Politecnico Di Milano
A.A. 2015/2016
Software Engineering 2: ”MyTaxiService”
Design Document

Paramithiotti Andrea (Matr. 788794)
Rompani Andrea (Matr. 854052)
Zoia Lorenzo (Matr. 852392)

v1.0
4/12/2015

Contents

1 Introduction

1.1 Purpose

This document represents the Design Document (DD).
The main goal of this document is to describe all the components of the system and their interaction.

1.2 Scope

The aim of the project is to create a brand new system for the management and the organization of a city taxi service. This system offers a mobile application and a web interface in order to give the customers the possibility to benefit from the taxi service. Furthermore, the system provides an additional communication interface for the taxi drivers.
The mobile application and the web interface accept requests and reservations for taxis from the users, with the possibility to organise a share ride among different users. The taxi driver is supposed to communicate his availability, acceptances and rejections of requests through the communication interface.

The system is created to simplify the access of passengers to the service and to guarantee a fair management of the taxi queues.

1.3 Definitions, Acronyms, and Abbreviations

1.3.1 Acronyms

- **DMZ:** Demilitarized Zone
- **DW:** Data Warehouse
- **DB:** Database
- **SOA:** Service-Oriented Architecture
- **RASD:** Requirements Analysis and Specification Document
- **GUI:** Graphical User Interface
- **OS:** Operating System

1.4 Reference Documents

This document refers to the RASD document of the previous deliver.
This document is conformed to the "IEEE Standard for Information Technology—Systems Design— Software Design Descriptions", (IEEE Std 1016™-2009 (Revision of IEEE Std 1016-1998)).

1.5 Document Structure

- **Introduction:** it gives a description of the document and the information about the project scope
- **Architectural Design:** it contains all the information about the architectures designed for the system and the components that compose them. It provides the Component and Deployment diagram in order to clarify the exact distribution of the components and their interaction. In this chapter all the motivations that have determined the choice of the architecture are also included
- **User Interface Design:** it contains all the mockups useful for the description of the user experience
- **Algorithm Design:** it tries to give an idea of a possible algorithm of a chosen system procedure
- **Requirements Traceability:** it explains how the requirements defined in the RASD map into the design elements that were defined in this document
- **References:** it contains the documentation
- **Revisions Table:** it contains the information of the changes for every revision

2 Architectural Design

2.1 Overview

The most important part of the System is structured using the SOA. For this reason, everything that finds place out of our physical system, so all the interfaces used by a person, are defined as Service Requestors. Every operations that the system has to provide are split into atomic services.
In order to manage every request coming from the outside and every message among the services there is a Broker between the service providers and the requestors.
Regarding the data managing, there are two different approaches:

- Storing all the sensitive data in a database, contained into a DMZ, in order to guarantee a high level of security.
- Using a DW, connected to an analyser software, to store all the other data in order to perform data mining operations.

The last component defined is an http Web Server used for the storage of every file needed by the Website.
Push Notifications are used to perform the communication from the service providers to the requestors or directly to the web server, depending on the user/Taxi driver interface involved.

2.2 High level Components and their Interaction

The components of the system are:

- The service providers, which are the software components that must be installed on the servers and that contain the logic in order to manage the services offered to the clients.
  In this system the providers are:
  - The requests manager that handles the clients requests for a normal ride
  - The shared rides manager that handles the clients requests for a shared ride
  - The reservations manager that handles the clients reservations
  - The registration manager that handles the users registration to the service, storing their information on the users database
  - The authentication manager that handles the users authentication to the system preventing unauthorized access into the system
  - The taxi manager, which performs the coordination of the queue and the Taxi drivers' availability
  - The emergency manager that handles the emergency messages from the taxi driver before and during a ride

- The service requestors, which are the interfaces that request the execution of a service. They all have an active internet connection in order to send the requests to the providers.
  In this system, according to the RASD description of the devices, the requestors are:
  - The Taxi driver interface that acts as the interface to the system for taxi driver
  - The web interface that is the web browser interface used by the clients to access the system functions and also eventually for administration purposes
  - The mobile interface that is the interface used by mobile devices to access the system functionalities

- The Broker, which is the central core of the system. It manages all the communications between the requestors and the providers or the providers themselves.
  It is composed of three elements:
  - **Service Broker**: this component receives the requests from the requestors through an internet connection and is able to forward them to the designated provider. Consequently, it has a direct connection to the internet and all the Service Providers.
  - **Notification Center**: this component allows the service providers to communicate to the requestors using the Push Notification messaging protocol. In order to perform his job, it is connected with internet and all the Service Providers, just like the Service Broker.
  - **Internal Message Dispatcher**: the aim of this component is to allow and manage the communication among the service providers, so it has a direct connection with each of them.

- The Web Server, which manages all the https requests and where all the files useful for the website are stored. It is also connected to the Service Broker and, together with it and the Notification Center, they are the only components that are connected to internet

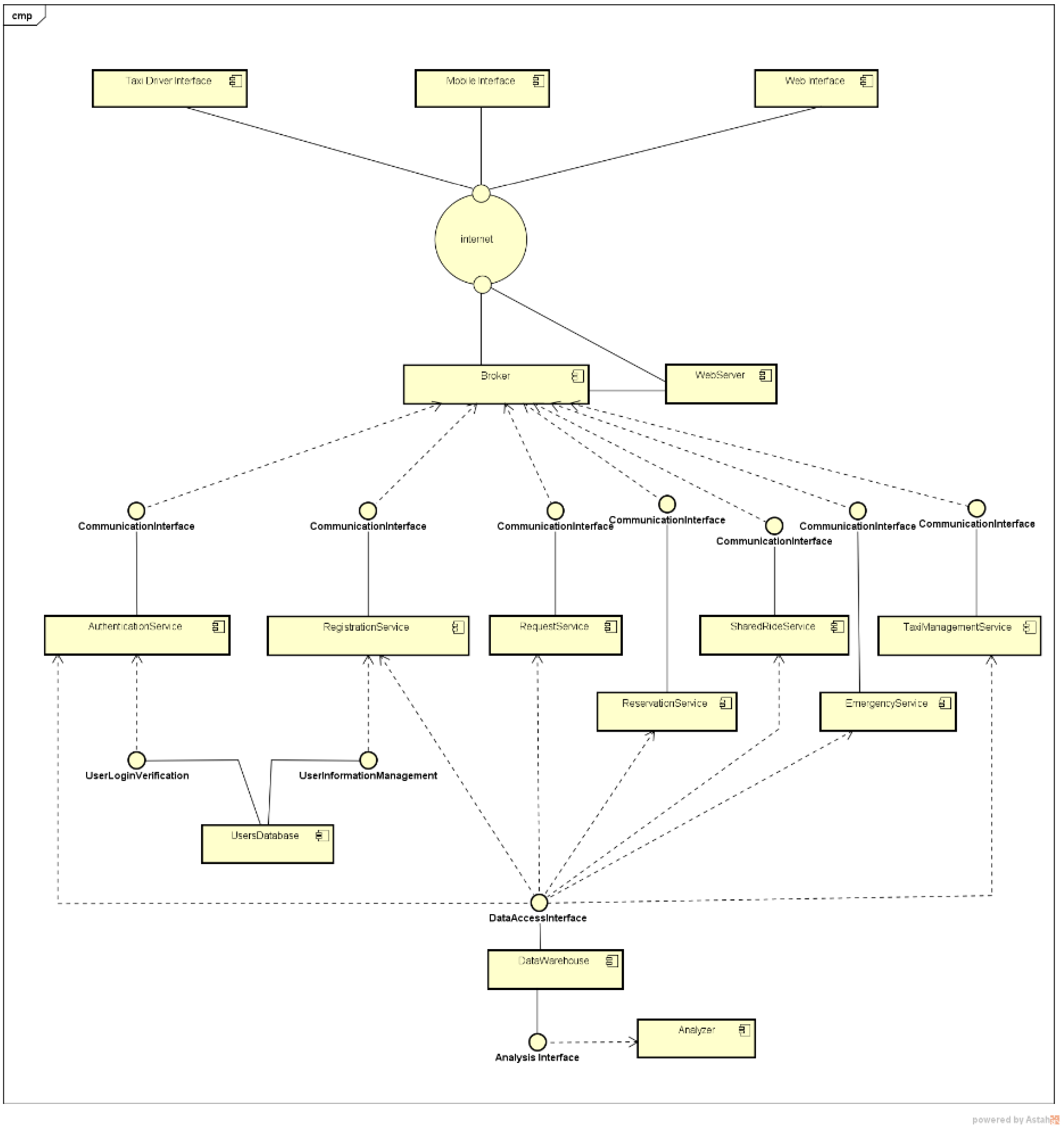
- A Data Warehouse, which is used directly by the providers in order to store or read any useful or sensitive data. It works with an OLAP technology for the storage of the data

- A Database, storing the users information, contained into a DMZ in order to increase the security and prevent unauthorized accesses to sensitive information

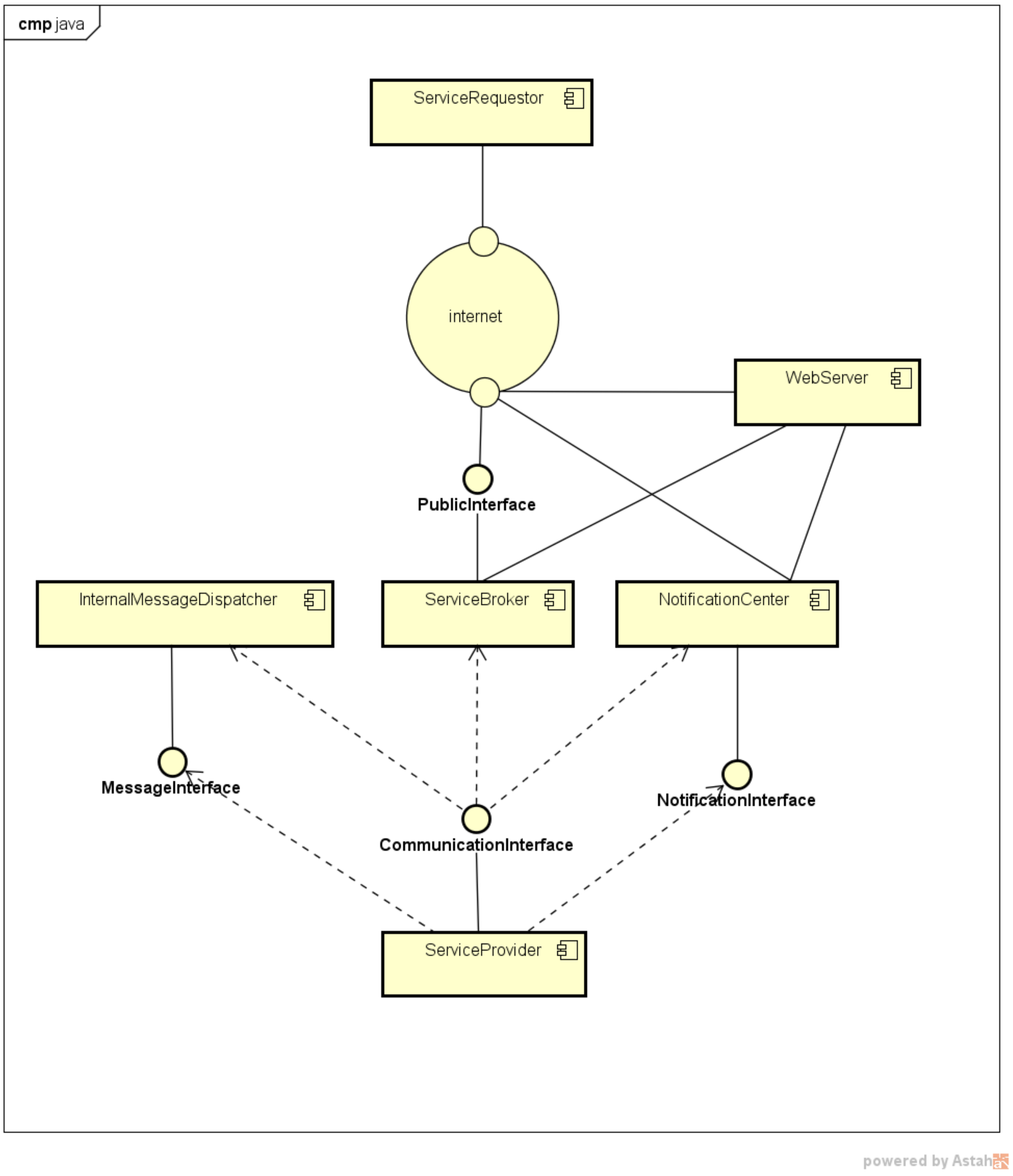
- The Analyser, a software application that have access to the data in the Data Warehouse and performs the statistical analysis

2.3 Component View

In this component diagram, the internal division of the Broker and the interfaces of each single part are implied, in order to increase the overall comprehensibility.



