POLITECNICO
MILANO 1863

Travlendar+ project Jerkenhag, Kverne, Rebner

Design Document

Deliverable:	DD
Title:	Design Document
Authors:	Jerkenhag, Kverne, Rebner
Version:	1.1
Date:	December 29, 2017
Download page:	https://github.com/Rebnersaurus/JerkenhagKverneRebner
Copyright:	Copyright 2017, Jerkenhag, Kverne, Rebner All rights reserved

Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose

The Travlendar+ is an application that will assist the user in managing their meetings, as well as the travel in between. The purpose of this document is to in depth explain the architectural decisions and structural patterns in the application Travlendar+. This document is addressed to the software development team.

1.2 Scope

The document will provide information and explanation about essential aspects of the application. These aspects will include:

- Creating a new account with login procedure.
- The storage of user's customized preferences.
- The algorithm for providing the optimal journey.
- Creating a new event.
- Interaction with external services.

Constraints:

- Regarding weather, we only focus on rain as a factor. However, it is possible as a future improvement to include other weather phenomena, such as extreme heat where longer distances of walking and biking would be ill-advised.
- In this document, due to time and resource constraints, we describe only one shared transportation service in detail which is BikeMi. However we do elaborate on how the shared bike and shared car is intended to work in general terms. Furthermore, the optimal solution would be to use the shared services API and only using the information when a request is made regarding shared services. In the meantime the data for the shared service we are using for now, BikeMe, has been stored in a table.

Assumptions:

- Taxi is not considered a public transport.
- For the prototype we have the city of Milan as a reference for transit functionalities.
- If the user utilizes it's own car or bike it is assumed that they will keep track of the location of their vehicle.
- We assume that the user has a stable internet connection with a minimum internet speed of 1MB/s.

1.3 Definitions Acronyms Abbreviations

Definitions:

- Event - A time block on a day in the calendar with two geolocations, the address of the event and the origin address to get to the event.
- Journey - The complete route to take to arrive at an event.

- Path - One piece of the journey, i.e. in a journey you might first have to take the bus to the train station and then the train to arrive at your meeting. In this case the journey would consist of 2 paths.
- Preferences - Concerns the importance the user places on the following points: Economy, Environment, Wetness and Speed
- Relevance - When Travlendar+ generates possible journeys for an event they will be ordered by relevance with the most relevant option on top.
- Economy - How important price is in the user for the relevance of a journey.
- Environment - How important carbon footprint of different MoT is for the user for the relevance of a journey.
- Weather Sensitive MoT - A MoT that is an outdoor activity and therefore affected by change in weather, like walking or taking the bike.
- Dryness - How unwilling the user is to travel by a weather sensitive MoT if there is a bad weather forecast.
- Flexible Break - Interval of the day that should be kept open for a break of a given duration at some point in this interval.
- S<Transport> - Refers to the shared version of a certain transport mood. (SCar = Shared Car)
- Selected journey - Refers to the journey Travlendar+ suggests or the user manually selects as active for the transportation to an event.
- Speed - The total travel time for a journey.

Acronyms:

- MoT - Method of Transportation
- FR - Functional requirements
- NFR - Non Functional requirements

1.4 Revision History

Version Date	**Changes**
Version 1.1 12/12/17	Minor changes
Version 2.0 27/12/17	Updated with respect to Implementation and Testing assignment

Table 1: Revision History

1.5 Reference Documents

1. Assignment document.
2. Software Engineering [1]

3. Fundamentals of software engineering [2]

1.6 Document Conventions

Due to the usage of a SQL-database in our system we decided to have a different approach in this document and give a view from a object-oriented aspect of a database-oriented system. Therefore our data can be thought of as following:

- Table: Array with instances of a ”class”.
- Row: Instance in that array.
- Column: Variable in that ”class”.

1.7 Document Structure

This document will first present the overall structure of the components of the application and their interactions. It will also describe the storage of different components in the database and continue with a description of interactions in between components with the use of different cases such as login.

After that the interactions between external API's will be explained. In the end of Architectural Design we will provide a description about the architecture of the application. The software application part will provide and in depth analysis of our relevance algorithm. Furthermore, we will explain the correlation from the RASD document to this DD document. In the end we will give a description of the implementation plan and test documentation.

2 Architectural Design

2.1 Overview

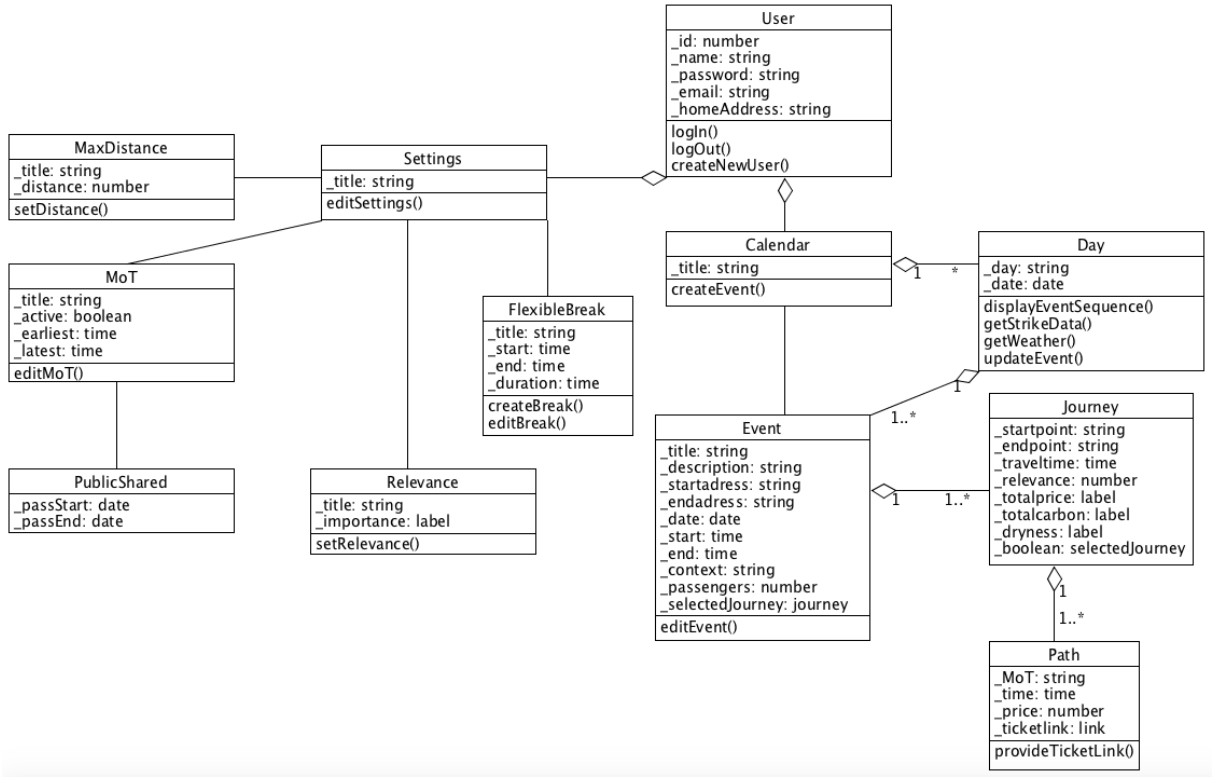

Figure 1: Structure from a users perspective

In Figure 1 is a visualization of the system from the user's perspective indicating how the user will interact with the different components of the application and it also provides an overview over the main parts. The user creates a profile and once logged in has access to their personal calendar and settings. The settings section includes different types of settings; transportation preferences, relevance and flexible break management. The actual calendar's main functionality is the creation and management of events. Each event generated has one or multiple journeys associated with it describing how to get from the start address to the end address of the event. These journeys consists out of one or a sequence of multiple paths that display the different MoT that are suggested.

