Travlendar+ Design Document

Sinico Matteo, Taglia Andrea

Version 1.1

November 2017

Table of Contents

1. INTRODUCTION

A. Purpose

In the RASD document have been presented the objectives and a general overview of Travlendar+ system. The purpose of this document is to give more technical details about software and hardware choices that we have found for the system realization. This document will focus on software design to provide the fundamental specifications that will allow the developers to implement the system.

B. Scope

In order to best describe the architecture and the runtime view of the application, we are going to use different type of diagrams in order to convey as much as possible the information on the software-to-be's structure. In particular there will be presented some UML diagrams:

- Component Diagram, to describe the high and the low level of the software's architectural structure
- Deployment Diagram, to describe the hardware's architectural structure and how the software is mapped on it
- Sequence Diagram, to describe the behavior at runtime of the software

It will be stated which architectural pattern we have chosen and how they are used, we will give a description of the most crucial algorithms and a description of the user interface layout. Finally, for the sake of completeness, it will shown how every requirement defined in the RASD document is mapped in this document.

C. Definitions, Acronyms, Abbreviations

C.1. Definitions

- Preferences of travel: the set of preference chosen by the user.
- Day-travel Proposal: the plan of the day computed by the application for the user, which include the user's appointment, the means of transport suggested to move from an appointment to the other and the estimated time of the movement, it could be made of more than one travel-piece.
- Travel Piece: it is intended as a single atomic unit as part of a more complex route made by different transport means and different times, all represented by travel pieces. In details a travel piece it's composed by the following field:
    - From: the starting location;
    - By: the means of transport chosen;
    - Leave by: the time at which is suggested to leave, in order to be on time.
- Application: by this term we refer both to the web-app and the overall system.
- Web-app: a software program that runs on a web server
- Appointment and Event are used as synonymous.

C.2 Acronyms

• API: application program interface.

C.3 Abbreviations

[Rm.n] The n-th functional requirement related to the m-th goal.
[Cn] The n-th component of the application.
[Cn.m] The m-th sub-component of the n-th component.
[RWn] The n-th run time view.

D. Revision history

1. 24/11/2017 Document v1.0 ready

F. Document Structure

1. Introduction: A brief description of the content and the purpose of the document

2. Architectural Design: This will be the main section of the document and will provide the design of the application. It will contains all the UML diagrams needed to specify how the software is going to be implemented (thus, the software design) and the architectural choices made in the design process.

3. Algorithms Design: This section will provide an insight on how the most crucial and important algorithms of our application will work.

4. User Interface Design: This section will provide a point of view over the design of the user interface. This has already been already discussed in the RASD through some wireframes, however in this document will provide a more complete explanation of the user interface layout.

5. Requirements Traceability: In this section we will provide the mapping between the requirements identified in the RASD and our design decisions.

6. Implementation, Integration and Test Plan: Here the plan and approach to the implementation stage is explained. Then an overview about the integration and test plan is done. More details on testing methodologies are given in order to accurately guide this crucial stage.