This diagram, instead, shows the division and the interfaces connection aforementioned in more details. In this diagram, all the service providers and the requestors are compressed in a single component in order to focus the attention on the interaction of the Broker's parts.

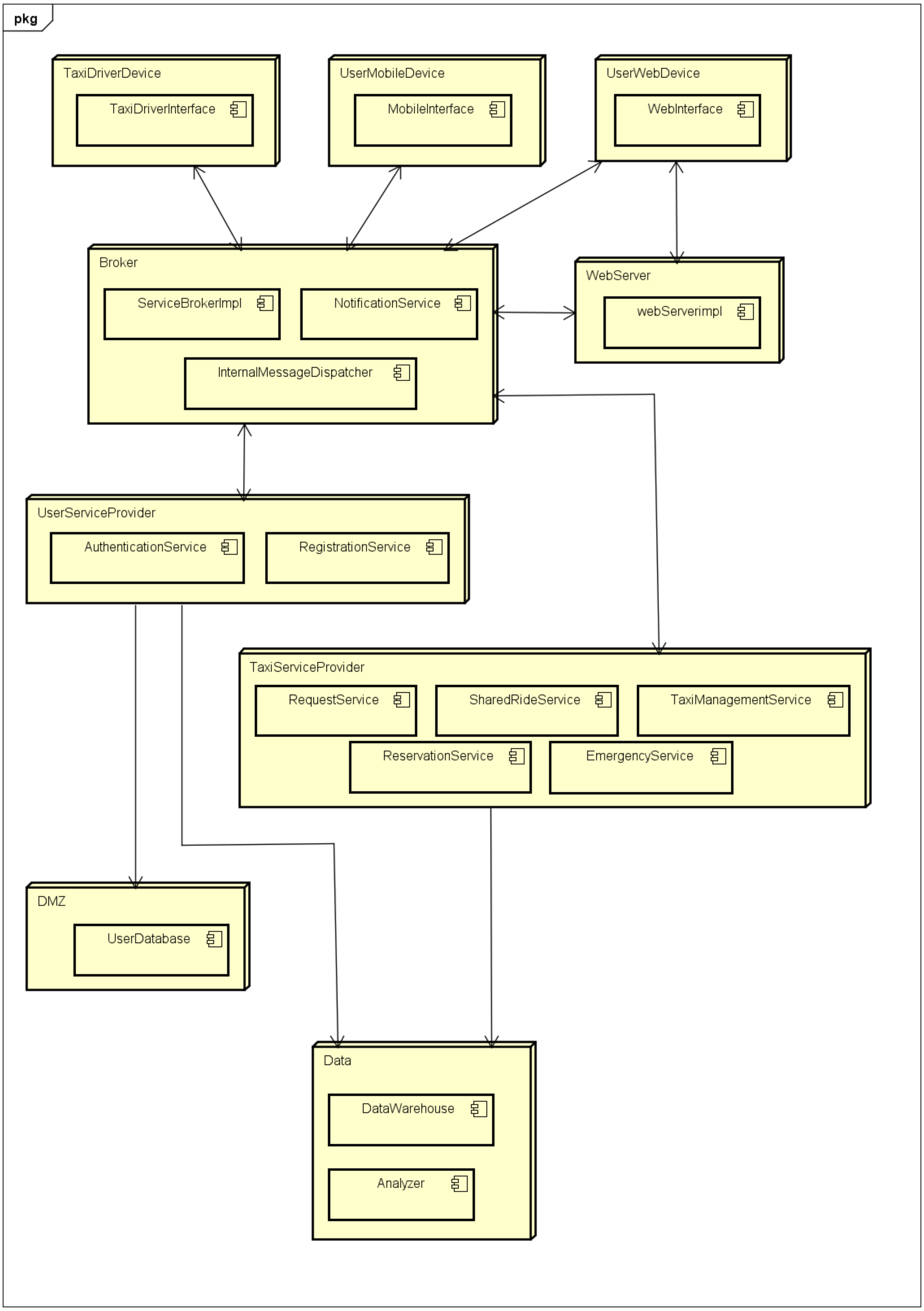


2.4 Deployment View

In this view are shown the interactions between the different functionalities, showing the logical division between services, in particular:

- The deployment of the users database into a DMZ to prevent unauthorized access to the users data as much as possible
- The presence of the data warehouse as the location of the data storage for everything not related to user authentication

The division in nodes of the deployment diagram is just logical and it does not represent a constrain for future hardware implementations.
Because of this reason, the components are grouped only by functionalities, as they were presented in the High level Components and their Interaction paragraph (2.2).

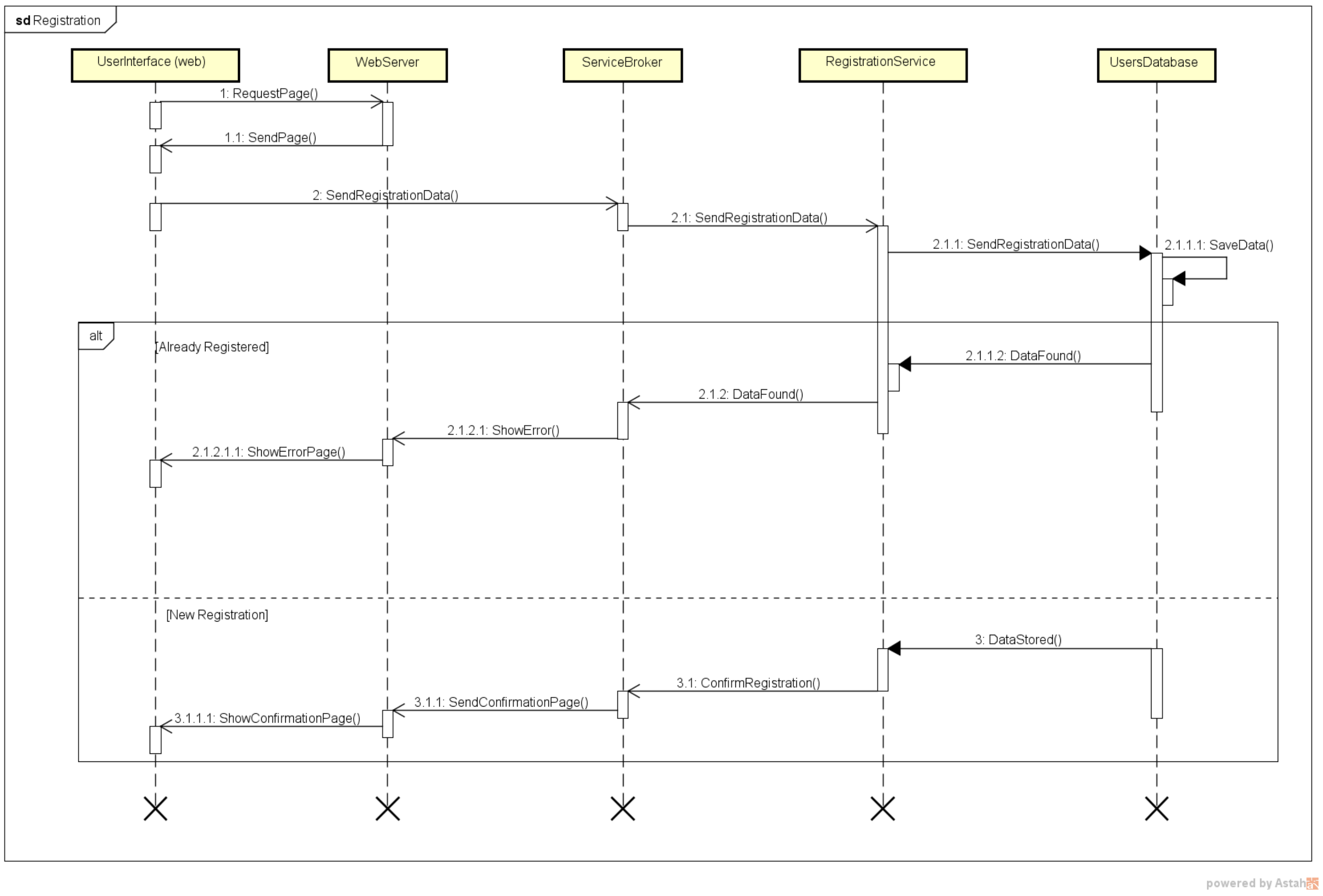

pkg
TaxiDriverDevice
TaxiDriverInterface
UserMobileDevice
MobileInterface
UserWebDevice
WebInterface
Broker
ServiceBrokerImpl
NotificationService
InternalMessageDispatcher
WebServer
webServerimpl
UserServiceProvider
AuthenticationService
RegistrationService
TaxiServiceProvider
RequestService
SharedRideService
TaxiManagementService
ReservationService
EmergencyService
DMZ
UserDatabase
Data
DataWarehouse
Analyzer

2.5 Runtime View

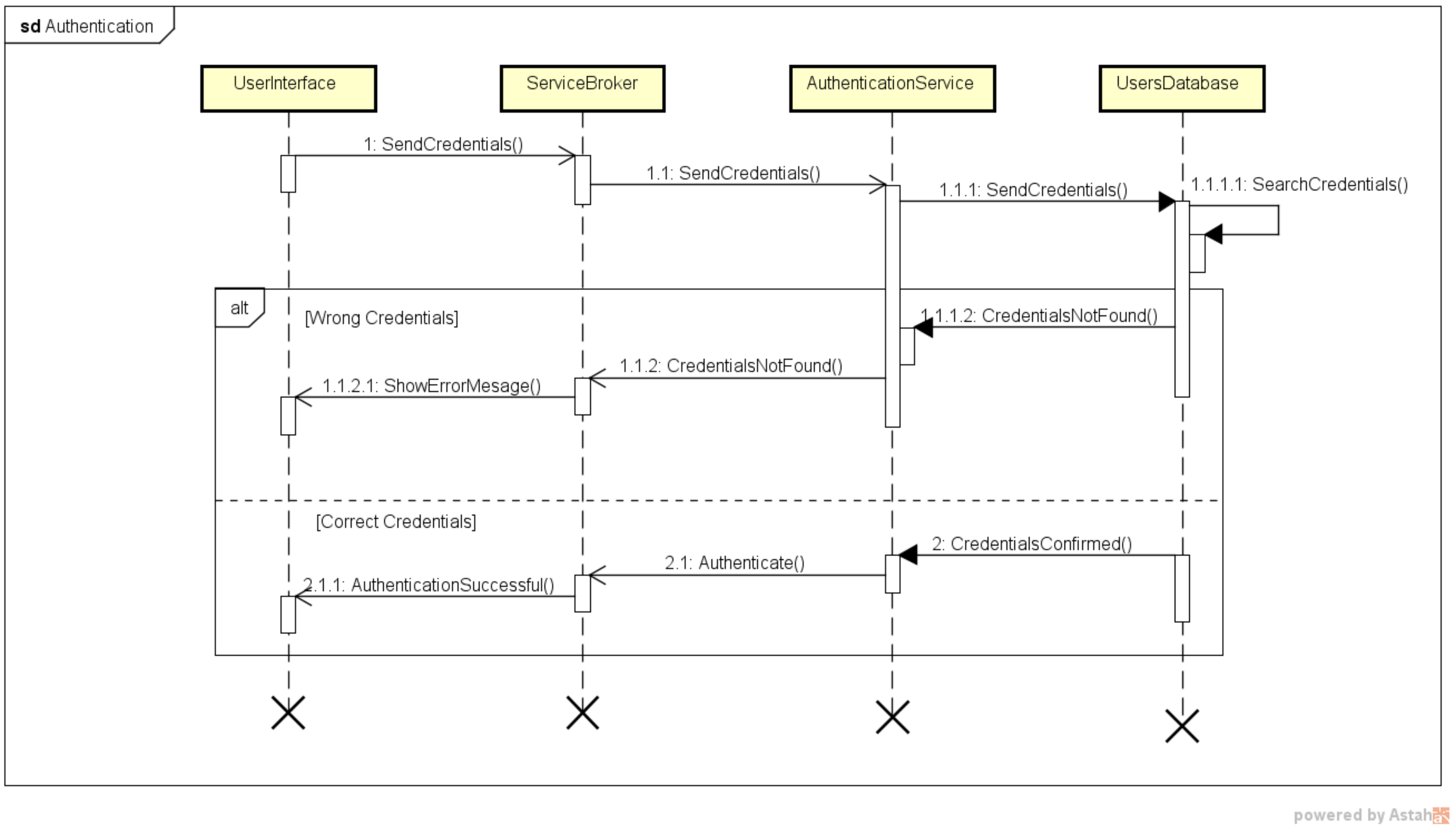
Here are shown the sequence diagrams that show how the components interact to accomplish the tasks described above and in the RASD document.