On the back-end, things will work a bit differently and the different parts will interact with each other in a less linear way. The data structure consists of a number of tables. Each of these tables corresponds to different components of the application. A row of a table is an instance of that component with its features contained in the value of each column of the table. Each row is distinguished by the unique row ID. Figure 2 shows a depiction of all of the tables in the data structure with the list of columns corresponding to that table and the functionalities associated with the table from the user's perspective.

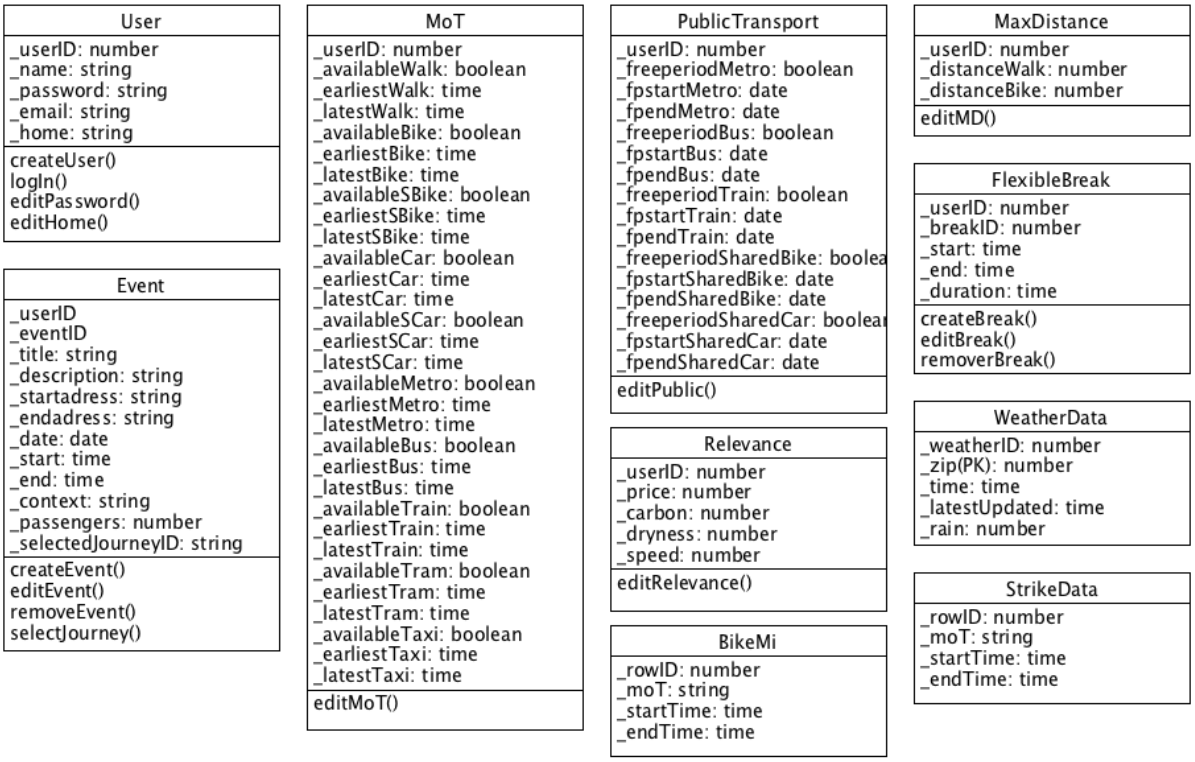

Figure 2: Component view

The user table is one of the most central components to the data structure since rows in many of the tables is connected to its user through the userID column. These tables main functionality is to store different pieces of information related to the user of the corresponding userID. This includes both tables regarding user settings but also objects created and interacted with by the users such as events and flexible breaks. There are also some tables separated from the user, namely the WeatherData table, StrikeData table and the table for the external service BikeMi. Each table will be described in more detail in the Component View section.

In Figure 3 is an example world of the tables related to the user table showing how the elements of these tables are connected to different users.