7. Effort Spent: working time spent by each team member.

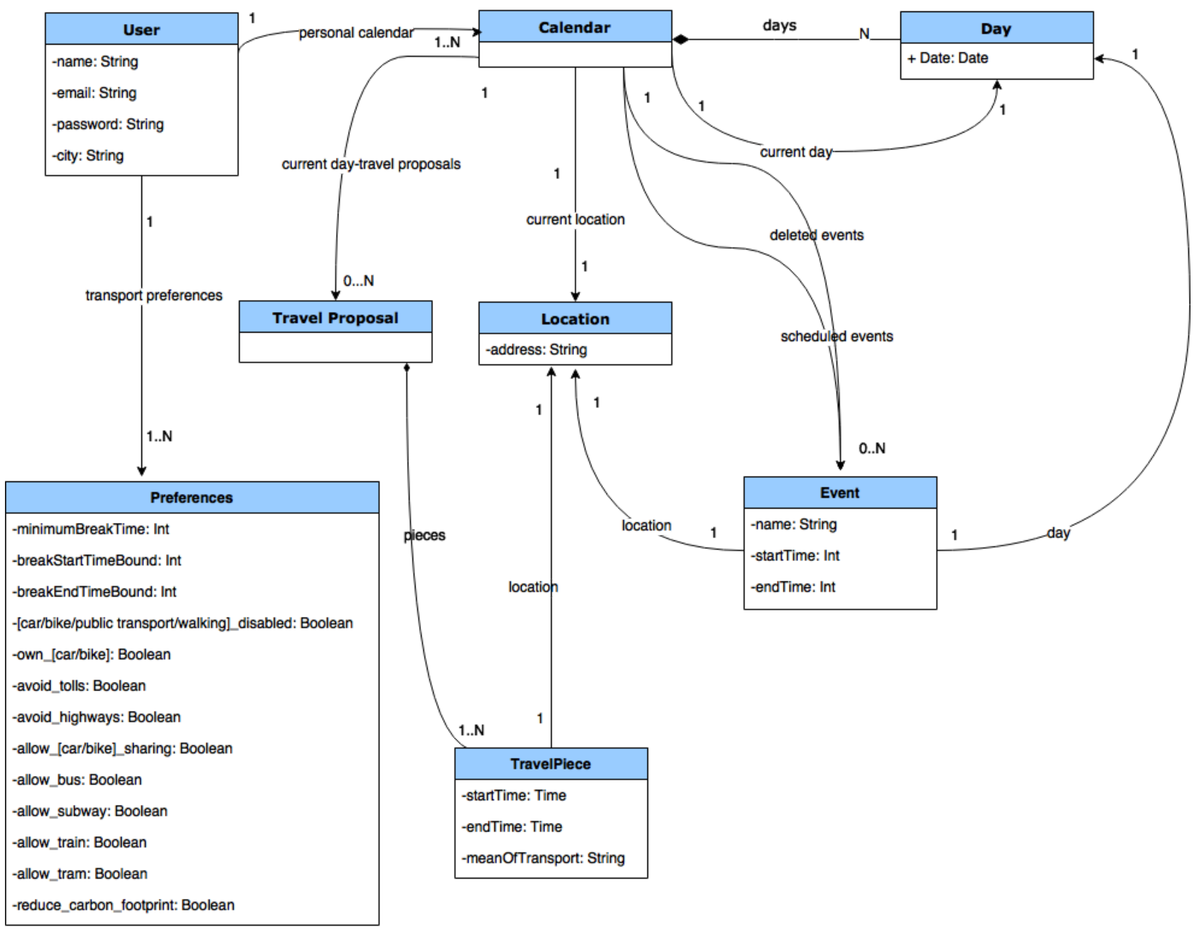
8. References

2. ARCHITECTURAL DESIGN

A. Overview:

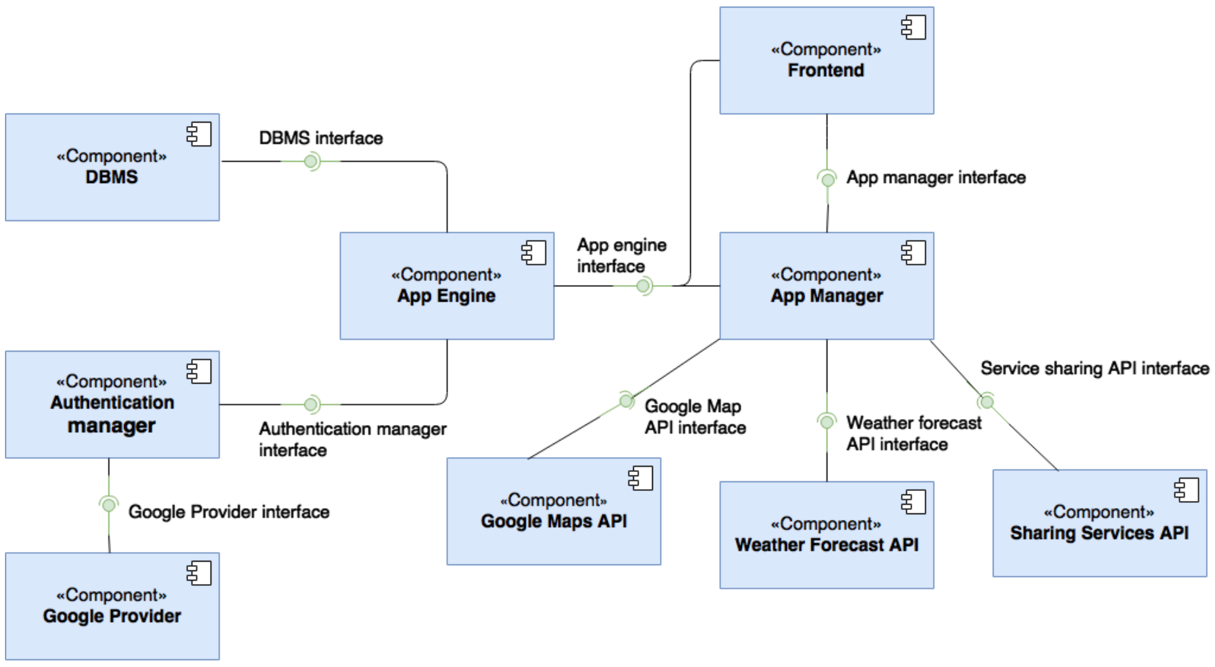
In this section is addressed the architectural design of Travlendar+. In the following subsections are depicted some of the most relevant shades of the system-to-build.

Therefore is given an insight over the software components of the system and the interfaces interleaved among them. Is also given a runtime view of the system to explain its behavior and is addressed the deployment of the software component to have a static view of what the system will looks like.
It also provided a more detailed class diagram with respect to the high level one provided in the RASD document:



B. High-level components and their interaction

In this section there is the component diagram of the application, that describes, at high level perspective the component of the application and the interactions between them.


«Component» Frontend
«Component» DBMS
DBMS interface
App manager interface
«Component» App Engine
App engine interface
«Component» App Manager
«Component» Authentication manager
Service sharing API interface
Google Map API interface
Weather forecast API interface
Authentication manager interface
Google Provider interface
«Component» Google Maps API
«Component» Weather Forecast API
«Component» Sharing Services API
«Component» Google Provider

B.1 DBMS [C1]

This is the data-base management system of the application that will manage all the data coming from the user and the application itself. These data in details are: user profile information (first name, last name, username, password, e-mail, sex, date of birth); user calendar (appointments) and user preferences (car, bike, public transport, walk, carboon footprint, break). The DBMS will provide an interface to allow the application to perform queries over the data.

B.2 Authentication Manager [C2]

This is the component of the application responsible both of the procedure of signing-in and signing-up. All the basic features, like password recovery, secure authentication will be guaranteed by the use of “Google Sign in”, which allows the user to sign in with its google account, without creating a new unuseful account. In order to do so the authentication manager needs an interface with Google sign-in, and has to provide an interface to the app engine.

B.3 Google sign-in provider API [C3]

Google Maps API Component takes care of the HTTPS requests and responses needed to allow the user to log in with his google account. API documentation available at https://developers.google.com/identity/sign-in/web/devconsole-project

B.4 App Engine [C4]

This component is the backend of the application, it’s the unique point of access to the data stored in the data-base and to the authentication manager. Thus it takes the HTTPS requests from the application and it translates it either in queries to the DBMS or in authentication request to the authentication manager. Then it retrieve the results to the rest of the application. So it’s the middleware between the data layer and the application logic layer.

B.5 App Manager [C5]

This is the core of the application, hence it’s responsible of receiving, either dispatching to the right component (app engine, google maps API, weather forecast API, sharing services API), or resolving all the request coming from the frontend of the application which are: sign-in, sign-up, edit profile information, edit calendar, evaluate day-travel proposal, edit preferences, show travel, (check the RASD document section B.2, B.3, and B.4 to have a complete view of this functionality). In order to do so it needs interface with all the component stated before, and it has to provide an interface to the frontend of the application.

B.6 Google Maps API [C6]

Google Maps API Component takes care of the HTTPS requests and responses needed to compute the estimated time it would take to get from one place to another by different means of transportation. API documentation available at https://developers.google.com/maps/documentation/directions/start

B.7 Weather Forecast API [C7]

Weather Forecast API Component takes care of the HTTPS requests and responses to the open weather service (API documentation at https://openweathermap.org/api) for weather forecasts data.

B.8 Sharing Services API [C8]

Sharing services API Component takes care of the HTTPS requests and responses to a urban mobility aggregator service that wrap in it all the most important sharing service available at the moment. This component will be bought choosing among the most known providers of this kind of service, of course, taking into account the solution which best will fit in the implementation phase.

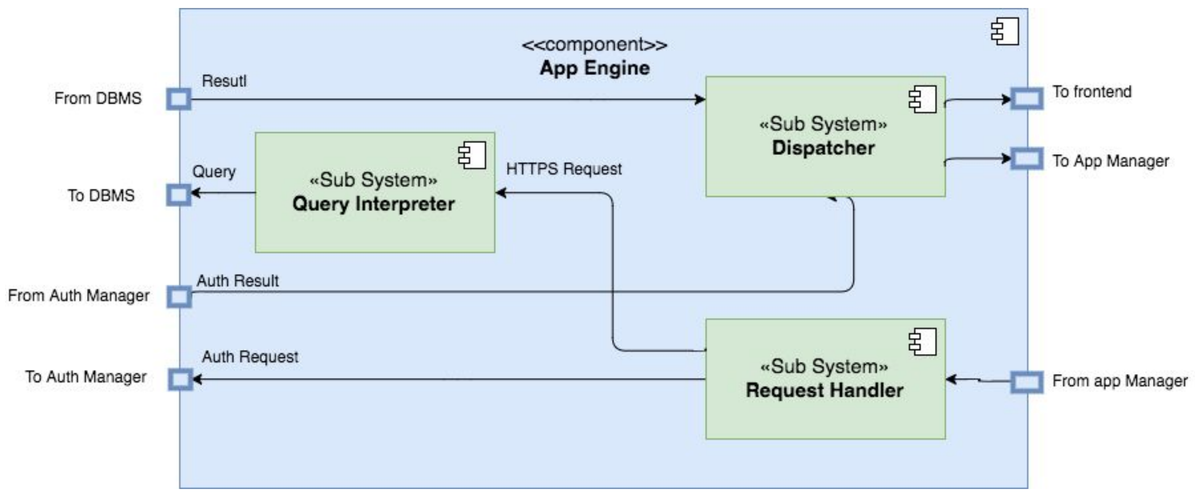
B.9 Frontend [C9]

This component it is the Graphic User Interface of the application. To see an example of it see the section 4 of this document. It requires interfaces both to the app manager and the app engine in order to perform request and to retrieve data and visualize it.

