Taxi Service
Design Description

Version 2.0

Revision History

Date	Version	Description	Author
2012-11-06	0.1	Initial Draft	DSD staff
2012-11-08	0.2	Added component diagram	Leon Dragić
2012-11-08	0.3	Chapter 1 Draft	Jelena Jerat
2012-11-08	0.4	Chapter 2 Draft	Leon Dragić
2012-11-09	0.5	Chapter 3 Draft	Marko Coha
2012-11-09	0.6	Chapter 4 Draft	Igor Piljić
2012-11-09	0.7	Chapter 4 Draft	Karlo Zanki
2012-11-09	1.0	First Version finalized	Luca Zangari
2012-11-14	1.1	Updated domain model and protocols	Marko Coha
2013-01-20	1.3	Updated UI for Customer client and Implementation modules / components	Leon Dragić
2013-01-20	1.4	Updated UI for Taxi client	Marko Coha
2013-01-20	1.5	Chapter 4.6 Draft	Igor Piljić
2013-01-20	2.0	Updated domain and database model, and dataflow diagram	Karlo Zanki

Table of Contents

1. Introduction 4
- 1.1 Purpose of this document 4
- 1.2 Intended Audience 4
- 1.3 Scope 4
- 1.4 Definitions and acronyms 4
  - 1.4.1 Definitions 4
  - 1.4.2 Acronyms and abbreviations 4
- 1.5 References 4

2. Software architecture 5
- 2.1 Conceptual design 5
  - 2.1.1 Main Server 5
  - 2.1.2 Taxi Client 5
  - 2.1.3 Customer Client 6
- 2.2 System specification 6
- 2.3 External Components 6

3. External interfaces 6
- 3.1 Hardware Interfaces 6
- 3.2 Software Interfaces 6
- 3.3 Communication Interfaces 6
- 3.4 User Interfaces 7
  - 3.4.1 Customer client Android application 7
  - 3.4.2 Taxi client Android application 9

4. Detailed software design 11
- 4.1 Implementation modules / components 11
  - 4.1.1 Deployment 11
  - 4.1.2 Domain model 13
- 4.2 Data flow / Interactions / Dependencies 14
- 4.3 Data Types / Formats 14
- 4.4 Database Model 15
- 4.5 Protocols 15
- 4.6 Interface description 16
  - 4.6.1 Basic responses 16
  - 4.6.2 Taxi client requests 16
  - 4.6.3 Customer client requests 20
- 4.7 Interfaces to External Systems 21

1. Introduction

1.1 Purpose of this document

This document describes the system architecture and design of Taxi service system. It is initially defined at the beginning of the project, after collecting all necessary requirements and discussing the system architecture in general. However, it is intended to be constantly updated and revised, as new features are defined and implemented. The development process of this project is iterative and therefore the system architecture changes and develops continually. This document will always reflect the current state of the system architecture, and every delivered item (prototype or final product) will be based on this document.

1.2 Intended Audience

This document is intended to be used by all team members during the system development. It will also be used by the project supervisor to get the insight into the project work. This is a technical document and it is not initially intended for the customers. However, technically educated customers may also get the general overview of the project using this document. Finally, this document is intended for the developers who will be able to get the detailed description of the system architecture in order to continue working on this project in future.

1.3 Scope

The Taxi Service system is divided into three main components: Taxi Mobile Application, Customer Mobile Application and Server. This document will describe interfaces between components, as well as the interfaces to external components and users. It will also describe every component separately.

1.4 Definitions and acronyms

1.4.1 Definitions

Keyword	Definitions
TaxiService	Project name

1.4.2 Acronyms and abbreviations

Acronym or abbreviation	Definitions
MVC	Model-View-Controller design pattern
HTTP	HyperText Transfer Protocol
HTML	HyperText Markup Language
ORM	Object Relational Mapping
API	Application Programming Interface
JSON	JavaScript Object Notation

1.5 References

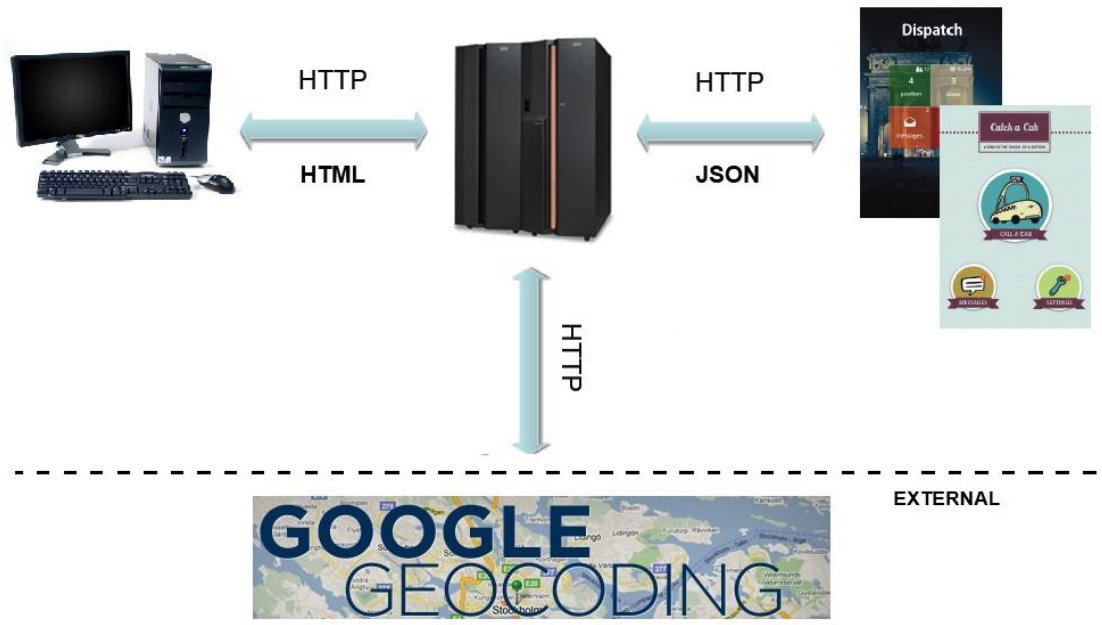
Project homepage: http://www.fer.unizg.hr/rasip/dsd/projects/taxi_service

