Politecnico di Milano
A.A. 2015-2016
Software Engineering 2: “MyTaxiService”

Design Document

Giuseppe Canonaco, Andrea Di Giosaffatte

4 Dicember 2015 – Version 1.0

Table of Contents

1. Introduction

1.1. Purpose

The redaction of this document is performed to guide the development phase in such a way to grant all the requirements deriving from the RASD document. The intended audience is formed by the developers of the software and the financial stakeholders.

1.2. Scope

This document specifies how the system is designed in terms of software components. It specifies how such components must be deployed over the physical tiers and how they interact at run-time. In conclusion it gives an insight of the design decisions taken during the design phase itself and it gives motivations about the non-trivial design decision taken.

1.3. Definitions, Acronyms, Abbreviations

This document relies on all the definitions, acronyms and abbreviations previously defined in the RASD document of the project, plus the following additions.

1.3.1. Definitions

- **Pending Arrangement**: an arrangement which needs a cab driver to be assigned.
- **Assignation**: when a queue manager chooses a cab driver to handle an arrangement.

1.3.2. Abbreviations

- **[RWn]** The n^{-th} run time view.

1.4. Reference Documents

- Assignments 1 and 2 (RASD and DD) which can be retrieved on beep.
- Structure of the design document which can be retrieved on beep.

1.5. Document Structure

This document is structured in 4 main sections each one of them addresses a different aspect of the design phase. The architectural design explains how the system is architected from a software and hardware point of view specifying also the run-time behavior. The algorithm

design explains how work the more relevant algorithms of the system. The user interface design gives the look and feel of the interfaces with the final users. The requirement traceability maps all the requirements coming from the RASD document into design elements which must grant those requirements.

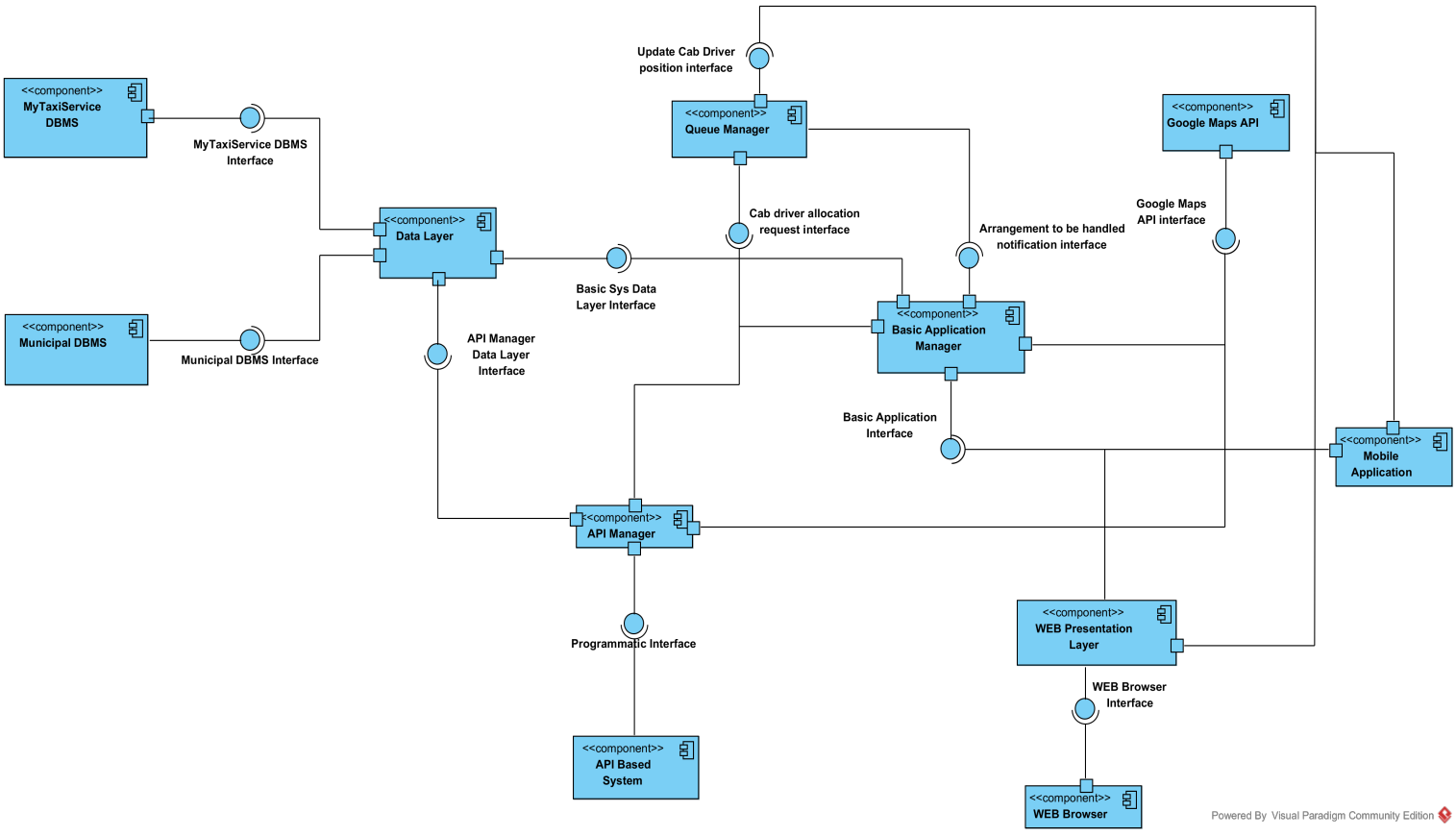
2. Architectural Design

2.1. Overview

In this section is addressed the architectural design of MyTaxiService. 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.

2.2. High Level Components and their Interaction

This section thoroughly depicts, with a high level prospection, all the components building our system and interacting with it.



MyTaxiService DBMS

This is the data base management system which will manage all the relevant information of the system-to-be. Such kind of information are the cab catcher profiles and the reservations made by the cab catchers. The DBMS, of course, will provide an interface which will allow the Data Layer to perform queries over the data. 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.

Municipal DBMS

This is the data base management system which manages all the information relevant to the town council. We have granted the access to a subpart of the set of data managed by this DBMS, this subset is composed by each profile of the cab driver which in turn will include its username, its password, its e-mail, its first name, its last name, its license number and its sex. This DBMS has, of course, an interface which will allow the data layer component to perform queries over the subset of data previously described.

Google Maps API

This component represents the piece of software encapsulating the set of services provided by Google to build location aware application. It offers an interface which will be mainly used to compute the estimated arrival time of a cab driver to the starting point of the ride. This component, due to the reason just stated, will be used by the API manager and the basic application manager.

WEB Browser

Is the component representing the software through which a user can access the web application.

API Based System

This component represents a generic system or application which will use a programmatic interface provided by our system. This programmatic interface will allow this component to use MyTaxiService on behalf of a cab catcher with previous authorization of the cab catcher itself.

Now we can address all the high level components which will be developed during the implementation phase.

Data Layer

This is basically a piece of software which will deal with the DBMSs, making them retrieve or store information which are relevant to the system to build. This component is a further level of abstraction over the data management. This is strictly necessary to use only one interface to access different kind of data which are stored in two different data bases. The other reason is to differentiate the way to access the set of data held by our system with respect to the

component which need to access our data. For this last reason this component will provide two interfaces to access data, one for the API manager and the other for the basic application manager.

API Manager

This is the component of our system which is used to handle all the calls coming from the API based systems and offers the programmatic interface due to this reason. Undoubtedly, to properly answer these calls the API Manager has to use other components which are the data layer, to access or to store data, the queue manager, to ask for a cab driver dispatch with respect to a particular pending arrangement, and the Google API, to compute the estimated arrival time of an assigned cab driver to the meeting point of the relative arrangement.

Mobile Application

This component is the view of the user over the system, so by means of this component the user performs actions over the system and sees notifications about the actions performed over the system itself. When performing an action over the view this component sends information to the system-to-be (the sent information could be the position of the user, data related to an arrangement or data related to the profile of the user, depending on the performed action). If the user handled is a cab driver, information like the availability and the answers to assignations are directly sent to the queue manager due to efficiency reasons. In all the other cases information are sent to the basic application manager.

