Politecnico di Milano
AA 2017-2018

Software Engineering Project

RAVLENDAR+

DD

Design Document

Version 1.01

Marco Prosdocimi - 898395
Enrico Ruggiano - 900040
Giacomo Vercesi - 899928

Contents

Introduction **2**
- Purpose . . . 2
- Scope . . . 2
- Definitions . . . 2

Architectural Design **2**
- High-level components and their interaction . . . 2
  - Applicational Servers . . . 3
  - Firewalls . . . 3
  - Databases . . . 3
  - Auxiliary Server - Scraper & API Module . . . 4
- Component View . . . 4
  - Server Side . . . 4
  - Cient Side . . . 5
- Deployement View . . . 5
- Runtime View . . . 5
  - Registration . . . 6
  - Add Event . . . 7
  - Buy Ticket . . . 8
  - Notification . . . 9
- Component Interfaces . . . 9
- Architectural Styles . . . 10
  - Overall Architecture . . . 10
  - Frontend Endpoint Server . . . 10
  - Web Scraping and API . . . 10
  - Optimal Path Server . . . 11
- Other Design Decisions . . . 11

Algorithm Design **11**
- Transport Categories . . . 11
- Transport Switch Penalties . . . 12
- Main Algorithm . . . 12

User Interface Design **13**
- User Interface MockUp . . . 13
- UX Diagram . . . 13
- Gui Scheme . . . 14

Requirements Traceability **14**

Implementation, Integration an Testing **16**
- Elements to be Integrated . . . 16
  - BackEnd . . . 16
  - Scraping Module . . . 16
  - FrontEnd . . . 16
- Component Integration sequence . . . 16
  - BackEnd . . . 16
  - FrontEnd . . . 17

Effort Spent **17**
- Prosdocimi Marco . . . 17

Ruggiano Enrico 17
Giacomo Vercesi 18

References **18**
External Link 18
Tool Used 18

depth 2

Introduction

Purpose

In this document it will be analyzed all the design features needed to develop Travlendar+ system.

We suppose that the readers are familiar to what is the Travlendar+ system and all its features. If it is not, we suggest them to read the RASD document before continuing.

Scope

We will focus on mapping all the functions analyzed on the RASD document on different components and modules in a specific structure.

We will do considerations about the Architecture structure of the system and the single components involved. This analysis will cover not only the Logic part of the application but the UI too.

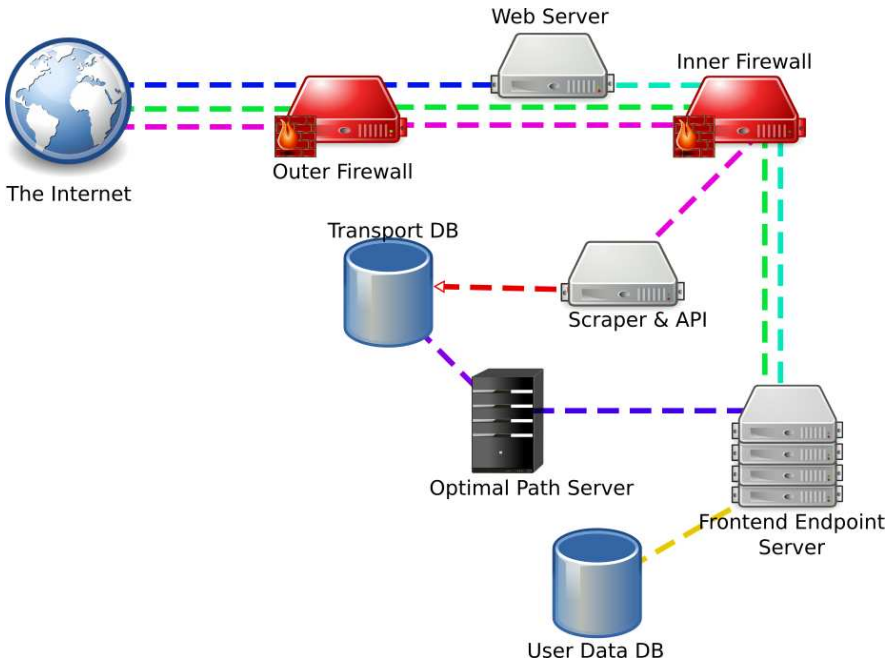
Definitions

- *Best Path Algorithm*: the algorithm which computes the *Best Path*. See *RASD - Definitions* for more information
- *FES*: Frontend Endpoint Server. See *Architecture Styles* for details.
- *OPS*: Optimal Path Server. It the server which calculates the Best Path Algorithm. See *Architecture Styles* for details.
- *OSM*: Open Street Map library. See External link for more informations.
- *RASD*: Requirements analysis and specification document.
- *REST*: Representational state transfer. See external link to have more information.
- *UI*: User Interface of the system.

Architectural Design

High-level components and their interaction

The architecture of the system is a 3 Logic Tiers.



It is mainly composed by these elements:

- Applicational Servers
- Firewalls
- Databases
- Auxiliary Server

Applicational Servers

Server used to provide the main service and logic of Travlendar+. As in the figure above, the Applicational Servers are:

1) **Web Server** which provides all the html forms and hypertext layout of the System.
2) **Frontend Endpoint Server** which dispatches and elaborates all the client requests in a server side and safe environment
3) **Optimal Path Server** which has the function of computing the Best Path Algorithm and arrange a route for the user.