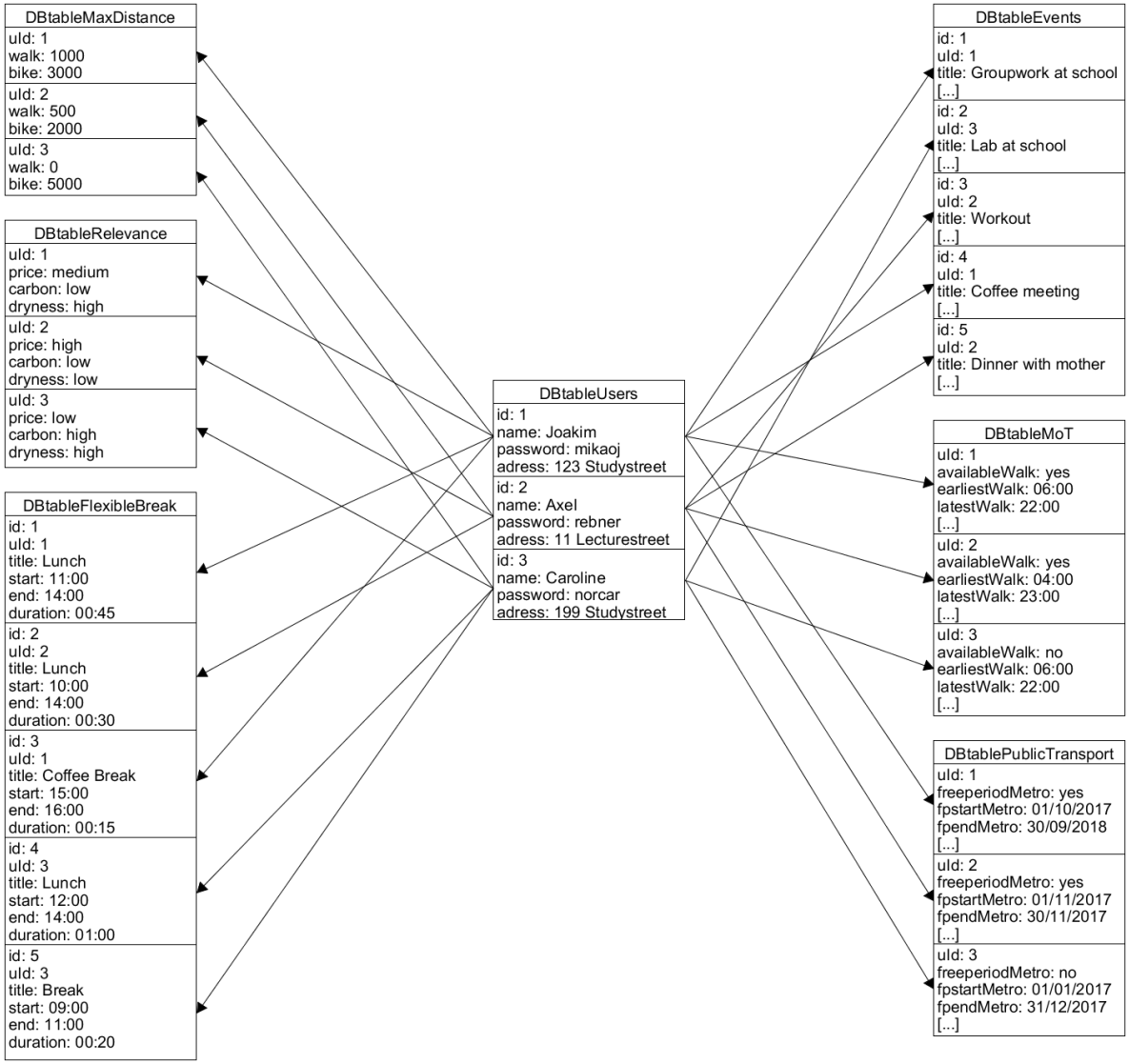

Figure 3: Relation for user id

The id from DBtableUsers is the primary key for accessing the users data from the other tables. As visible in Figure 3 every row in all of the tables has a uId(userID). Those tables that can have multiple rows for the same user, such as DBtableFlexibleBreak and DBtableEvents, also has a unique id for every row in the table but still uses the uId for access to the right user. For the other tables where the user can only have one row it is also the case that the user must have that row, i.e. for every user there must be one and only one corresponding row in table DBtableMoT.

2.2 Component View

In this section a more in depth description of the different tables is given. Many interactions between these occurs through a link of userID but there are some tables that are separate from the user table.

2.2.1 Tables related to user

userID: **number**	**name:** **string**	password: string	...

Table 2: Usertable

This table stores all the users of the systems and the most critical aspect of it is the unique userID column as this works as the key that links the rows of other tables to a certain user. Beyond this the user has a name, password and a home address. The name and password is utilized for logging in. The home address works as a default setting for the event creation. For the first meeting of a day it will be set as the start address and for the last meeting of the day it will be set as the end address.

The functionalities associated with this table consists of the creation of a new user, log in to the system as well as password and address edit.

userID: number	**eventID:** number	title: string	...

Table 3: Event Table

The event table allow for multiple rows for the same user, as one user can create multiple events. To keep track of the different events belonging to the same userID each event row has a unique eventID. Other columns of the event row represents the information a user types in when adding a new event. The rows of the Event table is therefore highly interactive from a user perspective as most of its data can be set directly by the user and can also be edited at a later point. A noteworthy column is the selected Journey which stores, by default, the most relevant journey computed or one that has manually been selected by the user. Depending on the selected journey the columns containing the levels of price, carbon, dryness and speed.

There are a number of functionalities associated with the event table and some of the main operations of the application takes place here. These are createEvent(), editEvent(), and removeEvent().

As part of the process of generating a new event row the potential journeys and paths between the start address and the end address must be computed. This involves filtering the journeys depending on the settings a specific user have regarding the different MoTs and the time slots in which these MoTs should be active. Further, the computed journeys for the destination of the event will be sorted by relevance according to the importance the user has granted the different relevance factors; price, carbon and dryness along with the factor of minimizing travel time. The algorithm for this will be discussed in more detail in the Algorithm Design section in this document.

An already existing event can also be interacted with either by editing it or removing it. When an event gets edited some of the column values of the event row opens up for alteration. Depending on the changes to the row, like addresses and time, new journeys may have to be calibrated from scratch and sorted as if the event was created a new. The edit function also enables the user to select another journey to the event if they are dissatisfied with the one Travlendar+ has suggested as the most relevant. Finally, an event can also be removed which leads to the termination of the associated event row.

The tables below are all related to the settings functionality of Travlendar+. In general each user has only one row of each table linked to them and the functionalities associated with the settings table is to alter the column values of that row. The only exception to this is the FlexibleBreak table where a user can have a number of breaks which also implies that these rows can be created and removed.

userID: number	availableWalk: boolean	earliestWalk: time	...

Table 4: MoT Table

The MoT table has the most extensive rows in the system as these includes all the user settings regarding the different transportation means. As Travlendar+ supports 10 different MoT this leads to a

great quantity of setting combinations. This table concerns itself with the properties of transportation setting that is general for all MoTs, whether or not they are Private or Public. The 3 general settings relevant for each MoT are the following; if the user wants the MoT to be active, the earliest time for usage and the latest time for usage during the day.

userID: **number**	freeperiodMetro: boolean	fpstartMetro: date	 ...

Table 5: Public Transport Table

The PublicTransport table adds the pieces of information that comes along with a transport being a public MoT. These are the settings that go beyond the general settings that was dealt with by the MoT table. For a public MoT the user has the option to add a seasonal pass. The columns here are consequently if a seasonal pass is active, its start date and its end date.

userID: **number**	distanceWalk: number	distanceBike: number

Table 6: Max Distance table

For the MoT:s walking and biking a max distance is of interest for natural reasons. With max distance means the maximum for that MoT in one journey. This is taking care of by the MaxDistance table which add these upper limitations to these MoT:s for the user which id corresponds to the row in this table. Worth noting here is that the max distance for bike is for both the private and shared option.

userID: **number**	price: number	carbon: number	 ...

Table 7: Relevance table

The Relevance table keeps track of the importance that a certain user attaches to price, carbon and dryness when travelling on a scale from 0-3 where 0=None, 1=Low, 2=Medium and 3=High. The actual interplay of these for the computation of the relevance of journeys for an event is taken care of by an algorithm. As stated earlier a closer look will be taken on this algorithm in the Algorithm Design section.

userID: **number**	**breakID:** **number**	title: start	 ...

Table 8: Flexible Break table

The FlexibleBreak table works a bit differently from the other ”settings tables” as it is possible for one user to have multiple breaks. This means that a FlexibleBreak row requires a unique id column along with the userID it is associated with. Other columns for this table is the start and end time for the flexible break interval as well as the duration of the actual break. The functionalities associated with the FlexibleBreak table are the optionalities to create breaks, edit existing breaks and remove breaks from a user’s calendar.

