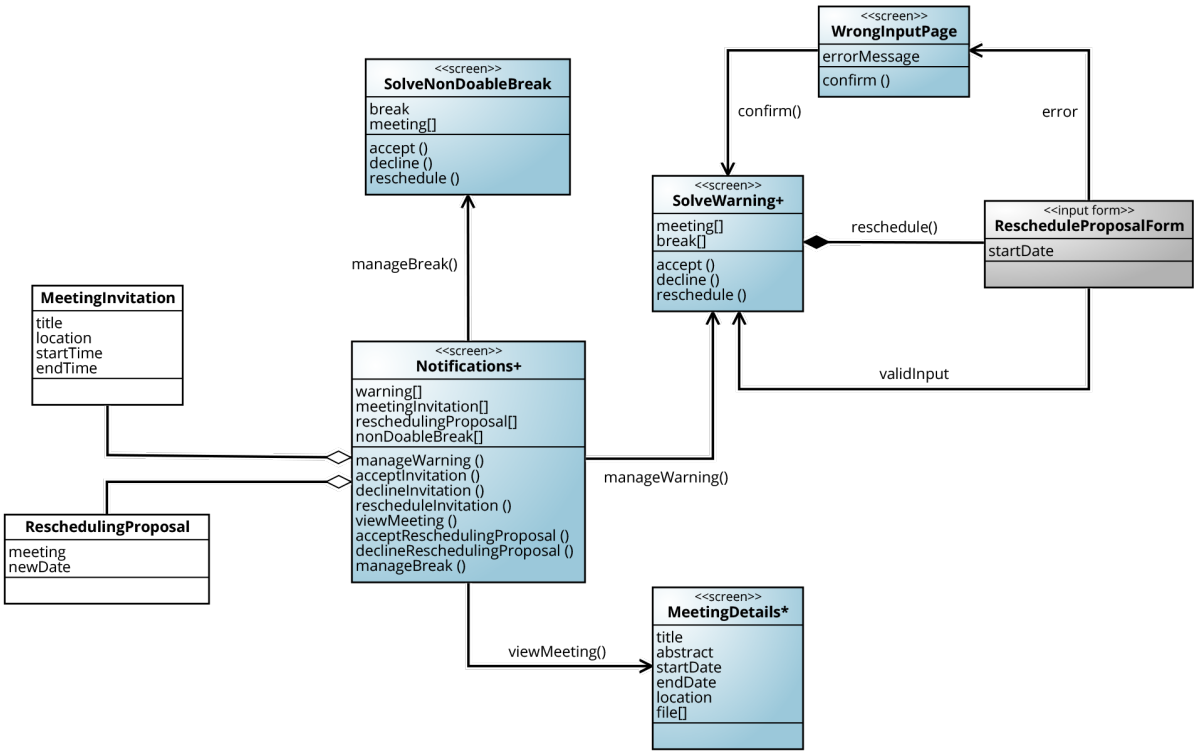

Figure 13: Notifications UX

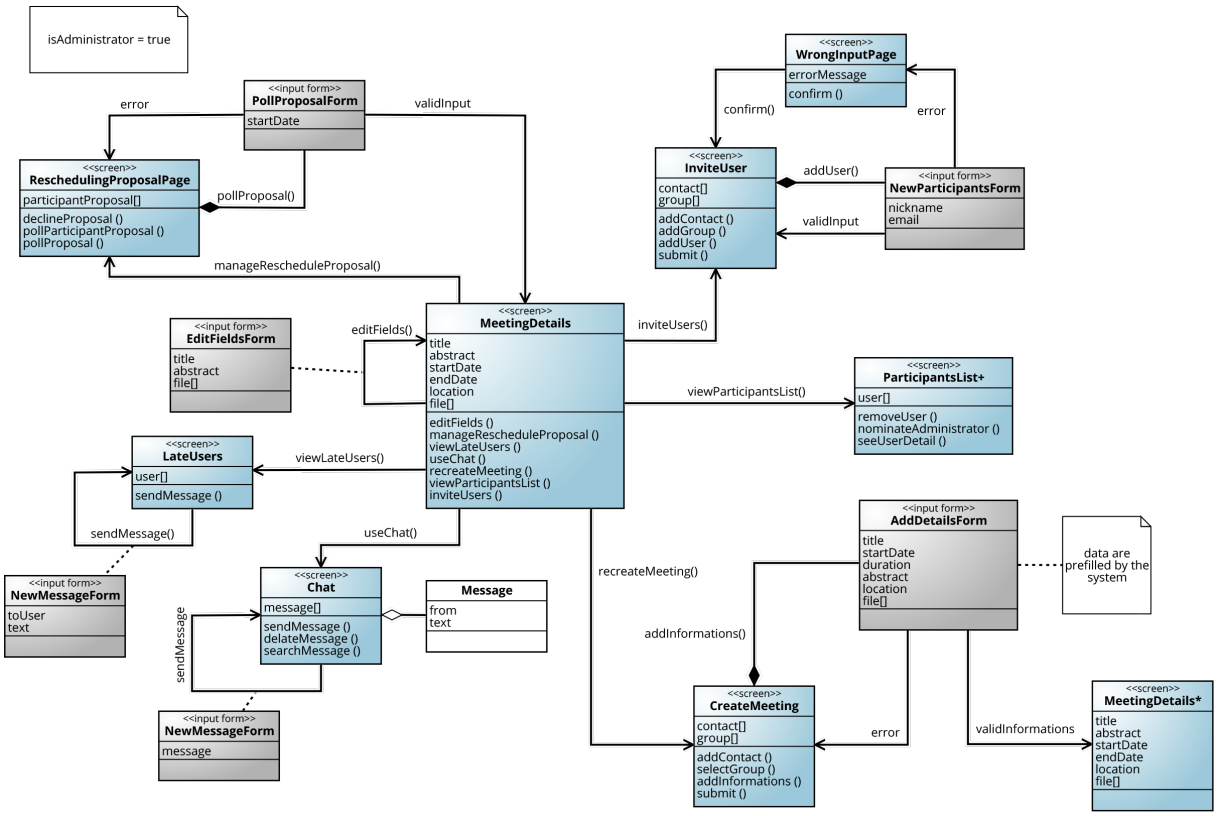

Figure 14: Manage Meeting UX

3.2 Mockups

In this section we are going to show some mockups of the system regarding the mobile application. On the RASD document we have presented some web application mockups and how they can be derived from mobile ones; for this reason we suppose UI developers will be able to derive correct web application mockups from the following ones.
During the design of mockups we have followed exactly the structure that is presented on UX diagrams; we won't show all the screens that are on the UX diagrams, however a lot of them have been created trying to clarify precisely how the user should interact with the system.
The mockups we are going to present are the following:

- **Homepage**: This mockup shows the external homepage of the system. This can be seen by anyone who downloads the application and want to register or login.
- **Calendar Page**: In this mockup we show the main page of the system where users are able to seen their meetings and travels. We offer the possibility to easily switch between daily, weekly and monthly views and create a new meeting. Moreover we show the meeting and travel details page where users can see in detail their events.
- **Notifications**: We have designed a notification page where a warning and an invitation are pending. The user can click on warning to open the overlapping meetings page where it can accept, decline or reschedule each of them; a rescheduling proposal page has been designed too. In addition, the user can choose to see in detail the meeting to which it has been invited.
- **Settings**: This mockup shows the current status, the current default location and daily breaks. We have designed also a page where the user can edit its preference list, marking travel means that it doesn't want to use as inactive. Moreover we have created mockups related to the process of adding a new constraint. The system suggests the user all the possibilities it can choose from while creating the new constraint.