Firewalls

Protection is a quality driver essential for the System. We decide to put only two firewalls, which are:

1) **Outer Firewall** which has the function to screen external packets with a light level of protection. A complete protection for a Web Server is not necessary.
2) **Inner Firewall** which has to protect all the server side from malicious packets. It must guarantee an high level of protection, especially for the databases access.

Databases

Of course the System needs to have databases. Those are:

1) **User Data DB** which stores all personal information and preferences of the accounts and member registred on the System.
2) **Transport DB** which stores all the information needed to compute the Best Path Algorithm. For istance it has stored public transports timetables. This is due to absence of valid external API. (see Scraper Section for more information)

Auxiliary Server - Scraper & API Module

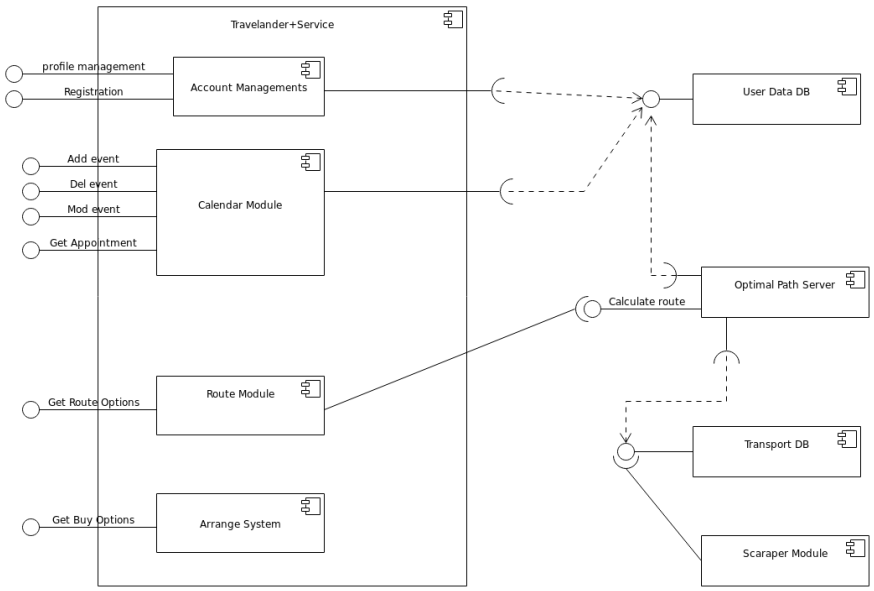
To compute the Best Path Algorithm are needed a lot of external information, such public transport timetables, geographical position car sharing system stations and so forth. External API which could query databases of third parties could be not exist. So it must implement a system which currently gives all these informations. For this reasons it could be used Auxiliary Servers which have the function to populate the Transport Database.

A good tool is the use of **Scraper & API Module Server**.

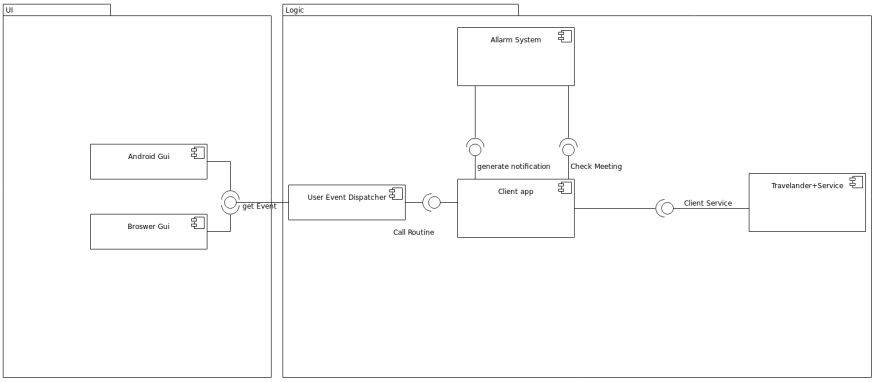
Component View

The main function offered by the system can be summarized in those structure of components.

Server Side

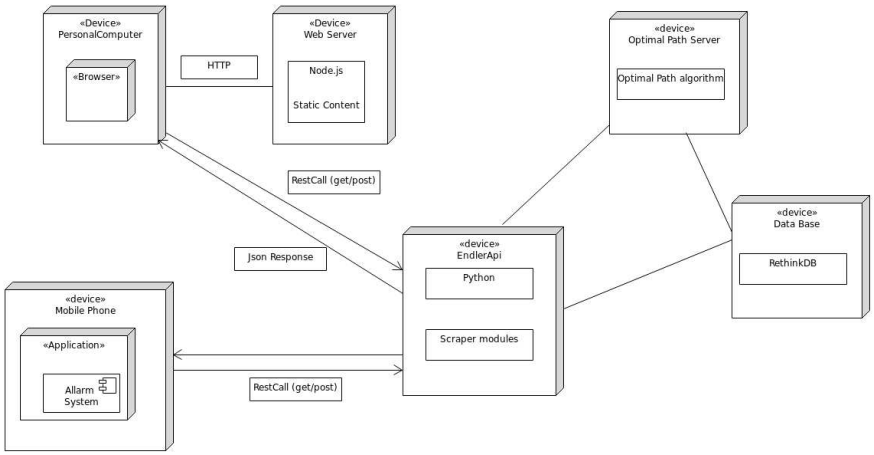


Cient Side



Deployement View

Other userful information about the deployment of the BackEnd components can be represented on the diagram below.