2. Software architecture

2.1 Conceptual design

The system consists of 3 main components: Taxi client application, Main server and Customer client application. Main server provides the core system functionality. Because of the specific system requirements (Android clients and web users), the main server is built using MVC design pattern. MVC enables us to easily transform the output of methods, meaning it is possible to return either HTML, or JSON string to the users, depending on the request they made.

System Design



The main server and the database are deployed on the cloud which has many real benefits. It is easily configurable, no need for maintenance, it increases the availability of the service and gives greater freedom for the users cause data can be accessed from almost everywhere.

The application for taxis will be deployed on any Android device with Android 4.0 OS or greater and the application for customers will be deployed on any Android device with Android 2.3.3 OS or greater.

2.1.1 Main Server

Main server is the central part of the system. It provides the core functionality of the system. Its main purpose is receiving requests from clients, processing them, and sending results of processes to the client. The main resource that server needs is connection with Google Geocoding API which is an external component of the system and the connection with the database. Users of the server are taxi client, customer client and web client.

2.1.2 Taxi Client

Taxi client is making specific request to the main server in order to establish functionality of the client that is needed for taxi driver to perform his work. The taxi client gets all the information from the server. The main resource that taxi client needs is the Android device with GPS chip which provides the current location of the taxi. Users of taxi client are taxi drivers.

2.1.3 Customer Client

Customer client provides the functionality for ordering a taxi at the current position of the device. The client sends the request to the server. The main resource that customer client needs is the Android device with GPS chip which provides the current location the customer. The users are people who install the Android application on their Android device.

2.2 System specification

Server is hosted on the Windows Azure cloud. It is developed in .NET framework, using MVC design pattern.
Taxi client application is hosted on Android device with OS 4.0 or greater. It is developed in Java.
Customer client application is hosted on Android device with OS 2.3.3 or greater. It is developed in Java.
The MSSQL database is used. Entity framework is used as ORM.

2.3 External Components

The system is using Google maps for showing the specific positions on the map.

3. External interfaces

3.1 Hardware Interfaces

One of the main functions of this product is taxi tracking, i.e. using GPS to locate a taxi in any given time. Each of the taxi client applications must, therefore, be installed on an Android device with a built-in GPS. The other client application, the one used by customers, will also have to access GPS data in order to provide an accurate location of the customer ordering a taxi. If it would be required by other features, a GPS could be used for calculating the velocity and similar parameters of a taxi vehicle. The administrator application will be developed as a web application and will not require any specific hardware.

3.2 Software Interfaces

The server part of the system will provide a REST web service for client applications to use. The service will provide an interface for the core functionalities of the product, such as registering a taxi in the database, tracking a location of a taxi, taking orders from clients, etc. The interface is described in detail in the Protocols section below. It will not be open to public, but such an API could be developed in the future.

3.3 Communication Interfaces

Each of the client applications will have to be connected to a 3G network at all times, in order to be able to send request to the REST service and receive results. All of the data will be sent over a 3G network. The customer client app will place orders and receive confirmations from the service, and the taxi client app will receive orders and respond to them via a 3G network.

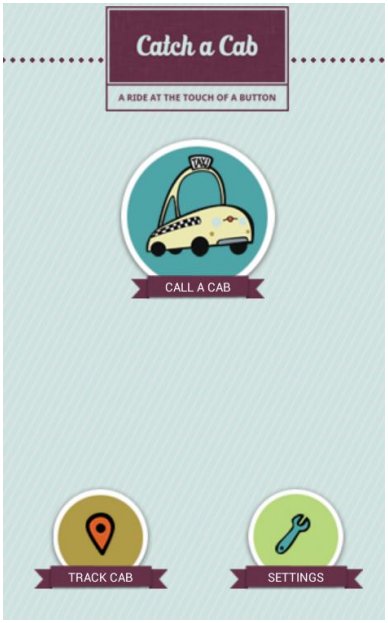
3.4 User Interfaces

3.4.1 Customer client Android application

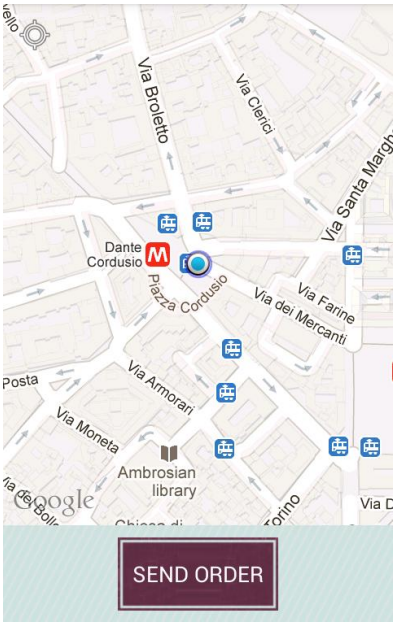
The Android application for customers “Catch a Cab” consists of 4 main parts:

- **Dashboard**
- **Call a Cab**
- **Track Cab**
- **Settings**
-

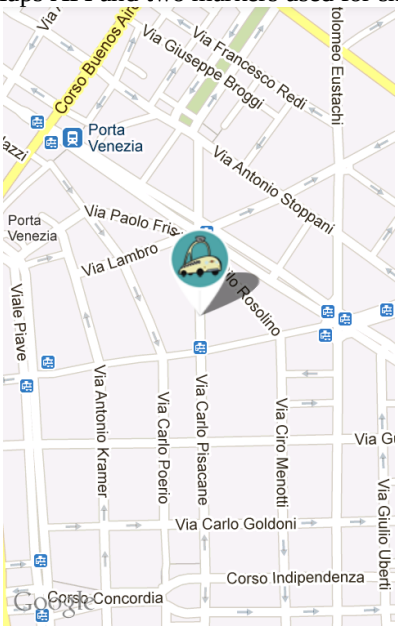
The dashboard is the starting activity for “Catch a Cab” application. It consists of three buttons. The center button is used to call a taxi, the lower left button to track the taxi after the order has been made, and the lower right button to open the settings panel.



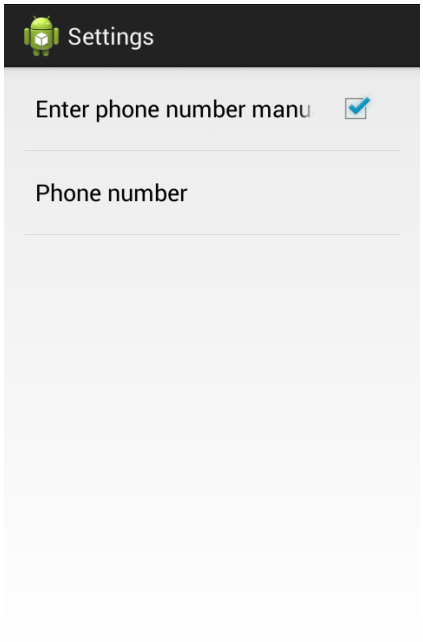
The Call a Cab activity is the main activity for „Catch a Cab“ application. It provides the core system functionality – Ordering a Cab. It consists of map which is provided by Google Maps API and button for sending the order to the main server.



The Track Cab activity is used for tracking the cab which is selected to pick the customer up. The UI consists of a map which is provided by Google Maps API and two markers used for showing the pickup and taxi position.