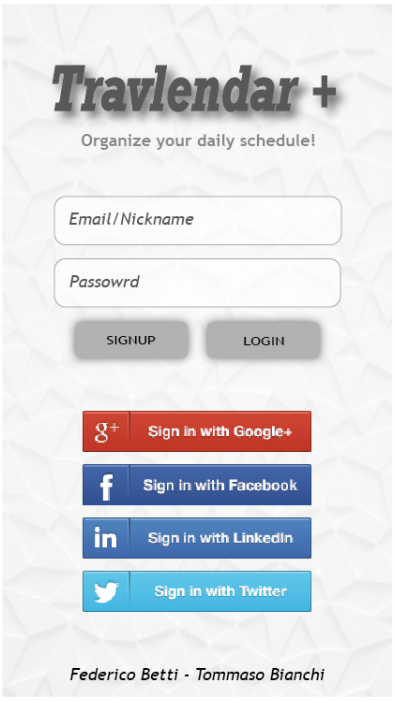

Figure 15: External Homepage

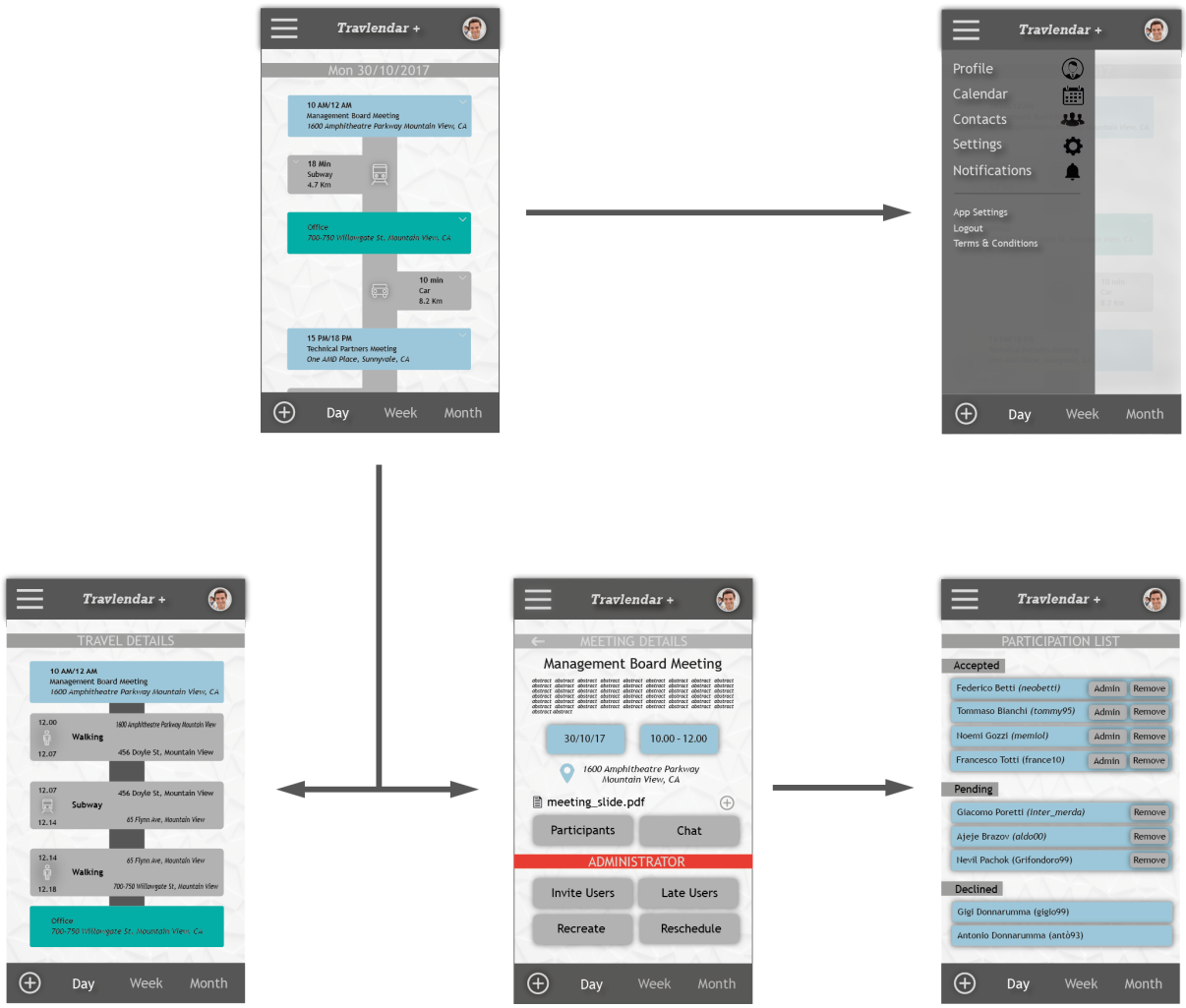

Figure 16: Calendar Page Mockup

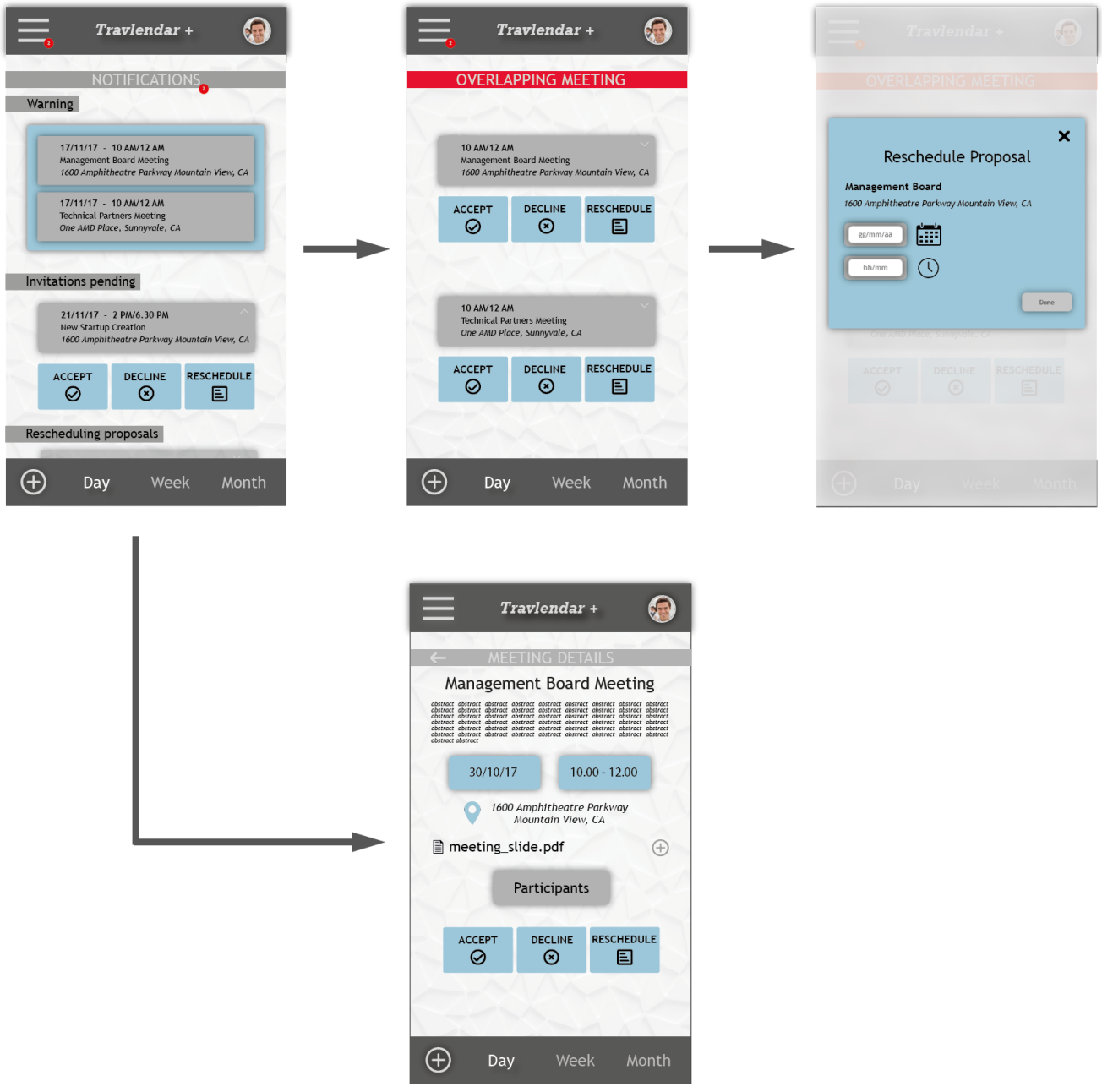

Figure 17: Notifications Mockup

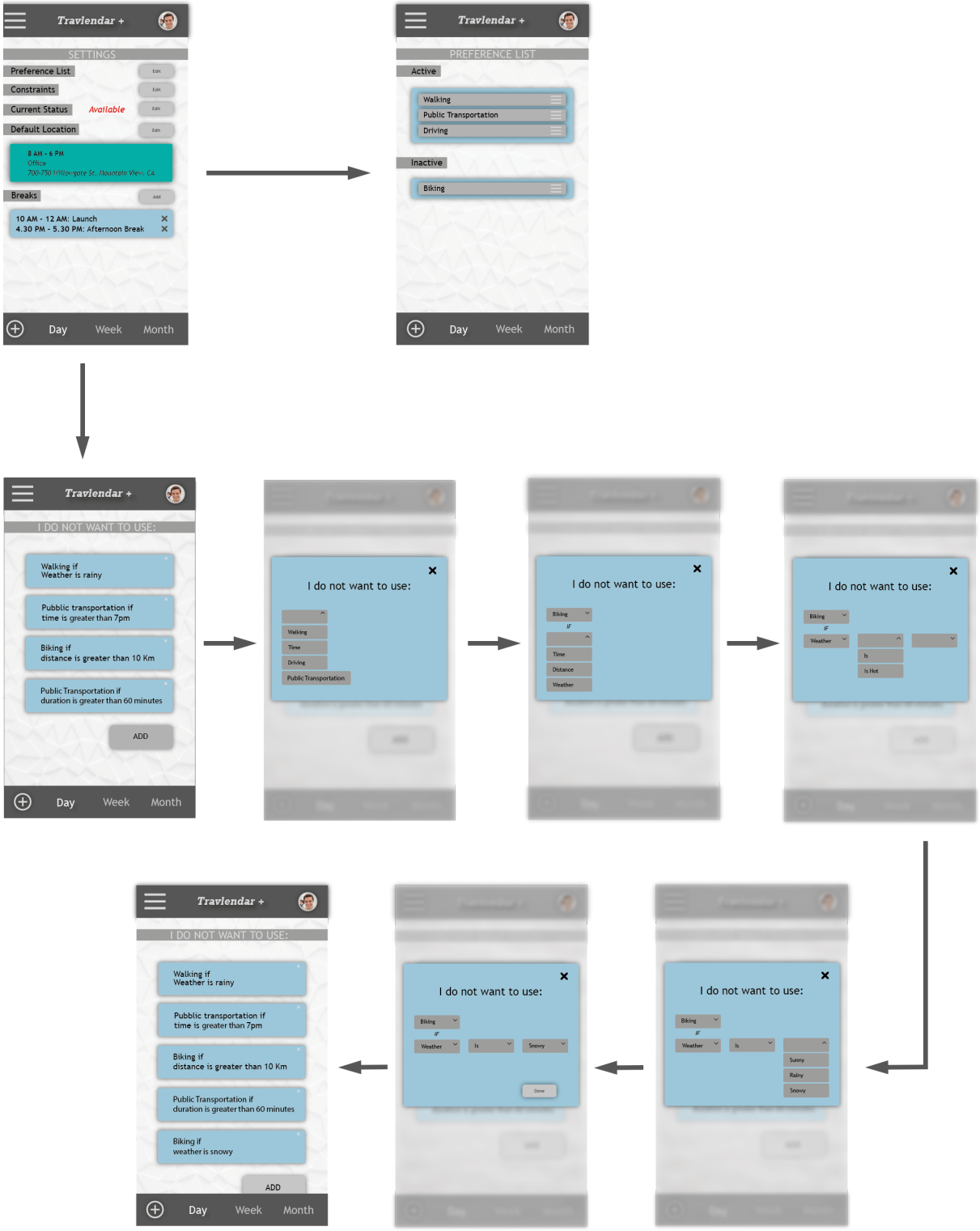

Figure 18: Settings Mockup

4 Algorithm Design

4.1 Description of the Problem

The most critical algorithmic problem to solve in Travlendar+ system is the insertion of a new meeting in a user's schedule. This is not as simple as it could seem because for every new meeting in a schedule a lot of things need to be taken into account and, possibly, recalculated and updated, such as whether the other meetings remain consistent, where flexible breaks can be placed, how to travel between subsequent meetings and so on. Some important choices have to be made, such as if to travel from a meeting to a default location and then to the next meeting or directly from one meeting to the other. In addition to that, this algorithm is likely to run quite often, as our users are likely to have some meetings each day and possibly many more invitations, plus as many flexible breaks as they want. Our implementation must therefore take all of this into account and reach a compromise between efficiency and precision, in order to build meaningful schedules without stressing our system too much.