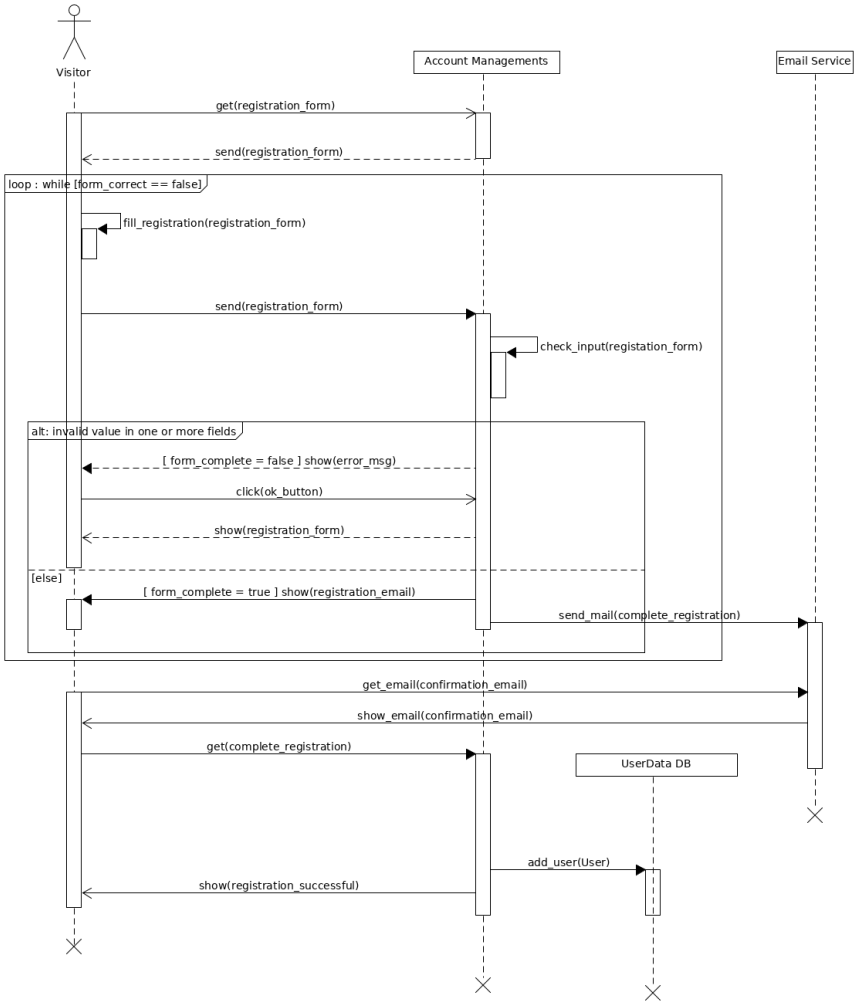


Runtime View

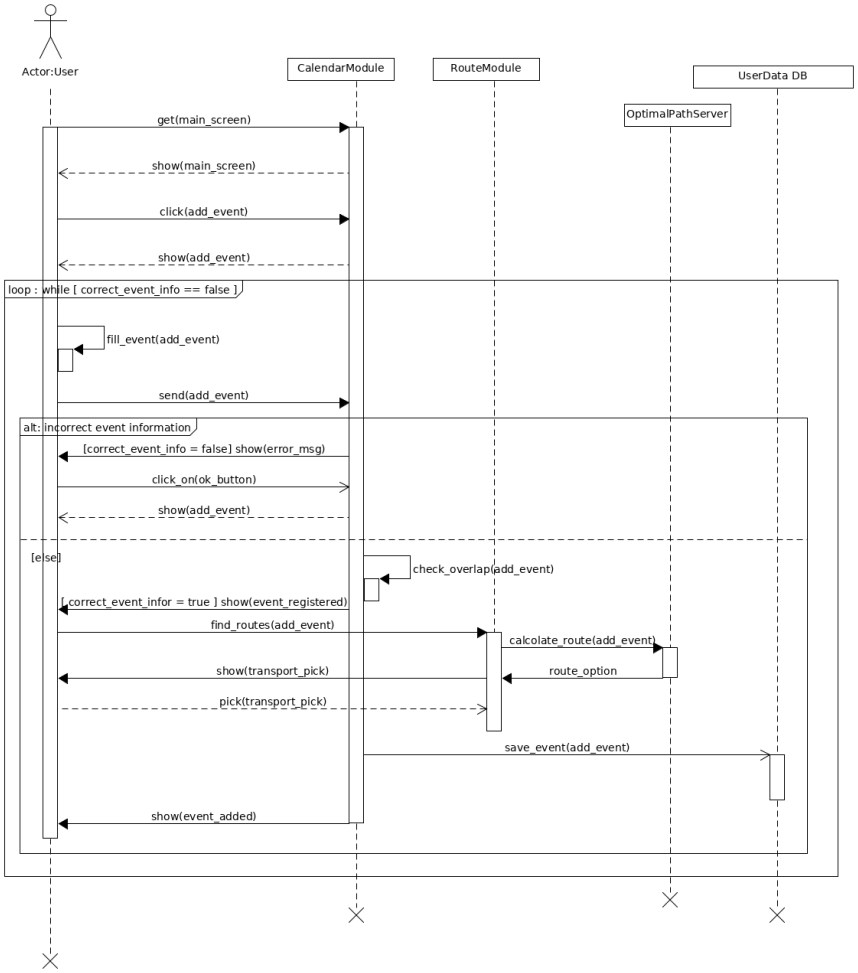
A schematic representation of the runtime System can be summarized in those diagrams. We will analyze the main functions of Travlendar+:

- Registration
- Add Event
- Buy Ticket
- Notification

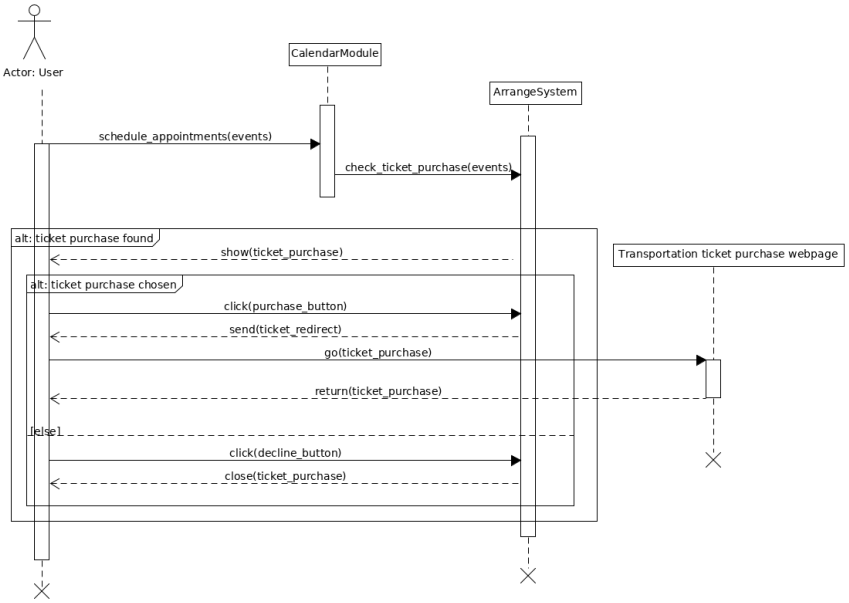
Registration



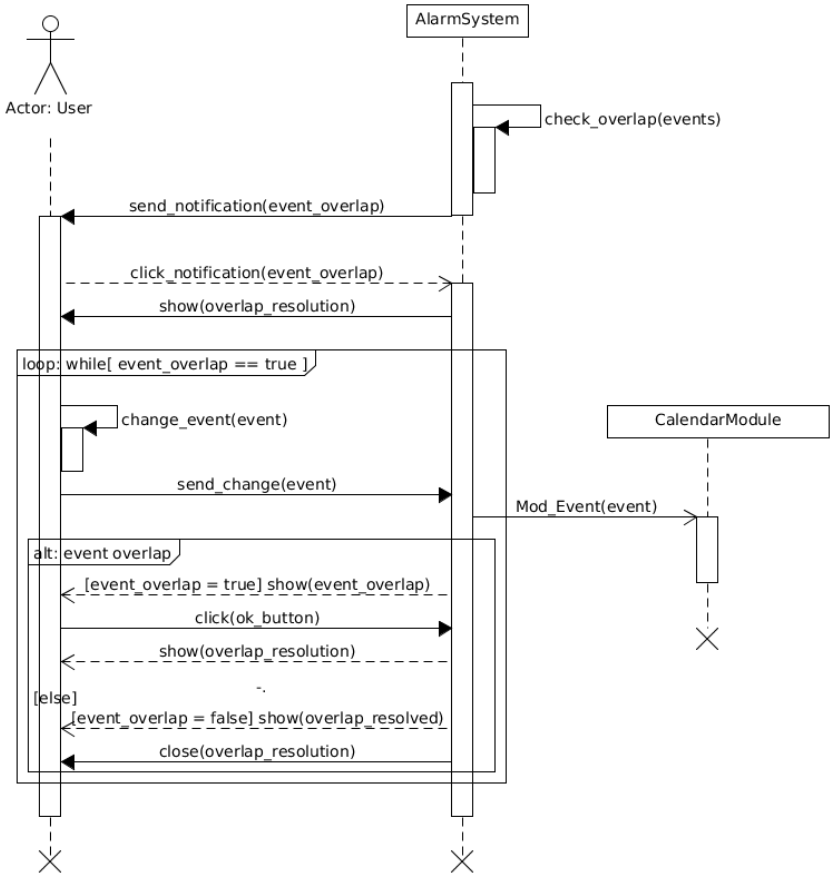
Add Event


Actor:User
CalendarModule
RouteModule
UserData DB
OptimalPathServer
get(main_screen)
show(main_screen)
click(add_event)
show(add_event)
loop : while [correct_event_info == false]
fill_event(add_event)
send(add_event)
alt: incorrect event information
[correct_event_info = false] show(error_msg)
click_on(ok_button)
show(add_event)
[else]
check_overlap(add_event)
[correct_event_infor = true] show(event_registered)
find_routes(add_event)
calcolate_route(add_event)
show(transport_pick)
route_option
pick(transport_pick)
save_event(add_event)
show(event_added)