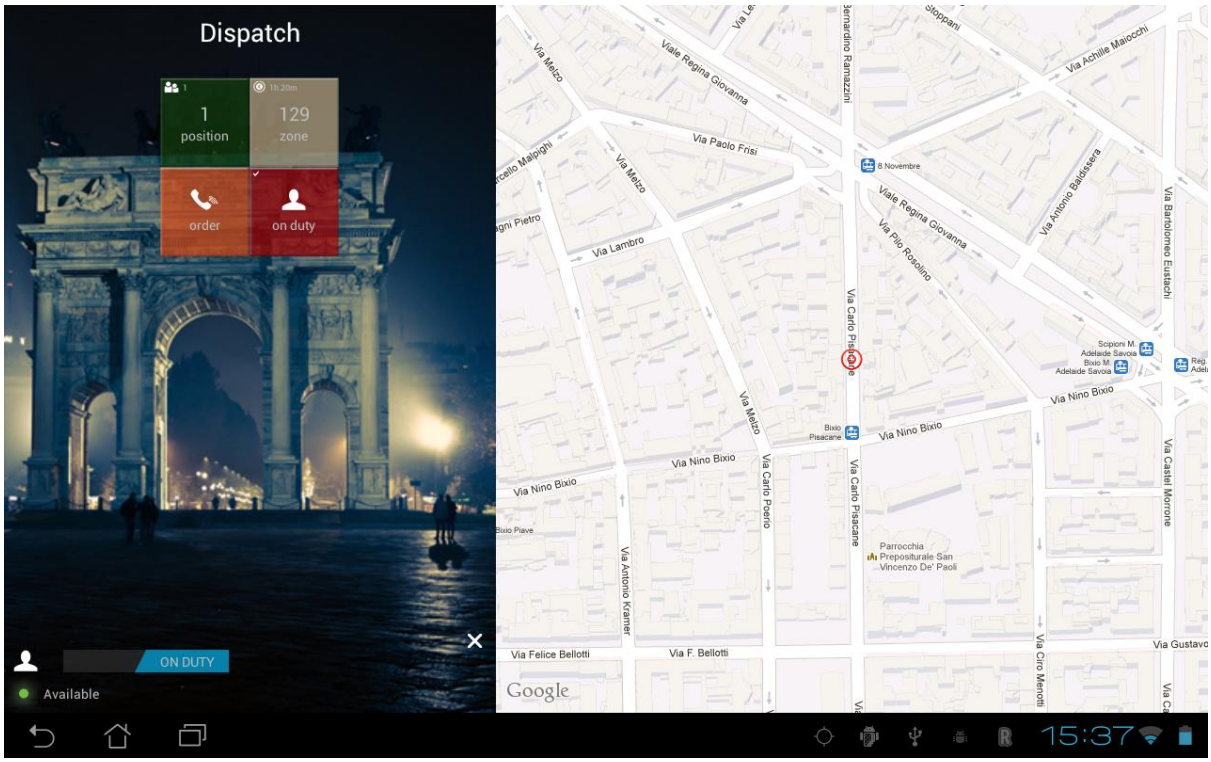


The Settings activity is used for selecting the preferences.

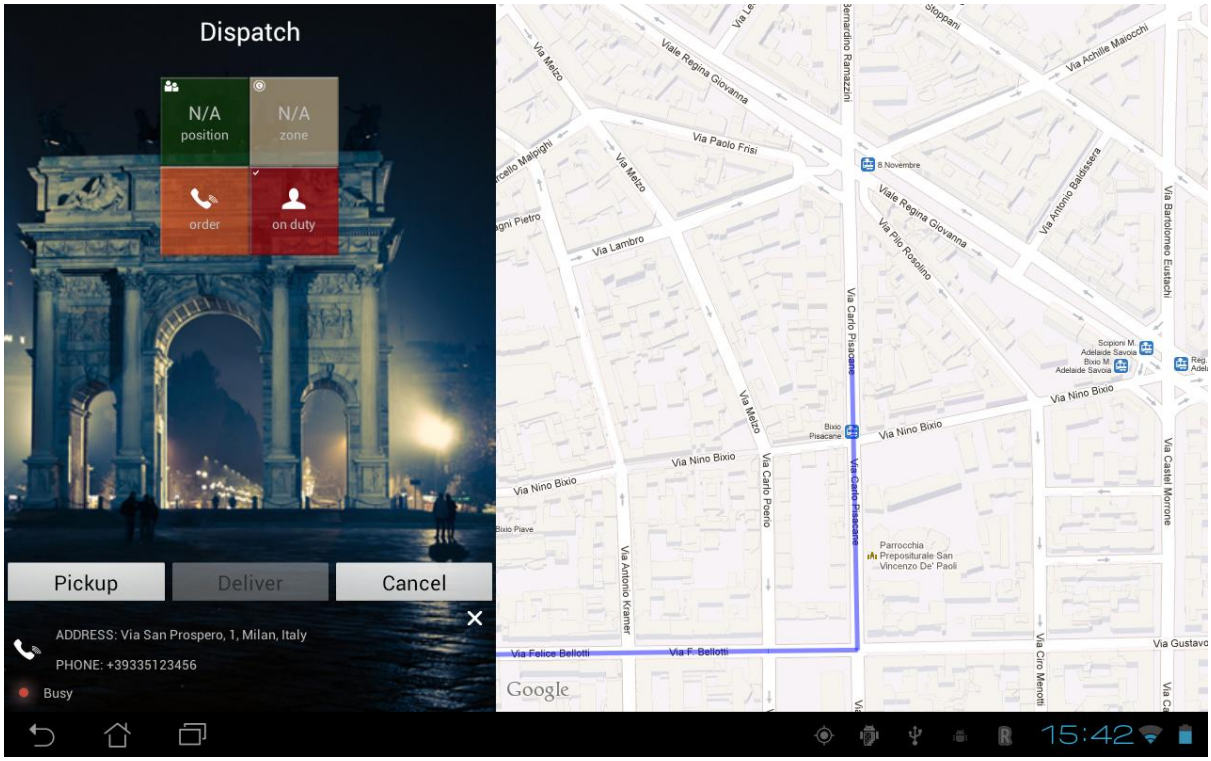


3.4.2 Taxi client Android application

The Taxi client application consists of one screen which is divided into two parts. On the left side of the screen is the main dashboard and next to it, on the right side, is the Google map view.
The dashboard for the taxi client application consists of four main buttons. The first button displays the number of taxis in the queue and the position of the taxi in the queue. The second button is used to give the user information about the zone he is currently in. The order button is used for managing received orders, such as picking up a customer and delivering the customer to a location. The last button is a status button - it shows the user's current status, which can, for example, be on duty or off duty, and a checkmark showing the status of the connection between the user's Android device and the server. The lower part of the dashboard contains a switch which is used for changing the taxi status, and an icon which describes the current taxi status. The right part of the screen contains a map which shows the taxi's current location.



When an order is active, the dashboard part is updated with the customer's address and phone number, and options are shown for interaction with the customer. The map also shows the nearest available route to the customer's address.



