Design Document

Politecnico di Milano

• Antonino Caminiti (mat. 835724)
• Daniele Gonella (mat. 827091)
• Matteo Scerbo (mat. 899823)

1. Introduction

A. Purpose

This document represents the D.D. (Design Document). Our goal is to give a detailed and complete description of the system in terms of its structural components and its architecture.

This document is directed to the software developer team who is going to implement the architectures, and the testers.

B. Scope

B.1. Description of the given Problem

We are going to design and implement a web application named “Travelendar+”.

The System will allow the users to

- Find the shortest available itinerary given the location of departure and destination, with possibility of modification in case of unforeseen circumstances
- Further customize said itinerary by stating their preferences of transport and desired pauses or breaks, customizing them by specifying how much the break should last and the given timeslot
- Buy and/or book tickets for public transports

The users will have to register (by inserting a username and a password) to be able to use the system. Every user has a set of travel preferences, which can be customized in general as well as for each itinerary, if necessary.

The main purpose will be to offer a quick, efficient, and reliable application to schedule the quickest routes complying with all the user’s events within the limits of feasibility.

B.2. Actual System

The users will be logging in with their username (or, alternatively, their e-mail address) and password. The set of preferences will be the basis for all travels, and the users will only need to give as inputs the locations of departure and the destination, the time at which they should arrive to the destination, and the type of event.

After that, they will also be able to customize the preferences for the specific trip (such as some means of transport to avoid) and they will receive the shortest itinerary given these inputs.

The user will also have the further option to arrange the trip by buying/booking tickets for public transport if needed, once the itinerary has been set.

C. Definitions Acronyms, Abbreviations

C.1. Definitions

- Event: Locations the user has to go to within a certain deadline for a given timeframe
- Trip: The description of route, including the transports, the user takes on to get from the starting location to the event
- Step: A single part of a trip, corresponds to one mean of transport
- Break: An optional pause to consider from all trips and events, it has to be within a chosen timeframe and last for at least a chosen amount of time

C.2. Acronyms

DB: Database
DBMS: Database Management System
RASD: Requirement Analysis and Specification Document
DD: Design Document

C.3. Abbreviations

[Gn] - nth goal
[Dn] - nth goal
[Rn] - nth functional requirements

D. Revision History

- 1.0 – First complete draft (25/11/2017)

E. Reference Documents

F. Document Structure

Introduction: this section introduces the design document. It contains a justification of his utility and indications on which parts are covered in this document that are not covered by RASD.

- **Architecture Design:** this section is divided into different parts:
  1. Overview : this sections explains the division in tiers of our application
  2. Component view : this sections gives a more detailed view of the components of the applications
  3. Deployment view: this section shows the components that must be deployed to have the application running correctly.
  4. Runtime view: sequence diagrams are represented in this section to show the course of the different tasks of our application
  5. Component interfaces: the interfaces between the components are presented in this section
  6. Selected architectural styles and patterns: this section explain the architectural choices taken during the creation of the application

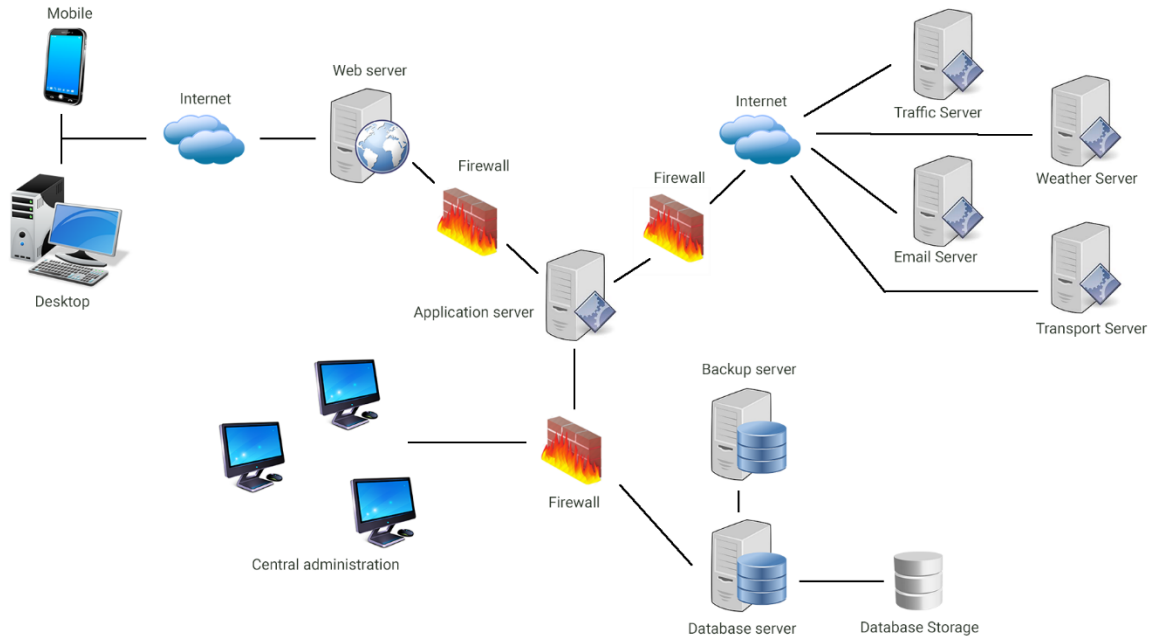
7. Other design decisions

- **Algorithms Design:** this section describes the most critical parts via some algorithms. Pseudo code is used in order to hide unnecessary implementation details in order to focus on the most important parts.
- **User Interface Design:** this section presents mockups.
- **Requirements Traceability:** this section aims to explain how the decisions taken in the RASD are linked to design elements.
- **Implementation, integration and test plan:** in this section we identify the order in which developers plan to implement the subcomponents of the system and the order in which we plan to integrate such subcomponents and test the integration.