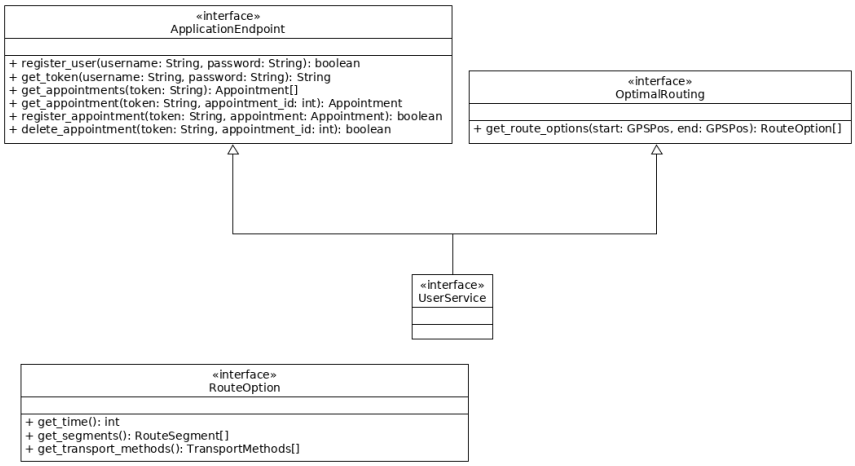
Buy Ticket


Actor: User
CalendarModule
ArrangeSystem
schedule_appointments(events)
check_ticket_purchase(events)
alt: ticket purchase found
show(ticket_purchase)
Transportation ticket purchase webpage
alt: ticket purchase chosen
click(purchase_button)
send(ticket_redirect)
go(ticket_purchase)
return(ticket_purchase)
[else]
click(decline_button)
close(ticket_purchase)

Notification



Component Interfaces



Architectural Styles

Overall Architecture

The software is divided in multiple tiers. On the server side there will be the following components:

1) Frontend endpoint server
2) Web scraping and API daemon to obtain transportation and weather data
3) RethinkDB document-oriented database
4) Optimal Path server

The client will be thin in both the web and android version.

Frontend Endpoint Server

The frontend endpoint server's purpose is to provide REST endpoint in JSON, it is the main and only interface between the client and the server. The API calls will be done through HTTP and will use a token infrastructure for authentication. All API calls will be processed through a secure connection. It will also include a portion of the business logic needed by Travlendar+. It will also integrate a proxy architecture with the OPS.

The following endpoints will be present in the first (v1) release:

- /v1/register_user
  - POST - registers the user into the system
- /v1/get_token
  - POST - user provides credentials and obtains a token to use for further transactions
- /v1/get_appointments
  - GET - receive a JSON describing the appointments specified in the date range
- /v1/get_route_options
  - POST - provides the routing alternatives given a source and destination point
- /v1/register_appointment
  - POST - registers an appointment, if it is specified the appointment is modified
- /v1/delete_appointment
  - POST - deletes the appointment

Web Scraping and API

This server is needed to obtain information pertaining the various transportation methods. As a architectural choice API will be favoured over raw website scraping if possible.

Optimal Path Server

The optimal path server's purpose is to calculate the optimal transportation options given a starting and ending point and user preferences. The OPS will exclusively communicate with the FES via a JSON REST API. It will make use of a OSM routing library to handle the route computation.

In the first release there will be the following endpoints:

- /v1/calculate_path
    - calculates a list of optimal paths

Other Design Decisions

- RESTful with transition support architecture system.
- RethinkDB was chosen based on the need to store data such as geojson and other information in a structured manner, hence precluding the use of a relational database.
- Both GUI will employ an MVC pattern to manage the interaction between the GUI and the backend requests

Algorithm Design

The main algorithm of this application resides in the optimal path server. The OPS' job is to provide to the endpoint server the travelling options needed at the core of the application.

Transport Categories

The algorithm considers viable transport options based on the trip distance:

- 0 - 0.5km (short)
    - foot
    - bike sharing
- 0.5-10km (city)
    - foot
    - bike/bike-sharing
    - metro/bus
    - car-sharing
    - suburban train
- 10-20km (ex-city)
    - car/taxi
    - metro/bus/suburban train
- 20-100km (region)
    - car/taxi
    - train

- 100+ km (long)
    - airplane
    - car
    - train

Transport Switch Penalties

Each mode of transport has a inherited "transfer delay", which is put to account for events such as parking the car, moving through the station and buying the transport ticket. It also allows to priviledge routes with less modes of transportation, while allowing fast multi-transport options to be displayed.

Main Algorithm

Here follows an example of the path code, written in pseudocode.