4. Detailed software design

4.1 Implementation modules / components

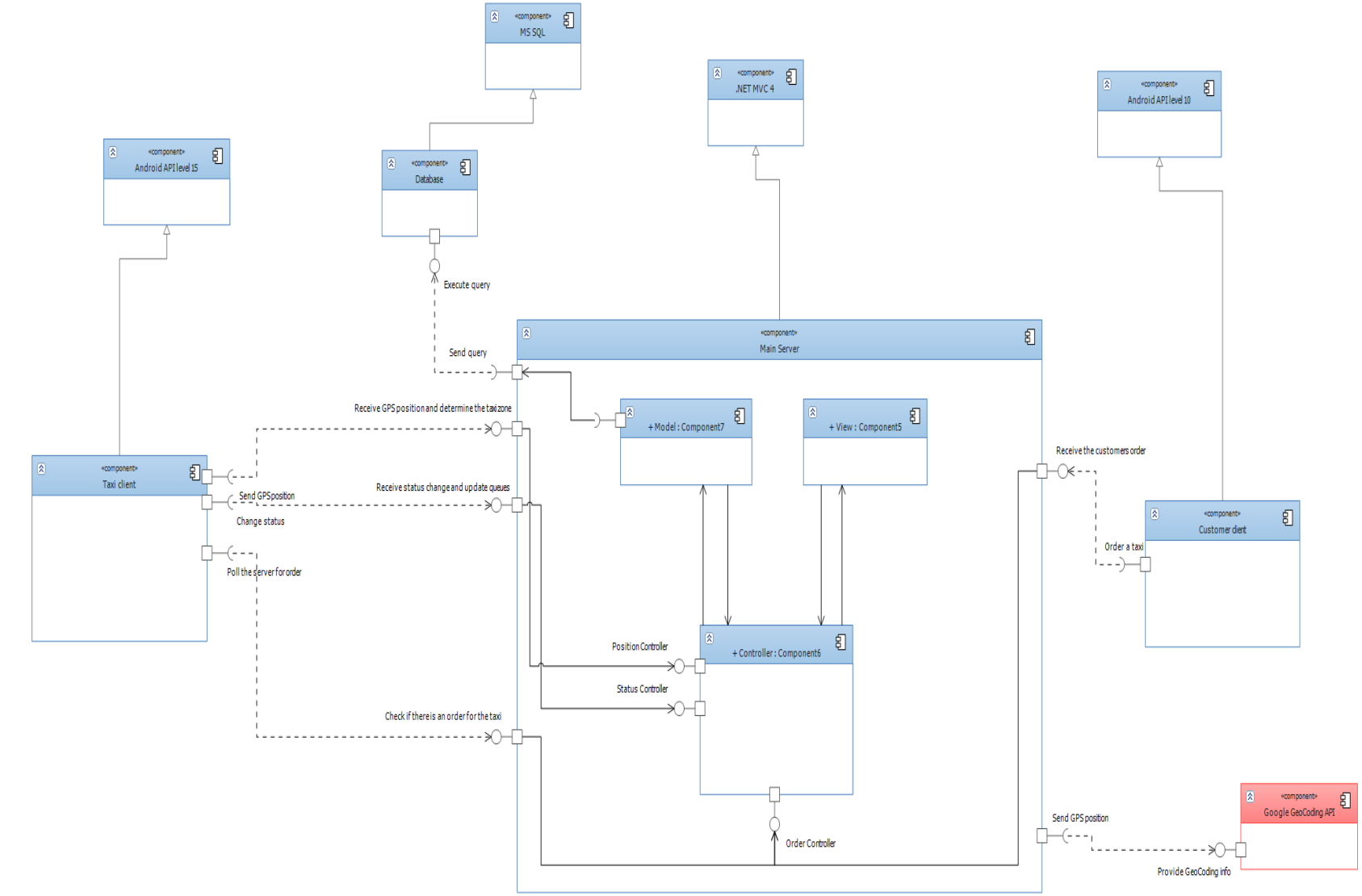
Product contains of three main modules, Customer client application “Catch a Cab”, Taxi driver’s client application “Dispatch” and a main server which coordinates the communication between these two components and merges them into a complete and unique system. Taxi driver’s application tracks taxi movement, updates taxi’s location and enables taxi driver to get information about potential pickup requests. Customer’s application allows customers to order a taxi and track the taxi selected for pickup. Main server collects and handles the data received from Taxi driver’s application and Customers application. It manages virtual queues and forwards order requests received from customers to the taxi drivers. Main server also provides a web application for customers. The web application has the same features as the Android client application for customers “Catch a Cab”. It is used for ordering and tracking the taxi.

This kind of deployment is natural since the product contains of two parts, one that is going to be used by the customers and the other that is going to be used by the taxi drivers. Third part, the Main Server provides a way to integrate these two modules.