2. Architectural Design

A. Overview

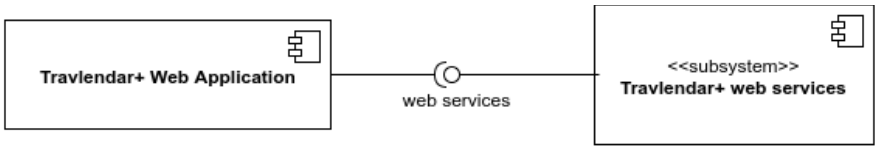
Our system is a web page-based application available for both mobile and desktop browsers and hosted on a web server, while the main services are handled by a separate application server in order to increase security and maintainability of the system. The application server communicates with external servers for information on traffic, weather, and transport options, as well as for sending E-mails. The application server saves the data on a database managed by a database sever. Firewalls protect both the application and database servers from all external threats, and a backup server is always available in case the main database server fails. The central administration computers can work with both the application server and the database server.

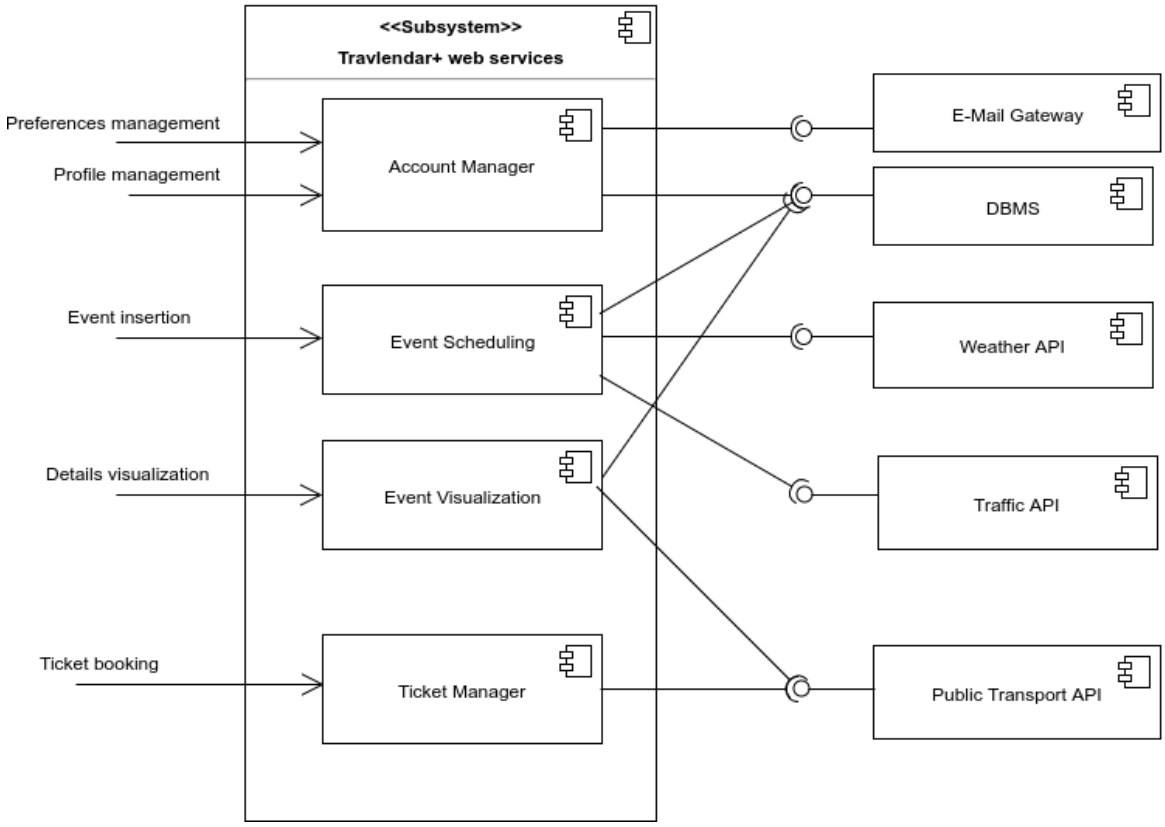


B. Component view

The following diagrams show the main components of the system and the interfaces through which they interact.

- The client side consists of only the web application on the customer's device.
- The server side has several subcomponents, as shown in the second picture.
- These subcomponents manage different services offered to the customer, each interacting with different third parties

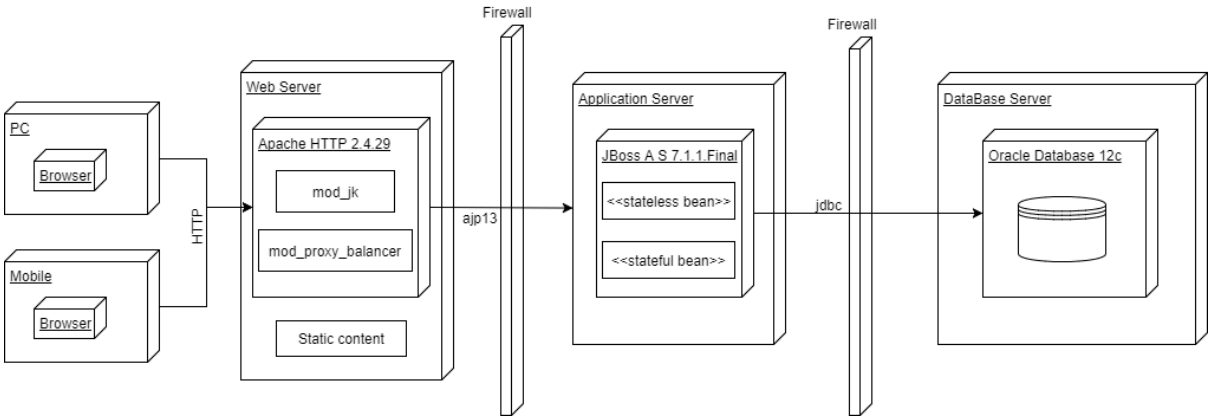




C. Deployment view

The system architecture is divided into 4 tiers and it is based on JEE framework.

- The first tier is the client tier: the web application on the customer's device.
- The web tier (tier 2) contains the web server implemented with the Apache HTTP platform, which is composed of the static content module, and mod_jk and mod_proxy_balancer, connectors which are used respectively for the connection with the servlet server and the load balancer.
- The application tier (tier 3) consists of JBoss which handles the java beans and all the business logic.
- The data tier (tier 4) is mainly composed by the Database Server. The communication between tier 3 and tier 4 is performed via JDBC connector which uses jdbc protocol.