WEB Presentation Layer

To this component is entrusted the rendering of all the pages to be displayed by the browser (this component is obviously called by a browser only when a cab catcher or a cab driver is using the web application). If a taxi driver sends to the web presentation layer an availability state update or an answer to an assignation this information are directly sent to the queue manager due to efficiency reasons. All the other operations like requests, reservations, profile information editing, authentication, signing up and arrangement deletion are asked to be performed to the basic application manager.

Queue Manager

This component handles all the taxi queues within the system. It receives the cab driver answers to assignations and information about the availability status of each cab driver both directly from the mobile application and from the web presentation layer. It asks the basic application manager to notify cab drivers about pending arrangements to be handled. It is worth to make the queue manager a stand-alone component because is used in the same way by the API manager and the basic application manager. Moreover it withholds the current state of each cab driver within the system and this such state must be shared by the API manager and the basic application manager. The API manager and the basic application manager ask this component to dispatch a cab driver for a pending arrangement.

Basic Application Manager

This component handles all the requests coming from the mobile application and from the web presentation layer except for the availability status update and the answers to

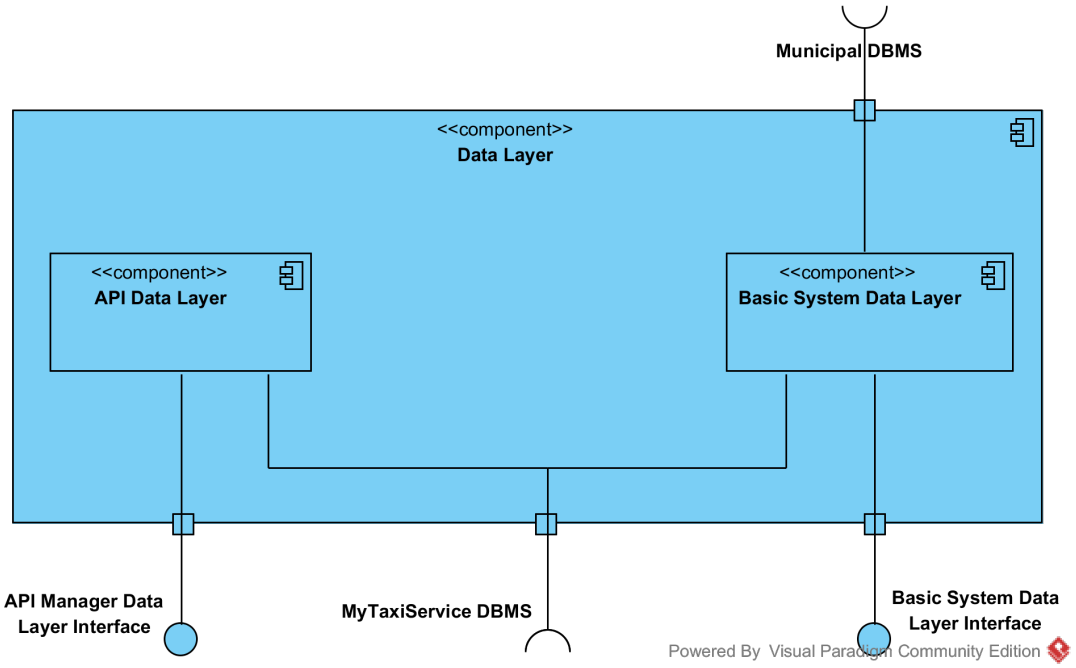
assignations, coming, both of them, from cab drivers (which means from either the mobile application or the web presentation layer while there is a cab driver connected to them). So it will handle arrangement requests, profile editing requests, authentication requests, sign up requests and arrangement deletion requests. To this component is also entrusted the management of all the notifications to the users. It uses the Google Maps API to compute the estimated arrival time of a cab driver. It calls the queue manager when there is need to assign a cab driver to an arrangement currently in pending. Of course this component uses the data layer each time there is need to deal with permanent data (when there is a reservation request for example or a user profile editing request etc.).

2.3. 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.

DATA LAYER

The data layer has two sub-components to separate the management of requests, intended in terms of queries and data storage, coming from the API manager and from the basic application manager.



API Data Layer

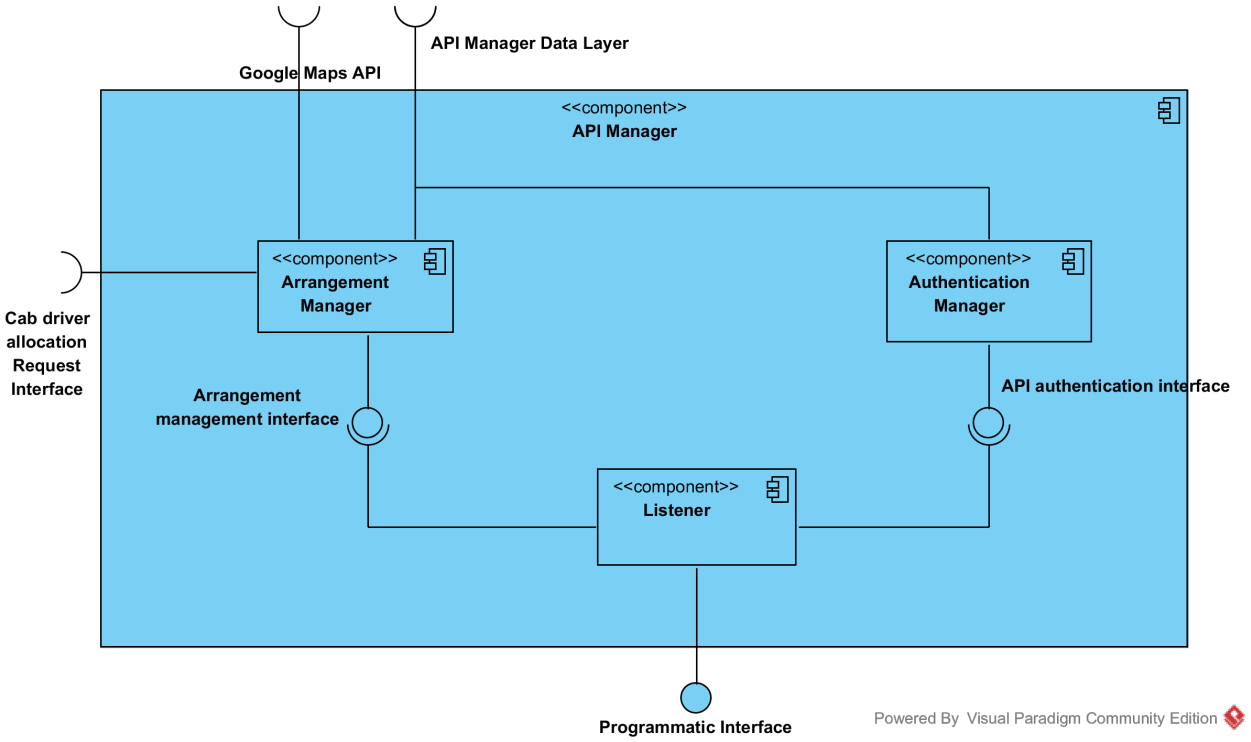
To the API data layer sub-component is entrusted the management of all the requests coming from the API manager which can't perform operations on behalf of a cab driver. For this reason the API data layer can't access the municipal DBMS. Eventually we can say that the API data layer handles the queries and the storage operations requested by the API manager. Such kind of operations are needed to complete authentication procedures or to store reservations.

Basic System Data Layer

To the basic system data layer sub-component is entrusted the management of all the requests, intended in terms of queries or data storage, coming from the basic application manager. This component has access to both the municipal and MyTaxiService DBMSs because the basic application manager has to handle requests related to both cab drivers and cab catchers.

API MANAGER

The API Manger is decomposed into three blocks the arrangement manager sub-component, the authentication manager sub-component and the listener sub-component. This decomposition is due to the fact that the API manager has to cope with two main kind of operations which are the authentication of the API based system (which allows, if successfully done, the API based system to perform operations on behalf of a cab catcher) and the management of the arrangements coming from the API based system.



Listener

This sub-component will dispatch all the calls coming from the API based system. To properly perform this function this sub component offers the programmatic interface, through which the API based system can ask for the execution of operations, and uses the interfaces offered by the other two sub-components to have the requested operation executed.