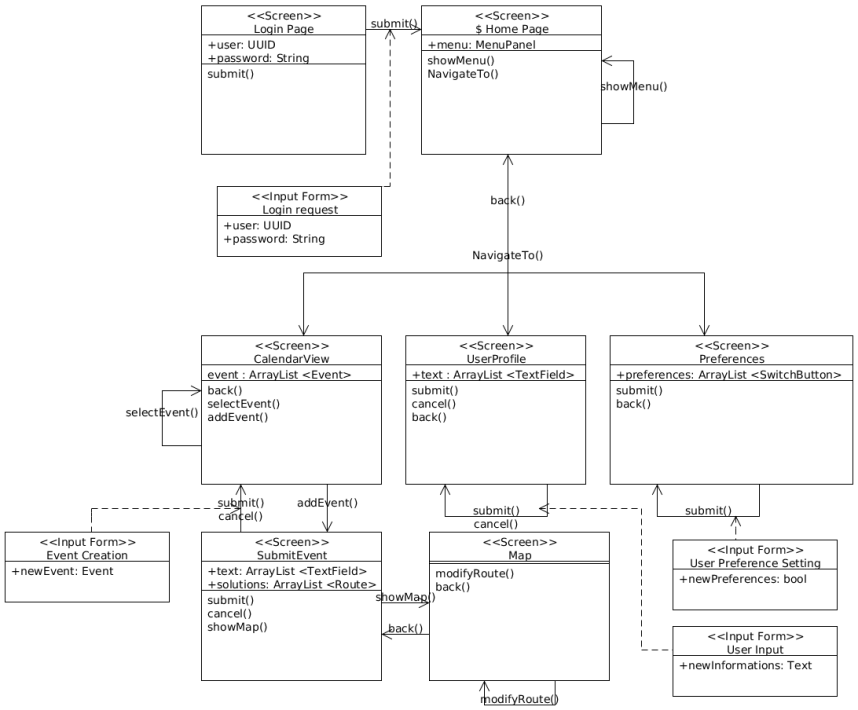
```
compute_path(start_coord, end_coord){
    min_bound = calc_foot_time(start_coord, end_coord);
    return compute_path_bound(start_coord, end_coord, min_bound);
}

compute_path_bound(start_coord, end_coord, time_bound){
    result = [];
    distance = calc_distance(start_coord, end_coord);
    if( distance == 0 ){
        return valid_null;
    }
    transports = get_transports(distance);
    for( single_transport : transports ){
        single_path = transport.compute_path_nearest(start_coord, end_coord);
        if( single_path.valid &&
            single_path.time + single_path.time_penalty < time_bound ){
            remaining_time = time_bound - (single_path.time + single_path.time_penalty);

            begin_distance = calc_distance(start_coord, single_path.start);
            end_distance = calc_distance(single_path.end, end_coord);

            begin_bound = begin_distance / (begin_distance + end_distance);
            end_bound = end_distance / (begin_distance + end_distance);

            result_begin = compute_path_bound(start_coord, single_path.begin);
            result_end = compute_path_bound(single.path_end, end_coord);

            if( result_begin.valid && result_end.valid ){
                result.add( result_begin + single_path + result_end );
            }
        }
    }
    return result;
}
```

User Interface Design

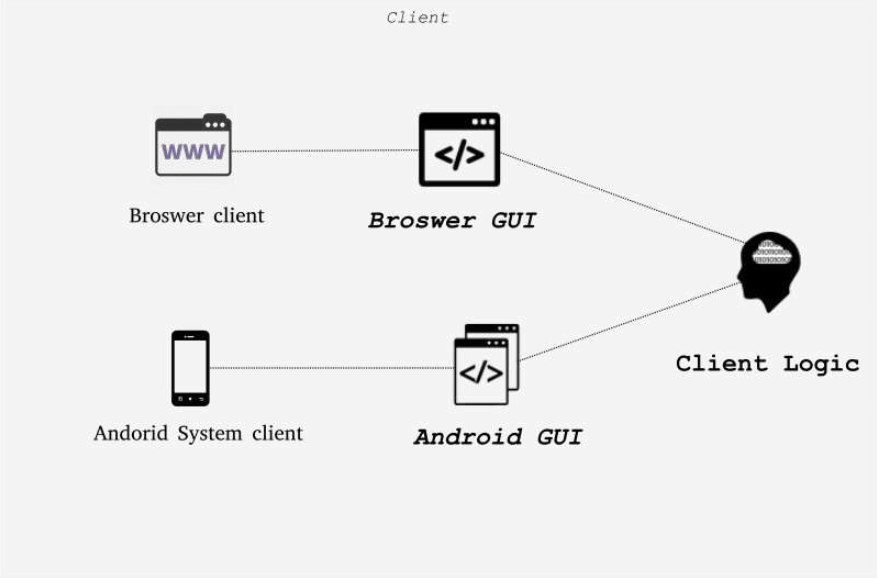
User Interface MockUp

see *External Interface Requirements* on RASD document.

UX Diagram



Gui Scheme



Gui elements can be reassumed on those elements:

1) **Broswer Gui** which is composed by all the html pages and insert section of the client broswer interface
2) **Android Gui** which is composed by all the Activities and java classes of the android layout app.

The Gui interfaces face the same application logic module, which is described in above sections. Broswer Gui and Android Gui arrange their elements following the UX diagram above to give to the user the same experience.