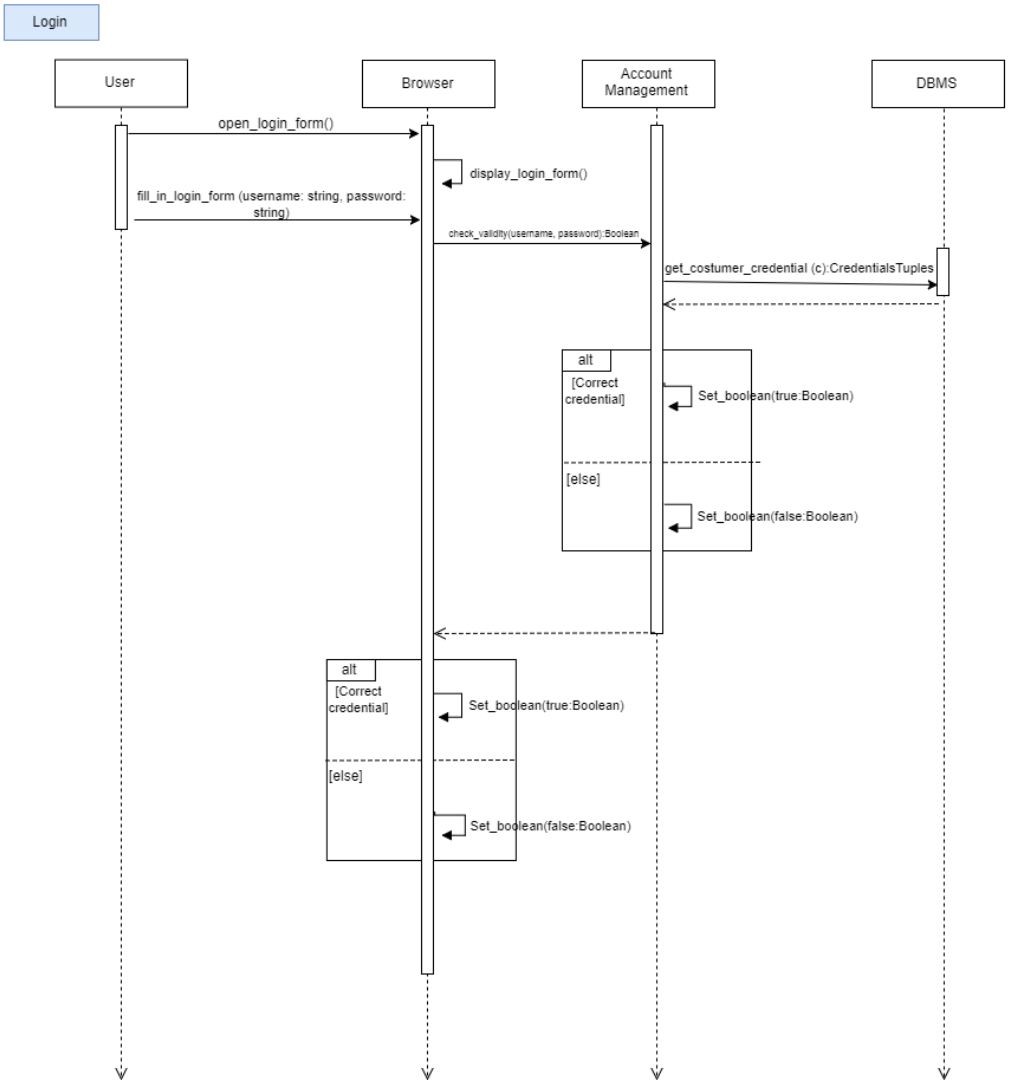


Recommended implementation

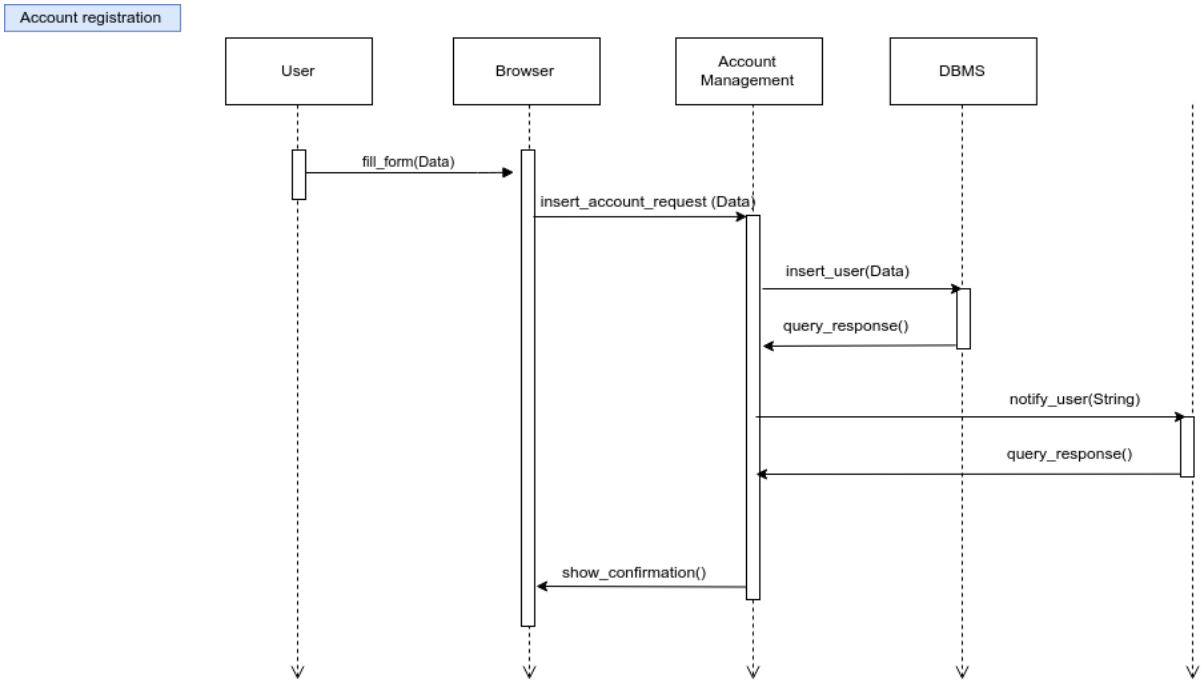
- Client tier: simply a web browser: no implementation on our part.
- Web tier: the web pages may be implemented using HTML 5.0, Java Script and CSS.
- D.2.Script engine tier: the dynamic content of web pages may be generated using Java servlets.
- Application tier: The EJB application server uses stateless java beans and stores the data (and the state with the client) on the database using JPA and mapping the object with the data through entity beans.
- Data tier: the database may be implemented with Oracle database 12c.
- For web, script engine, application and data tier a server such as Oracle SPARC T7 may be a good solution because it suits well with JEE applications and Oracle DBMSes.