Arrangement Manager

This sub-component will handle all the arrangement forwarded by the listener(in other words all the arrangement belonging to an API based system). Of course, to properly perform this

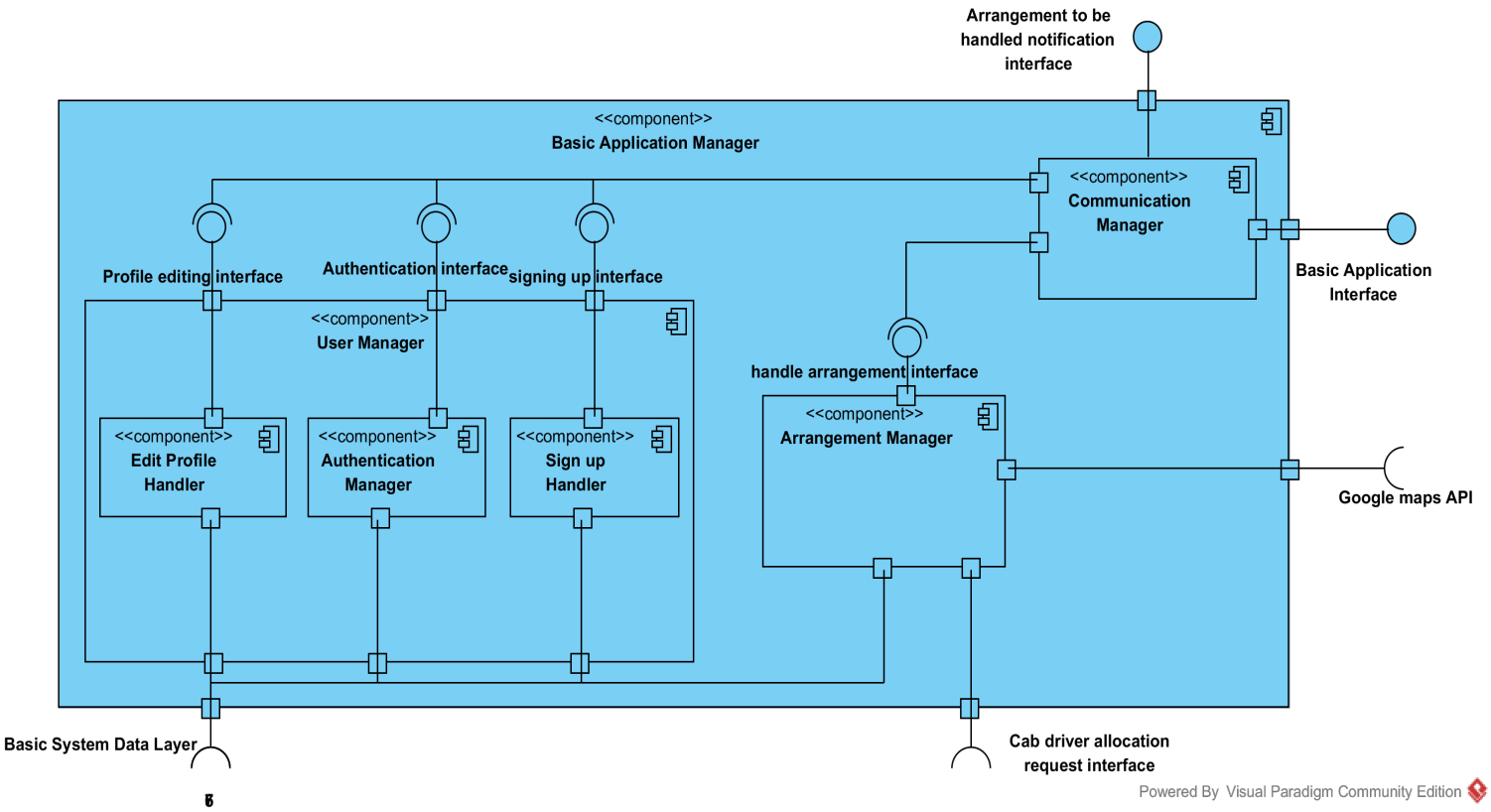
function the arrangement manager has to exploit the interface offered by the API data layer, the interface offered by the Google Maps API and the interface offered by the queue manager. The API data layer is used to store reservations or to delete them if not yet in handling. The Google Maps API is used to compute the estimated arrival time of a cab driver to the starting point of the ride, this estimated arrival time will be, of course, bound to the arrangement itself. The queue manager is used to dispatch a cab driver for a pending arrangement.

Authentication Manager

This sub-component will handle all the authentication requests forwarded by the listener. Of course to provide an authentication functionality this sub-component must access the data-set containing the information through which authenticate. For this reason it uses the API data layer.

BASIC APPLICATION MANAGER

The basic application manager is divided into three sub-components which are the communication manager, the arrangement manager and the user manager. This decomposition is due to the fact that the basic application manager has to provide a way to notify the users, has to handle the arrangement belonging to the cab catchers and has to allow operations over user profiles.



Communication Manager

This sub-component receives all kinds of operation requests both from the mobile application and from the web presentation layer. The only two operation requests which are not sent to the communication manager are the availability state updates and the assignation answers coming from the cab drivers (for simplicity sake coming from cab drivers means that are coming either from the web presentation layer or the mobile application but there's a cab driver connected to those components). Of course this component handles all the notifications which must be sent to the cab catchers due to arrangement handling or to the cab drivers due to assignations handling performed by the queue manager.

Arrangement Manager

This sub-component handles all the arrangements belonging to cab catchers. So it has to store the reservations through the basic system data layer and has to ask the queue manager to dispatch a cab driver with respect to a pending arrangement. Moreover the arrangement manager computes the estimated arrival time of a cab driver to the starting point of the ride. It can delete a reservation not yet in handling through the basic system data layer.

User Manager

The user manager is in turn decomposed into edit profile handler, authentication manager and sign up handler. The decomposition is due to the fact that it has to offer three different functionalities.

Edit Profile Handler
To this component is entrusted all the editing operations over the profiles. Of course to perform this operations it will use the basic system data layer.

Authentication Manager

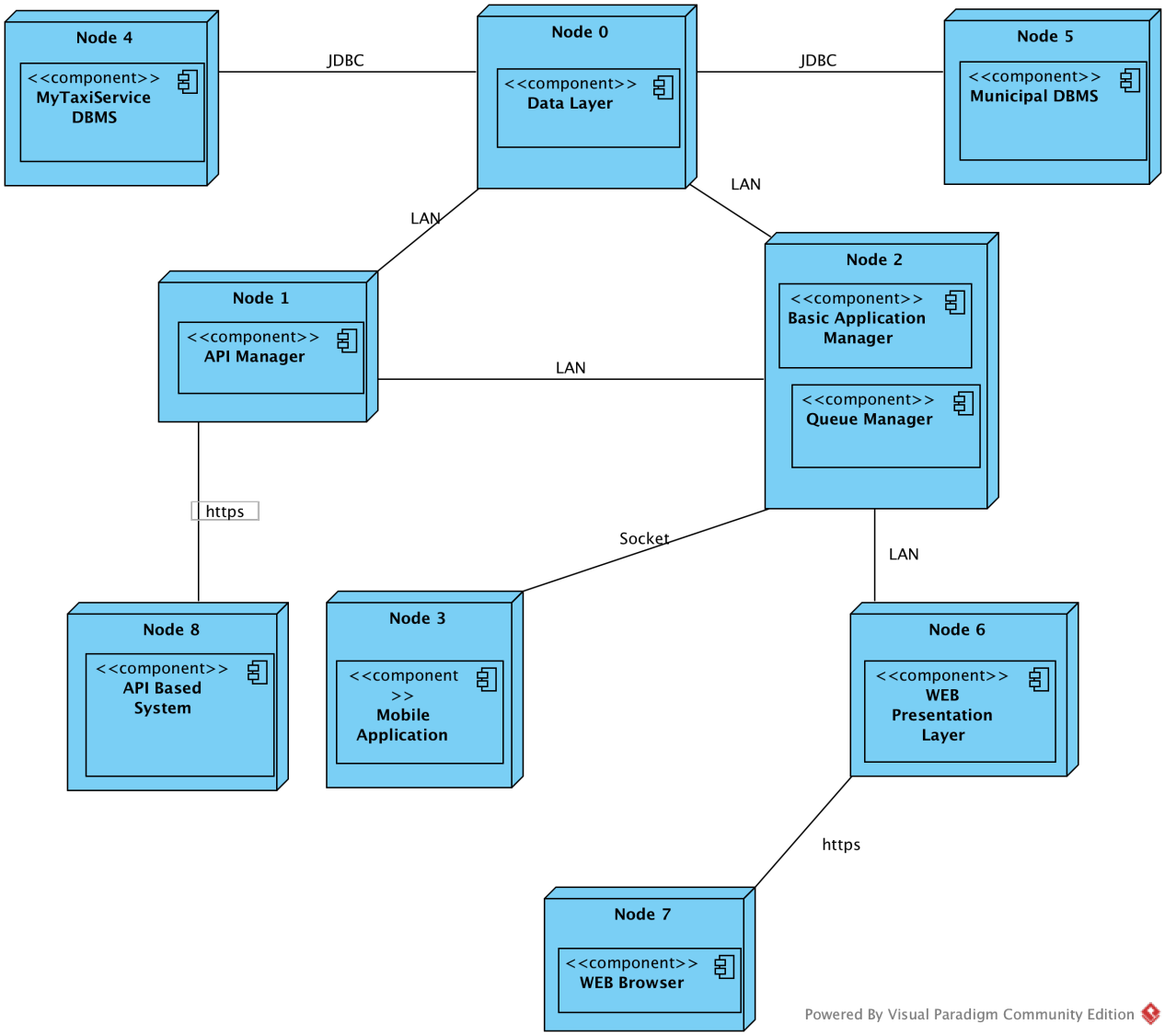
This component will handle all the authentication operations for cab drivers and cab catchers. It has to access the basic system data layer due to the previously stated reason.