2.2.2 Tables separated from user

Below tables are separated from the user table. The data contained in these tables are not available to the user unless the information in a row is affecting Travlendar+ responses to the requests of the user.

zip: **number**	**time:** **time**	lastupdated: timestamp	 ...

Table 9: Weather Data table

The WeatherData table includes information on the weather prognosis in different regions. This is done by having different rows in the table for different zip-codes detailing the amount of rain in that area at a certain time. There will be several rows for each zip-code as the prognosis for multiple points in time will be available for the same area. The WeatherData rows also includes a latest updated column value in order to not make redundant updates if the latest update was recent.

The table will be updated as Travlendar+ pulls information from the external weather prognosis service used, OpenWeatherMap. New rows will therefore be added continuously as the third party provides information as time proceeds and information about later points in time is available. Old rows will likewise be removed when the point in time they are describing are in the past and no longer of interest.

id: **number**	mot: string	startTime: time	endTime: time

Table 10: Strike Data table

The StrikeData table keeps track on upcoming strikes. Strike data will be gathered through monitoring the news channels of different MoT providers. If a strike has been discovered from one of the third-party services monitored a row will be added to this table indicating the MoT affected and the interval in time when the strike takes place. Consequently, if there are no upcoming strike prognoses this table is empty. If a row is added to the table users who have events with selected journeys affected by the strike will be given a warning.

For shared service transportation a separate table will be kept for each third party. For this document due to time constraint we are only describing one of these services here.

id: **number**	mot: string	startTime: time	endTime: time

Table 11: BikeMiSites table

The BikeMiSites table indicates the position of all pick-up/drop-off locations of BikeMi bikes. These positions are made when looking at journey options utilizing the shared bike MoT. The table rows include longitude, latitude, address, zip code, availablebikes and availableslots. The availablebikes column value is relevant for the pick-up location, if the closest pick-up location has a value of 0 on this column Travlendar+ should not direct the user towards it. Instead it should point the user towards the closest pick-up location with availablebikes>0. Similarly the drop-off location needs availableslot>0 for the system to pick it as the place for the user to drop-off the shared bike.

In the prototype of Travlendar+ all positions will already be stored in the database. For future development the aim is to move away from this solution since this would require a unique table for each bike/car sharing service used. Instead we aim to be able to directly interact with the API and shared service and use the data only when there is a request for it.

2.3 Deployment View

Travlendar+ is intended as a web application that will be accessible by a network or Internet connection. The system is intended to run on any up to date web browser. As it is located on the web it is meant to

be accessible through personal computers and different types of smartphones and tablets.

The application will have somewhat complex runtime dependencies since it is utilizing a number of third-party applications to provide its functionalities. These interactions will be looked at in more depth in the following section.

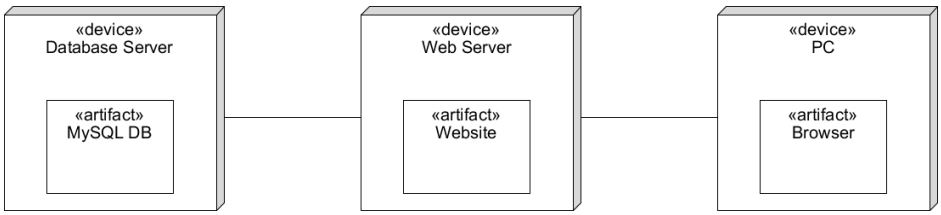

Figure 4: Deployment View

2.4 Runtime View

This section describes in detail the sequence of events for some of the key processes of Travlendar+ and the different actors involved in the system. The processes described is the login of a user, creation of an event and modification of user settings by the use of sequence diagrams.

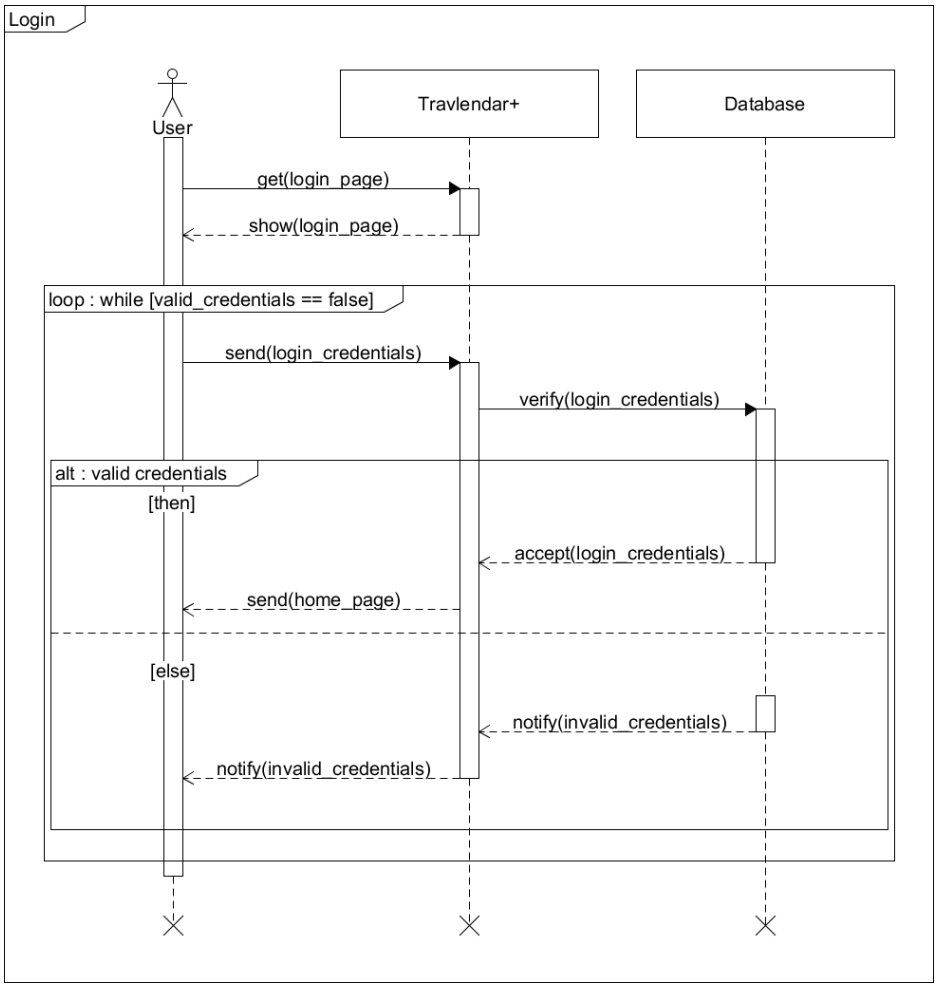

Figure 5: Sequence diagram for the login of a user

When logging into the Travlendar+ service the user fills out username and password, this is then hashed and checked against the database which in turn sends back the userID if the credentials are correct and a notification if they are not. If the credentials are correct the userID is stored in the session of the user as to work as a key for accessing the rest of the data in the program. All of the credentials and events are connected to this userID.