D. Runtime View

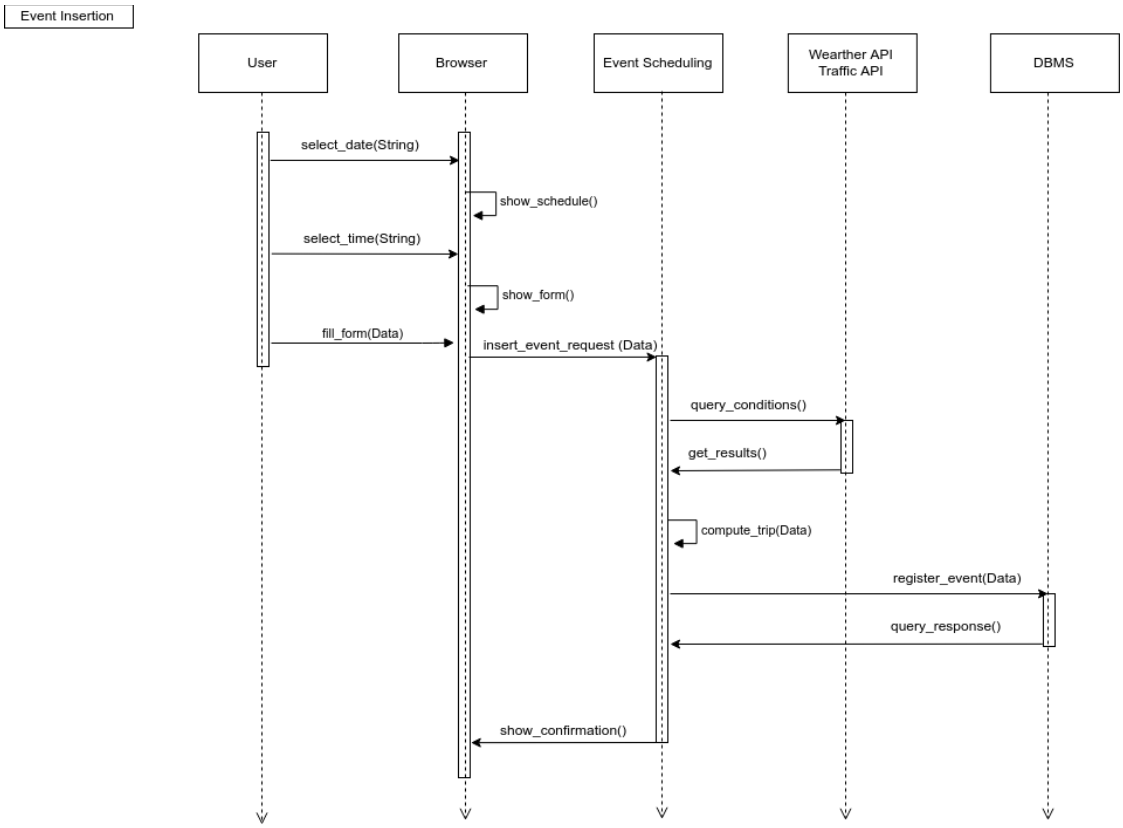
D.1.Login



D.2. Account Registration

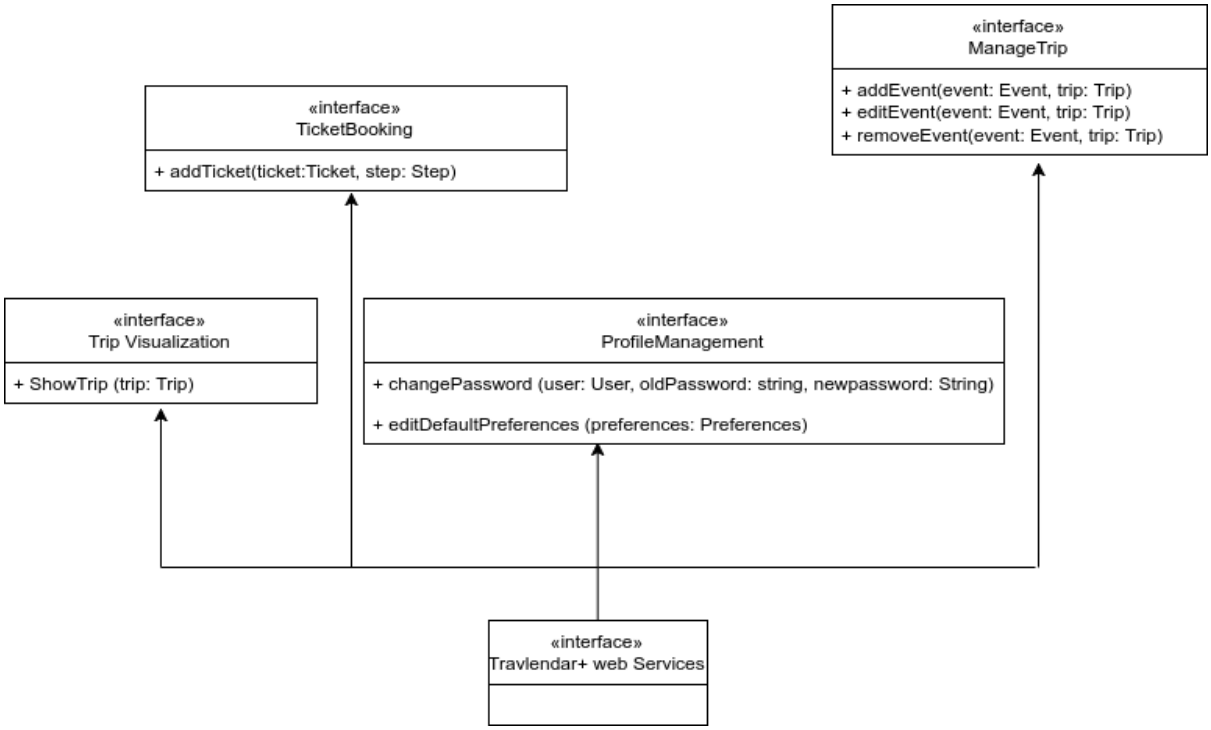


D.3 Event Insertion



E. Component Interfaces

Travlendar+ Web Services gathers all the methods needed to provide the client with functionalities such as schedule a trip, visualize the trips yet to take, add and event and manage their profile.



F. Selected architectural styles and patterns

Selected design patterns:

- **Proxy pattern** allows for object level access control by acting as a pass through entity or a placeholder object. (Used in some components of the component diagrams)

Recommended architectural pattern for implementation:

- **Model-View-Controller** pattern divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user. This is one of the most common and e_ective ways to avoid a dangerous level of coupling between the various parts of the whole system.

Recommended design patterns for implementation:

- **Factory pattern** exposes a method for creating objects, allowing subclasses to control the actual creation process. It is particularly useful if applied in combination with the MVC pattern.