C. Component view

In this section is provided an insight of those components whose behavior must be refined to have a comprehensive view of the system to build.

C.1 App Engine

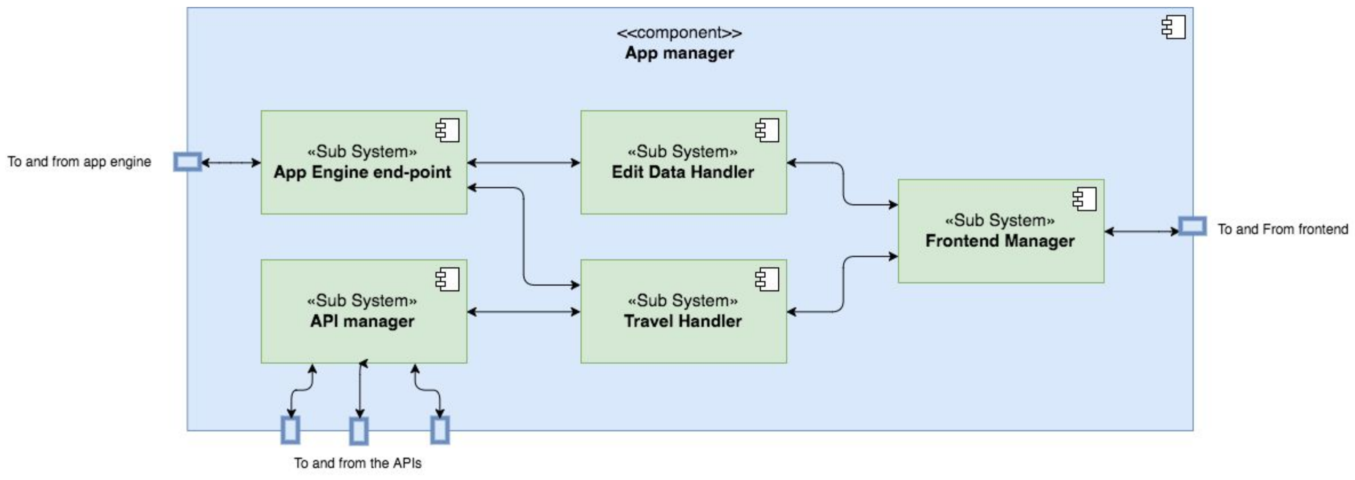


The app engine component is composed of 3 sub system:

- Request Handler
- Dispatcher
- Query Interpreter

that are three piece of software which accomplish different task.
The **Request Handler [C4.1]** receives the HTTPS requests from the app manager and decides either to direct these requests to the query interpreter or to redirect them directly to the Authentication Manager, simply reading the header of the HTTPS requests.
The **Dispatcher [C4.2]**, the other way round, receives the results from the Authentication Manager and from the DBMS and then sends them either to the frontend or to the app manager, respectively if they are information that update the view of the application or if they are just requested by the app manager to compute the day travel proposal.
The **Query interpreter C[4.3]** receives the HTTPS requests from the Request Handler and then translate them into queries understandable by the DBMS.

C.2 App Manager



The App manager is composed of 5 sub system:

- FrontEnd Manager

- Edit Data Handler
- Travel Handler
- App Engine end-point
- API Manager

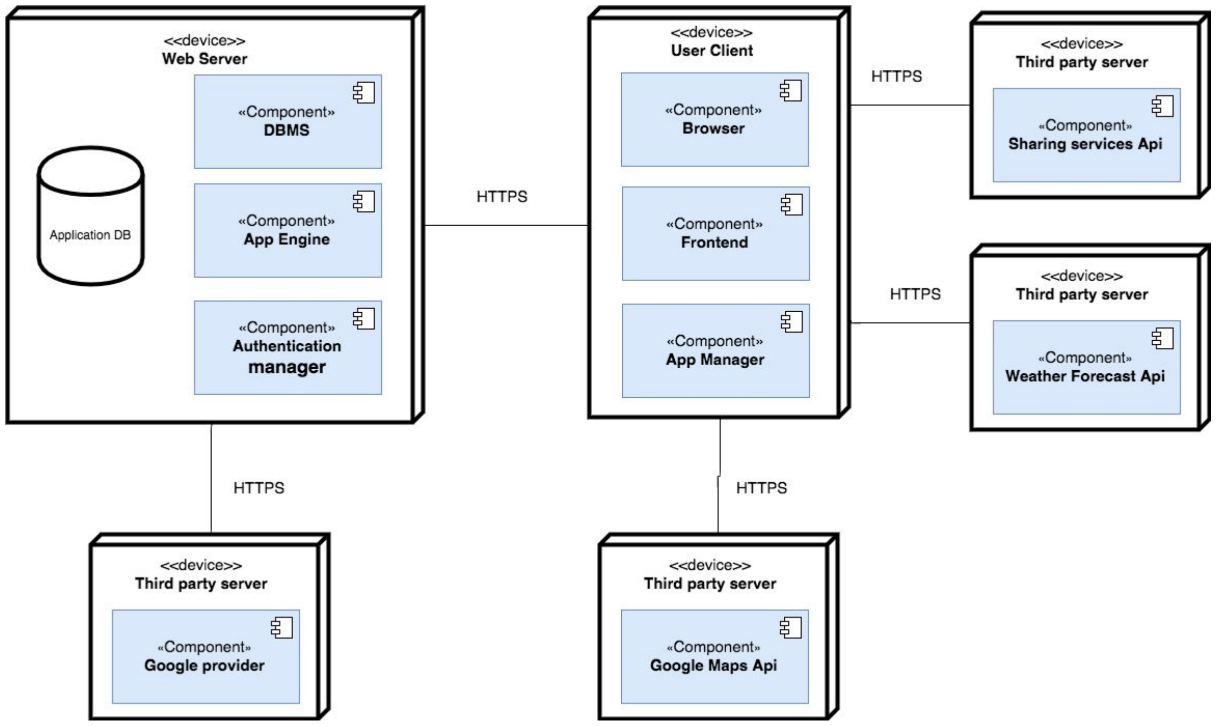
each of them is a different piece of software which accomplish a different task.
The **FrontEnd Manager [C5.1]** is responsible of receiving the requests from the frontend and dispatch them to the correct sub system: either the edit data handler if the request it's a query or a modification of the data, the travel handler if it's a request to compute the day travel proposal either default or not, or the App engine end-point if it's an authentication request.
The **Edit data handler [C5.2]** handles all the query and data modification request (edit calendar, edit profile, edit preferences, show travel)[1]. The sub-component is responsible of checking the feasibility of the data inserted or modified:

- edit calendar: the date, the starting and the ending hour of the appointment have to exists on the calendar; if the appointment day is the current day the start hour has to be greater than the current hour.
- edit profile: the date of birth has to be an existing day on the calendar.