4.1.1 Deployment

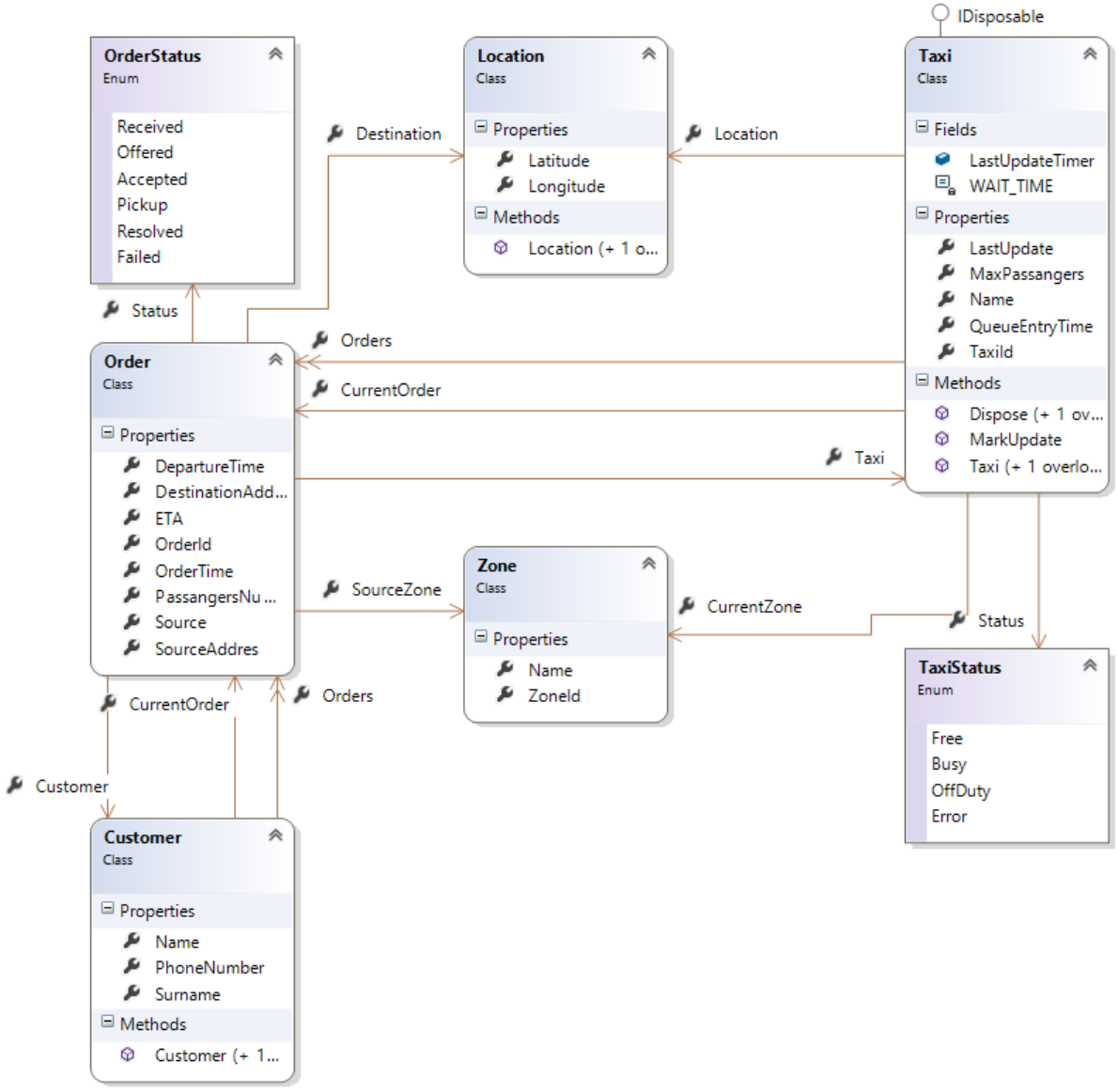
Taxi driver’s application can be deployed on any Android device with API 15 or higher placed inside of the taxi. Customer application can be deployed on any Android device that supports API 10 or higher. The main server is deployed on Windows Azure cloud same as the database.