- **Observer pattern** lets one or more objects be noti_ed of state changes in other objects within the system. It is practically essential for the application of the MVC pattern.
- **Visitor pattern** allows for one or more operations to be applied to a set of objects at runtime, decoupling the operations from the object structure. It contributes to the overall decoupling of the system.

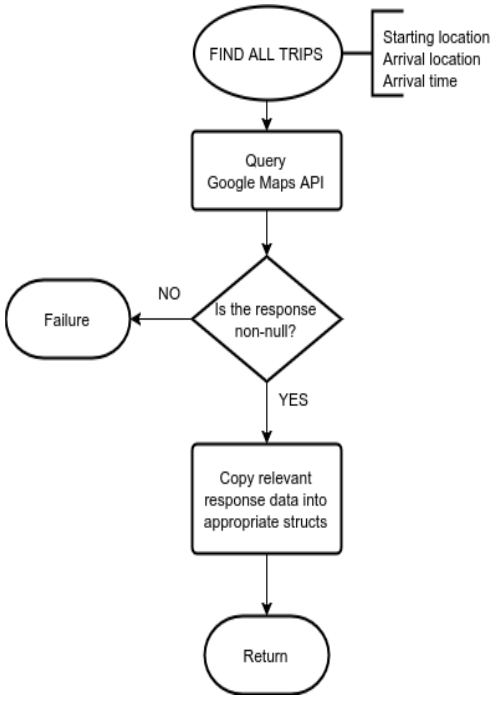
G. Other Design Decisions

None in particular

3. Algorithm design

3.1. Find all trips

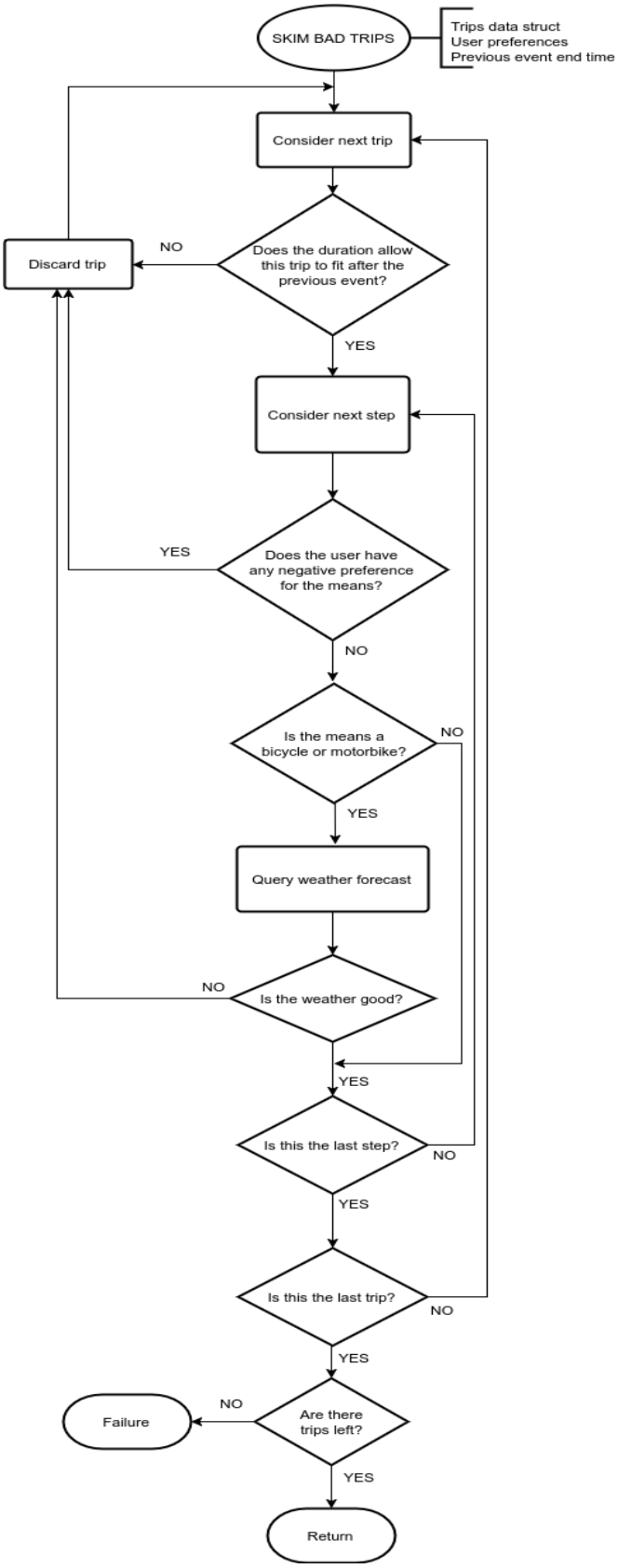
This algorithm takes as input the endpoints of the trip to be scheduled and the requested arrival time.
Various options are obtained by querying an external server, that of the Google Maps Directions API, including parameters such as "alternatives=true" to get more than one option and "traffic-model=pessimistic" to consider the worst case scenario.
The API response is presented in JSON form, and our algorithm parses it memorizing the important data (duration with traffic, means, locations... of each step of each option) in a similarly-arranged struct.
If the response was void of possible routes, the procedure fails as the trip can't be scheduled.