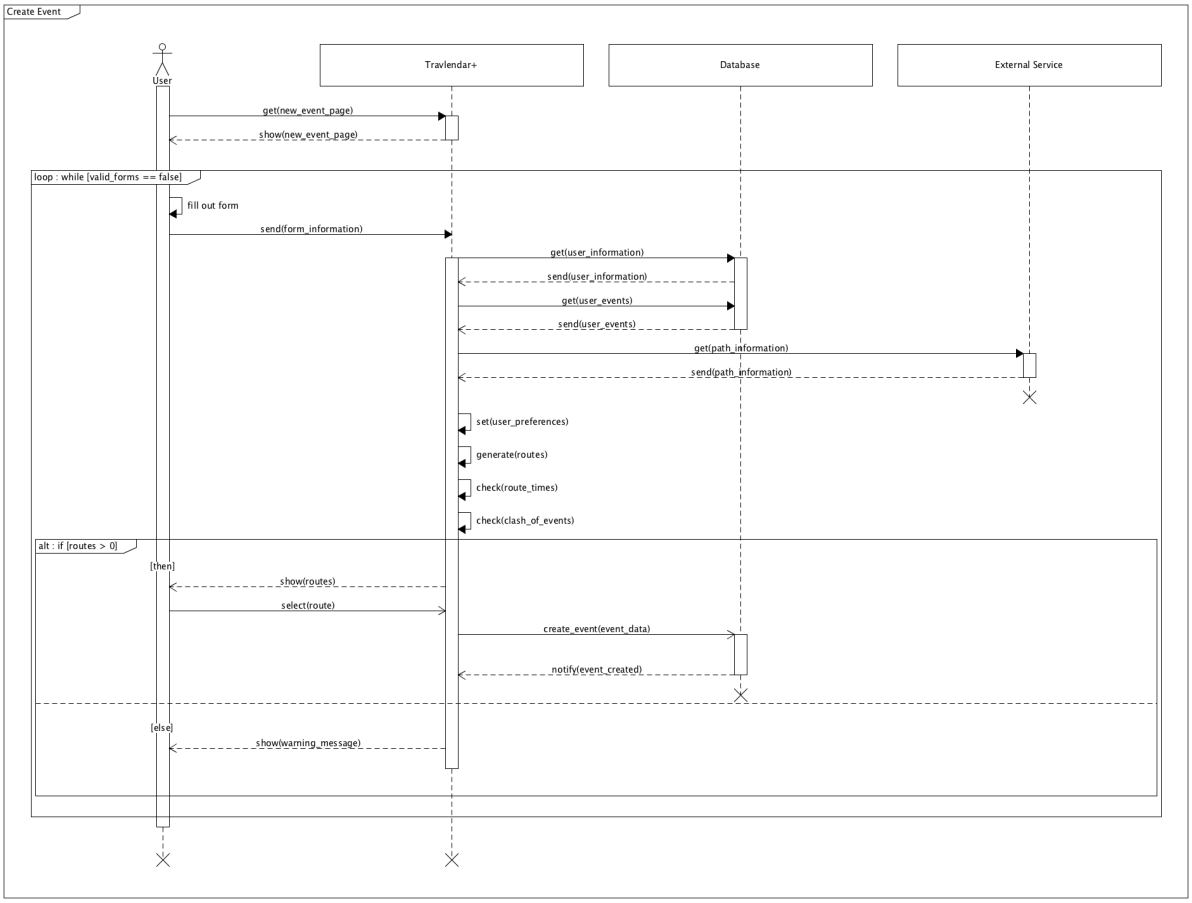

Figure 6: Sequence diagram for the creation of an event

When creating an event the userID obtained at the login is used to get all of the information regarding preferences. This is then used to filter the search of the external service for providing the journey. Travlendar+ also check for clashes of other events connected to the userID or to any clashes of flexible breaks. If everything checks out the event is created with a connection to the userID.

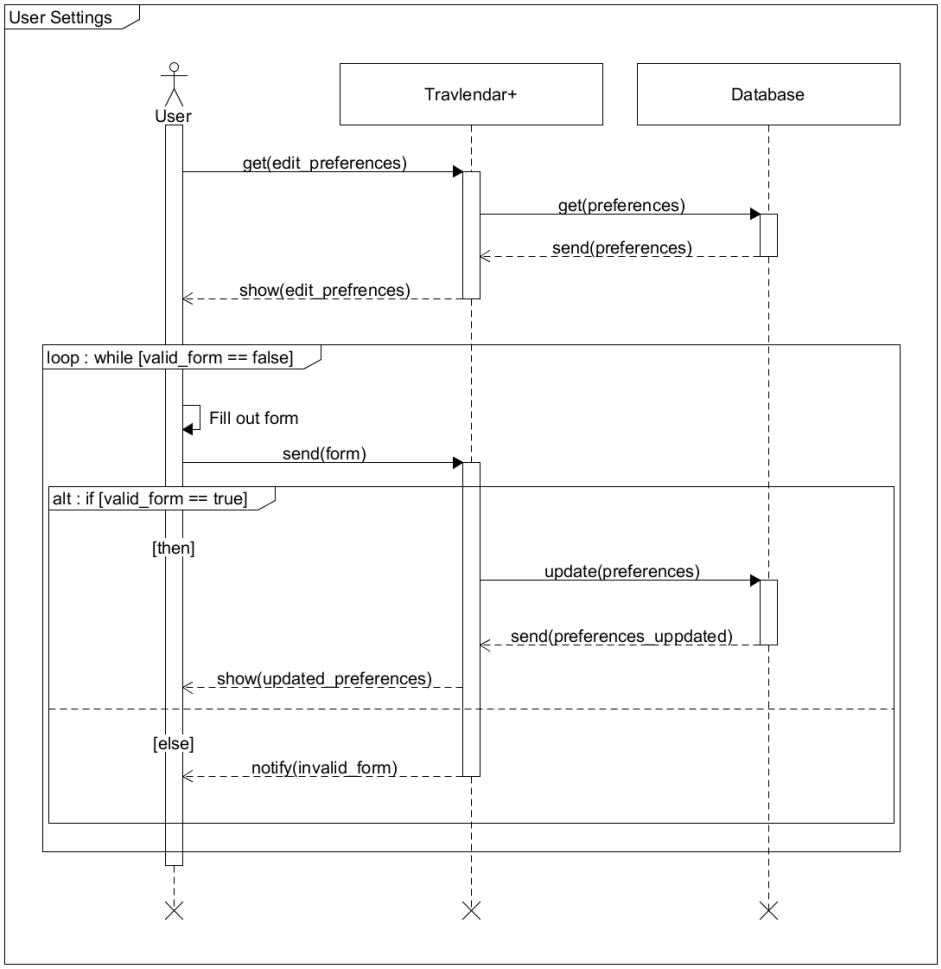

Figure 7: Sequence diagram for the edit of a user's settings

When entering the settings page the database is connected and all of the settings loaded in, if the user clicks save then the form will be checked and if the entered data is valid the server sends its new data to the database to be stored.