The **Travel Handler [C5.3]** is the real core of the application because is the component responsible of the computation of the day travel proposal. Thus it has to be able to access both the DBMS and the APIs and also to the frontend to retrieve the results. A detailed explanation of the algorithm used to compute the day travel proposal is shown in the section 3 of this document.
The **App engine end-point [C5.4]** both wraps the requests from the app manager components in an HTTPS request ready to be sent to the app engine and, the other way round, unwraps the HTTPS responses in an understandable format for the app manager components.
The **APIs manager [C5.5]** redirects to the correct API the requests from the travel handler.

[1] please refer to the RASD document (sections 3.B.2 and 3.B.3) to have a detailed view of all the functionalities offered by the application.

D. Deployment view



The criterion which has lead the deployment is dictated by the choice of adopting a classical two tier client-server implementation[2].
All the APIs are in a stand-alone block because we see them as black-box to which sending requests and from which receiving responses to utilize in the other component of the application, all the requests and the responses are sent and received by HTTPS requests.
The core of the application ,and obviously the frontend, are deployed in the user client, while the data management and the security features are kept together in a unique tier in order to reduce the cost and to improve the performance of the application. The two tier communicate via standard HTTPS protocol.

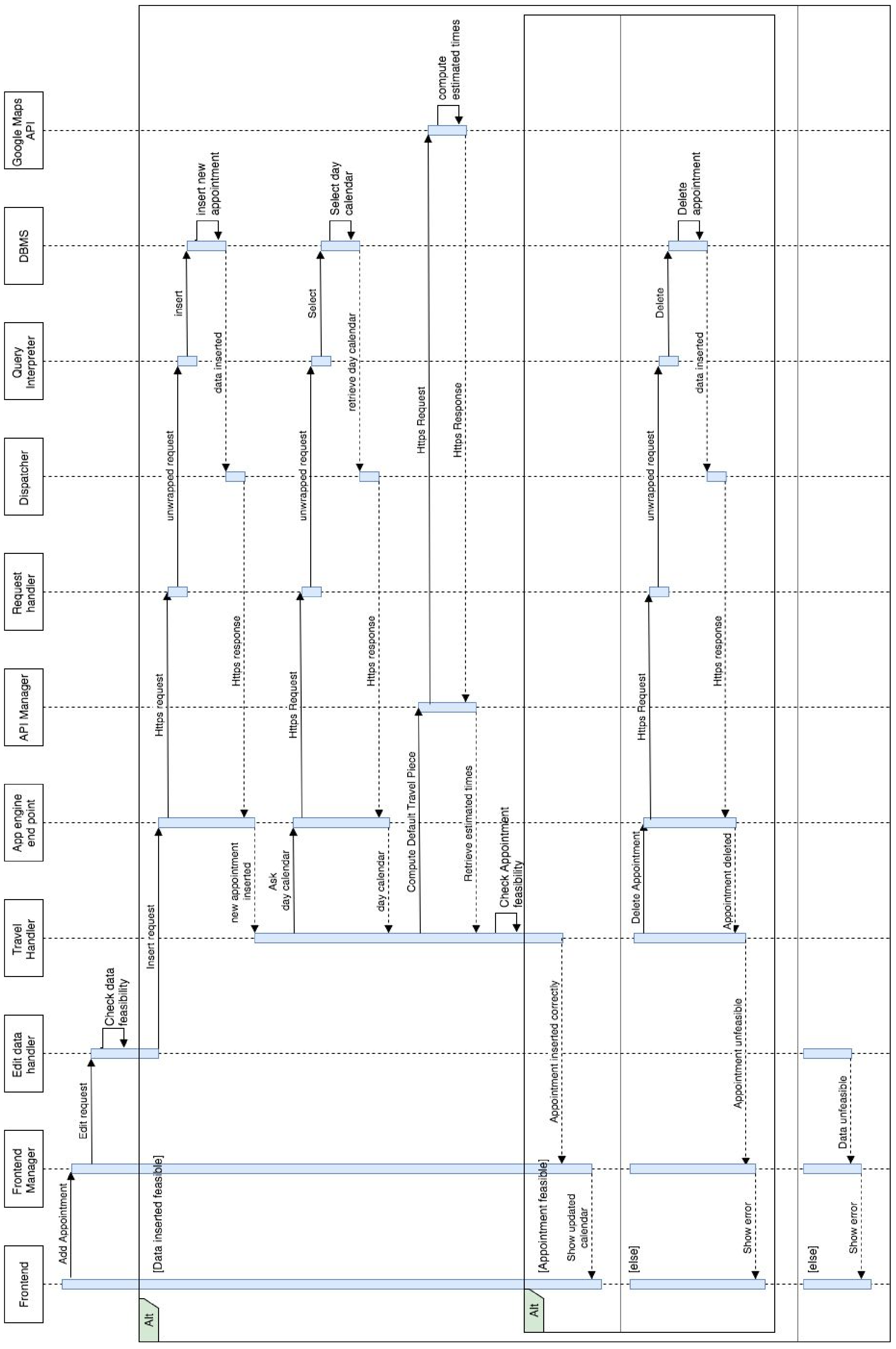
E. Runtime view:

Here it is described how the components interact to accomplish specific tasks:

[RW1]

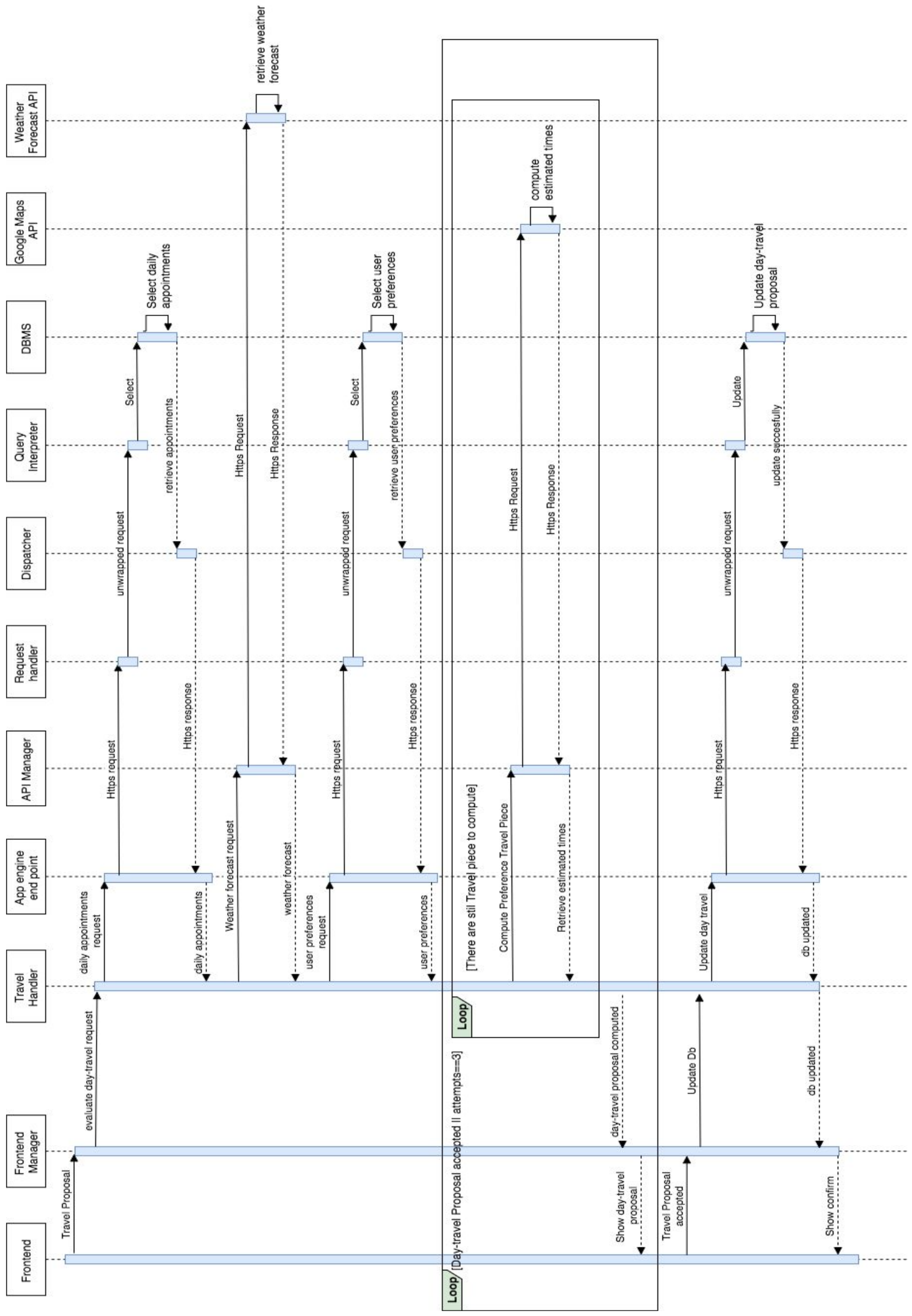
The following runtime view refers to the second scenario of the RASD document (SC2), and shows how an appointment is added to the personal user calendar.