The request of every web page is always made by the user web interface or a specific service to the web server, which answers sending the requested web pages.
It was decided to show this process only in the first sequence diagram, as it would have been too complex and chaotic to show this process every time that a web page was requested.

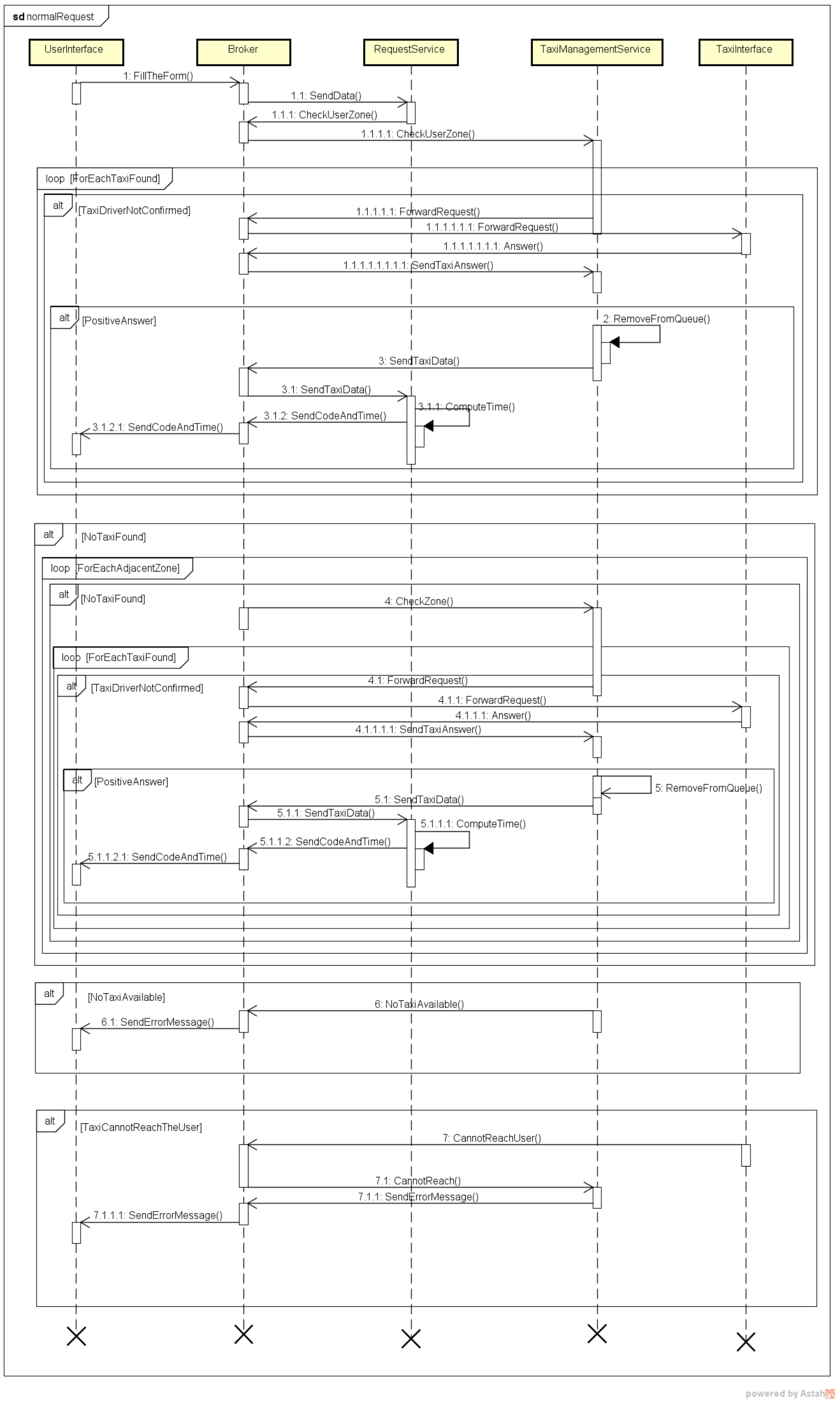
2.5.1 Registration



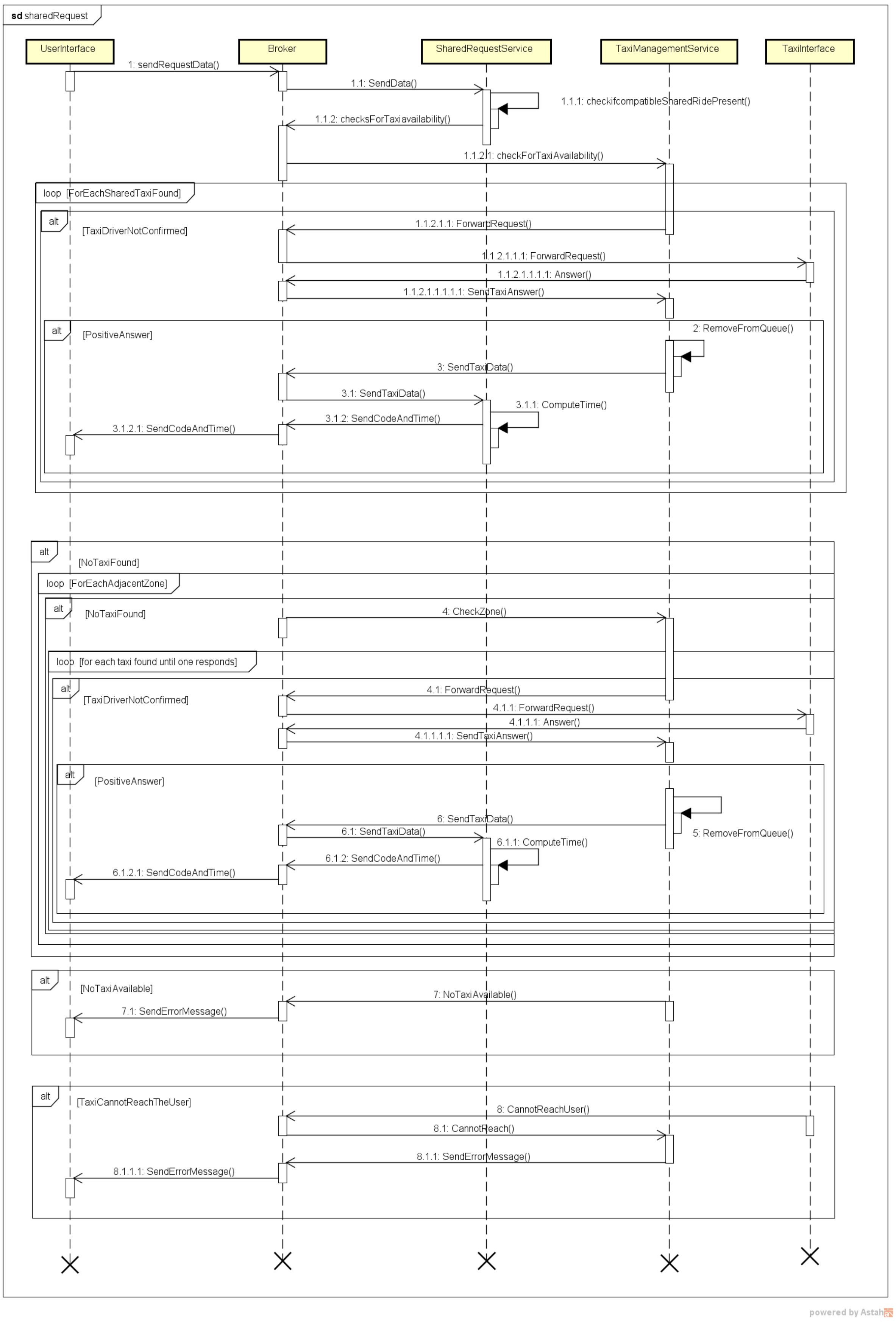
2.5.2 Authentication



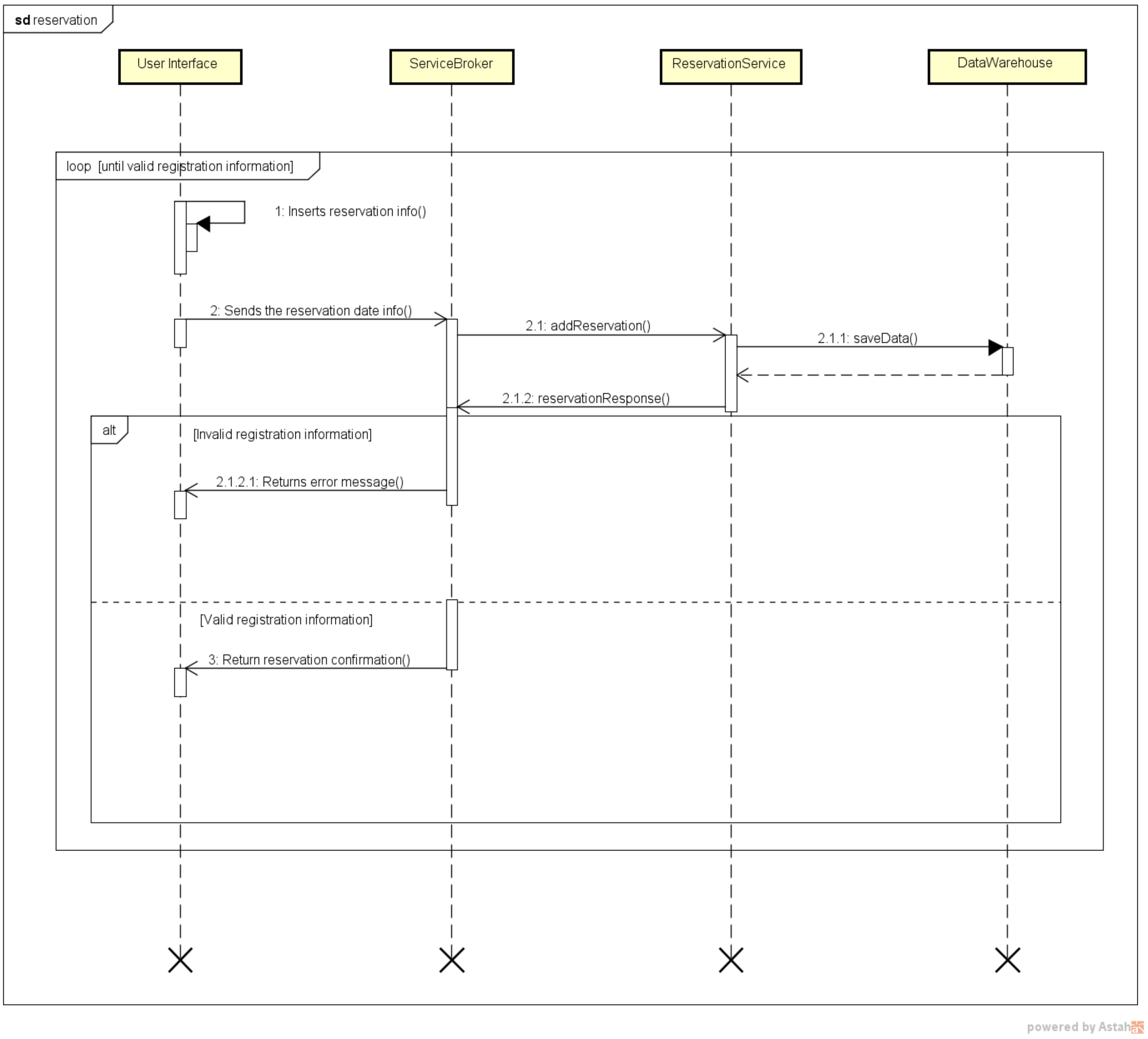
2.5.3 Normal Request



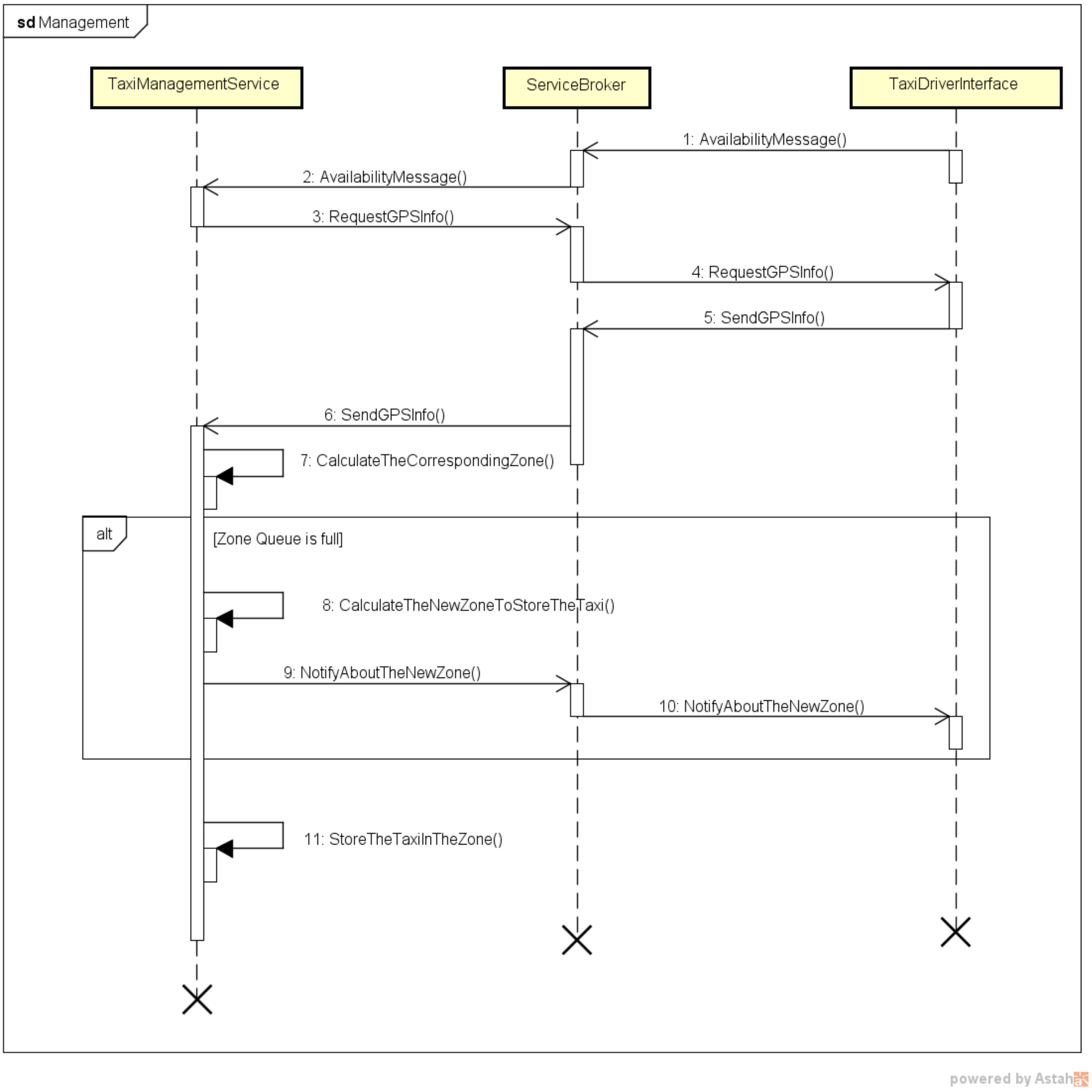
2.5.4 Shared Request



2.5.5 Reservation

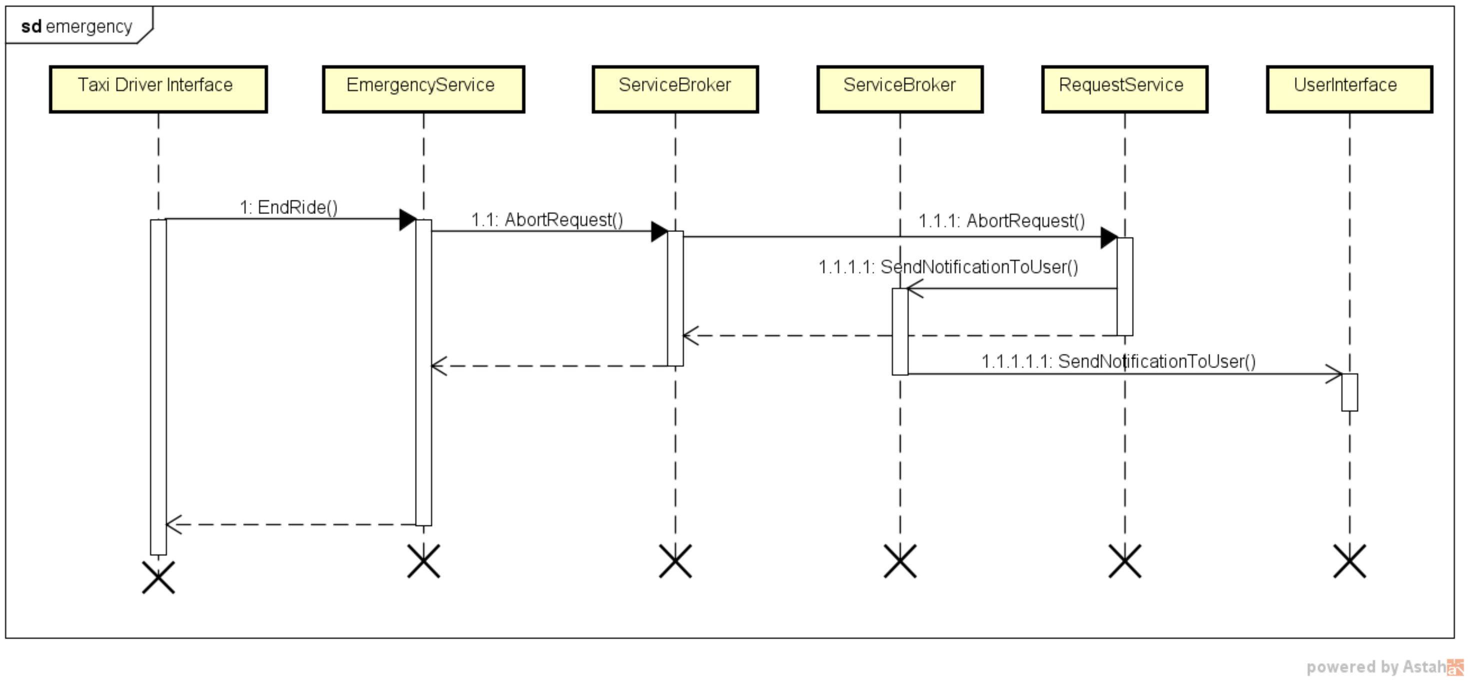

sd reservation
User Interface
ServiceBroker
ReservationService
DataWarehouse
loop [until valid registration information]
1: Inserts reservation info()
2: Sends the reservation date info()
2.1: addReservation()
2.1.1: saveData()
2.1.2: reservationResponse()
alt
[Invalid registration information]
2.1.2.1: Returns error message()
[Valid registration information]
3: Return reservation confirmation()
powered by Astah

2.5.6 Management



2.5.7 Emergency

This sequence diagram contains the flow of events of a ride cancellation done by the taxi driver and the communication of the cancellation to the user interface.
In the case that the user requests another taxi, the sequence diagram follows the steps of the 2.5.3 except for the fact that the request starts from the Emergency Service, which itself notifies the user of the arrival of the new taxi.



2.6 Component Interfaces

Component	Interface	Interface method Prototype	Parameters			Output
Service Broker	Public Interface	DispatchMessage	Source	Destination	Content	Result Data
Internal Message Dispatcher	Message Interface	DispatchMessage	Source	Destination	Content	Result Data
Notification Center	Notification Interface	SendNotification	Destination	Message		Success
Authentication Service	Communication Interface	Login	User Login Credentials			Login Success state
						Login related Information
Registration Service	Communication Interface	Registration	User Data			Registration success state
Request Service	Communication Interface	NewRequest	Starting Location	User		Request
		LinkTaxi	Request	Taxi		Success
		AbortRequest	Request	Motivation		Success
		StartReservation	Reservation	User		Request