4.2 Description of the Algorithm

The work of the algorithm is divided in two main functions: `insert_meeting` to insert a new meeting into the user's schedule and `best_travel` to compute the best travel between two locations with a given travel mean, taking into account the user's preference list and constraints. To simplify the execution of the algorithm and to speed it up, inconsistent meetings will be treated as non-existent, so that we can avoid to consider them when calculating travels; this is a fairly strong assumption, but it is needed to avoid all sort of problems tied to overlapping meetings, where it is unclear how to travel between them.

The first function, `insert_meeting`, does its job following this steps:

1. Find all the meetings overlapping with the one that we are inserting and mark them as inconsistent because they are clearly in conflict; if at least a meeting is found, mark also the new one as inconsistent and terminate the function. While doing this, keep track of all overlaps and store them; we will need this information to update meetings' consistency after a conflict is solved.
2. Try to compute a travel for arriving and for leaving the newly inserted meeting taking into account the previous meeting, the following one and the default locations; if this is not possible, terminate the function marking the new meeting and the conflicting one as inconsistent.
3. Adjust the effective time of all the flexible breaks overlapping with the new meeting; if this is not possible, mark the break as not doable. While doing this, keep track of all overlaps and store them; we will need this information to update breaks' doability after a conflict is solved.
4. Link the travels to the meeting and store everything, then terminate successfully.

The second function, `best_travel`, does its job following this steps:

1. Fetch a path from the External Shortest Path Provider for each travel means in the user's preference list.
2. Discard all the paths that are not compatible with the user's constraints.
3. Build the wighted preference list with the path calculated above.
4. Return the path corresponding to the first item of the weighted preference list.

4.3 Pseudocode of the Algorithm

Algorithm INSERT MEETING

global variables
conflict_set := set of pairs of meetings that are in conflict
break_overlap_set := set of pairs meeting-break that overlap

```
1:  function INSERT_MEETING(new_meeting, user)
2:    overlapping_meetings := all meetings of user that overlap with new_meeting
3:    for all meeting in overlapping_meetings do
4:      mark meeting as inconsistent
5:      add (new_meeting, meeting) to conflict_set
6:    end for
7:    if overlapping_meetings is not empty then
8:      mark new_meeting as inconsistent
9:      return
10:   end if
11:   defloc_before := the default location before new_meeting
12:   defloc_after := the default location after new_meeting
13:   arriving_travel := BEST_TRAVEL(defloc_before.location, new_meeting.location, user)
14:   leaving_travel := BEST_TRAVEL(new_meeting.location, defloc_after.location, user)
15:   before_meeting := the last consistent meeting such that its end date is before the start date of new_meeting
16:   after_meeting := the first consistent meeting such that its start date is after the end date of new_meeting

17:   if before_meeting is not NULL and arriving_travel.duration > new_meeting.start_date - before_meeting.leaving_travel.end_time then
18:     arriving_travel := BEST_TRAVEL(before_meeting.location, new_meeting.location, user)
19:     if arriving_travel.duration > new_meeting.start_date - before_meeting.end_date then
20:       mark new_meeting as inconsistent
21:       mark before_meeting as inconsistent
22:       add (new_meeting, before_meeting) to conflict_set
23:       return
24:     else
25:       before_meeting.leaving_travel := arriving_travel
26:     end if
27:   end if

28:   if after_meeting is not NULL and leaving_travel.duration > after_meeting.arriving_travel.start_time - new_meeting.end_date then
29:     leaving_travel := BEST_TRAVEL(new_meeting.location, after_meeting.location, user)
30:     if leaving_travel.duration > after_meeting.start_date - new_meeting.end_date then
31:       mark new_meeting as inconsistent
32:       mark after_meeting as inconsistent
33:       add (new_meeting, after_meeting) to conflict_set
34:       return
35:     else
36:       after_meeting.arriving_travel := leaving_travel
37:     end if
38:   end if

39:   overlapping_breaks := all breaks of user that overlap with new_meeting and its travels
40:   for all break in overlapping_breaks do
41:     UPDATE_BREAK(break, new_meeting ,arriving_travel, leaving_travel)
42:     add (new_meeting, break) to break_overlap_set
43:   end for
44:   new_meeting.arriving_travel := arriving_travel
45:   new_meeting.leaving_travel := leaving_travel
46:   return
```


Algorithm BEST TRAVEL

```
1:  function BEST_TRAVEL(from, to, user)
2:    weighted_list := empty array of the same size as user.preference_list
3:    for all travel_mean in user.preference_list do
4:      path := SHORTEST_PATH(from, to, travel_mean)
5:      for all constraint in user.constraints do
6:        if path is not compatible with constraint then
7:          path := NULL
8:          break
9:        end if
10:     end for
11:     if path is not NULL then
12:       insert path in weighted_list
13:     end if
14:   end for
15:   apply weigths to weighted_list
16:   sort weighted_list by increasing path duration
17:   return the first element of weighted_list
```