3.2. Skim Bad Trips

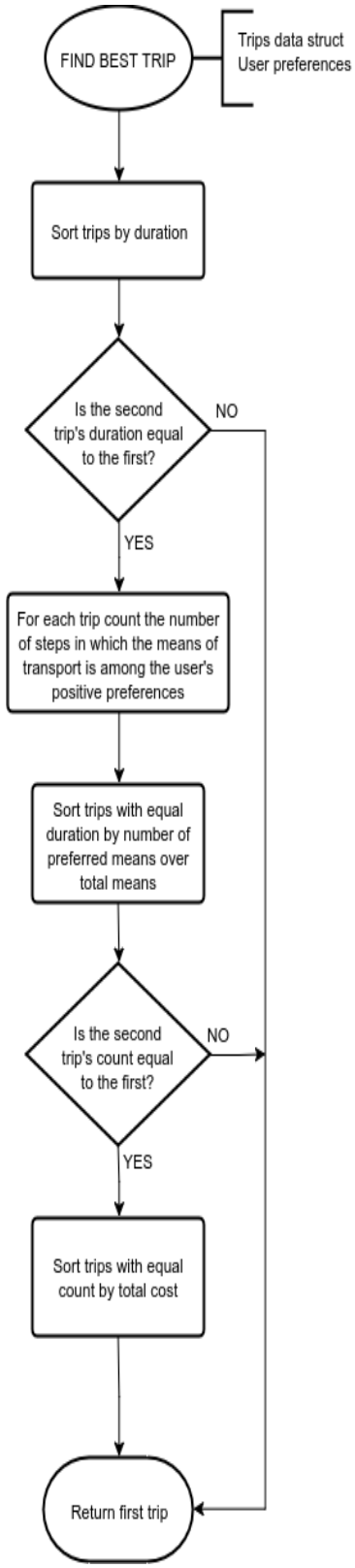
The second part of the trip-scheduling procedure removes trips that are definitely not the best option from the strict created by the previous algorithm, taking also into account the user's preferences and the ending time of the user's previous event.
Before all else, the duration of each whole trip is checked to ensure that it is doable starting from the user's previous event. The option is discarded if it can’t fit in the schedule.
Then, each step of the trip undergoes further inspection: if the user has a negative preference for the step's means of transport, the whole route is discarded; if the step's means of transport is a bike the external server for weather is queried and the route is discarded in case of rain.

If the struct is empty at the end of the procedure, the algorithm returns an error as the trip is impossible.



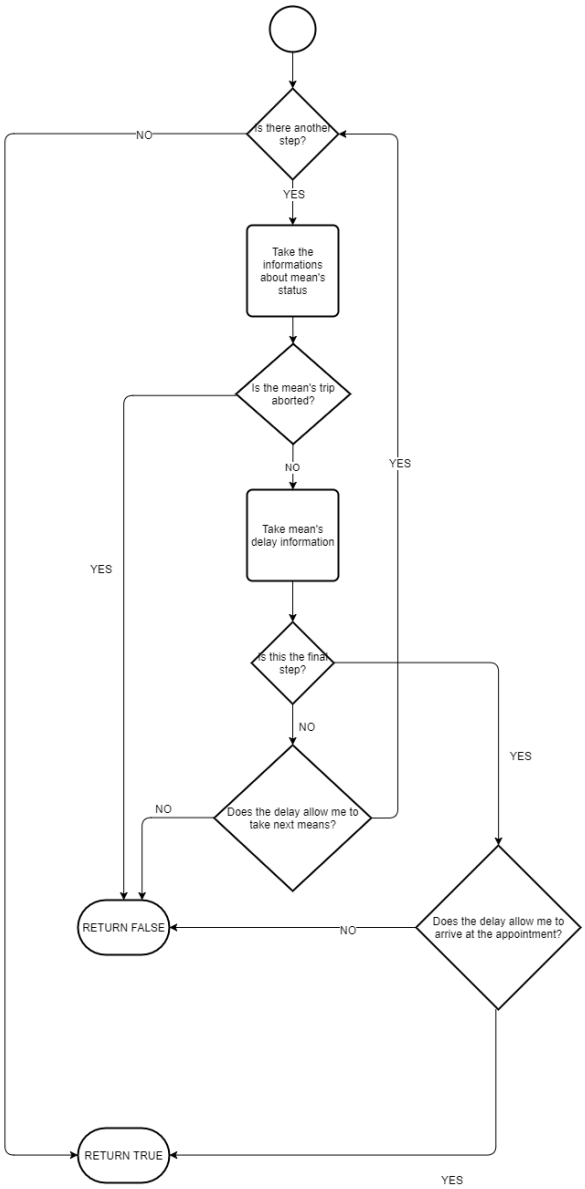
3.3. Best Trip

The last part of the trip scheduling procedure chooses the best route between the ones remaining in the data structure. The duration of each trip is taken into account with the highest consideration, the user's positive preferences on means of transport are second, and lastly the price of the route is considered.
If all routes are equal under all accounts, the first one in the list is chosen.



3.4. Check Function

Every time a user requests the details of a trip, some information about each step (such as the delay of the means of transport) is updated.If the change in the status makes the trip, in any way, unfeasible (a delay could render it impossible to begin the next step, or to get to the event on time), it is computed again from scratch.If the trip can no longer be scheduled, the event is canceled and the user is alerted.