Reservation Service	Communication Interface	ReserveTaxi	Reservation Info	User		Reservation
		CancelReservation	Reservation ID	User		Success
Emergency Service	Communication Interface	EndRide	User	Taxi Driver		Success
		SendNewTaxi	Location	User		Taxi
Taxi Management Service	Communication Interface	FindTaxi	Starting Location	Request		Taxi
		SendTaxiAnswer	Request	Taxi Driver		Success
		SetAvailability	Taxi Driver	Value		Success
Shared Ride Service	Communication Interface	StartSharedRide	Starting Location	Destinat ion	User	Request
		FindCompatibleRide	Starting Location	Destinat ion	User	Request
Users Database	User Information Management	RegisterUser	User Data			Registration Success State
						Registration Errors
	User Login Verification	Login	User Login Credentials			Login Success State
Data Warehouse	Data Access Interface	Save Data	Key	Data		Saving success state
	Analysis Interface	Analyze	Type of analysis	Data		Analysis Results

2.7 Selected Architectural Styles and Patterns

The main architectural style designed for this system is the SOA.
In fact, as described in the High Level Components and Their Interaction paragraph (2.2), it is possible to recognize the typical SOA three-ways division. There are both the Service Requestors and the Service Providers, other than the Service Broker that allows the communication between each element.
It was decided to design the project with this architecture for these reasons:

- **Good management of a big number of users**
  Thanks to the division in atomic services that the system provides, the application is able to manage a large amount of different requests in parallel

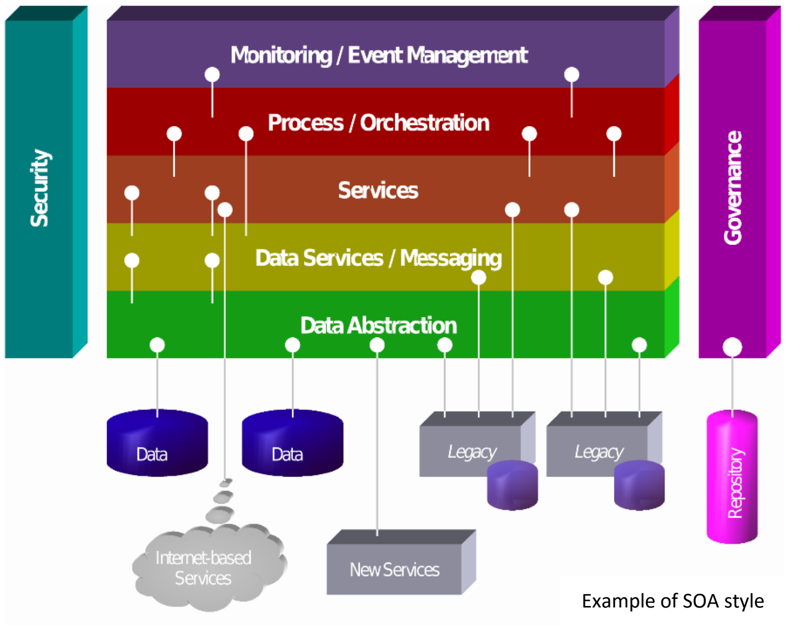
- **Future developments**
  The division in services ensures a simple way for the addition of new implementations and service upgrades made with the APIs considered in the RASD

- **Scalability**
  The division in services allows a better management of the resources also in the case of an unexpected increase of the requests

- **Reliability**
  Having the possibility to add new servers to our cluster, a high level of availability can be guaranteed. For example, we can have one or more auxiliary servers that in case of failure can be easily used in order to keep on managing the external requests in a transparent way towards the external users.