5 Implementation, Integration and Test Plan

5.1 Implementation and Integration Strategy

In this section we are going to present the strategy that the development team should follow to implement Travlendar+ system. As we can see in the Component Dependency Diagram all the system's components have been designed in such a way that they have the least amount of dependencies, and for this reason a lot of them can be developed in parallel. In fact, all the components that are in the same column of the graph can be easily implemented by different teams. Following this process, every time that a component needs an interface from another one, that one should have been already implemented. However, this development strategy would lead to build a testable system only at the end, like in a Waterfall model, because the Meeting Controller component, that is the one of the core components of the system, would be implemented almost at the end of the process. Since modern implementation strategies are based on mid-term versions of the system, that can be shown to stakeholders, it would be very limiting to adopt this model because it will not build any prototype (e.g. alpha and beta versions) until the end of the process, denying the possibility of corrections and modifications on the go.
For this reasons we propose the following plan to implement the Travlendar+ project:

1. Implement the Constraint Engine, the External Shortest Path Adapter and the Break Manager in parallel in order to have a solid foundation for the development of the other components.
2. Implement the Travel Planner component.
3. Integrate the Travel Planner with the Constraint Engine and the External Shortest Path Adapter.

   At this point the system should be able to find the best travel between pairs of locations. subject to given constraints.
4. Implement Meeting Scheduler component.
5. Integrate the Meeting Scheduler with the Travel Planner component.

   At this point the system should be able to properly build and update schedules.
6. Implement the Meeting Controller component. Since it has still some unsolved dependencies, we will use stubs to simulate the behaviour of all the component that are not been developed yet. We will do this to be able to reach a point in which the core of the system is testable sooner.
7. Integrate the Meeting Controller with the Meeting Scheduler.

   At this point the main functionality of the system (i.e. create meeting to organize daily schedule) should be completed.
8. Implement the External Login Adapter and the Authentication Engine integrating them together.
9. Implement the User Search Engine.
10. Integrate the User Search Engine with the Meeting Controller component.
11. Implement User Controller.
12. Integrate the User Controller with the User Search Engine, the Constraint Engine and the Authentication Engine.

    At this point a user should be able to use out system.
13. Implement Rescheduling Engine and integrate it with Meeting Controller.

14. Implement Notification Engine and integrate it with the Meeting Controller and the Rescheduling Engine.

    At this point a Beta version could be shipped.

15. Implement the File Uploader, the Chat Engine and the Late User Engine in parallel.

16. Integrate the File Uploader, the Chat Engine and the Late User Engine with the Meeting Controller, replacing the previously inserted stubs.

17. Implement the External Geolocalization Adapter and the Real-Time Indications Engine integrating them together.

    At this point the first working version of Travlendar+ can be considered completed.