[2] please refer to section 2.G of this document to have a detailed explanation on the advantages offered by a two tier architecture.


Frontend
Frontend Manager
Edit data handler
Travel Handler
App engine end point
API Manager
Request handler
Dispatcher
Query Interpreter
DBMS
Google Maps API
Alt
[Data inserted feasible]
Add Appointment
Edit request
Check data feasibility
Insert request
Https request
unwrapped request
insert
insert new appointment
data inserted
Https response
new appointment inserted
Ask day calendar
Https Request
unwrapped request
Select
Select day calendar
retrieve day calendar
Https response
day calendar
Compute Default Travel Piece
Https Request
compute estimated times
Https Response
Retrieve estimated times
Check Appointment feasibility
Alt
[Appointment feasible]
Appointment inserted correctly
Show updated calendar
[else]
Delete Appointment
Https Request
unwrapped request
Delete
Delete appointment
data inserted
Https response
Appointment deleted
Appointment unfeasible
Show error
[else]
Data unfeasible
Show error

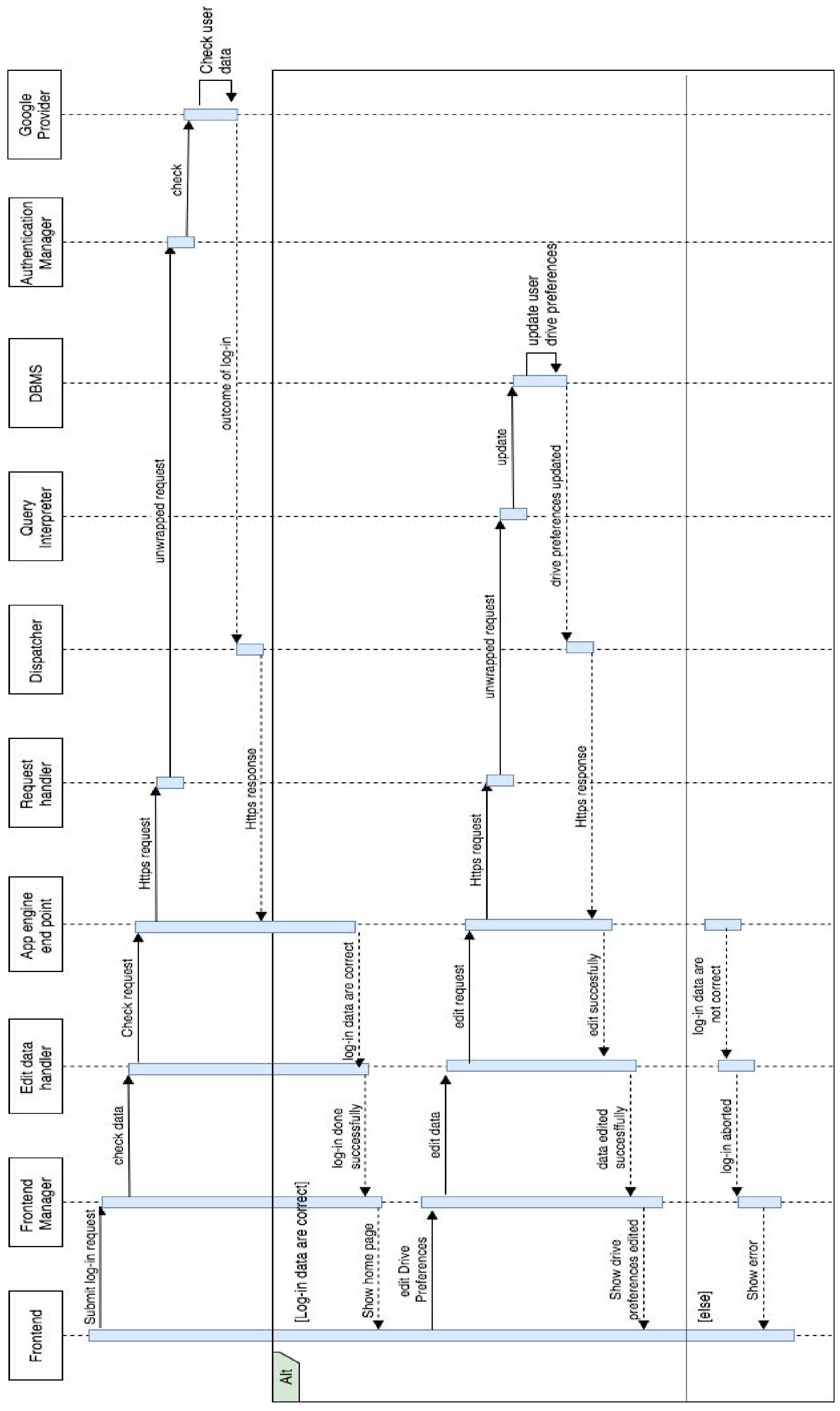
[RW2]

The following runtime view refers to the first scenario of the RASD document (SC1), and shows the core of the application: how it computes day-travel proposal.



[RW3]

The following runtime view refers to the third scenario of the RASD document (SC3), and shows how is managed the log-in request and an update of the user preferences.



F. Component interfaces

The application architecture is taught to be simple and to minimize the type of message that the component can send each other. To make this possible each component has two sub-component: the **"receiver"** that manage all the requests which arrives from the other components of the application, and the **"dispatcher"** which is responsible to dispatch either all the internal request to the right components, or the results produced in the current component, to the components which have requested them before. Then we can have one receiver and one dispatcher for each component or only one.
Hence the App manager has the Frontend Manager, the App engine end-point and the API manager: the first is the all-in-one receiver, the second and the third are the dispatchers.