4. User Interface design

4.1 Mock up

Refer to the mockup provided in the R.A.S.D. – Section 3.1.1

5. Requirement Traceability

[G1] Allows a visitor to register an account | [R1]-[R2]

Travlendar+ Web Services → Account manager

[G2] Allows the user to set a new password in case he forgot the old one | [R3]
Travlendar+ Web Services → Account manager

[G3] Allows the user to set preferences | [R4]-[R5]
Travlendar+ Web Services → Account manager

[G4] Allows the user to edit said set of preferences | [R4], [R7]-[R8]
Travlendar+ Web Services → Account manager

[G5] Allows the user to request for an optimized trip | [R4], [R9]-[R15]
Travlendar+ Web Services → Event Scheduling

[G6] Allows the user to edit and arrange trips or completely cancel them | [R4], [R12]-[R13], [R16]-[R19]
Travlendar+ Web Services → Event Scheduling

[G7] Allows the user to set breaks by giving their timeframes and durations | [R4], [R15], [R20]-[R27]
Travlendar+ Web Services → Event Scheduling

[G8] Allows to book/buy tickets for their trips | [R4], [R28]
Travlendar+ Web Services → Ticket Manager

[G9] Allows to book/buy tickets for their trips | [R4], [R29]
Travlendar+ Web Services → Ticket Manager

6. Implementation, integration and test plan

A. Entry Criteria

Before beginning the verification activity of integration system we need to have available external APIs, in particular

- The *DBMS* fully configured and operative to test all the other components that needs access to database
- The external *E-Mail Gateway* should be already available in order for integration tests involving the *Account Manager* component
- Same thing regarded the *Weather API* and *Traffic API* in order to make integration tests involving the *Event Scheduling* Component
- The external *Public Transport API* should be already available in order for integration tests involving the *Event Visualization* and *Ticket Manager* components

To test the integration of the two components, minimum percentage of competition of each component with respect to its functionalities should be required:

- 60% of *Account Manager*; (at least user creation and login features)
- 60% of the Event Scheduling
- 75% of the Event Visualization
- 75% of Ticket Manager

B. Elements to integrate

As seen from the previous schemes

- Front End Component: Web Application
- Back-End Components: Travlendar+ Web Services
- External Components: All the components which refer to functionalities that refer by the DBMS and the other third party systems

The main integrations of back-end components with external components are the following

- **Account Manager, E-mail Gateway, DBMS**
- **Event Scheduling, DBMS, Weather API, Traffic API**
- **Event Visualization, DBMS, Public Transport API**
- **Ticket Manager, Public Transport API**

The integration of back-end components with front-end components only consists of that between the web application and the web services.

C. Integration Testing Strategy

The testing phase will follow a bottom up strategy: we start by integrating the components that don't depend on other ones to function properly, or that only depend on already developed ones. Once all single components are fully tested, we finish by testing the subsystems which the previous components take part in, allowing us to begin to test a component of a subsystem as soon as it's finished (or near completion), thus allowing us to have immediate feedback to parallelize the development phase and the testing phase.

Furthermore, we focus first on the most critical component (critical-module-first approach), as any eventual bug may lead to dangerous consequences (especially to the customer) if not fixed immediately, in this order

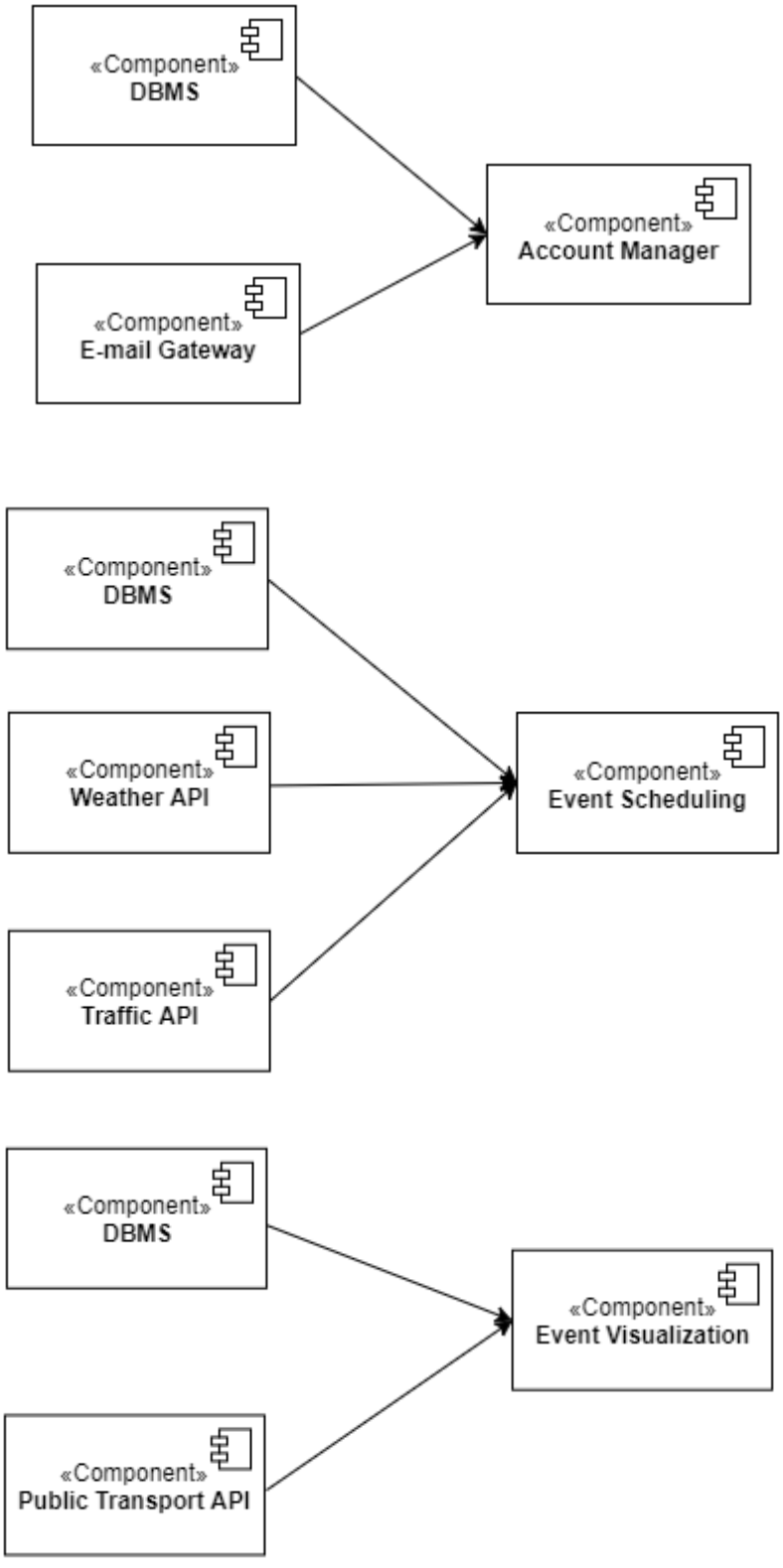
1. *Event Scheduling*
2. *Account Management*
3. *Event Visualization*
4. *Ticket Booking.*