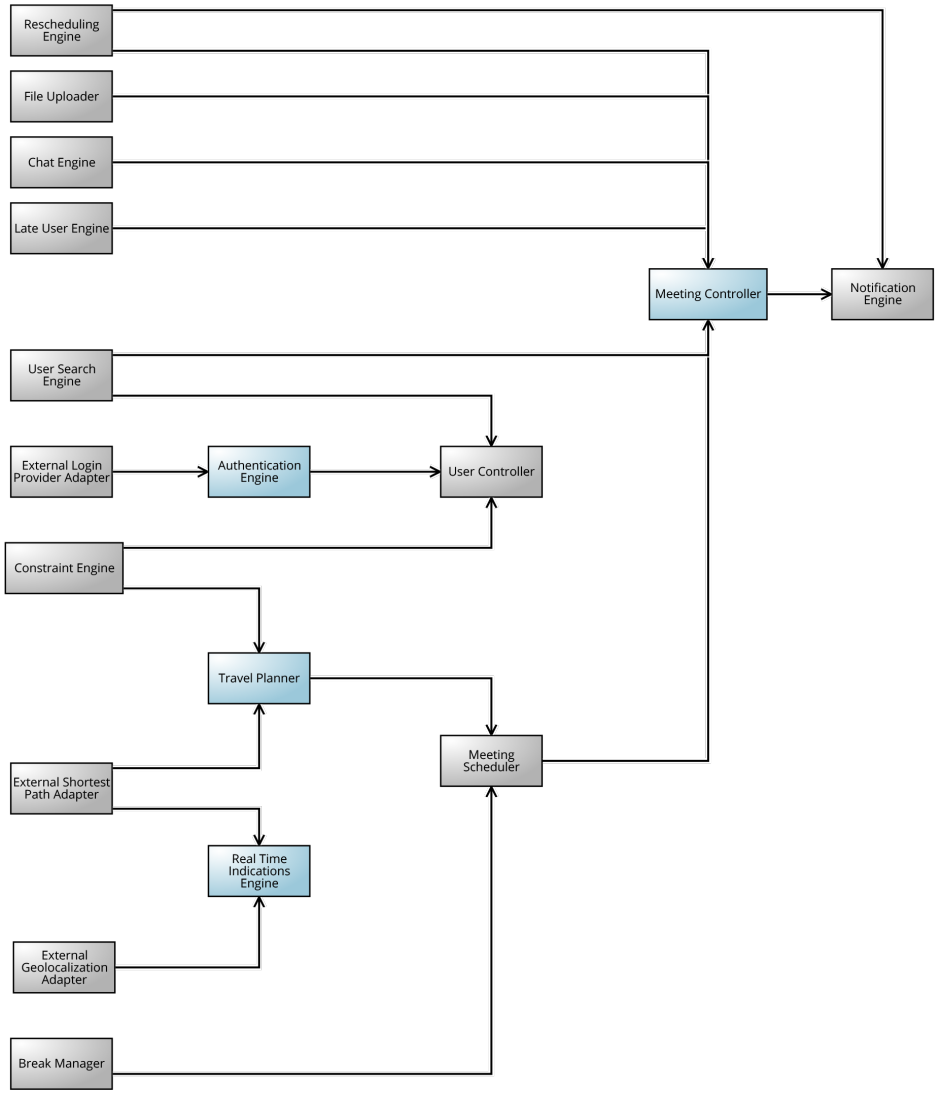

Figure 19: Component Dependency Diagram

5.2 Test Plan

Continuous and automated testing will be done from the very beginning of the implementation phase of the Travlendar+ project. This way, as soon as the development of a component is finished we can know if it works as expected and, if so, we can move on rapidly to the next step. We can also perform integration testing shortly after all the involved components are ready.
In this section for each component in the system core we present a list of tests to perform during the implementation phase, together with the main integration tests. This does not aim to be a complete list of all possible test cases, hence developers should still add some more specific ones while writing down the code.

5.2.1 Unit Tests

Component	Authentication Engine
Unit tests	• Test registration. • Test login. • Test fetching the user that sent a request. • Test the redirect to the Complete Registration page.

Table 2: Authentication Engine Tests

Component	External Login Adapter
Unit tests	• Test correct communication towards an external login provider. • Test correct communication towards multiple different external login providers.

Table 3: External Login Adapter Test

Component	User Search Engine
Unit tests	• Test searching for a user given its nickname, its email or parts of them. • Test searching for a non-existent user.

Table 4: User Search Engine Tests

Component	User Controller
Unit tests	• Test the creation of a user. • Test the editing of a user's profile (in particular the change of password and primary mail). • Test the addition/removal of a contact. • Test the creation of a group. • Test the signup to an existing group. • Test the addition of an auto-decline status and the refusal of all meeting invitations in that period. • Test the edit of the preference list. • Test the addition/removal/modification of default locations.

Table 5: User Controller Tests

Component	Constraint Engine
Unit tests	• Test checking if a constraint holds or not.

Table 6: Constraint Engine Tests

Component	Meeting Controller
Unit tests	• Test the creation of a meeting. • Test the editing of the fields of a meeting. • Test the recreation of a meeting. • Test the approval/refusal/rescheduling of a meeting. • Test the invitation of a user selecting it from the contacts of an administrator.

Table 7: Meeting Controller Tests

Component	Chat Engine
Unit tests	• Test sending messages. • Test retrieving messages, both one at a time and in bursts. • Test sending messages from different users at the same time.

Table 8: Chat Engine Tests

Component	Late User Engine
Unit tests	• Test the visualization of the late user dashboard. • Test the dispatch of messages to late users.

Table 9: Late User Engine Tests

Component	File Uploader
Unit tests	• Test uploading files with different extensions and different sizes. • Test downloading files.

Table 10: File Uploader Tests

Component	Travel Planner
Unit tests	• Test planning travels between random points.

Table 11: Travel Planner Tests

Component	Real-Time Indications Engine
Unit tests	• Test the activation of the real-time indications engine component when a meeting is approaching.

Table 12: Real-Time Indications Engine Tests

Component	External Shortest Path Adapter
Unit tests	• Test correct communication towards the external shortest path provider. • Test that a path is always returned by the external shortest path provider.

Table 13: External Shortest Path Adapter Tests

Component	External Geolocalization Adapter
Unit tests	• Test correct communication towards the external geolocalization provider. • Test that is always possible to retrieve the correct position of a user via the external geolocalization provider; requires someone physically going around.

Table 14: External Geolocalization Adapter Tests

Component	Meeting Scheduler
Unit tests	• Test inserting a meeting in an empty schedule and in one with already a lot of meetings. • Test schedule update after a warning is solved. • Test the correct visualization of a schedule.

Table 15: Meeting Scheduler Tests

Component	Break Manager
Unit tests	• Test the update of a break multiple times by different meetings. • Test the update of a break by a meeting that makes it undoable and creates a warning.

Table 16: Break Manager Tests

Component	Rescheduling Engine
Unit tests	• Test the creation of a new rescheduling proposal. • Test the polling of a rescheduling proposal by an administrator. • Test voting on a poll and sending the last vote, that should trigger an automatic rescheduling approval/rejection.

Table 17: Rescheduling Engine Tests

Component	Notification Engine
Unit tests	• Test the gathering of all the notification for a user who has lots of meeting invitations, breaks, warnings, etc. • Test the automatic push of notifications on the application client.

Table 18: Notification Engine Tests

5.2.2 Integration Tests

Components	Authentication Engine - External Login Adapter
Integration tests	• Test registration (i.e. first time login) via external login provider. • Test login via external login provider.

Table 19: Authentication Engine - External Login Adapter Integration Tests

Components	User Controller - Constraint Engine
Integration tests	• Test adding/removing a constraint for a user.