cmp TaxiService



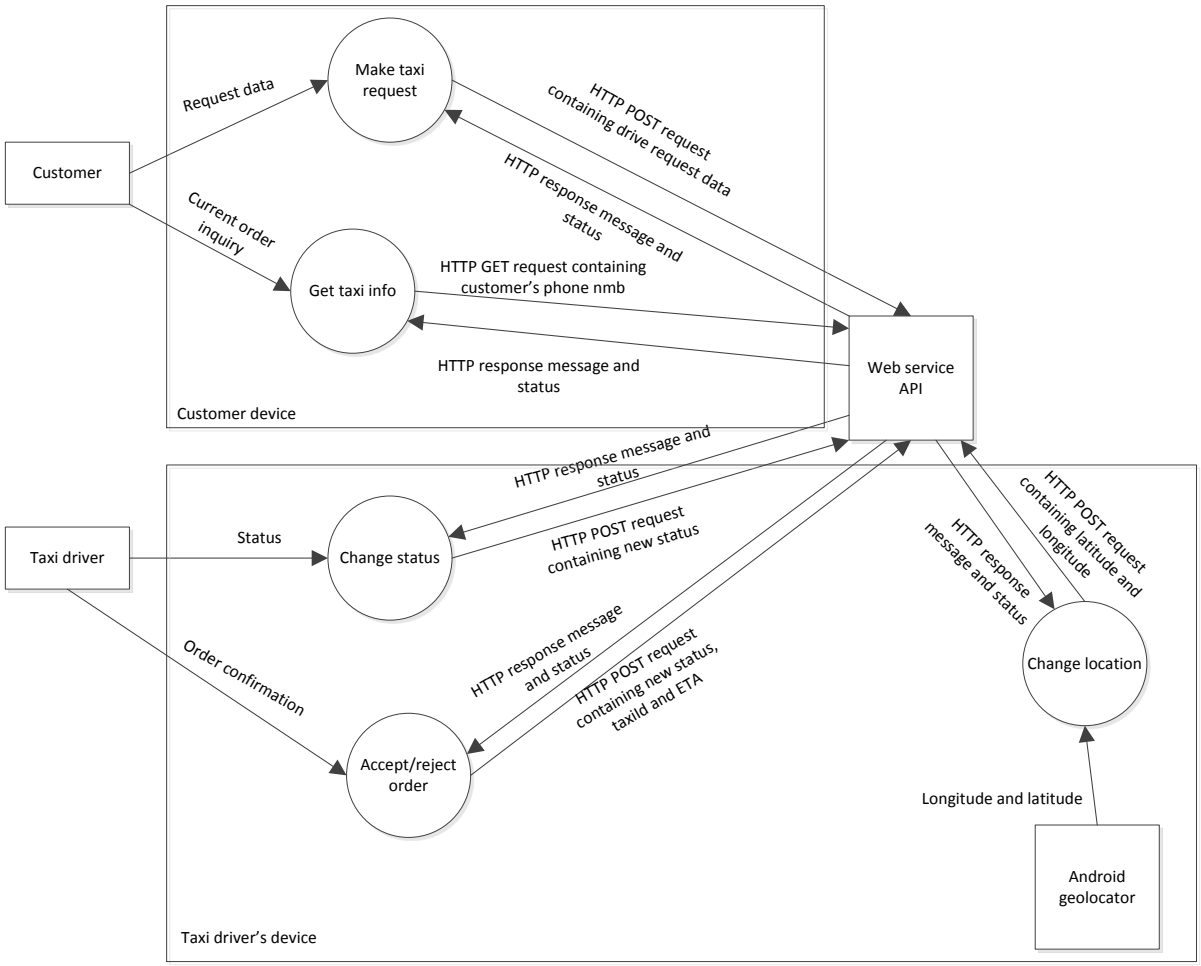
4.1.2 Domain model

Domain model describes core concepts of the product. Current domain model represents concepts needed for the final product, and it represents the objects and logical constructs that exist in the real world. In the domain model we have an object named Taxi which is representation of a real taxi. It contains information about taxis location, orders, passenger capacity and taxis Id and current status. It also contains information about time at which taxi entered the specific queue, and this information is used for the queue management. Last update timer field enables tracking taxi status, after the timer expires, taxi automatically changes its status to "off duty".
Location object represents geographical location described with longitude and latitude. We also have object named Customer which represents a customer from the real world, it stores customer's phone number and some personal information like name and surname.
Order object models customer's order for a taxi drive. It contains the source and destination location. It also has information about order time, departure time and estimated time of arrival. Destination address and source address are equivalent to the real location described with the latitude and longitude. It also contains information about the customer who made the order and the taxi which received the order.
Zone object represent zones in which taxis are placed, each zone has one virtual query in which taxis are placed and which is managed by our system. Also there are two enumerations defined, one describing possible status of the order, and the second one describing the status of the taxi.



4.2 Data flow / Interactions / Dependencies

Data inside of the system is being exchanged between the main three components, Customer application, Web service and Taxi driver's application. Taxi driver's application and Customer application communicate with the web service in the same way, by making HTTP REST requests. Each HTTP POST request contains a JSON string in its body, which is used to pass data associated with the request to the server. The communication is always one-way, from client applications to the web service which then sends back a HTTP response with a JSON string in its body containing success status and a short description.



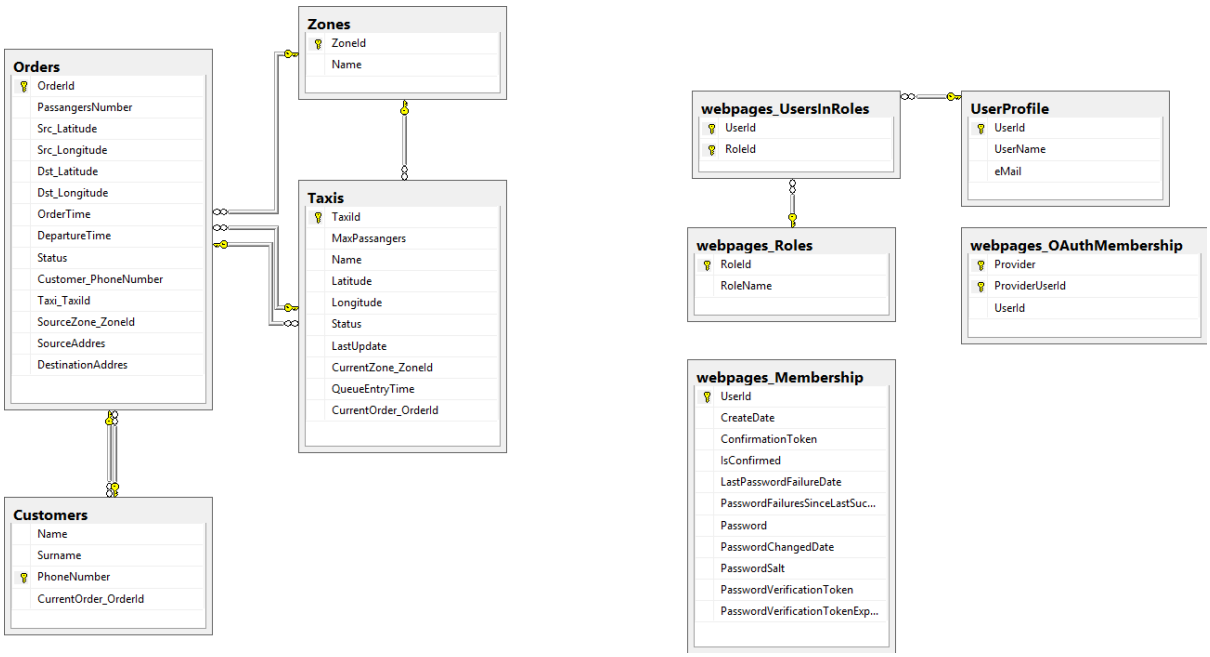
Dependency between different parts of the system is reduced by simple extraction of the interfaces. All interaction between the components is pointed towards the interface instead to the concrete implementation. This way the coupling is reduced, because it doesn't matter how the component is implemented as long as it implements the interface. This also enables separate testing of different system modules because data received through interaction can be mocked.