Sign up Handler

To this component is entrusted the registration of new cab catchers within the system, to perform this it needs to access the basic system data layer.

2.4. Deployment View

In this section is addressed the deployment of the component previously described.



The criterion which has lead the deployment is a reasonable distribution of the workload among the tier without overloading the internal network because of the weight and number of messages to be exchanged among the internal tiers (for internal tiers we mean the tiers connected through the internal network which is in turn composed by all the lan and jdbc[1] connections). This is of course a tradeoff to cope with, because if too much tiers are created to better distribute the workload this will heavily increase the number of messages exchanged over the internal network, instead if the number of messages exchanged over the internal network is reduced too much then the workload over the tiers will turn out to be increased. Of course moving toward one or the other direction will introduce different kinds of cost which are related to the computation power of a particular tier or to the speed of the connections among the tier which have to exchange messages.

[1] To understand why JDBC is the chosen connector between the data layer and the DBMSs, have a look at the section "Other Design Decision" in this document.

The data layer is deployed over a stand-alone tier because of security precautions. In this way at least this component runs over a tier which is not directly touched by external entities. Moreover the API manager component is deployed over a stand-alone tier because the number of requests coming from third-party systems could be way larger than the number of requests coming from our basic users and to make more secure the management of third-party systems. The web presentation layer component is deployed over a stand-alone tier in order to separate the business logic of the basic application from the web presentation itself. This is due to offloading the tier hosting the business logic of the basic application, and also results into security precautions. The basic application manager component and the queue manager component are deployed over the same tier because this speeds up the communication between the two components making MyTaxiService swifter. Of course the MyTaxiService DBMS will be bought from one of the most known vendors of this kind of service but the choice of the vendor is postponed to the implementation phase in order to choose the best fitting solution with respect to the implementation itself.

2.5. Runtime View

In the section below it's shown how the components interact to accomplish tasks related with the use cases specified in the RASD document. There are 4 runtime views modelled as sequence diagrams, each of which embodies several relevant use cases.

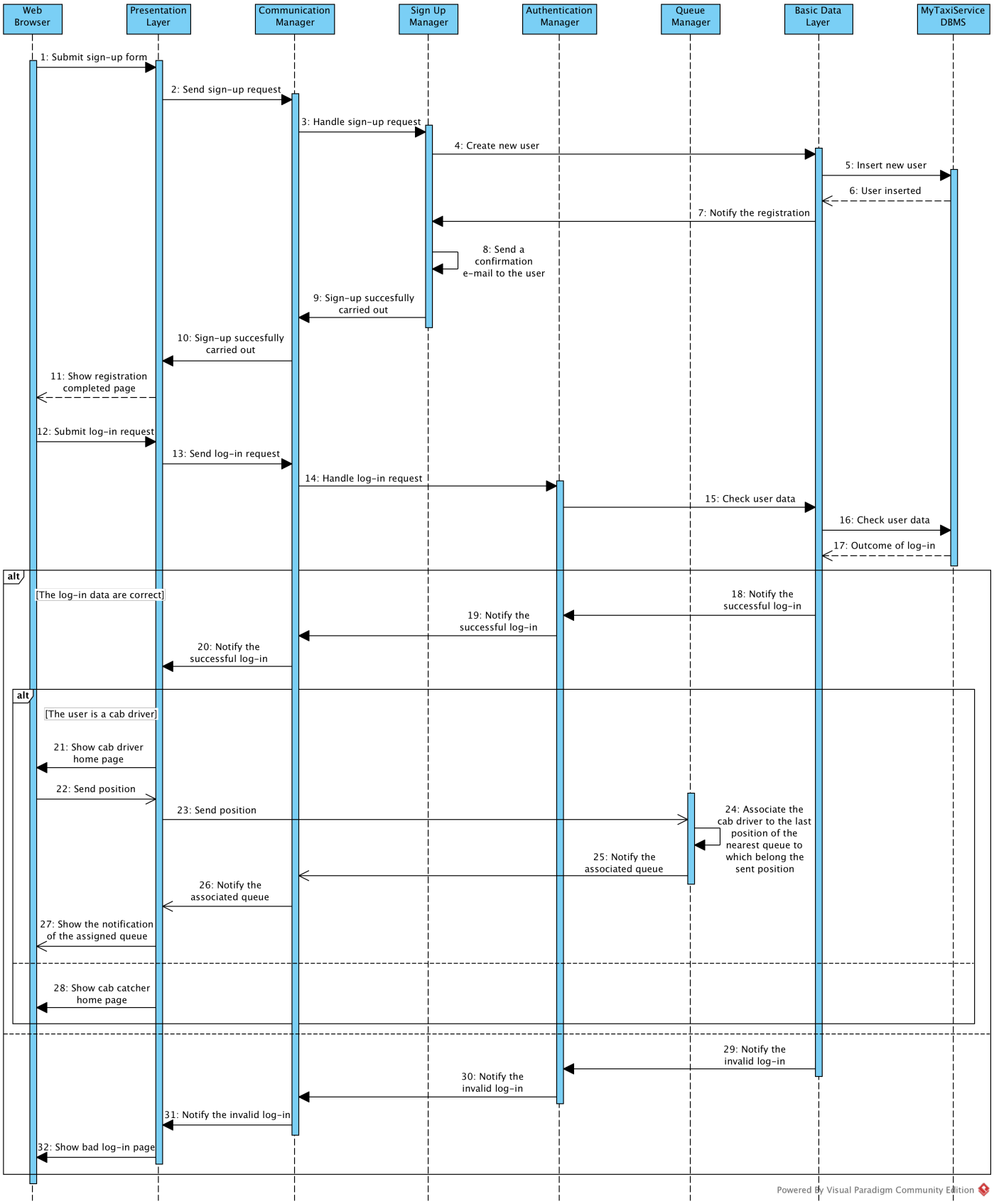
[RW1] This runtime view shows the interaction between components in a scenario in which somebody, through a web browser, wants to register to the service as a cab catcher. Subsequently, the diagram shows the interaction also for a scenario in which a user (a cab catcher or a cab driver) wants to access the service from the web. All the operations and the exchange of messages are shown, even in case of failure of the log-in.

[RW2] This runtime view shows the interaction between components in a scenario in which a cab catcher makes a request through the mobile app. The diagram shows the interaction also when cab drivers answer to a request. All the operations and the exchange of messages are shown, even in case of failure of the request.