Table 20: User Controller - Constraint Engine Integration Tests

Components	Meeting Controller - Meeting Scheduler
Integration tests	• Test the receipt of a meeting invitation that triggers an update of the schedule.

Table 21: Meeting Controller - Meeting Scheduler Integration Tests

Components	Meeting Controller - User Search Engine
Integration tests	• Test the invitation of a user by searching it.

Table 22: Meeting Controller - User Search Engine Integration Tests

Components	Meeting Controller - Chat Engine
Integration tests	• Test that it is not possible to send messages by a user who's not in the meeting.

Table 23: Meeting Controller - Chat Engine Integration Tests

Components	Meeting Controller - Late User Engine
Integration tests	• Test that it is not possible to access the dashboard by a non-administrator. • Test that it is not possible to access the dashboard long before the starting time of the meeting.

Table 24: Meeting Controller - Late User Engine Integration Tests

Components	Meeting Controller - File Uploader
Integration tests	• Test that it is not possible to download a file by a user who's not in the meeting.

Table 25: Meeting Controller - File Uploader Integration Tests

Components	Travel Planner - Constraint Engine
Integration tests	• Test planning a travel subject to a given set of constraints.

Table 26: Travel Planner - Constraint Engine Integration Tests

Components	Real-Time Indication Engine - External Geolocalization Adapter - Travel Planner
Integration tests	• Test the indications by following them and checking they are right; requires someone physically going around.

Table 27: Real-Time Indication Engine - External Geolocalization Adapter - Travel Planner Integration Tests

Components	Meeting Scheduler - Break Manager
Integration tests	• Test inserting a meeting in an empty schedule with a lot of breaks.

Table 28: Meeting Scheduler - Break Manager Integration Tests

Components	Meeting Controller - Rescheduling Engine
Integration tests	• Test the approval/rejection of a rescheduling proposal by an administrator.

Table 29: Meeting Controller - Rescheduling Engine Integration Tests

Components	Meeting Controller - Notification Engine
Integration tests	• Test the approval/refusal/rescheduling of a meeting via the Notification Engine.

Table 30: Meeting Controller - Notification Engine Integration Tests

6 Appendix

6.1 Effort Spent

Date	Tommaso Bianchi	Federico Betti
09/11	2	4,5
10/11	0	6
11/11	0	1,5
12/11	1,5	3,5
13/11	3,5	2
14/11	2	0
15/11	5,5	5,5
16/11	0	1
17/11	3,5	3,5
18/11	4	0
19/11	1,5	3
20/11	2	3
21/11	2,5	3
22/11	2,75	1
23/11	3,5	2
24/11	6	6
25/11	2,5	0

Table 31: Effort Spent

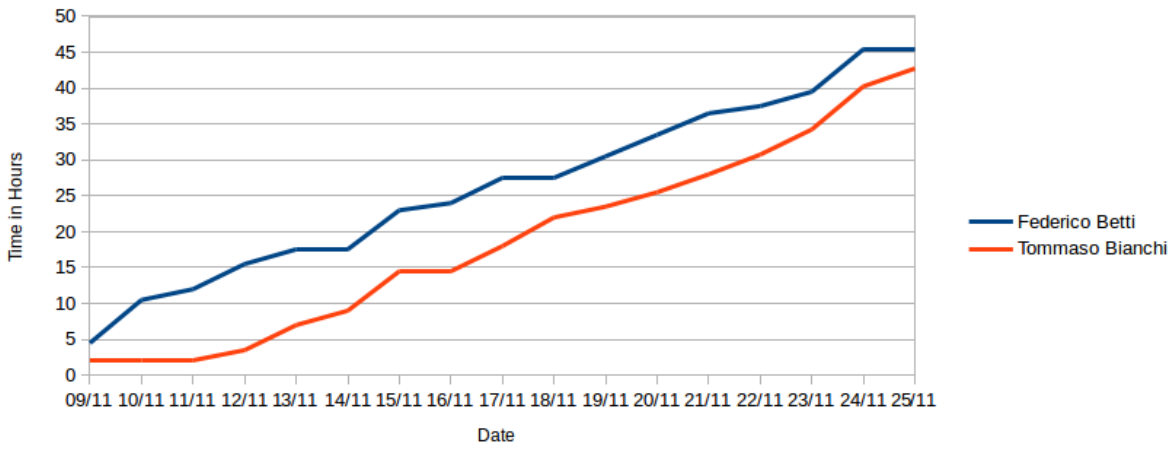

Figure 20: Effort Spent

6.2 Tools Used

- **TeXstudio**: a cross-platform open source LaTeX editor
- **Git**: a version control system for tracking changes in computer files and coordinating work on those files among multiple people
- **Signavio**: a tool for creating UML diagrams; we used it for the class diagram and the use case diagram
- **Astah**: a tool for creating UML diagrams; we used it for the sequence diagrams and the state chart diagrams
- **Adobe Illustrator**: a vector graphics editor; we used it for the mockups of the user interface
- **Photoshop**: a raster graphics editor; we used it for some small editing of the images exported from the other tools

6.3 Revision History

This is the first version of the document, released on 26/11/2017. All subsequent modification will be tracked in the github repository and briefly listed here.