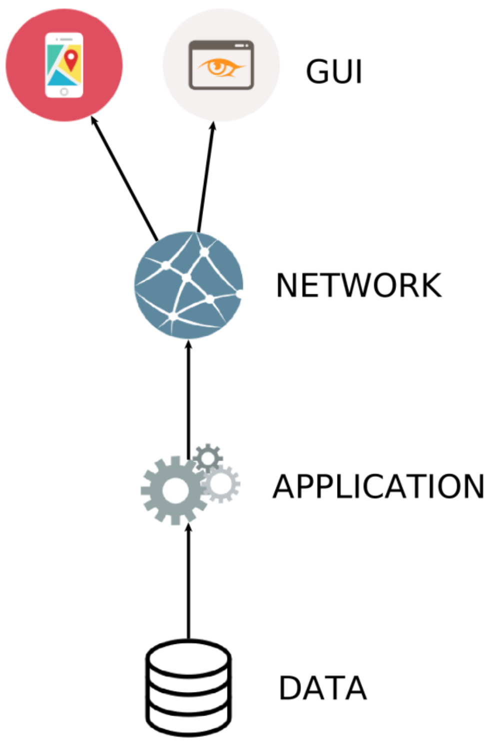
- **Maintainability**
  The quite small range of our application makes it easy to handle the maintenance of the individual services. If the application becomes bigger, this point will forcedly become an issue

The most important issue in this design is how to perform the communication between each service.
As told in the paragraph above and explained in the component diagrams, each service exhibits a communication interface. Thanks to this, each service is able to guarantee some methods to the others and to the users/taxi drivers. In this way, it is possible to perform a call to these methods by sending a message containing some identification codes (as an example: both the ID of the caller and the called) and useful parameters to the dispatcher and then, if necessary, waiting for the answer.
Moreover, it was decided to allow the communication only through the Internal Message Dispatcher because it is not necessary that every service knows the physical location of the others; this guarantees a better scalability and reliability.



Example of SOA style

The other architectural style designed for this system is a Client-Server, regarding everything about the management of the website. As shown by the image below, In order to give the possibility to use our website from every html 5 compliant browser, a Two-tiers architecture was chosen, where both the data and the application find place in the system server and the GUI is handled directly by the browser of the device.



2.8 Other Design Decisions

It was decided to divide the Broker into three parts for this reasons:

- **Better distribution of work**
  The division of the request amount allows us to avoid an overloading of a single element and to perform more different requests at the same time
- **Better quality of service**
  A malfunction of the internal message dispatcher can be masked keeping online the Service Broker and performing all the atomic operations like the authentication or the registration

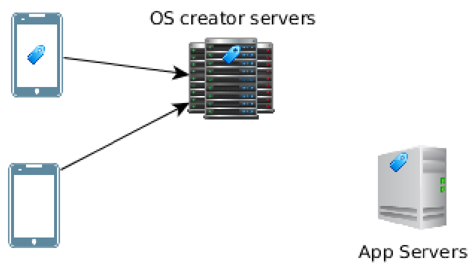
The Notification Center stores a mapping of the User IDs and the User Interfaces in order to univocally link every user to the related interface.

It was decided to use a Data Warehouse in order to exploit its peculiarity of knowledge extraction from data. In fact, the aim is to use it for the prevision of a possible future increase of the requests and the creation of statistics useful for the economy of the company.
Usage data can also be used to manage the work division on the various components in the best way. Finally, through the traffic analysis, it is possible to forecast the time of the day when there are both the maximum and the minimum usage peaks and consequently to adapt the behaviour of the system.

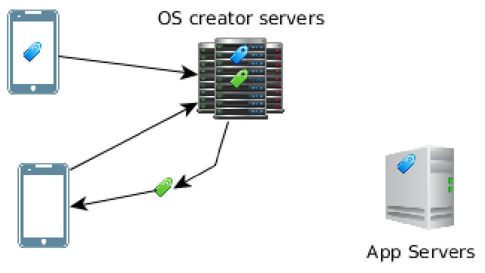
In order to send the requests to the taxi drivers and to communicate to the clients which taxi is arriving to pick him, it was needed a way to communicate unilaterally from the services to the taxi drivers and from the services to the mobile application. The solution is to use push notifications.
This functionality uses a technology developed for mobile devices, it creates a unique link between the phone and the OS creator servers, saving battery life and reducing usage of mobile data connection. All the steps required in order to create this link are presented in the Image below.

Simple explanation of the push notification service used in this application

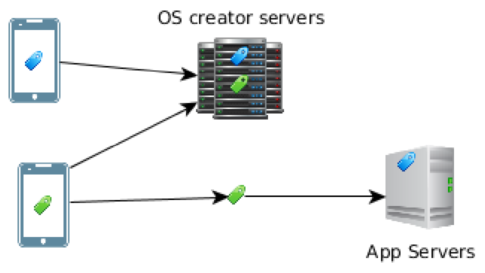
1 Phone application connects to the os creator servers



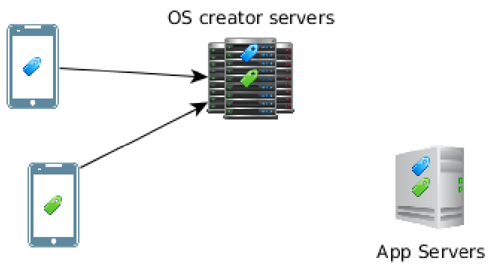
2 Phone application asks for a token specific for the <application,phone> pair



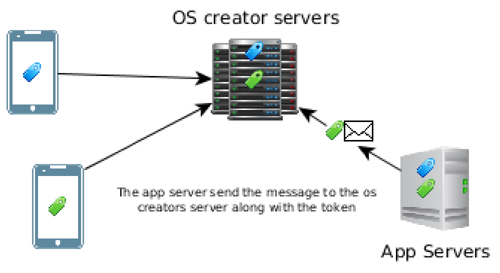
3 Phone application communicates the token to the application servers



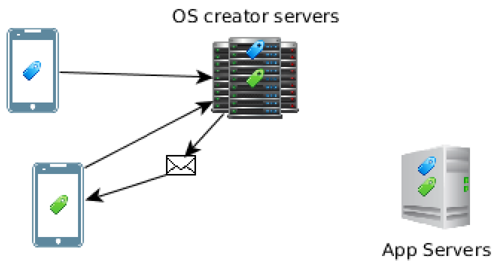
4 Application servers store the token



5 Server need to communicate unilaterally with the specific phone


OS creator servers
The app server send the message to the os creators server along with the token
App Servers

6 OS creator server sends the message to the correct mobile application



3 Algorithm Design

The abstraction level of the documents has always been high and it was decided to appoint the implementation of the behaviour of each component to the developers. For this reason, the definition of detailed algorithms would be inaccurate and out of place.

It was decided to show an idea about the management of the available taxis and the queues in the context of a normal request.
In the pseudo code below, the expressions specifying the calls made to the other services are written as pseudo object oriented method calls like the one below:

ServiceBroker.**Service**(service_parameters).**method**(method_parameters)

Where:

- ServiceBroker: is the interface object of the service broker
- Service: is the name of the service that the call wants to access
- service_parameters: are the parameters to send to the interface
- method: is the method of the interface to call
- method_parameters: are the parameters to send to the method

3.1 Request

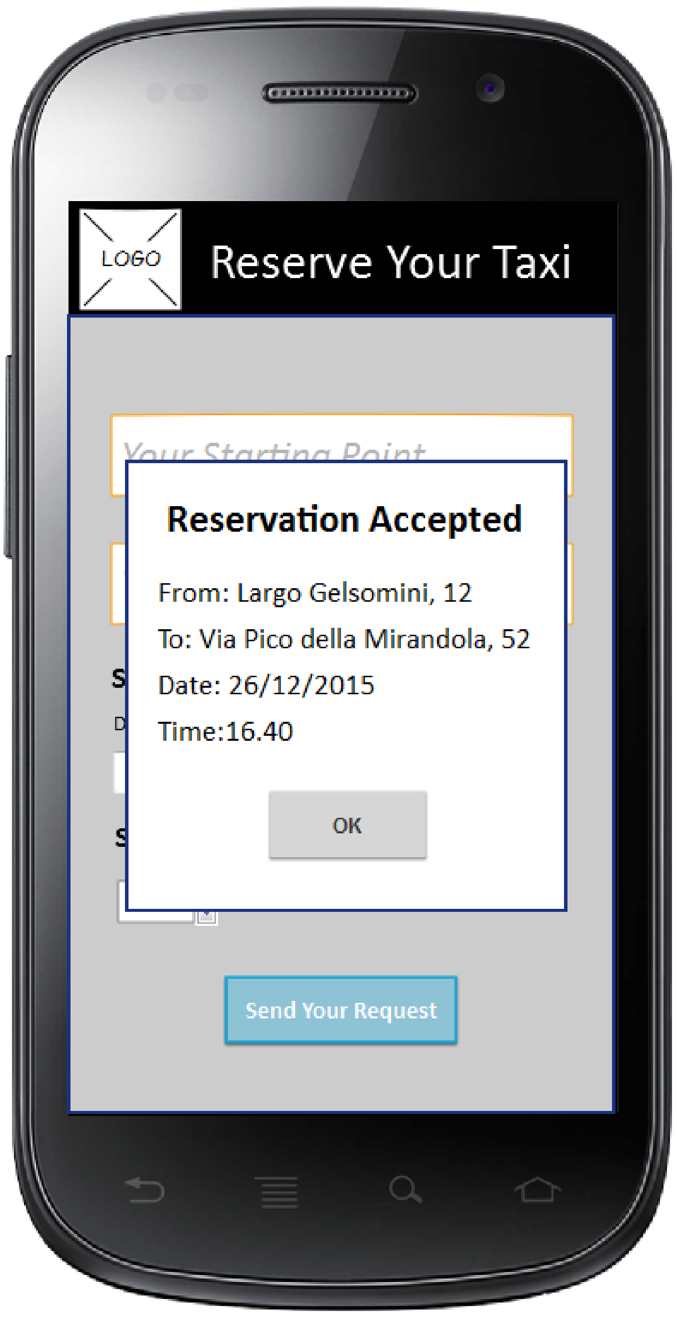
```
//support function used to call a taxi and wait for the response
Boolean callTaxi(zone) {
    if (taxi_driver_not_confirmed){
        //send the request to the taxi interface
        ServiceBroker.TaxiInterface(taxi).ForwardRequest(clientzone);
        //waiting for taxi driver response
        while (answer not received){
            Wait_for_response();
        }
        if (taxi_answer_positive) {
            RemoveFromQueue(zone,taxi);
            //Links the taxi to the request on the request service algorithm
            ServiceBroker.RequestService.linkTaxi(request,taxi);
            return true;
        }
    }
    return false;
}
```

```
Taxi findTaxi(startingLocation,Request){
        Taxis = taxi queue of the zone of the starting location;
        TaxiFound=false;
        SelectedTaxi=nil;
        foreach taxi in Taxis {
            if (callTaxi(taxi)){
                TaxiFound=true;
                SelectedTaxi=taxi;
                break;
            {
        }
        if (!TaxiFound) then
            Zones=list of the zones each containing its queue;
            foreach (zone in Zones) {
                  Taxis = taxi queue of the zone: zone
                      foreach (taxi in Taxis) {
                          if (callTaxi(taxi)){
                              TaxiFound=true;
                              SelectedTaxi=taxi;
                              break;
                          }
                  }
                      if (TaxiFound){
                          break;
                      }
            }
        }
        if (!TaxiFound) {
            //no taxi can be fund so the request is aborted
            ServiceBroker.RequestService.abortRequest(request,"no taxi anywhere");
        }
}
```

4 User Interface Design

In addition to the mock-ups defined in the RASD document, this chapter presents other examples showing some situations that were not taken into account in the RASD document.

The following two mock-ups show how the visualization of a notification sent by the server is displayed in the app.