G. Selected architectural styles and patterns:

The architecture choice for Travlendar+ web application consists of a classical two tier client-server with a pretty thick client, as most of the core algorithms will be computed client-side exploiting modern powerful devices affordable for the vast majority of the population. Adopting a two tier architecture is dictated by these few advantages:

- Application can be easily developed due to the simplicity of the architecture
- Maximum user satisfaction is gained with accurate and fast prototyping of application through robust tools

Components of the application have been splitted up amongst tiers based on a Model View Controller (MVC) pattern. As shown in the Component Diagram in section 2.B of the current document, the entire model is placed on the server side, while the view and controller will run on clients. Please refer to section 3 of the current document for an in depth view of the controller main algorithms.

As better described in the following section, from a client point of view the application will be a pioneer of the new Progressive Web Applications first proposed as soon as two years ago. It will be a web applications with app-like user experience, including offline capabilities.

It is mandatory to state the actual architecture onto which will be based Travlendar+ application, as it dictates the architectural style being used. Indeed, the server will be completely handled by the Firebase development platform which takes away server managing issues to the developers which will totally rely on the scalability and security offered by this platform. In this architecture, Travlendar+ dynamic content and user data is stored and retrieved from Firebase.

This pattern allows the developer team to focus on the very contents delivered to the user and it will be required much less caring about the back-end.

H. Other design decisions

The main design choice concerns the intention of shaping Travlendar+ has a Progressive Web App. Here we report the main characteristics according to Google Developers:

- Progressive - Work for every user, regardless of browser choice because they're built with progressive enhancement as a core tenet.
- Responsive - Fit any form factor: desktop, mobile, tablet, or forms yet to emerge.

- Connectivity independent - Service workers allow work offline, or on low quality networks.
- App-like - Feel like an app to the user with app-style interactions and navigation.
- Fresh - Always up-to-date thanks to the service worker update process.
- Safe - Served via HTTPS to prevent snooping and ensure content hasn't been tampered with.
- Discoverable - Are identifiable as "applications" thanks to W3C manifests and service worker registration scope allowing search engines to find them.
- Re-engageable - Make re-engagement easy through features like push notifications.
- Installable - Allow users to "keep" apps they find most useful on their home screen without the hassle of an app store.
- Linkable - Easily shared via a URL and do not require complex installation.

One of the main reasons which led to this design direction is the possibility of having Travlendar+ installable in a way which is not the usual application that needs to be downloaded from the store. Web accessibility was one the main priorities has it is the fastest way to engage users for the first time. At that point a link to the app store would require the download and consequent installation of the app. This is something we have tried to avoid as we want things to be quick and straightforward for the users. The capability of an installation-like approach where no actual download and no actual device space wasting is required is something Travlendar+ project needs to have great impact into the market. Progressive Web Application gives this capability exploiting new browsers feature (service-worker in this case).
This is a real effort that will be made in implementation stage to explore new technologies in order to give a real boost the the project as to make it stepping out from competitors.

3. ALGORITHM DESIGN:

The most relevant algorithms are described below in terms of input necessary for the algorithm to be executed, output that it should return, logic steps it should follow during execution. These high level descriptions would then become the algorithms specification for the developers team.

A. Compute Travel Algorithm

The actual core of the whole application, the algorithm looks for the optimal route to get the user from one appointment to another taking into account his personal preferences.

INPUT :
- List of user personal Events from current day, each one including:
  - event location
  - start time
  - end time
  - name
- User's break time preferences (all integer values):
  - minimum time
  - lower start time bound
  - upper end time bound
- User's travel preferences (all binary values):
  - [car/bike/public_transport/walking]_disabled

- own_[car/bike]
- avoid_highways
- avoid_tolls
- allow_[car/bike]_sharing
- allow_[public transport - bus]
- allow_[public transport - subway]
- allow_[public transport - train]
- allow_[public transport - tram]
- reduce_carbon_footprint
- home location

We cannot make any assumption about the availability of the Preferences input data after user first login as the user is not forced to provide some at first access. This is the reason why there will be needed to set a first login profile with some default Preferences, making it possible to assume that input data will always be available allowing the algorithm to always work.
As for the list of Events, even if we get an empty list the algorithm manages the case.
We need to add to user's travel preferences the field "home location" in order to better organize the user's day travel proposal, because it's supposed that the first and the last event of an average user's day are the departure and the return to his home.

EXECUTION :

1. prepare output list of size Events + 1
2. n = 0
3. for each couple of Events i,j do //note that first and last event location will be home
    a. if (weather forecasts expect rain for that time) disable walking and cycling
    b. if (reduce_carbon_footprint == true) try to compute the travel piece in this order: walking, bicycling, transit, driving.
    c. else() try the other way round;
    d. if (not car_disabled && (own_car || allow_car_sharing)) API requests with origin = i.location, destination = j.location, arrival_time = j.startTime, mode = driving, avoid preferences
    e. if (event feasible with current preferences) create TravelPiece with info received;
    f. else() try with the following in the priority order.
    g. same for bike. if
    h. same for public transport -> append transit options based on preferences
    i. same for walking.
    j. n++

OUTPUT :

- List of Travel Piece which will be interleaved with Events to make a full day schedule. Each Travel Piece is composed of:
    - mean of transport
    - starting location
    - time to start travelling

B. Add Event to Calendar

Another important algorithm which deserves more attention is the one concerning the act of adding a new event by the user. It is extremely relevant to have a first high level definition of the very operations made to ensure no unschedulable event is put into the calendar, as it would make the day travel computation not feasible and thus breaking down the main feature of Travlendar+.

INPUT :

- New Event data:
    - day
    - event location
    - start time
    - end time
    - name

EXECUTION :

1. API requests with origin = previousEvent.location, destination = nextEvent.location, mode = transit, arrival_time = nextEvent.startTime
2. if (request.time_to_start_travel <= previousEvent.endTime) throw error pop up
3. if (request.time_to_start_travel between newEvent.day.breakTimeStart and newEvent.day.breakTimeEnd && not newEvent.day.scheduleNewBreak(newEvent)) throw error pop up

// note that newEvent.day.scheduleNewBreak takes an Event as input and tries to find a suitable time where a break could be placed taking into account user's preferences and the new event inserted into the day schedule. Returns false if it fails. Returns true and updates day break time start and end if it succeeds