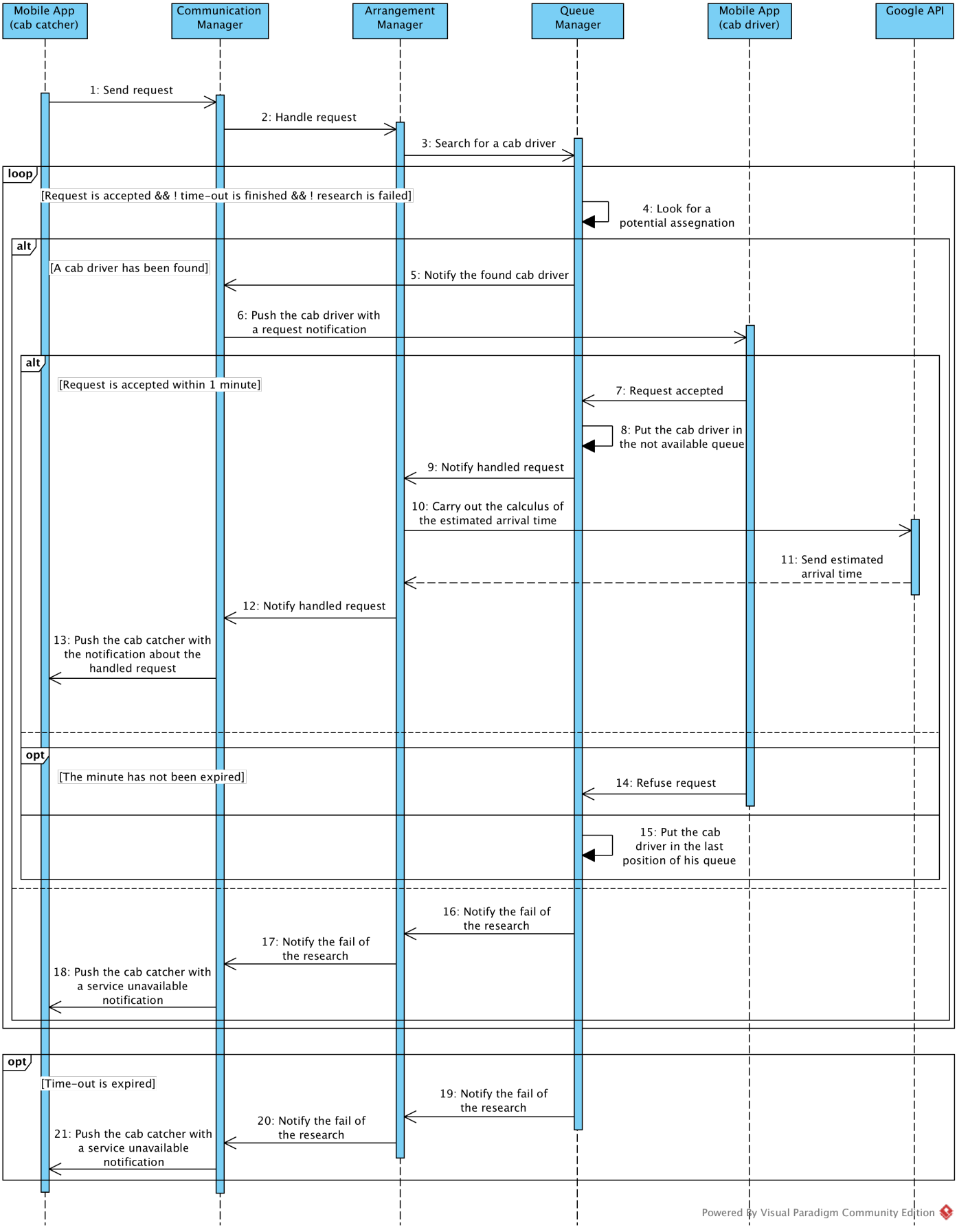
[RW3] This runtime view shows the interaction between components in a scenario in which a cab catcher makes a reservation through the mobile app. In the case of an allocated reservation, the diagram shows the interaction also when a cab catcher decides to delete it. All the operations and the exchange of messages are shown, even in case of failure of the reservation.

[RW4] This runtime view shows the interaction between components in a scenario in which an API based system asks for authorization to the basic service. After receiving it, the API based system makes a request through the programmatic interface offered by the basic service. All the operations and the exchange of messages are shown, even in case of authorization or request failure.

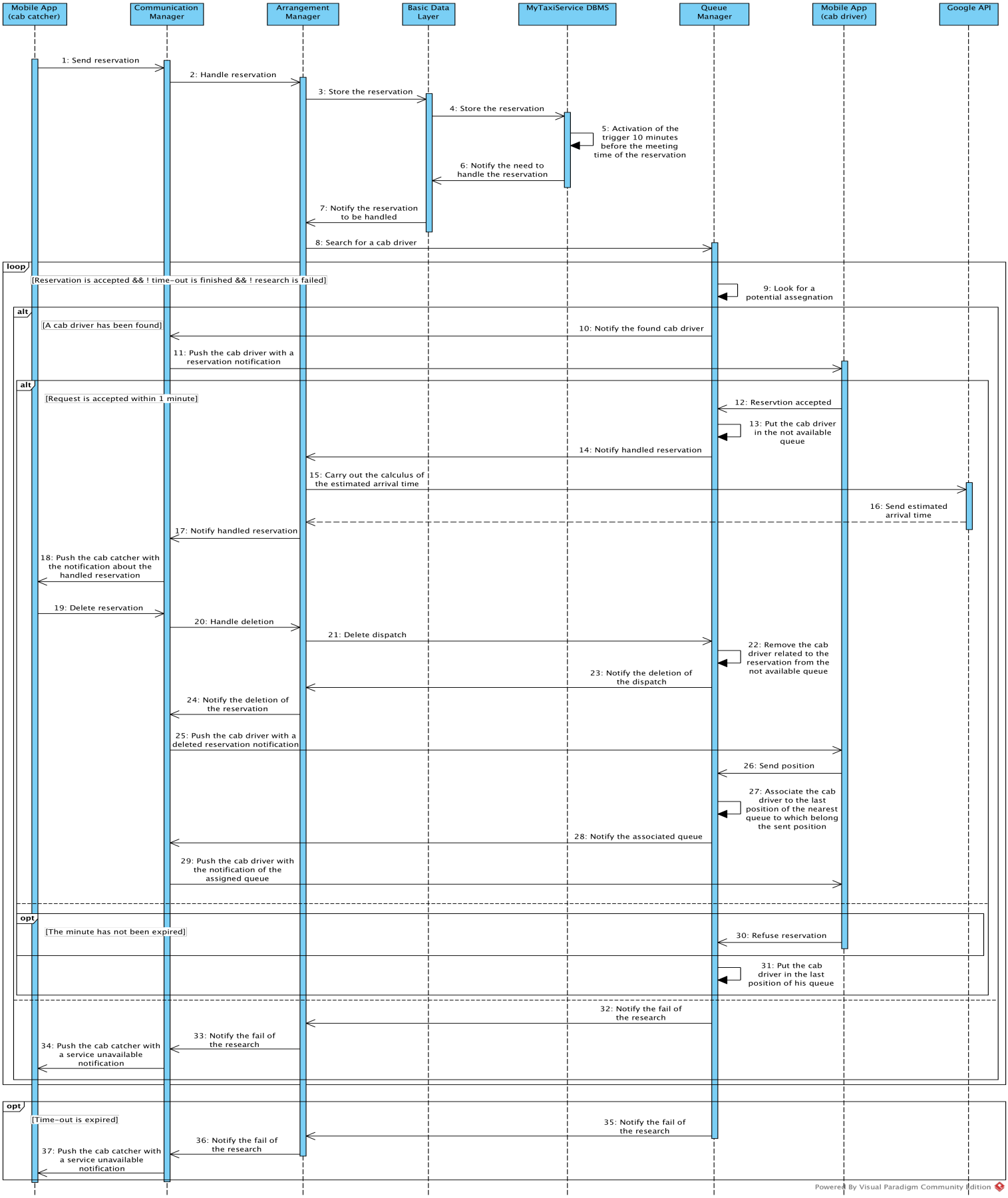
[RW1]


Web Browser
Presentation Layer
Communication Manager
Sign Up Manager
Authentication Manager
Queue Manager
Basic Data Layer
MyTaxiService DBMS
1: Submit sign-up form
2: Send sign-up request
3: Handle sign-up request
4: Create new user
5: Insert new user
6: User inserted
7: Notify the registration
8: Send a confirmation e-mail to the user
9: Sign-up succesfully carried out
10: Sign-up succesfully carried out
11: Show registration completed page
12: Submit log-in request
13: Send log-in request
14: Handle log-in request
15: Check user data
16: Check user data
17: Outcome of log-in
alt
[The log-in data are correct]
18: Notify the successful log-in
19: Notify the successful log-in
20: Notify the successful log-in
alt
[The user is a cab driver]
21: Show cab driver home page
22: Send position
23: Send position
24: Associate the cab driver to the last position of the nearest queue to which belong the sent position
25: Notify the associated queue
26: Notify the associated queue
27: Show the notification of the assigned queue
28: Show cab catcher home page
29: Notify the invalid log-in
30: Notify the invalid log-in
31: Notify the invalid log-in
32: Show bad log-in page
Powered By Visual Paradigm Community Edition