Client can access to Travlendar+ services in two different ways:
- Broswer: Mozilla, Chrome any broswer with html5, javascript support
- Android App: Avaiable for any Android 5.x Systems (API level 21)

The different GUI will be as much as possible similar focusing on the Material Design guidelines. (see *External Link* in *References*)

Communication between Gui and client logic is *Event Based*:
- Html5 Window Events and Javascript for the Broswer Client
- onCreate(), onStart() methods and Intents between Activities for Android Client.

Requirements Traceability

Referring to RASD document we can track the requirements on the components described.

Goals	Requirements	Components	Description
C1- Show a personal calendar of 'Events' submitted.	CR1- The system must store the submit of 'Event' of the User	Calendar Module, User DB	Can be used Local storage to have more Availability
	CR2- The system must have a visual calendar	Calendar Module, Broswer GUI, App GUI	
	CR3- The system must have a page navigation system	Calendar Module, Broswer GUI, App GUI	
C2- Let the User submit 'Events'	CR1- The system must store an 'Event' with Specified option	Calendar Module, User DB	
	CR2- The system must not let the User to create 'Event' in the past days.	Calendar Module	
	CR3- The system must provide a message of 'Warning' if the new Event overlaps with other	Calendar Module, Alarm System	
C3- Notify the User when the 'Event' is about to start	CR1- The system must warn the User that can miss or be late to an appointment	Alarm System, UserDB	Alarm System could query the UserDB to have informations on Users events or just access on Local storage
	CR2- Alarm System' can be configurate by the User	Alarm System, UserDB	
	CR3- The 'Alarm System' if activated must start before the start time of the 'Event'	Alarm System	
C4- Let the User to create 'Flexible Event'	CR1- 'Flexible Event' flag when the User is creating a new 'Event'	Calendar Module, Broswer GUI, App GUI,UserDB	
	CR2- A 'Flexible Event' can be overlapped to an another 'Event'	Calendar Module	
	CR3- A 'Flexible Event' can be copied and pasted on the Calendar and be repeated on several days of the same Week	Calendar Module	
	CR4- A 'Flexible Event' can be easly suppressed.	Calendar Module	
M1- Geolocate the gps coordinates of the Event	MR1- The system must provide gps API and be able to locate the position on a graphical map.	Route Module, App GUI, Broswer GUI	
M2- Calculate a list of possible shortest routes	MR1- The 'Best Route' Algorithm must return a list of shortest routes.	Optimal path server	
M3- Calculate the Extimation time of arrival at the destination	MR1- The system with the support of external API can calculate an Extimation time of arrival for a specified 'Best Route'.	Optimal path server	
M4- Let the User choose a route from the list provided	MR1- The system must provide a grphical list in which are presented all the possible 'Best Routes' and details of the itinerary.	Route Module, App GUI, Broswer GUI	
	MR2- The system must wait a choice of the User to save the route for the specified 'Event'.	Route Module, App GUI, Broswer GUI	
M5- Let the User modify the "Best Route" adding 'Costraint' for 'Intermidiate Locations', preferred 'Vehicle', max distance with a specified 'Vehicle' or max time on a specified 'Vehicle'.	MR1- The system must provide a graphical feature in which the user can modify the path adding location on the virtual maps.	Route Module, App GUI, Broswer GUI	
	MR2- The 'Best Route' Algorithm must update the Extimate time of arrival at destination depending on the geographical position of the 'Intermidiate Locations' added or the new 'Vehicle' speed average chosen.	Optimal path server	
	MR3- In case of 'Costraint' too much strict the system can return a 'Warning' message notifing the User that a 'Best Route' does not exist with that 'Costraint'.	Optimal path server	
M6- Consider on the possible 'Vehicle' avaible all the public transports of the city, railway stations, aeroports, train stations, car and bike sharing systems, bike, car and by foot.	MR1- The system must query information on timetables of the public transports of the city.	Optimal path server, TransportDB, Scraper Module	
	MR2- The system must notify on the virtual map stations of the public transports of the city.	Route Module, TransportDB, App GUI, Broswer GUI	
M7- Notify with a Warning message if the route chosen by the User is not good and he/she may arrive on late at the 'Meeting' because of its Extimation Time too long.	MR1- Before subitting the 'Event', the system must check if the time of the 'Event' and the 'Extimation' time of Arrival of the corrisponding 'Best Route' overlap with other 'Event' time start.	Route Module	
M8- Suggest a "Best Route" to the User with a 'Vehicle' which is appropriate for the day time of the appointment, the geographical location, the type of the meeting and the weather	MR1- The system must provide a "Suggested Route"	Optimal path server	
U1- Let the User to sign in to the Service filling an online form.	UR1- The system must provide a registration form to the User.	AccountManager, UserDataDb	
	UR2- The User is not signed in until all the fields of the form are not filled and valid.	AccountManager	
	UR3- The system must verify if the information on the registration form are valid.	AccountManager	
U2- Let the User to login to their personal User page and update their informations.	UR1- The system must provide an update function on the User profile page.	AccountManager, App GUI, Broswer GUI	
	UR2- The system must verify if the new informations are valid.	AccountManager	
U3- Let the User to login to their personal User page and update their informations.	UR1- The system must store the preference or dislike of the User	AccountManager, UserDataDb	
U4- Let the User buy online tickets for the majority of the public transports involved on the route chosen.	UR1- The system must provide an "Arrange System".	Arrange System	
	UR2- The "Arrange System" can query external systems and reserve vehicles for user.	Arrange System, TransportDB, Scraper Module	
	UR3- The "Arrange System" can redirect the user to secure pages in which can be buyed tickets for tranports inolved on the route chosen.	Arrange System	
U5- (optional) Let the User modify settings for the Algorithm 'Best Route'such activating 'Green Mode' or enabling options like "No traffic lighters", "No Schools at 16:00", "Show Autovelox".	UR1- The system must store all the setting of the Algorithm chosed by the User	UserDataDB	
U6- (optional) Let the User submit on his/her page the availability of public transports subcription, driver licence, coupons for special transports for best result on Algorithm 'Best Route' calculus.	UR1-The system must use if available those information when calculating the 'Best Route'	Optimal path server, UserDataDB	