4. do the same with nextEvent instead of previousEvent

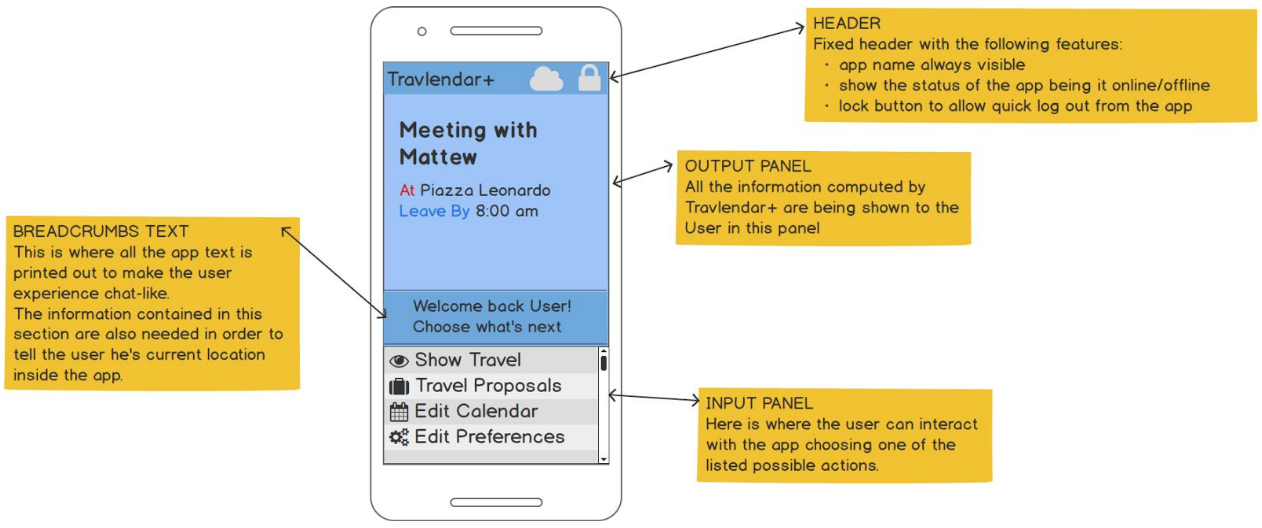
OUTPUT :

- User's calendar is updated with a new event or a pop up alerts the user that the event is not schedulable and user's calendar does not change

4. USER INTERFACE DESIGN:

User experience general concept has already been discussed and justified in the RASD document (please refer to section 3, sub-section A.1) , also a first sketch on how the user interface could look like have been presented in the same section. Below the UI is explored further giving more precise details for the development stage.
It has been decided to have a fixed layout that will adapt to any screen (even though smartphone devices are the target which we are mainly interested in).

UI layout must be the same for each page of the application. Home Screen sample is being used to show the sections into which the general UI layout should be splitted down:



HEADER
Fixed header with the following features:
· app name always visible
· show the status of the app being it online/offline
· lock button to allow quick log out from the app
OUTPUT PANEL
All the information computed by Travlendar+ are being shown to the User in this panel
BREADCRUMBS TEXT
This is where all the app text is printed out to make the user experience chat-like.
The information contained in this section are also needed in order to tell the user he's current location inside the app.
INPUT PANEL
Here is where the user can interact with the app choosing one of the listed possible actions.
Travlendar+
Meeting with Mattew
At Piazza Leonardo
Leave By 8:00 am
Welcome back User! Choose what's next
Show Travel
Travel Proposals
Edit Calendar
Edit Preferences

Header section indicators are needed to show to the user the progressive web app offline feature which makes it easy to check whether the app is online (simple cloud icon) or offline (cloud with slash icon).
The header must also hold the locker which is needed for quick log out from the app as to help preserving users' privacy.

The output panel could sometimes be slidable if the output content doesn't fit the available space for the container. Only a few very intuitive exception are allowed to let the user use the panel with some inputs inside.

Breadcrumbs text should always have an informal style.

Input panel should provide list items with icons along with text to make it as quick and clear as possible for the user to understand what he can do from that screen. The panel can be vertically slidable if the list items don't fit the available space.
There must always be an option to go back (except for the home page) to previous screen.

5. REQUIREMENTS TRACEABILITY:

Goal	Requirement	Use case	Component	SubComponent
G1	R1.1	UC2	C2 / C3	-
G1	R1.2	UC1	C2 / C3	-
G1	R1.3	UC3	C2 / C3	-
G1	R1.4	UC7	C5	-
G1	R1.5	UC4	C5	C5.2

Goal	Requirement	Use case	Component	SubComponent
G2	R2.1	UC8	C5 / C4 / C1	C5.2 / C4.3

G2	R2.2	UC7	C5 / C4 / C1	C5.2 / C4.3
G2	R2.3	UC9	C5 / C4 / C1	C5.2 / C4.3
G2	R2.4	UC11	C5 / C4 / C1	C5.2 / C4.3
G2	R2.5	UC12	C5 / C4 / C1	C5.2 / C4.3
G2	R2.6	UC14	C5 / C4 / C1	C5.2 / C4.3
G2	R2.7	UC15	C5 / C4 / C1	C5.2 / C4.3
G2	R2.8	UC13	C5 / C4 / C1	C5.2 / C4.3
G2	R2.9	UC16	C5 / C4 / C1	C5.2 / C4.3

Goal	Requirement	Use case	Component	SubComponent
G3	R3.1	UC17 / UC18 / UC21	C5 / C4 / C1	C5.2 / C5.3 / C4.3
G3	R3.2	UC17 / UC20	C5 / C7	C5.3 / C5.5
G3	R3.3	UC17 / UC18	C5 / C6	C5.3 / C5.5
G3	R3.4	UC17	C5 / C9	C5.3
G3	R3.5	UC17	C5 / C4 / C1	C5.2 / C5.3 / C4.3
G3	R3.6	UC17	C5	C5.3

Goal	Requirement	Use case	Component	SubComponent
G4	R4.1	UC7	C5 / C6 / C4 / C1	C5.3 / C5.5 / C4.3
G4	R4.2	UC18 / UC7	C5 / C6 / C4	C5.3 / C5.5 / C4.3
G4	R4.3	UC7	C5 / C4 / C1	C5.2 / C5.3 / C4.3

Goal	Requirement	Use case	Component	SubComponent
G5	R5.1	UC19 / UC17	C5 / C4 / C6 / C7 / C1	C5.3 / C5.5 / C4.3
G5	R5.2	UC19	C9	-
G5	R5.3	UC19 / UC7	C9	-
G5	R5.4	UC19 / UC7	C9	-
G5	R5.5	UC18 / UC7	C9	-