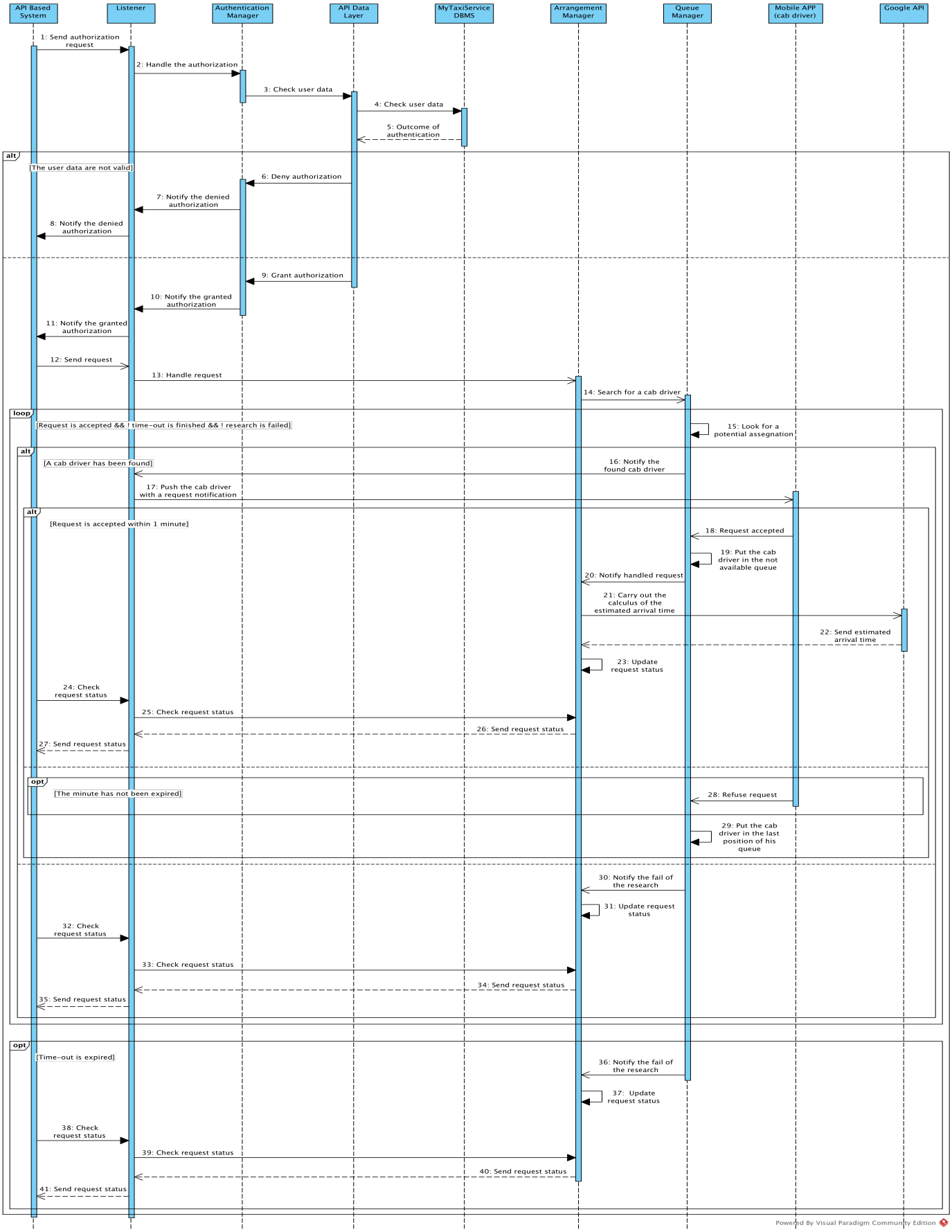
[RW2]


Mobile App (cab catcher)
Communication Manager
Arrangement Manager
Queue Manager
Mobile App (cab driver)
Google API
1: Send request
2: Handle request
3: Search for a cab driver
loop
[Request is accepted && ! time-out is finished && ! research is failed]
4: Look for a potential assegnation
alt
[A cab driver has been found]
5: Notify the found cab driver
6: Push the cab driver with a request notification
alt
[Request is accepted within 1 minute]
7: Request accepted
8: Put the cab driver in the not available queue
9: Notify handled request
10: Carry out the calculus of the estimated arrival time
11: Send estimated arrival time
12: Notify handled request
13: Push the cab catcher with the notification about the handled request
opt
[The minute has not been expired]
14: Refuse request
15: Put the cab driver in the last position of his queue
16: Notify the fail of the research
17: Notify the fail of the research
18: Push the cab catcher with a service unavailable notification
opt
[Time-out is expired]
19: Notify the fail of the research
20: Notify the fail of the research
21: Push the cab catcher with a service unavailable notification
Powered by Visual Paradigm Community Edition

[RW3]


Mobile App (cab catcher)
Communication Manager
Arrangement Manager
Basic Data Layer
MyTaxiService DBMS
Queue Manager
Mobile App (cab driver)
Google API
1: Send reservation
2: Handle reservation
3: Store the reservation
4: Store the reservation
5: Activation of the trigger 10 minutes before the meeting time of the reservation
6: Notify the need to handle the reservation
7: Notify the reservation to be handled
8: Search for a cab driver
loop
[Reservation is accepted && ! time-out is finished && ! research is failed]
9: Look for a potential assegnation
alt
[A cab driver has been found]
10: Notify the found cab driver
11: Push the cab driver with a reservation notification
alt
[Request is accepted within 1 minute]
12: Reservtion accepted
13: Put the cab driver in the not available queue
14: Notify handled reservation
15: Carry out the calculus of the estimated arrival time
16: Send estimated arrival time
17: Notify handled reservation
18: Push the cab catcher with the notification about the handled reservation
19: Delete reservation
20: Handle deletion
21: Delete dispatch
22: Remove the cab driver related to the reservation from the not available queue
23: Notify the deletion of the dispatch
24: Notify the deletion of the reservation
25: Push the cab driver with a deleted reservation notification
26: Send position
27: Associate the cab driver to the last position of the nearest queue to which belong the sent position
28: Notify the associated queue
29: Push the cab driver with the notification of the assigned queue
opt
[The minute has not been expired]
30: Refuse reservation
31: Put the cab driver in the last position of his queue
32: Notify the fail of the research
33: Notify the fail of the research
34: Push the cab catcher with a service unavailable notification
opt
[Time-out is expired]
35: Notify the fail of the research
36: Notify the fail of the research
37: Push the cab catcher with a service unavailable notification
Powered By Visual Paradigm Community Edition

[RW4]


API Based System
Listener
Authentication Manager
API Data Layer
MyTaxiService DBMS
Arrangement Manager
Queue Manager
Mobile APP (cab driver)
Google API
1: Send authorization request
2: Handle the authorization
3: Check user data
4: Check user data
5: Outcome of authentication
alt
[The user data are not valid]
6: Deny authorization
7: Notify the denied authorization
8: Notify the denied authorization
9: Grant authorization
10: Notify the granted authorization
11: Notify the granted authorization
12: Send request
13: Handle request
14: Search for a cab driver
loop
[Request is accepted && ! time-out is finished && ! research is failed]
15: Look for a potential assegnation
alt
[A cab driver has been found]
16: Notify the found cab driver
17: Push the cab driver with a request notification
alt
[Request is accepted within 1 minute]
18: Request accepted
19: Put the cab driver in the not available queue
20: Notify handled request
21: Carry out the calculus of the estimated arrival time
22: Send estimated arrival time
23: Update request status
24: Check request status
25: Check request status
26: Send request status
27: Send request status
opt
[The minute has not been expired]
28: Refuse request
29: Put the cab driver in the last position of his queue
30: Notify the fail of the research
31: Update request status
32: Check request status
33: Check request status
34: Send request status
35: Send request status
opt
[Time-out is expired]
36: Notify the fail of the research
37: Update request status
38: Check request status
39: Check request status
40: Send request status
41: Send request status
Powered By Visual Paradigm Community Edition

2.6. Component Interfaces

MUNICIPAL DBMS

This is an Oracle DBMS. To deepen the knowledge about the interface offered by this kind of database management system follow this link:

http://www.oracle.com/technetwork/database/features/oci/index-090945.html

Of course, to perform operations over this component must be used a connector, JDBC, which allows to access the database, performing queries and storing new data, from a java application. To have a comprehensive view of how to use this connector see:

http://docs.oracle.com/javase/7/docs/technotes/guides/jdbc/ and

http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136101.html

To our application is granted access only to a subset of the data stored on this database.