Reservation Accepted

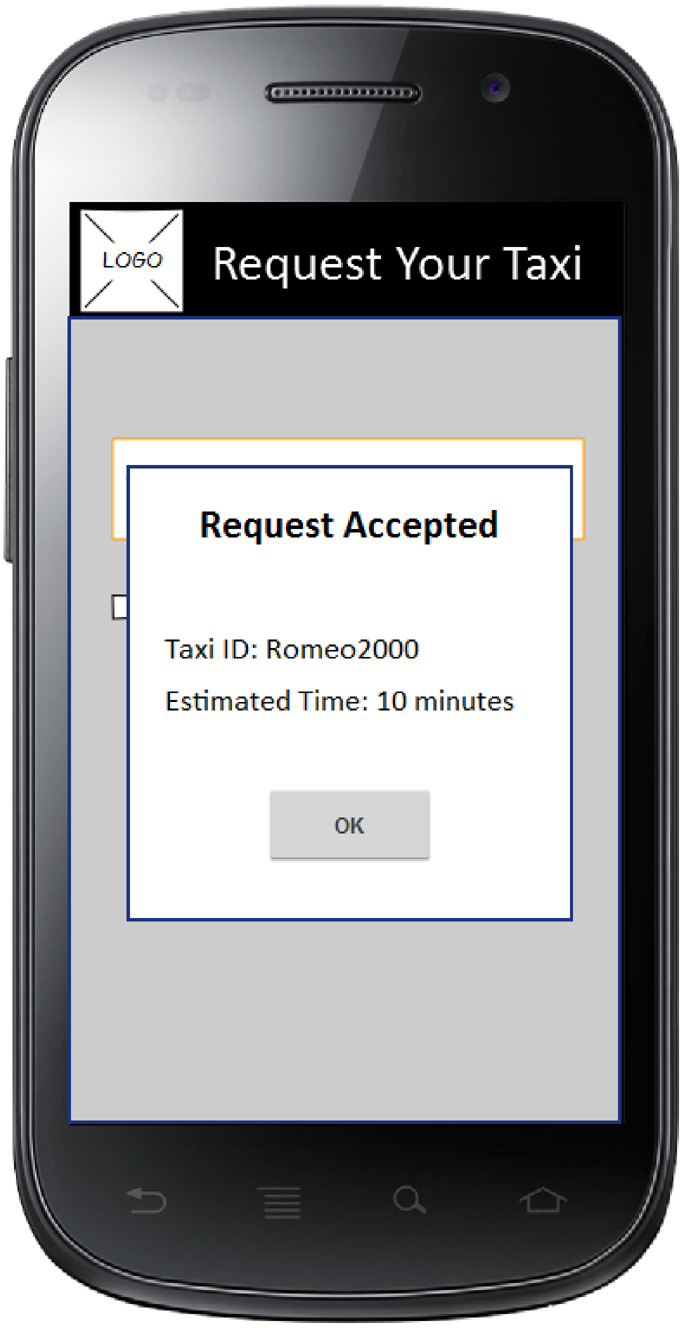
From: Largo Gelsomini, 12

To: Via Pico della Mirandola, 52

Date: 26/12/2015

Time:16.40

OK



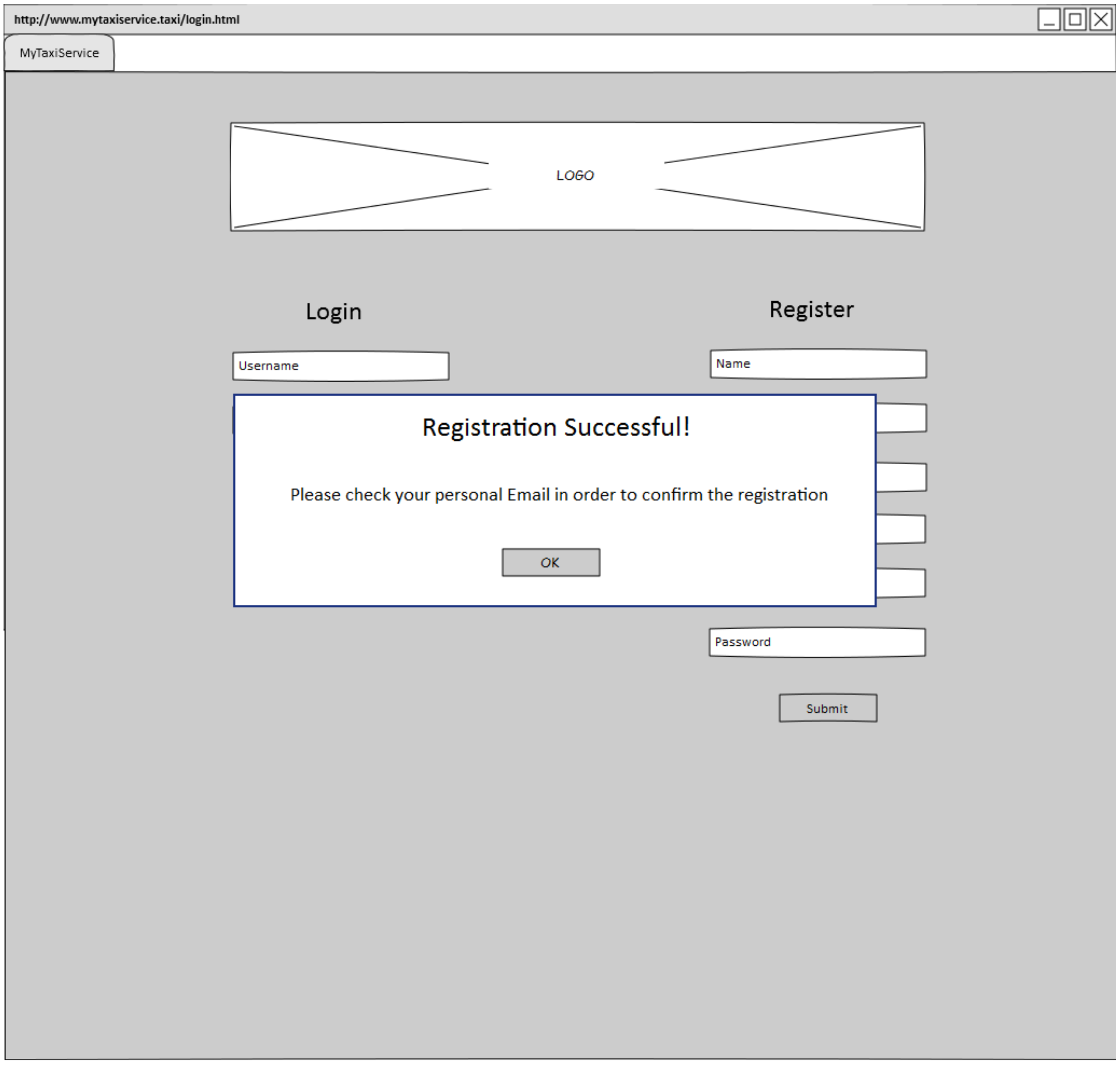
Request Accepted

Taxi ID: Romeo2000

Estimated Time: 10 minutes

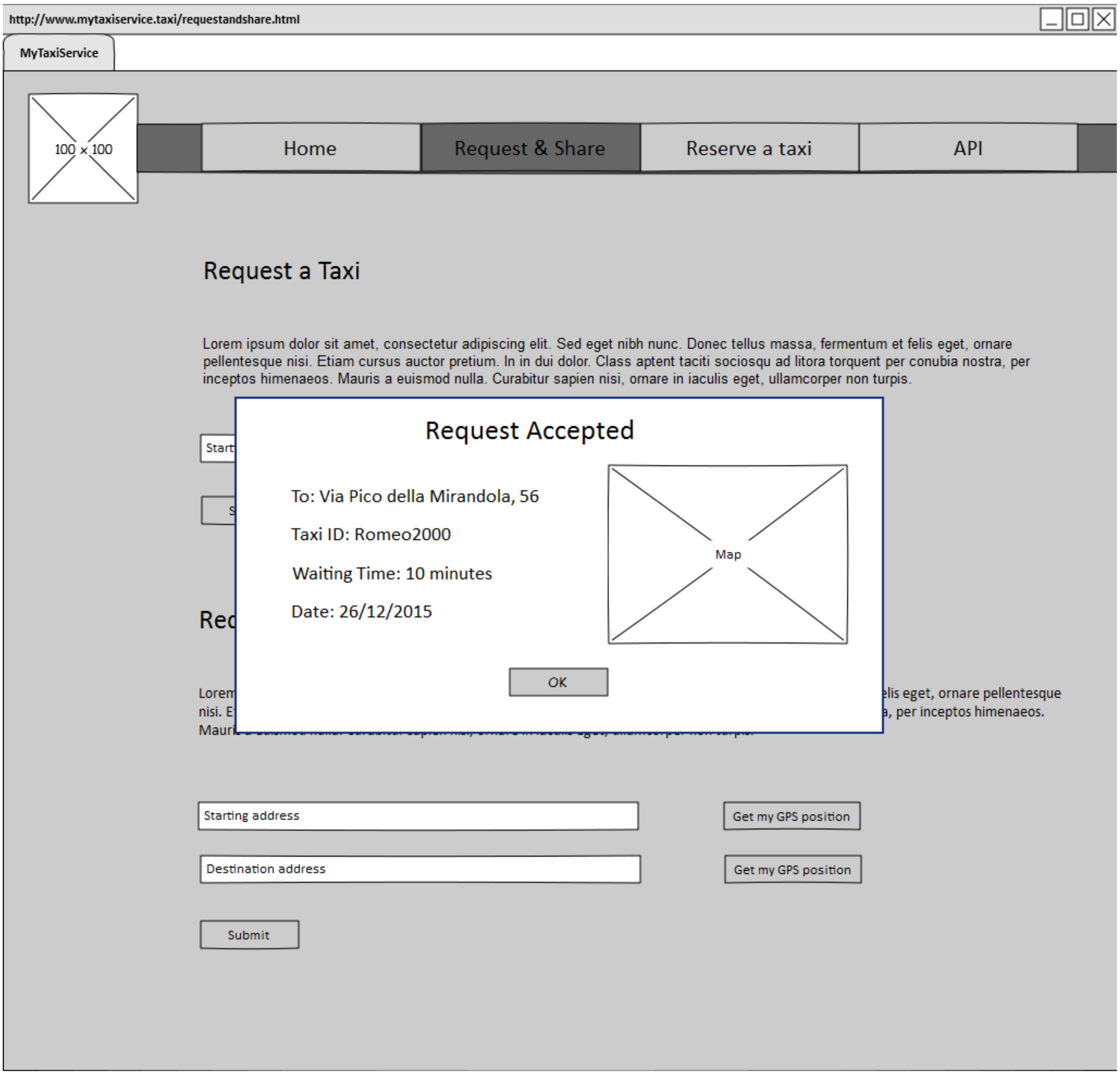
OK

The two mockups below show the visualization of a notification in the Website.



Registration Successful!

Please check your personal Email in order to confirm the registration



Request a Taxi

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed eget nibh nunc. Donec tellus massa, fermentum et felis eget, ornare pellentesque nisi. Etiam cursus auctor pretium. In in dui dolor. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Mauris a euismod nulla. Curabitur sapien nisi, ornare in iaculis eget, ullamcorper non turpis.

Start

S

Req

Lorem ... elis eget, ornare pellentesque nisi. E... a, per inceptos himenaeos. Mauri...