2.5 Component Interfaces

https://developers.google.com/maps/documentation/directions/intro#Routes

Travlendar+ will interact with some external services in order to deliver some of its functionalities. These external services will be described in more detail below.

- Google Maps - For calibrating possible routes between 2 positions and the MoTs used for these. This API will also be used for providing links to ticket purchase if the MoT requires it and it is available.
- Shared bike service (BikeMi) - For locating pick-up locations for shared bike
- Shared car service - For locating pick-up locations for shared car
- OpenWeatherMap - For gathering weather prognosis
- Public Transportations home pages/twitter feeds - For gathering strike data about Public transportation

2.5.1 Google Maps

When making a request to the Google Maps API the required parameters are origin (start address), destination (end address) and key (The API key of Travlendar+). Other parameters that are optional but will be utilized by Travlendar+ is mode (this can either be walking, driving, biking or transit) and alternatives (alternative road to the first one suggested). If the mode is transit the options can be narrowed down further by include a transit_mode parameter detailing which type of public transport the user wants to utilize. If the user has deactivated a MoT in his or her settings this mode will be left out when the request is sent and in the case of public transportation the transit_mode will be excluded. If all public transportation modes have been disabled the entire mode=transit will be omitted from the request.

The Google Maps API will be used to gather information of possible routes between 2 positions. In most cases this will be a simple input of start address and end address and filtering based on types of mode. However, the API interaction will work a bit differently to include shared services, such as shared bike and shared car. For shared bike Travlendar+ will locate the closest pick-up/drop-off location to the start address and the closest pick-up/drop-off location to the end address. For shared car it is assumed that the end address will work as a drop-off location for the service and thus the only extra route is to get to the closest pick-up location from the user's start address.

2.5.2 Shared bike journey

The journey will consists of 3 separate Google Maps routes put together. These 3 are:

- Start address to pick-up location: Mode - Walking/Transit
- Pick-up location to drop-off location: Mode - Bike
- Drop-off location to end address: Mode - Walking/Transit

2.5.3 Shared car journey

The journey will consists of 2 Google Maps routes put together. These are:

- Start address to pick-up location: Mode - Walking/Transit
- Pick-up location to end address: Mode - Car

When Google Maps API has returned the potential routes they will be given a number based on the relevance for the requesting user derived from that user's settings regarding importance of price, carbon footprint, weather sensitivity and time. Travlendar+ will then promote the route granted with the lowest number as the selected journey for an event and store it. The other journeys will be discarded. For more detail on how this algorithm works please refer to the Algorithm Design section.

When a transit mode is part of the route retrieved by Google Maps, as part of the Google Maps response, an URL is provided for the home page of the company providing the transport service. This will be exploited for providing the user of Travlendar+ with a link to ticket purchase if part of the journey that is selected for an event has a MoT that requires it. This link will be accessible in the journey description of an event in the calendar.

The interaction with the Google Map API will be triggered when an event is created, edited or if the user manually wants to select another journey than the one suggested. The Google Map API will also be interacted with if the weather or strike prediction requires a selected journey to be discarded. In this case Travlendar+ will notify the user that a change has been made.

2.5.4 Shared bike and car service

As described in the Google Maps API the external services for shared bike and shared car systems will be used to locate the closest pick-up/drop-off locations to the start address and end address for an event. This will be

2.5.5 OpenWeatherMap

OpenWeatherMap (OWM) will be used to extract information about the weather prognosis for upcoming dates. OWM offers different forecast deals but since the project is on a small budget the project team has decided to use the best option which has been made available for free. This was the API for a 5 day / 3 hour forecast, which means a prognosis for the upcoming 5 days in 3 hour intervals.

When the routes collected from the Google API has been received by Travlendar+ and the journey suggested as the selected journey contains weather sensitive MoT:s, such as walking and biking, Travlendar+ will interact with OpenWeatherMap. If the weather forecast for the time of the event is indicating bad weather the selected journey might be replaced by a less weather sensitive one if dryness is high priority for the user making the request.

The weather will also be checked in the background when a user refreshes Travlendar+. If the weather prognosis changes to the worse it will interact with pre-existing selected journeys in a similar way for events with a user that values dryness.

2.5.6 Strike update

Travlendar+ will monitor the homepages and other potential newsfeeds of the companies of the public MoTs supported after signs of strike and the date affected by the strike. This will naturally be done differently based on the different way the companies present the news. The aim is to cover the most reliable feed of strike news for each type of MoT.

If a strike is noticed from one of the sources monitored a warning will be sent to the user and Travlendar+ will make changes to events impacted by the strike. Journeys selected for an event on the strike date using the MoT affected by the strike will be recalibrated and if possible another journey excluding this MoT will be suggested as selected journey instead.

2.6 Selected Architectural Styles And Patterns

As an architectural style for our software system we have used the Client-Server Model. The Client-Server Model uses a one-to-many relationship with it's clients where you have one single server that can interact with multiple users at the same time. The server hosts the data and most of the processes and the user can connect to it through a network or an Internet connection. The client makes requests that are passed along to the server from which the server then provides a response.

This architecture style made sense for us on many levels since we have a set of isolated users who are independently interacting with the web application. By the utilization of the users userID as a key, each user has their own unique environment of the server which they alone can interact with. The requests the users will send in might be to add new data to the server in forms of new user profiles, events or breaks. The users can also edit pre-existing instances of these as well as alter their user settings.

With this architectural style we also use the pull and push protocol to interact with our users. When the application is in use, the server extracts data (pull) from the weather and strike-site's and use them to notify (push) the users. The pull protocol is also in use when the users request data from the application' server.

2.7 Other Design Decisions

2.7.1 Keeping as few variables in RAM as possible

We researched an alternative way of keeping the data in the RAM of the web server instead of loading the data when it was needed. The alternative was to, when the user logs in, load all of the user settings into the RAM of the server for faster access when navigating to the settings page or creating a new event. This seemed as a good idea at first, the CPU and the database would then have to work less but at the expense of RAM-memory.

Our calculations showed that the used RAM per logged in user would increase almost by 50 times its current size. The CPU would work less, but after realizing that the workload would not even be cut in half we noticed that it was not worth it, especially when we do not know the exact specifications of the CPU. Furthermore, since the user is expected to not change their personal settings that frequently and only create an event about 1 out of 4 times that the user visits the website the data would not even be needed in the majority of the visits. And even so, when an event is created all of the user's events must be checked, as well as the user's flexible breaks, which still would put load on the CPU and the saved power from the saved settings would become a small portion of what would actually be used.

Therefore we decided to stick with the loading of data as they are needed, keeping down the RAM usage as much as possible.