MyTaxiService DBMS

This data base management system will be chosen during the implementation phase taking into account the solution which best fit the constraints deriving from that phase with relation to the cost of the solution itself. The only constraint is that the chosen solution must offer an interface accessible through JDBC. To have a look at how JDBC works see:

http://docs.oracle.com/javase/7/docs/technotes/guides/jdbc/ and

http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136101.html

DATA LAYER

This component offers two interfaces: the basic system data layer interface and the API manager data layer interface.

Basic system data layer interface

It offers all the functionalities to access and store data over the two DBMSs our system works with. Of course this functionalities are at a higher level than those offered by the JDBC connector. The basic functionalities offered are: modify, to edit user profiles; delete reservation, to delete a reservation not yet in handling; authentication, to check the credentials of a user; registration, in case of cab catcher sign up; store reservation, to save a reservation over the data base.

API manager data layer interface

It offers functionalities to access data over the MyTaxiService DBMS. This functionalities are at a higher level than those offered by the JDBC connector. The main functionalities offered are: authentication, to check the credentials; store reservation, to save a reservation coming from an API based system; delete reservation, to delete reservation which are not yet in handling.

API MANAGER

It is composed by three sub-component so we describe the interfaces offered by them.

Programmatic interface

It offers all the functionalities needed by an API based system to access the basic service. This functionalities are: request, to allow an API based system to yield a request; reservation, to allow the API based system to make a reservation; delete arrangement to allow an API based system to delete an arrangement; check status, to allow an API based system to check the status of a particular arrangement.

Arrangement manager interface

It offers all the functionalities to handle in a proper way an arrangement. Therefore it has the handle functionality which allows the handling of new arrangement, the check status functionality which allows to check the status of a particular arrangement and the delete arrangement functionality to delete a particular arrangement.

Authentication manager interface

It offers the authentication functionality for an API based system.

BASIC APPLICATION MANAGER

It is divided into three sub-components so here are described the interfaces offered by these sub-components.

Arrangement to be handled notification interface

This interface allows the queue manager to ask for a cab driver notification. Therefore it has to offer the dispatched cab driver notification functionality which notifies a cab driver when an assignation to an arrangement has been performed.

Basic Application interface

It has to offer all the functionalities which allow the web presentation layer and the mobile application to render the view. This functionalities are: authentication, to grant access to the service; taxi request, to make a taxi request; reservation, to make a reservation; profile

editing, to edit profile information; sign up, to register new cab catchers to the service; delete arrangement, to delete arrangement previously asked.

Handle Arrangement interface

It has to offer all the functionalities to allow the communication manager to properly request operations over arrangements. This functionalities are: handle arrangement, which asks to handle a new arrangement; delete arrangement, which asks to delete a previously done arrangement.

Sign up interface

It has to offer the functionality to register a new cab catcher over the system.

Authentication interface

It has to offer the functionality to authenticate a user which wants access to the service.

Profile editing interface

It has to offer the functionality to edit the profile of a user.

GOOGLE MAPS API

Google Maps API Interface

This interface is mainly used to compute the estimated arrival time of a cab driver to the starting point of the ride. To have an insight of this interface see:

https://developers.google.com/maps/documentation/distance-matrix/intro

WEB PRESENTATION LAYER

Web browser interface

This interface allows to surf the web pages representing the web application and of course collects the information related to the actions performed by the user over the web pages.

QUEUE MANAGER

Update cab driver position interface

This interface allows the mobile application and the web presentation layer, when handling a cab driver, to directly send information about availability status updates and answers to arrangement assignations. It has to offer the change availability function, to change the availability status of a cab driver, the refuse arrangement function, to refuse an arrangement assignation, and the accept arrangement function, to accept an arrangement assignation.

Cab driver allocation request interface

This interface has to offer all the functionalities to properly perform cab driver allocation requests. Therefore has to provide the allocate cab driver function, to request the queue manager to dispatch a cab driver which will handle an arrangement, and the delete dispatch function, to stop the dispatch procedure of a cab driver to a deleted arrangement or if the cab driver was already dispatched to switch him from the unavailable-cab-driver-queue to the available one.

2.7. Selected Architectural Styles and Patterns

Some functionalities of the system are designed using a client-server architecture like for example the log in functionality in which a user sends his log-in information to the basic application manger, through the mobile application or through the web presentation layer, and it answers with the data to be rendered by the web presentation layer or the mobile application. Other functionalities like this are the profile editing, the sign-up, the arrangement deletion (only if considered with respect to the cab catcher, because if considered with respect to the cab driver the deletion of an arrangement could be an event that must be notified. The notification must occur when the cab driver was already dispatched), the request and reservation made by API based systems because the basic system doesn't notify third-party systems about the progress over arrangements but is the third-party system which asks for the status of a particular arrangement (of course these two functionalities if seen from the point of view of a cab driver are event based, because every time there is an arrangement to handle this must be notified to a cab driver). The check status functionality for the reason we've just stated is also client-server.
There are some functionalities in our system which are event-based. This functionalities are: request and reservation(made by cab catchers) because when completely handled the system must push the notification to the interested cab catcher and when it's time to assign a cab driver to either a reservation or a request the system must push a notification to the proposed cab driver.
The system is also a service based application because it exploits third-party APIs (like Google maps API) and in turn offers an API itself.
The model-view-controller architectural pattern has been applied because of the separation of concerns principle. The web presentation layer and the mobile application components represent the view of our application, the basic application manager with the queue manager represent the controller of the application and the data bases (the basic one and the municipal one) represent the model (notice that some part of the model is contained within the RAM of the tier hosting the queue manager and the basic application manager due to the queues and the arrangement in handling).
To conclude we have a 5 tier architectural pattern for the web application(client-web server-business logic-data layer-DBMS) and a four tier architectural pattern for the mobile application and the API offered service (client-business logic-data layer-DBMS). The mobile application and the web application are both thin client.

2.8. Other Design Decisions

To grant high availability (more precisely the 99.9% requested by the RASD document) must be implemented an active-active redundancy pattern over all the parts of the system which are considered critical for the service. This parts are the API manager, the queue manager combined to the basic application manager, the data layer. In the deployment view they figured over single tiers because the redundancy pattern must be transparent to the clients of those components (the web presentation layer has not this pattern applied because is not considered a critical part of the system, in fact someone can access the service anyway through the mobile application if the web presentation layer is down).This pattern has been chosen not to waste resources in our system, indeed all the redundancy elements participate in answering the requests (we don't have redundancy elements which are passive). To have a further look on how this kind of pattern works see pattern 4 in this document:

http://hillside.net/plop/2006/Papers/Library/High%20Availability%20Patterns.pdf

Another important constraint is that one over the MyTaxiService DBMS which must be an active database to provide a smoother way to deal with the reservations. Once registered, a reservation, will be associated with a trigger which, when the time to handle a reservation has come, will fire notifying the data layer which will make the arrangement manager handle the reservation.

The language through which implement this system in the development phase must be java because it offers a platform through which smoothly implement this kind of systems (java EE). It follows that JDBC must be the connector through which using the interfaces of the DBMSs because is the advised one by oracle itself.

3. Algorithm Design

3.1. Cab Driver Assignation to a Pending Arrangement

The arrangement manager of the basic application manager (with the arrangement manager of the API manager works in the same way) calls the allocate cab driver function of the cab driver allocation request interface offered by the queue manager. Using this function the arrangement manager is asking the queue manager to look for a cab driver which will handle a pending arrangement. The queue manager access the position of the starting point of the ride, which is contained in the pending arrangement passed through the call previously done by the arrangement manager. The queue manager uses this position as a criterion to choose the queue in which looking for the cab driver. It chooses the queue which has a position in the town minimizing the distance between that position and the starting point of the pending arrangement. Once the queue has been chosen, the queue manager checks if the queue is not empty. If so it gets the first cab driver in the queue and asks the communication manager to notify that cab driver of the assignation, which in turn can be refused by the cab driver itself. Otherwise if the queue is empty the queue manager reiterates the procedure previously describe to find another queue, of course, without considering the queue already checked. When the cab-driver answer reaches the queue manager, it has to decide whether to continue the search or not. If the answer to the previously done proposal is positive then the queue

manager switches the cab driver into the not available cab-drivers queue and notifies the arrangement manager with the cab driver which has decided to handle the pending arrangement. If the answer is negative the queue manager has to switch the cab driver which has just refused to handle the arrangement in the last position of the queue, and notify, as already described, the next cab driver in that queue which, of course, must be different from that one already notified. If there isn't another cab driver different from that one which has already refused the arrangement in the chosen queue, then, using the procedure already described, the queue manager finds another queue in which looking for a cab driver to assign to the pending arrangement. Every time a cab driver to be notified can't be found, the queue manager informs the arrangement manager that the assignation can't be performed. After asking the communication manager to notify the chosen cab driver, if there isn't an answer from that cab driver within a minute from the dispatch of the notification itself, then the queue manager looks for another cab driver which can handle the pending arrangement.