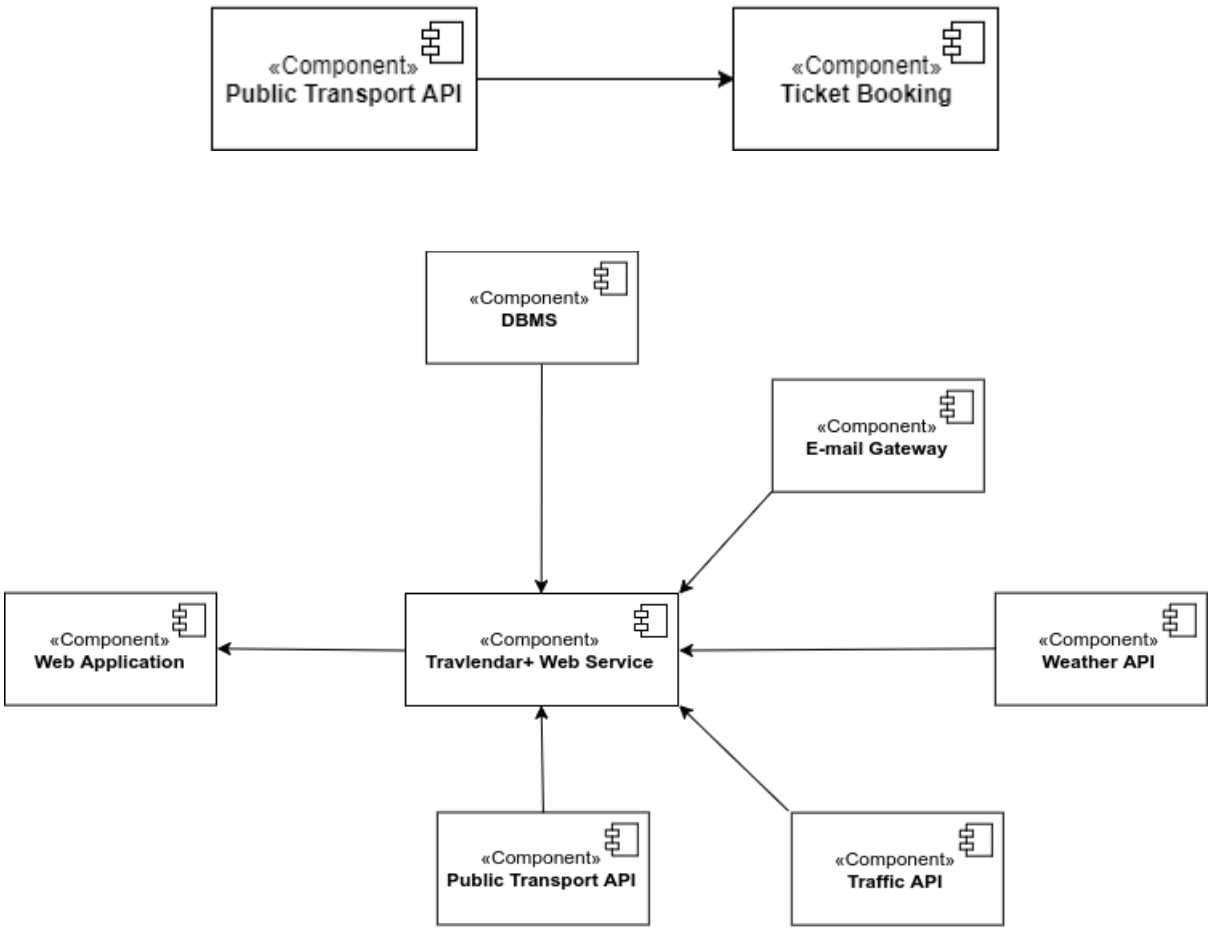
D. Software and Subsystem Integration sequence

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.

Three of the main components (*Ticket Booking* being the exception) of the customer web service are connected to the DMBS in order to function properly. Moreover, *Account Manager* requires *E-mail Gateway*, *Event Scheduling* requires *Weather API* and *Traffic API*,

Event Visualization requires *Public Transport API*. Instead, *Ticket Booking* requires only *Public Transport API*





As the trips we visualized need to be first of all scheduled, we need to implement *Event Scheduling* before *Event Visualization*

7. Effort Spent

- Antonino Caminiti
  - 8/11/2017: 3 hours – 30 minutes
  - 15/11/2017: 5 hours – 30 minutes
  - 17/11/2017: 1 hour
  - 22/11/2017: 5 hours
  - 25/11/2017: 2 hours
- Daniele Gonella
  - 8/11/2017: 3 hours
  - 15/11/2017: 5 hours – 30 minutes
  - 18/11/2017: 3 hours
  - 22/11/2017: 5 hours
- Matteo Scerbo
  - 8/11/2017: 3 hours
  - 15/11/2017: 5 hours – 30 minutes
  - 17/11/2017: 3 hours
  - 22/11/2017: 5 hours

25/11/2017: 1 hour

8. Tool Used

- Github -
- Draw.io - for UML Models