3 Algorithm Design

Focus on the definition of the most relevant algorithmic part This algorithm takes place in creatEvent()

For this section we have identified the calculation of the relevance for a journey as the most integral algorithm of the application to discuss since it is key for giving the user a personal experience when interacting with Travlendar+. We believe that this is the most relevant one to focus in on since this is where the customizability the application allows for really comes into play. We want the user to feel that the application understands the user's needs and as a rule of thumb, the fewer times a user feel that they need to manually switch the suggested journey provided by Travlendar+ the better.

This algorithm occurs during the creation of an event, when all the information has been set but it has not yet been saved. The information is used to fetch data from external services to provide one or more routes for the user. This algorithm can also be called when the user alters the information or navigates to the event and simply wants to look at the different alternatives again. The algorithm takes the information stored for a user in the column values of the row of the Relevance table associated with the userId of that user and utilizes it to give a journey a numerical value which it is then sorted by when connected to the event.

The journey with the lowest value will be the one that by default will be chosen as the selected journey for a generated event. This is done by looking at the paths that a journey consists of. Each of these paths are granted a number from 1-3 (Low, Medium, High) on the following factors: price, carbon footprint, wetness and speed. These factors are based on the different types of MoT the path consists of. Some of these are more or less static as in the context of carbon emission from a certain type of transport. Others are more dynamic as in the case of heavy rain compared to a weather sensitive MoT like walking. In the dynamic case the MoT for the path needs to be checked towards an external source.

Chain of events - User attempting to create an event:

1. Event information is passed along to the external source Google Maps API;
2. Google Maps API delivers journeys between the start address and the end address of the event;
3. Journeys are filtered based on the user's settings;
4. Journeys are breaked into paths and the paths are analyzed based on the different importance factors: price, carbon footprint, wetness and speed.
5. The paths are summed up and the journey is granted a total score on these factors;
6. Journeys are granted a relevance score compared to the user's preferences;
7. Journeys are sorted by the relevance score from lowest to highest;
8. Event row is created with the journey with the lowest score suggested as the selected journey;

3.1 Algorithm Code

```
float function Relevance() {
 float relevance = 0;

 foreach(Path p in Journey.paths) {
   Journey.duration += p.duration;
   Journey.cost += p.cost;
```



```
    Journey.co2 += p.co2;
    Journey.distance += p.distance;

    if(p.MoT == 'bike') {
      Journey.bikeDistance += p.distance;
      Journey.wet += weather(p.zip, p.time)*p.distance;
      } else if(p.MoT == 'walk') {
        Journey.walkDistance += p.distance;
        Journey.wet += weather(p.zip, p.time)*p.distance;
      }
    }

    float maxWetness = userWet;

    if(Journey.wet > maxWetness)
      return NULL;

    if(Journey.walkDistance > maxWalkDistance)
      return NULL;

    if(Journey.bikeDistance > maxBikeDistance)
      return NULL;

    float relevance = 0;

    relevance += Journey.time * userTime;
    relevance += Journey.cost * userCost;
    relevance += Journey.co2 * userCo2;
    relevance += Journey.wet * userWet;

    return relevance;
}
```

3.2 Potential future improvement

This algorithm is also interesting in the light that it will be continuously improved once Travlendar+ is out in public and usage statistics are coming in. The weights given to the different importance factors are an estimation at this stage but they are expected to be fine tuned when more usage data is coming in.