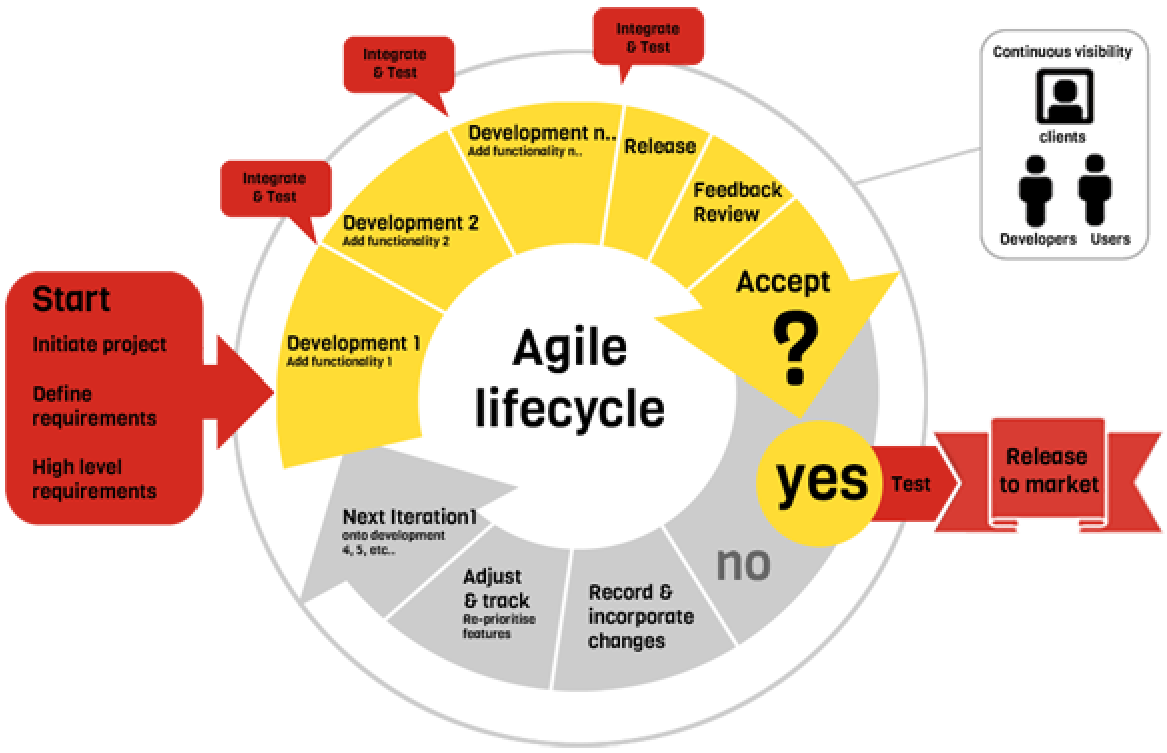
The sub-components C5.1, C5.4, C4.1, C4.2 are required to satisfy all the reqirements being the sub-components responsible to interface the differents components of the application.

6. IMPLEMENTATION, INTEGRATION AND TEST PLAN:

Implementation stage of the Travlendar+ app development will be tackled using an Agile approach. After a deep evaluation of the context into which the app has to be developed, there are many reasons which led to this choice:

Agile adapts well to changing priorities. Implementing agile ensures that the software which is going to be developed is up to date.
There will be intermediate releases which can really help estimating the quality of the work being done.
Also testing will benefit from it, thanks to the possibility of a much more focused (web-component granularity) software analysis. It is far preferable to test software in smaller pieces as it is being developed than develop a whole project and test it at the last stage. This is even more relevant for Travlendar+ as many features needs to be built on top of other. This way it can be sure to develop on top of well tested building blocks.

The Agile lifecycle which will dictate the implementation and testing phases can have a graphical representation in the picture below:



Web components will play a fundamental role in the implementation with such approach. They make possible to isolate various part of the web application by encapsulating behaviours and styles into separate logical components, which often also maps to a physical file level separation.

After very few features, and thus web components, have been implemented, they will be integrated in terms of both front end positioning inside the application and application data flow. This step will be placed right before release stage.

Once the release is approved, testing phase will take place. The top-down approach was, at first, taken into account, but then discarded because is mainly used when the project needs to speed up this phase due to deadline constraints, indeed, with this strategy the low level components are roughly tested.
Instead, testing will be done in a bottom-up manner in order to have a robust checkpoint which acts as a building block for next iterations. It will consist in both black-box and white-box testing based on "Web Component Tester which is an end-to-end testing environment built by the Polymer team. It enables testing of elements locally, against all of the installed browsers, or remotely, via Sauce Labs. It is built on top of popular third-party tools, including" :

- Mocha for a test framework, complete with support for BDD (Behaviour-Driven Development) and TDD (Test-Driven Development).
- Chai for more assertion types that can be used with Mocha tests.
- Selenium for running tests against multiple browsers.
- Accessibility Developer Tools for accessibility audits.

Google Insights tools will also have their say from a performance viewpoint. This is the black-box testing part where the application is evaluated by just looking at the outside behaviour without any look at the code. There will only be considered the network-independent aspects of page performance: the server configuration, the HTML structure of a page, and its use of external resources such as images, JavaScript, and CSS.

Lighthouse popular automated testing tool will also play an important role during the testing phase, giving some statistics on how well the application performs and how much it adheres to best practices.

Application components (please refer to section 2.B of the current document) implementation, integration and testing are splitted up into agile iterations as reported by the schema below:

1. Skeleton of the FrontEnd component implemented and tested.
2. Some of the App Manager heavily rely on data gathered from the DBMS, thus it seems reasonable to have the database schema built first. Indeed, this will be the next planned step. The schema will just be implemented here.
3. App Manager sub components are the most delicate to deal with and therefore a great testing effort should be spent here. At this step the first sub components to be implemented are the FrontEnd Manager alongside the Travel Handler. The components must then be integrated together before being carefully tested.
4. API manager and App Engine sub components will then be implemented and integrated with the currently available Travel Handler. Testing phase takes place.
5. The more trivial Edit Data Handler is implemented at this stage with consecutive integration with other sub components.

6. The whole finished App Manager is exposed to testing.
7. Integration between App Manager and FrontEnd takes place.
8. App Manager attaches to external API components being implemented at this stage.
9. App Engine implemented, integrated with the rest of the current system.
10. DBMS attached to App Engine to test App Manager with real data.
11. Heavy testing phase.
12. Authentication Manager implemented and integrated with the rest of the system.
13. System in now ready for final testing.

A. Definition of Views

Regarding the implementation it will be very useful to fix content of the application views:

- **Home View**
    - Output Panel

  display the Event-component regarding the next event of the current day.
    - Text Panel

  “Welcome back [[user.name]]! It’s [[currentDate]] and there is your next event of the day.”
    - Input Panel

  Show Travel, Compute Travel Proposals, Edit Calendar, Edit Preferences

- **Show Travel View**
    - Output Panel

  display the current Travel Proposal (list of Travel Piece components and Event components)
    - Text Panel

  “Here you can see the best Travel Proposal computed so far. You can compute more Proposals if you don’t like this one anymore. ”
    - Input Panel

  Compute other Proposals (same as Compute Travel Proposals), Go Back

- **Compute Travel Proposals View**
    - Output Panel

  after a potential loading, display one of the results of Compute Travel Algorithm (please refer to section 3.A) which is a Travel Proposal component
    - Text Panel

  “Alright! This is what we came up with as a travel plan for you. You can Accept the proposal or you can try new ones by tapping on Next proposal.”
    - Input Panel

  Accept Proposal, Next Proposal, Go Back

- **Edit Calendar View**
    - Output Panel

show the currently inserted events
    - Text Panel

"This is where you can add or remove events"
    - Input Panel

Add Event, Delete Event, Go Back

- **Edit Preferences View**
    - Output Panel

list currently set preferences
    - Text Panel

"Here you can edit all your personal preferences. Above you can find all the currently set preferences"
    - Input Panel

Edit Break Time Pref, Edit Transport Pref, Go Back

7. EFFORT SPENT:

The effort spent by the team is of 41 hours for each group member (on average).

8. REFERENCES

A. Bibliography

- IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications, which can be retrieved on the beep page of the course.
- Agile Lifecycle Image taken from http://cyclosys.com/Content/img/portfolio/
- Polymer Project https://www.polymer-project.org
- Google Insights https://developers.google.com/speed/docs/insights
- Google Developers/PWA https://developers.google.com/web/progressive-web-apps/

B. Used Tools

- Balsamiq Mockups 3
- Draw.io
- Google Docs