Implementation, Integration an Testing

Elements to be Integrated

BackEnd

1) Handler Api
2) **RethinkDB** 1) UserDB
   2) TrasportDB
3) Optimal Path Server

Scraping Module

Scrap modules are not required for the testing and implementation of the other components. they will be used once the system has been completed to populate the DataBases. They can be implemented, tested and integrated independently of the other modules.

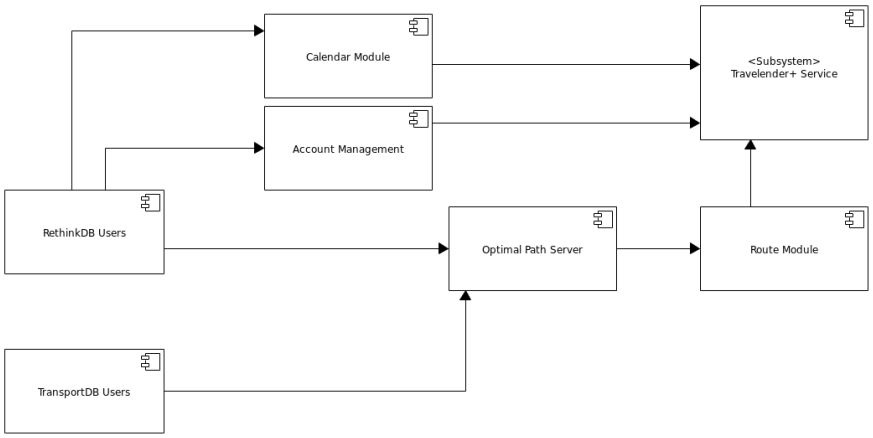
FrontEnd

1) GUI (android and Broswer)
2) Client Interface
3) Client logic
4) Alarm System

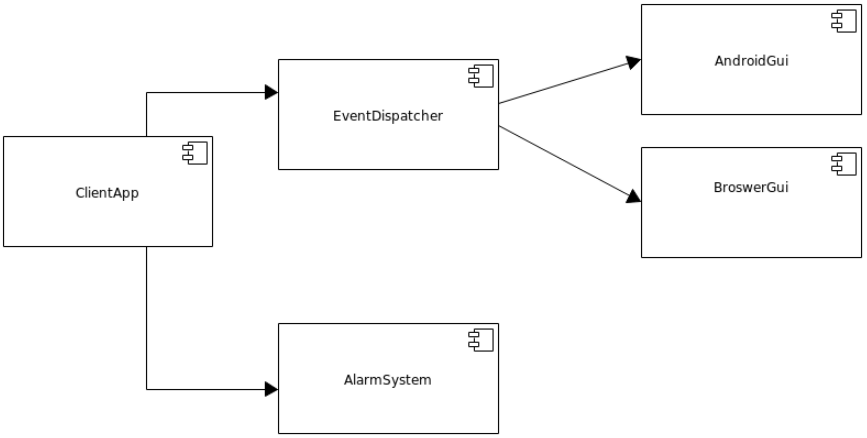
Component Integration sequence

In this section of the document will be described the order of integration of the components. As a notation, an arrow going from component A to component B means that A is necessary for B to function, so it must have already been implemented before performing the integration.

BackEnd



FrontEnd



Effort Spent

Prosdocimi Marco

6/11/2017 2h

12/11/2017 3h

13/11/2017 3h

14/11/2017 2h

15/11/2017 3h

17/11/2017 4h

20/11/2017 2h

21/11/2017 1h

24/11/2017 2h

26/11/2017 3h

Ruggiano Enrico

4/11/2017 1h

12/11/2017 2h

14/11/2017 5h

15/11/2017 3h

16/11/2017 2h

18/11/2017 3h

20/11/2017 3h

21/11/2017 1h

22/11/2017 2h

23/11/2017 2h

26/11/2017 3h

Giacomo Vercesi

7/11/2017 4h

13/11/2017 3h

15/11/2017 2h

17/11/2017 2h

22/11/2017 3h

23/11/2017 2h

24/11/2017 4h

25/11/2017 2h

26/11/2017 3h

References

External Link

REST architecture systems:

- https://en.wikipedia.org/wiki/Representational_state_transfer

Open Street Map library:

- https://wiki.openstreetmap.org/wiki/Geo::OSM_library

RethinkDB:

- https://www.rethinkdb.com/

Material Design:

- https://material.io/

Tool Used

Uml Diagrams:

- http://umletino.com/
- UMlet

Table:

- libreOffice Calc

Architecture schemes:

- Inkscape