4. User Interface Design

4.1. User Interface Design

In this section it is provided an overview on how the user interfaces of our system will look like. The overview contains both the "look and feel" perceived by a user that wants to access the service and the one perceived by a cab catcher or a cab driver. The choice of representing only the mobile version of the interfaces is due to the fact that the web application interfaces are the same, at least in the essential core.

[UI1] Application Home Page. This interface represents the start interface and it is the one shown when a not logged-in user opens the application.

[UI2] Cab Catcher Sign-Up.

[UI3] User Log-In.

[UI4] Cab Catcher Home Page.

[UI5] Cab Driver Home Page.

[UI6] Cab Catcher Reservation.

[UI7] Cab Catcher Arrangements. This interface shows all the information about the active arrangements of a cab catcher. Through this interface it is also possible to delete an arrangement, both undesignated and designated.

[UI8] User Edit Profile.

[UI9] User Change E-mail.

[UI10] User Change Password.

[UI11] User Password Recovery.

[UI12] Cab Driver Arrangement Notification. This interface is shown when an arrangement needs to be handled (accepted or declined) from a cab driver.

[UI13] User Operation Completion. This interface is shown every time an operation (like a request, a reservation, a deletion of an arrangement or a profile edit) is completed successfully.

5. Requirements Traceability

In this section it is explained how the requirements defined in the RASD document map into the design elements defined previously in this document. In order to properly understand the mapping of the requirements, it is important to state that this section uses the notation **[Rm.n]** and **[NFRn]** to refer, respectively, to the functional and to the non-functional requirements defined in the RASD document. Hence, to have the proper background before starting to read this section, have a look to the RASD document.

5.1. Mapping of the Functional Requirements

[R1.1] This requirement is mapped into the Authentication Manager component, that will provide the Log-In functionality.

[R1.2] This requirement is mapped into the Authentication Manager component, that will provide the Password Recovery functionality.

[R1.3] , **[R1.4]** , **[R1.5]** Those requirements are mapped into the Edit Profile component, that will provide all the Edit Profile functionalities.

[R2.1] This requirement is mapped into the Sign-Up Manager component, that will provide the Sign-Up functionality.

[R2.2] This requirement needs to be mapped into 2 different components that will provide the Request functionality: the Arrangement Manager component, that will correctly handle the request, and the Queue Manager component, that will search an assignation for the request.

[R2.3] This requirement needs to be mapped into 3 different components that will provide the Reservation functionality: the Arrangement Manager component, that will correctly handle the reservation, the Queue Manager component, that will search an assignation for the reservation, and the Data Layer component, that will interact with the DBMS of the application to properly store the reservation.

[R2.4] This requirement needs to be mapped into 3 different components that will provide the Delete Undesignated Arrangement functionality: the Arrangement Manager component, that will correctly handle the deletion of the arrangement, the Queue Manager component, that in case of a pending arrangement will stop the search of an assignation, and the Data Layer component, that in case of a reservation which is not in handling, will interact with the DBMS of the application to properly delete the reservation.

[R2.5] This requirement needs to be mapped into 2 different components that will provide the Delete Designated Arrangement functionality: the Arrangement Manager component, that will correctly handle the deletion of the arrangement and the Queue Manager component, that will remove the assigned cab driver from the not-available queue.

[R2.6] This requirement needs to be mapped into the Arrangement Manager component, that will interact with Google APIs to get the estimated arrival time of an assigned arrangement. This component will also provide the information about the code by which the cab catcher will recognize the taxi.

[R3.1] This requirement needs to be mapped into the Queue Manager component, that will provide the Set Availability functionality by pushing and popping cab drivers from the available and not-available queue.

[R3.2] , **[R3.3]** Those requirements needs to be mapped into the Queue Manager component, that will provide the Decline and Accept Arrangement functionality by pushing and popping cab drivers from the available, the not-available and the zone queues.

[R4.1] This requirement needs to be mapped into the API Manager component, that will provide a programmatic interface which will allow third-party systems to perform operations over the basic one.

[R4.2] This requirement needs to be mapped into 2 different components that will provide the API Request functionality: the API Arrangement Manager[2] component, that will correctly handle the request, and the Queue Manager component, that will search an assignation for the request.

[R4.3] This requirement needs to be mapped into 3 different components that will provide the API Reservation functionality: the API Arrangement Manager component, that will correctly handle the reservation, the Queue Manager component, that will search an assignation for the reservation, and the Data Layer component, that will interact with the DBMS of the basic application to properly store the reservation.

[R4.4] This requirement needs to be mapped into the API Arrangement Manager component that will provide the Check Arrangement Status functionality by keeping track of all the information about the active arrangements.

[R4.5] This requirement needs to be mapped into 3 different components that will provide the API Delete Arrangement functionality: the API Arrangement Manager component, that will correctly handle the deletion of the arrangement, the Queue Manager component that, in case of a pending arrangement will stop the search of an assignation, while in case of an already designated arrangement will remove the assigned cab driver from the not-available queue. Finally, the Data Layer component, that in case of a reservation in handling will interact with the DBMS of the basic application to properly delete the arrangement.

5.2. Mapping of the Non-Functional Requirements

[NFR1] This requirement needs to be mapped into the Presentation Layer component, that will guarantee that the web application will run on the main known browsers.

[2] In the previous sections this component is addressed just as “Arrangement Manager”. Here, it is called “API Arrangement Manager” to highlight the fact that the component is responsible for handling requests from API based systems.

[NFR2] This requirement needs to be mapped into the Mobile App component, that will guarantee that the mobile application will run on the main known OS.

[NFR3] This requirement needs to be mapped into 2 different components that will guarantee that the cab catcher won't have no more than one active request at a time: the Mobile App component, that will meet the needs of the requirement when the cab catcher will be using the mobile application, and the Presentation Layer component, that will modify the page shown to the cab catcher using the web application in order to accomplish the requirement.

[NFR4] This requirement needs to be mapped into 2 different components that will guarantee that each reservation will occur at least two hours before the ride: the Mobile App component, that will meet the needs of the requirement when the cab catcher will be using the mobile application, and the Presentation Layer component, that will accomplish the requirement when the cab catcher will be using the web application.

[NFR5] This requirement needs to be mapped into the Edit Profile component, that will guarantee that the code sent during the password recovery procedure, will be valid for 2 hours starting from the moment in which the code will be generated.

[NFR6] This requirement needs to be mapped into the MyTaxiService DBMS component that, through the use of a trigger, will notify the need of an arrangement allocation 10 minutes before the meeting time with the cab catcher.

[NFR7] This requirement needs to be mapped into the Queue Manager component that, through the use of a timer, will consider as refused an arrangement notification which is not answered by a cad driver within 1 minute.

[NFR8] This requirement will be accomplished with the use of an Active-Active Redundancy Pattern that will guarantee an availability equal to 99,9% of the system critical parts.

[NFR9] This requirement needs to be mapped into the Queue Manager component that, through the use of a timer, will guarantee that each arrangement will have a time-out equal to 20 minutes, starting from the first allocation attempt. The Queue Manager will consider as not designable an arrangement whose time-out will be expired.

6. References

6.1. References

- The RASD document of the project, previously redacted.
- The Microsoft Application Architecture Guide, 2nd edition.

7. Used Tools

7.1. Used Tools

The tools used to create the Design Document are:

- Microsoft Office Word 2011: to redact and to format this document.
- Visual Paradigm: to create the Component View, the Deployment View and the Runtime View.
- Proto.io: to create the Mockups of the User Interfaces.

For redacting and writing this document we have spent **35 hours** per person.