Request Accepted

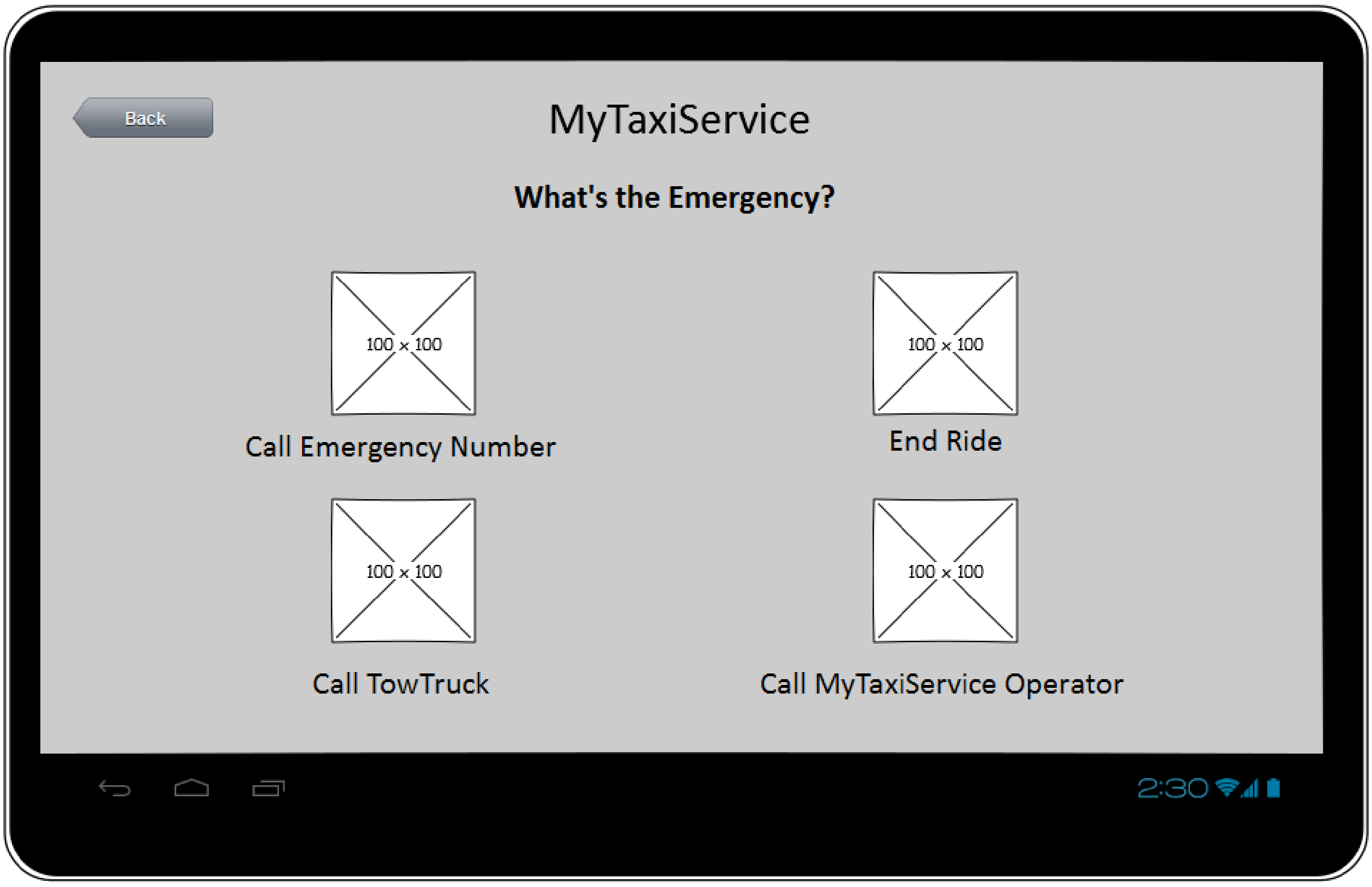
To: Via Pico della Mirandola, 56

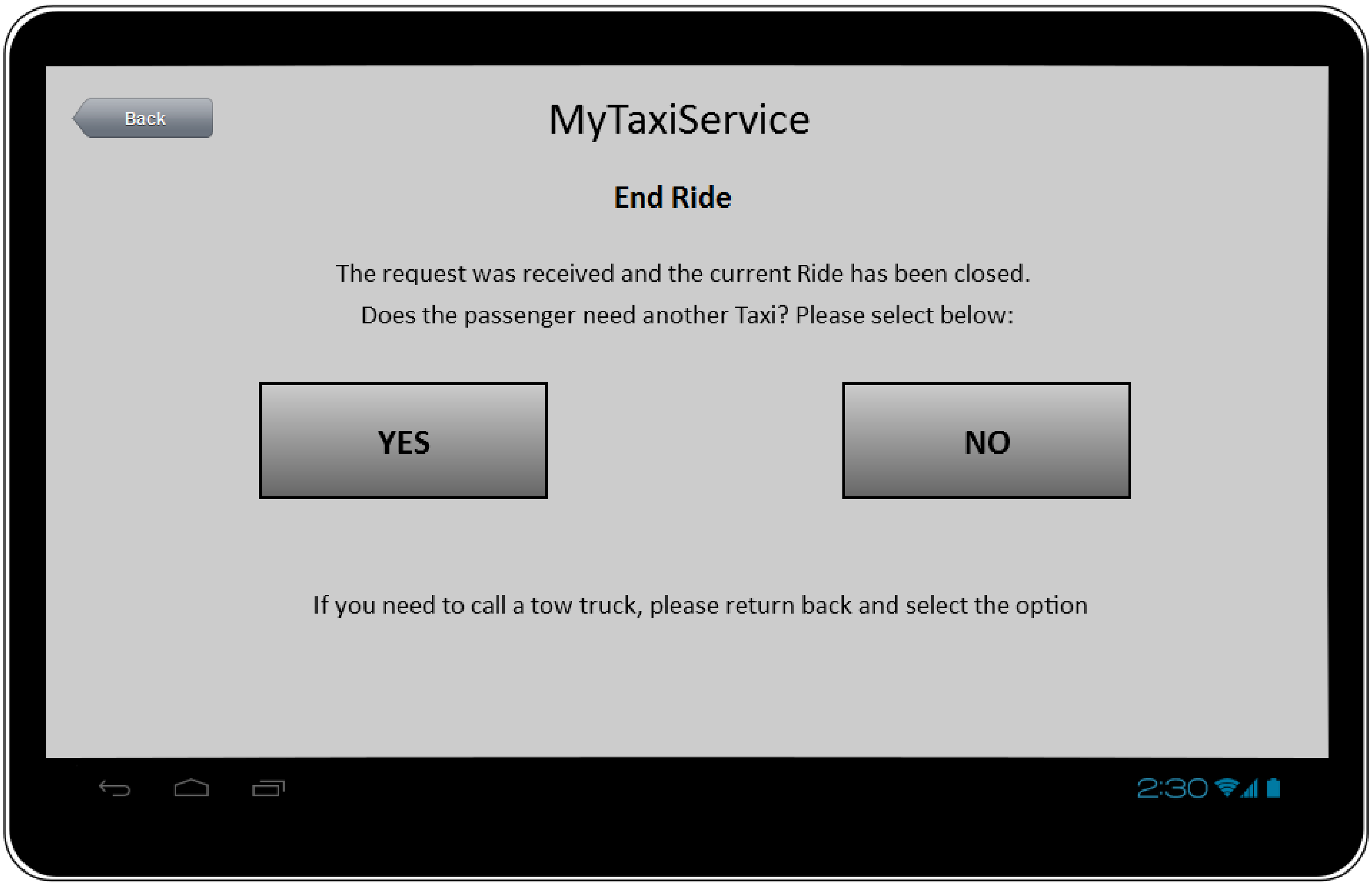
Taxi ID: Romeo2000

Waiting Time: 10 minutes

Date: 26/12/2015

The three following mock-ups illustrate all the layout concerning the emergency process.

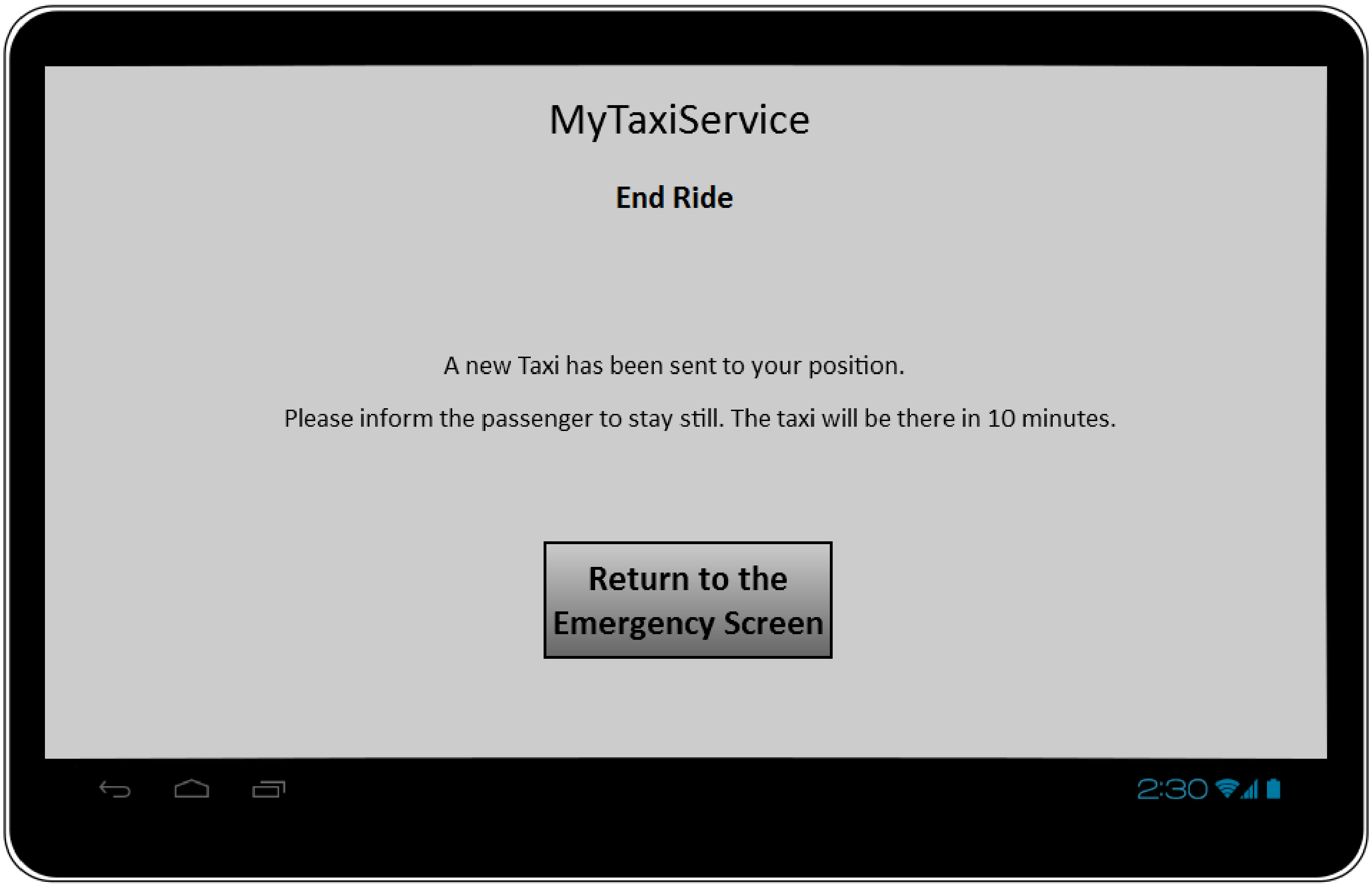




The request was received and the current Ride has been closed.

Does the passenger need another Taxi? Please select below:

If you need to call a tow truck, please return back and select the option



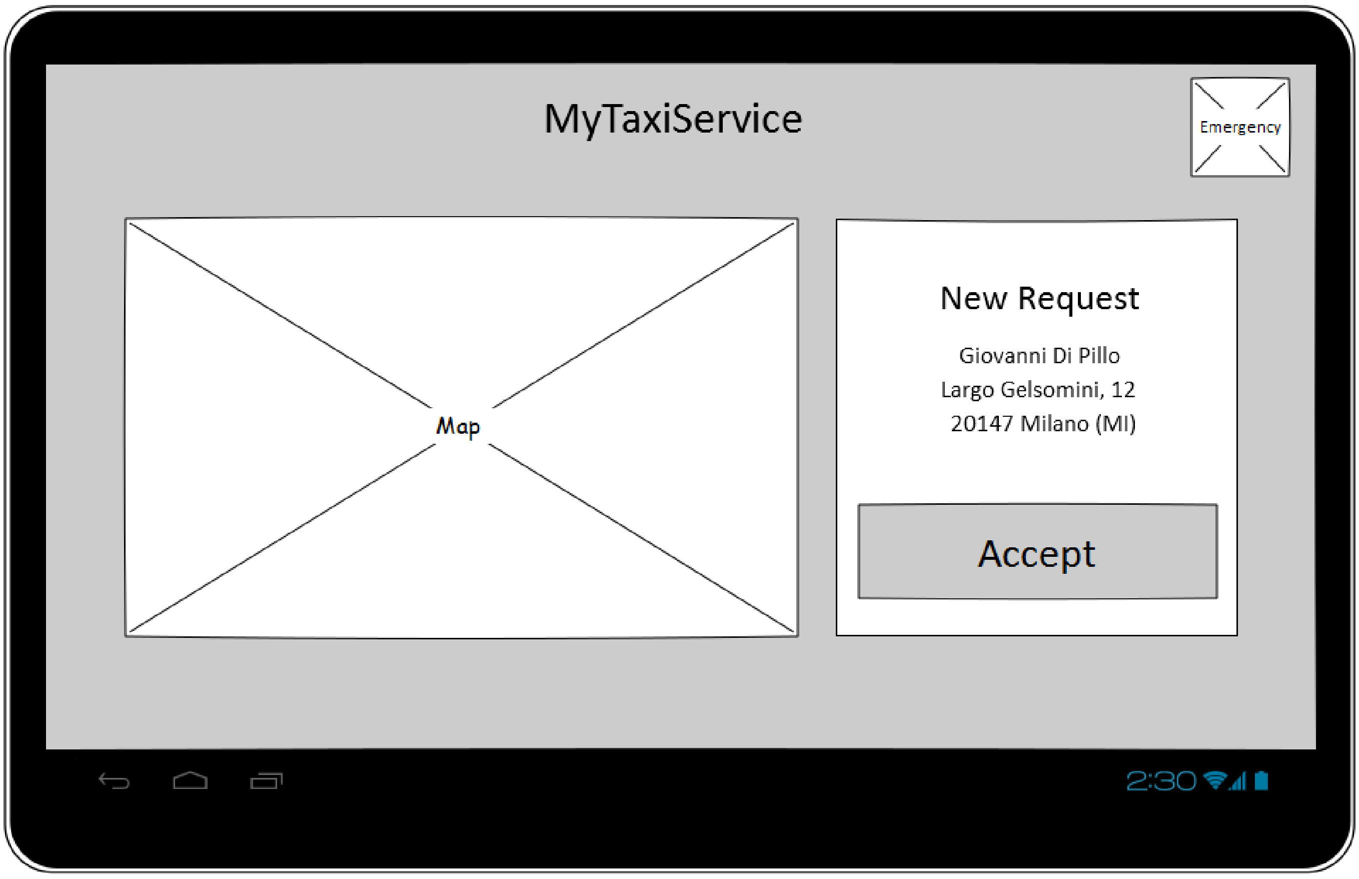
MyTaxiService

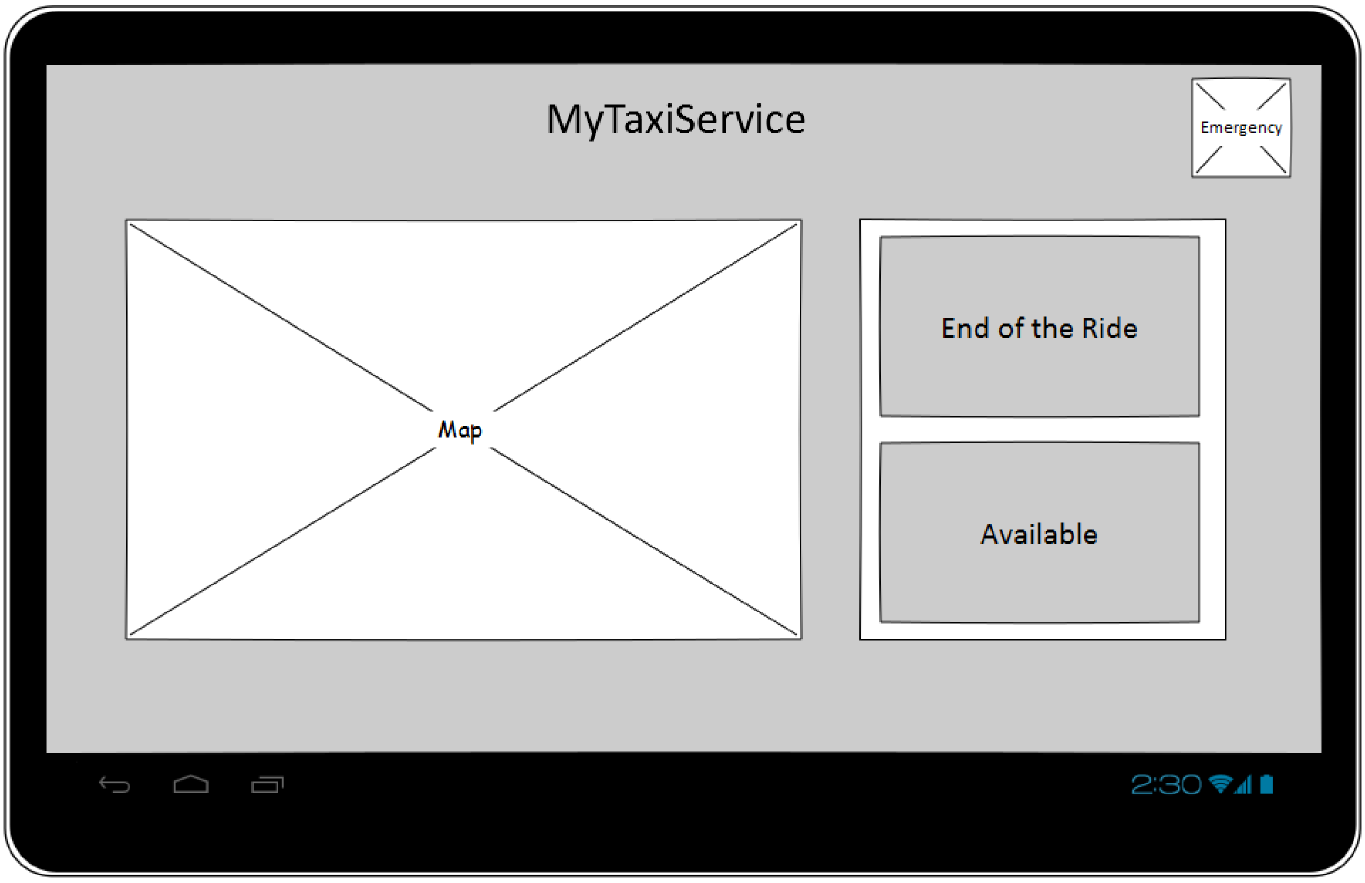
End Ride

A new Taxi has been sent to your position.

Please inform the passenger to stay still. The taxi will be there in 10 minutes.

The following mock-ups are an update of those presented in the RASD and, in addition, show the button used for the emergency functionality.





5 Requirements Traceability

This section links the requirements of the goals specified in the RASD to the component that satisfies it, also adding further information to better explain the link or the role of the component.

In order to increase the focus only on the service components, all the components that perform the communication are not mentioned here.

5.1 Goal 1

Requirement	Component	Further Information
The user must not be already registered	Registration Service	Same registration information cannot be used for another registration, at most the user must be provided with a way to recover his credentials
The username chosen by the user in the registration phase must be unique	Registration Service	
The user must confirm his email address in order to become a registered user	Registration Service	The system must be able to know that the email is under the user control, the method to implement this is up to the developer
The authentication data inserted must be correct in order to authenticate	Authentication Service	The authentication data provided by the user must correspond to the data stored in the system databases
If the user is not authenticated, he can only see the authentication or registration page	Mobile Interface & Web Interface	The app and the web will be developed in order to show all the functionality only after the user's authentication

5.2 Goal 2

Requirement	Component	Further Information
The user must be authenticated	Authentication Service	
The taxi driver must accept the forwarded request using his mobile app	Taxi Driver Interface	As the system structure, the Taxi driver can receive notification only by the app installed in his Device
The system must calculate the estimated time for the arrival of the taxi and inform the user	Request Service	The system knows the taxi location at least on the moment the taxi driver accepts the requests, so it can calculate a possible route and then an arrival time
The system must forward the request to an available taxi	Taxi Management Service	

5.3 Goal 3

Requirement	Component	Further Information
The system puts the taxis at the bottom of the queue of the area where they finish the rides	Taxi Management Service	
When a user makes a request, the system must send the nearest taxi available	Taxi Management Service & Request Service	Request service asks the Taxi management service for a taxi
If there isn't any available taxi when a user makes a request, the system must warn the user	Taxi Management Service & Request Service	For example in the case of a strike or an extremely busy time
A taxi driver can remove his availability and restore it whenever he wants, but that implies the loss of his priority in the queue	Taxi Management Service	The action can be performed by clicking on a dedicated button on the home page of the taxi driver interface

5.4 Goal 4

Requirement	Component	Further Information
The user must be authenticated	Authentication Service	
The system must dispatch a taxi for the reservation 10 minutes before the requested time	Reservation Service & Taxi Management service	
If something happens to the taxi while driving towards the user, the system must warn the user	Emergency Service	The taxi Driver must, through the emergency app page, warn the system and then the system will warn the User

5.5 Goal 5

Requirement	Component	Further Information
The users must be authenticated	Authentication Service	
In order to share a ride, the different users must be on the same route	Shared Ride Service	
The system must calculate the fee for every user, dividing the total fee and weighting it on the amount of kilometres done by each of them	Shared Ride Service	
There can't be more than 4 passengers in a taxi at the same time	Shared Ride Service	
If a shared ride is made for a single user, the system calculates the fee as a normal ride	Shared Ride Service	

5.6 Goal 6

Requirement	Component	Further Information
Allow the taxi driver to call the emergency number and a tow truck	Taxi Driver Interface	This Functionality is perform taking advantage of a Phone Calls app preinstalled in the device
Allow the taxi driver to cancel the current ride and, with the consent of the user, send a substitute taxi	Emergency Service & Request Service & Taxi Management Service	
Allow the taxi driver to call an operator	Taxi Driver Interface	This Functionality is perform taking advantage of a Phone Calls app preinstalled in the device

6 References

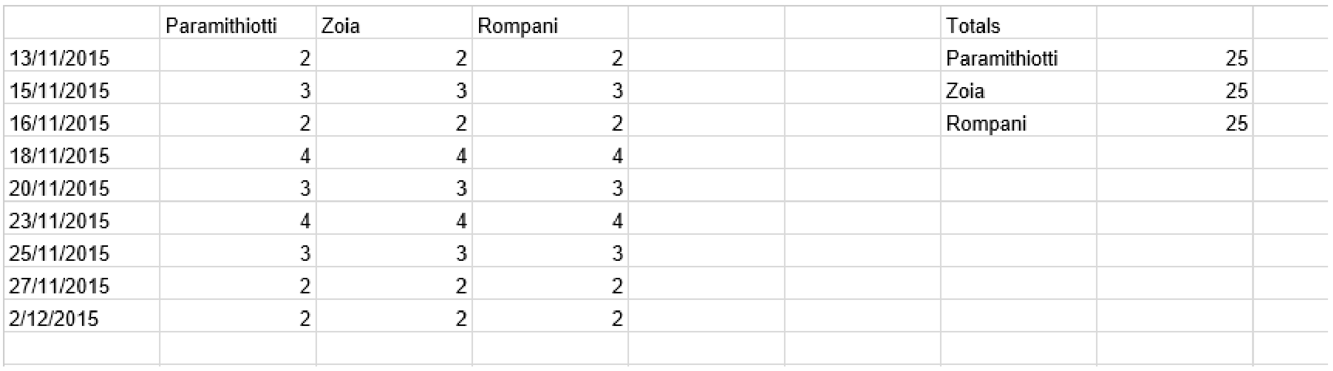
- IEEE Recommended Practice for Software Requirements Specifications - IEEE Std 830-1998
- IEEE Standard for Information Technology—Systems Design—Software Design Descriptions - IEEE Std 1016TM-2009
- Claus T. Jensen . SOA Design Principles For Dummies ® , IBM Limited Edition - John Wiley & Sons, Inc. 2013
- Wikipedia article about SOA architectural style: https://en.wikipedia.org/wiki/Service-oriented_architecture
- Slides regarding Java EE provided during the course
- Slides regarding Data Wharehouses of the Course "Sistemi Informativi"
- Java EE at a Glance - http://www.oracle.com/technetwork/java/javaee/overview/index.html

7 Revisions Table

Version	date	Modifications
1.0	2015/12/04	First version

8 Hours of Work

To avoid misunderstandings we always worked together and never done anything at home, so the hours of work are the same for every member of the group.



	Paramithiotti	Zoia	Rompani			Totals		
13/11/2015	2	2	2			Paramithiotti	25	
15/11/2015	3	3	3			Zoia	25	
16/11/2015	2	2	2			Rompani	25	
18/11/2015	4	4	4					
20/11/2015	3	3	3					
23/11/2015	4	4	4					
25/11/2015	3	3	3					
27/11/2015	2	2	2					
2/12/2015	2	2	2					