4 User Interface Design

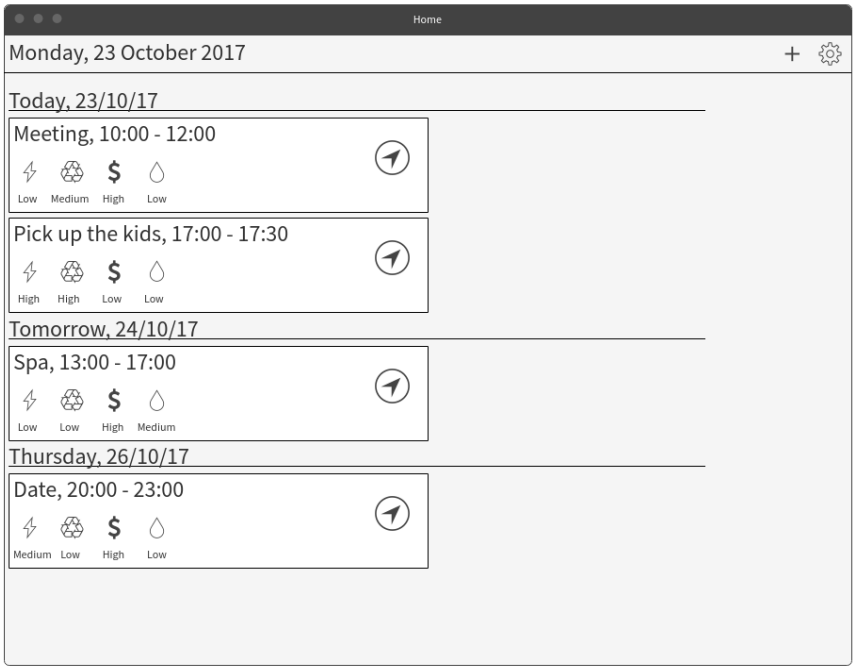

Figure 8: Home-screen

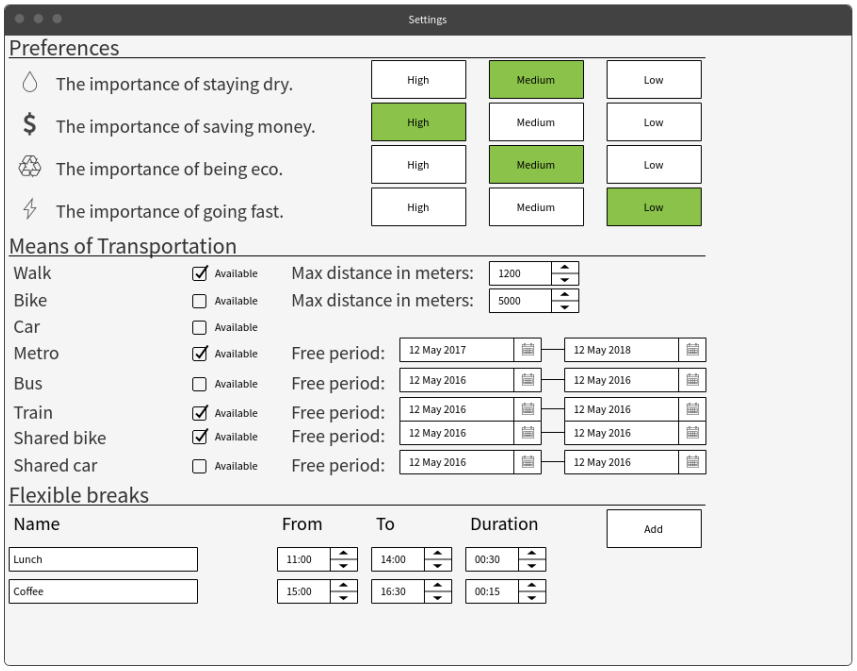

Figure 9: Settings-screen

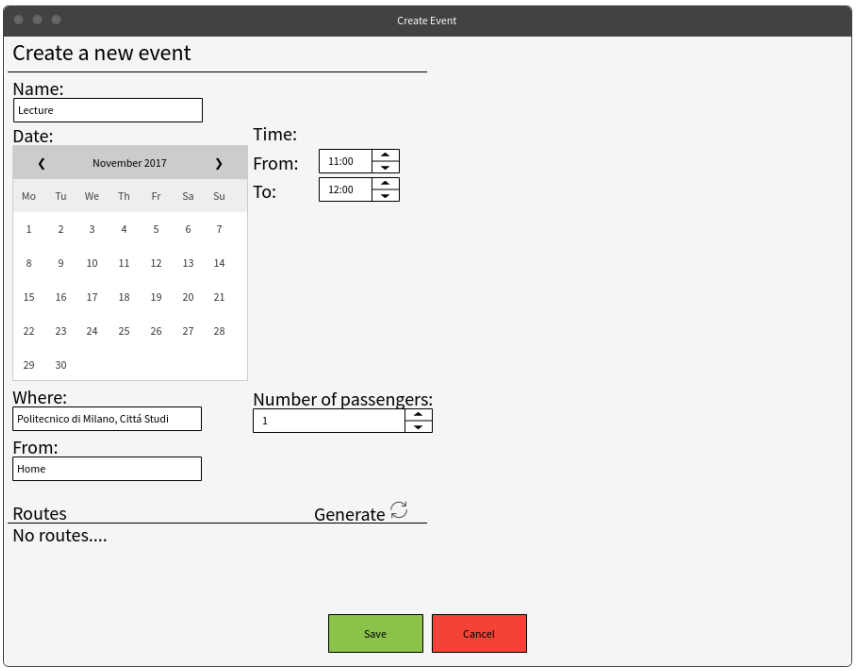

Figure 10: Event-screen

5 Requirements Traceability

In this section will see how this document corresponds to the RASD documents functional requirements. The table will show two columns of Goal ID and FR ID's, then the other column will specify the table or figure that corresponds to the FR, and the last column will give further comments if any.

Goal ID	FR ID	Figure ID	Comment
G1	**FR1** FR1.1 FR1.2 FR1.3 FR1.4	3 2	When the user accesses the calendar, the storage of information about each user is provided by using concepts as these figures provides.
G2	**FR2** FR2.1 FR2.2 FR2.3 FR2.4 FR2.5 FR2.6 FR2.7 FR2.8 FR2.9 FR2.10 FR2.11	2 4 5 6 9 11 7	This requirement needs a good storage system, since it concerns the users preferences. These tables show how the application will store and find such elements.
G3	**FR3** FR3.1 FR3.2 FR3.3 FR3.4	7 3.1	This algorithm is the essence of our application and this figure and table shows how the most optimal path will be chosen for the user.
G4	**FR4** FR4.1 FR4.1.1 FR4.1.2 FR4.1.3 FR4.1.4 FR4.1.5 FR4.1.6 FR4.2 FR4.3	3 6 9	These tables show how the events are created and how the components interact with each other.

G5	**FR5** FR5.1 FR5.1.1 FR5.1.2 FR5.2 FR5.2.1 FR5.2.2 FR5.2.3	8	The flexible break table provides information about the requirement concerning the users customized breaks.
G6	**FR6** FR6.1 FR6.2 FR6.3	6	The sequence diagram provides information of what will happen if there is a collision in events.
G7	**FR7** FR7.1 FR7.2	N.A.	The shortest path description does not have any figures, as it is a part of googles API of finding the shortest path. Our relevance algorithm supplements the API with choosing the most optimal journey for the user.
G8	**FR8** FR8.1 FR8.2 FR8.3 FR8.4	5	The Public Transport table, shows how the purchase of tickets is to be stored and used in the application.
G9	**FR9** FR9.1 FR9.1.1 FR9.1.2 FR9.2	6	The sequence diagram provides information about the unreachable destination process when creating an event. This corresponds to the unreachable destination requirement.
G10	**FR10** FR10.1 FR10.2 FR10.2.1 FR10.3 FR10.3.1	5 3 2	Each user is to have a personal profile. As these figures show, the application uses this database structure to store the certain elements. Also the first table shows how the login process is executed.

Table 12: Requirement Decision Diagram

6 Implementation, Integration and Test Plan

ID	Description	Preconditions	Deadline
I1	Create a shell of the website without any function.	None	26/11/2017
I2	Create all the tables in the database.	Working database	29/11/2017
I3	Make the user able to create an account and login.	I2	3/12/2017
I4	Make the user able to change its settings.	I3	7/12/2017
I5	Make the user able to create an event.	I3	10/12/2017
I6	Make the user able to manage an event.	I5	13/12/2017
I7	Create a functional weather table.	I2 & weather data	15/12/2017
I8	Create a functional BikeMi table.	I2 & bike data	17/12/2017
I9	Public transportation ticket implementation.	I6	19/12/2017

Table 13: Implementation

For the test plan please refer to the Implementation and Testing assignment document.

7 Effort Spent

7.1 Effort Spent

7.1.1 Axel Rebner

- 01/11/17: Kick-off and planning - 2 hours
- 10/11/17: Architectural Decisions and Design - 4 hours
- 11/11/17: Component view, sequence diagrams and run-time view - 4 hours
- 13/11/17: Algorithm design, component view, architectural styles and pattern - 6 hours
- 14/11/17: Implementation, Component interface - 5 hours
- 15/11/17: Requirements traceability, Component interfaces, Algorithm design - 6 hours
- 18/11/17: Overall review - 6 hours
- 20/11/17: Structuring document text - 3 hours

7.1.2 Caroline Kverne

- 01/11/17: Kick-off and planning - 2 hours
- 14/11/17: Architectural decisions - 2 hours
- 15/11/17: Requirements traceability, Component interfaces, Algorithm design - 6 hours
- 18/11/17: Overall review - 6 hours
- 20/11/17: Read through and minor changes - 2 hours

7.1.3 Joakim Jerkenhag

- 01/11/17: Kick-off and planning - 2 hours
- 10/11/17: Architectural Decisions and Design - 4 hours
- 11/11/17: Component view, sequence diagrams and run-time view - 4 hours
- 14/11/17: Implementation, Component interface - 5 hours
- 15/11/17: Requirements traceability, Component interfaces, Algorithm design - 6 hours
- 18/11/17: Overall review - 5 hours
- 20/11/17: Structuring document text and layout - 4 hours
- 23/11/17: User interface section and internal referencing - 1 hour

8 References

References

[1] Software Engineering: Principles and Practice, Hans van Vliet (c) Wiley, 2007.

[2] Fundamentals of software engineering (2. ed.), Prof. Dr.-Ing. Axel Hunger, April 2009.