4.3 Data Types / Formats

Whole documentation will be written in both, .doc and .pdf formats.
For communication between server and clients team has decided to use JSON format that will be part of HTTP requests and responses. Examples of some JSON templates that were made for this communication are given in section 4.5.

4.4 Database Model

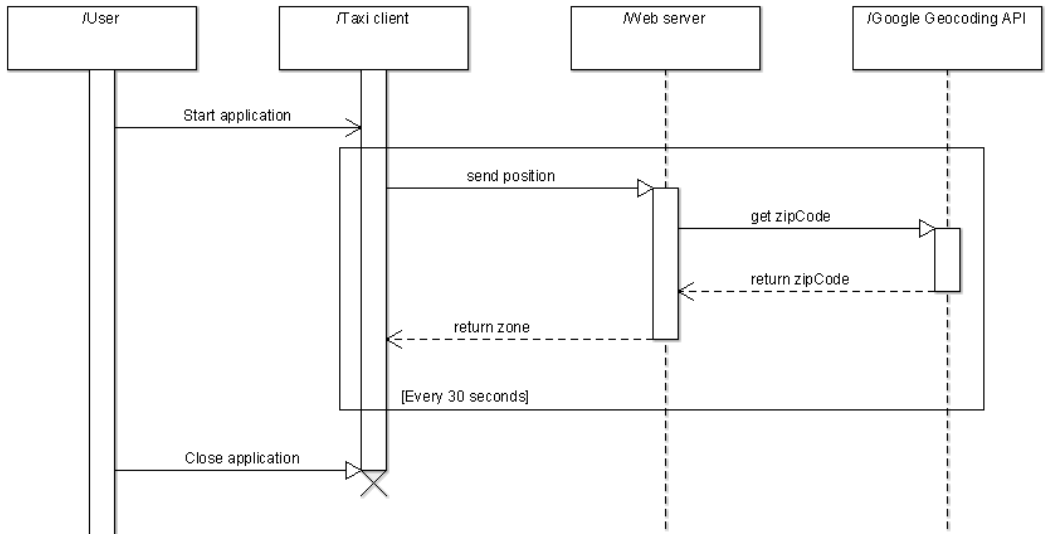
Database diagram represents the final version of the database and it visualizes all entities that are persisted in the database. Orders, Zones, Taxis and Customers tables are used for storing information from domain model, and the data that is being stored in this table is equivalent to the data described in the chapter 4.1.2.
Other tables visible on the database diagram are used for storing the data used for web pages membership provider and are automatically generated.



4.5 Protocols

HTTP is used for communication between server and clients. Every HTTP POST request has JSON string in its body and response from server is JSON string as well. Examples of JSON messages and detailed description of communication can be found in figure 4.6.

Sequential diagram that describes implementation of first two features (taxi sends its position, server determines zone) is given in the picture below.



4.6 Interface description

In this figure all messages that can be sent in communication between server and clients will be described in details.

Following conventions are used, unless differently specified:

- All URLs are in lowercase.
- Parameter's names should be in lowercase or in camelCase starting with a lowercase letter.
- Values are alphanumeric strings. They don't have a fixed capitalization definition. However, some parameters require a capital case value (i.e. OK/ERROR responses).

4.6.1 Basic responses

Following responses can be used by the server in general cases, when no other response has been defined yet.

4.6.1.1 OK

This response should be used after a successful request and when no other parameters have to be passed back.

Response parameters	Mandatory	Description / possible values
status	Yes	The outcome of the request. The only possible value is OK

Example:

```
{
        "status":"OK"
}
```

4.6.1.2 ERROR

This response should be used after a wrong request (i.e. badly formatted request, missing some parameters, wrong parameters format, etc).

Response parameters	Mandatory	Description / possible values
status	Yes	The outcome of the request. The only possible value is ERROR
description	Yes	The description of the error, if it's possible. An empty value is allowed, but discouraged. Whenever possible, define specific errors for a call.

Example

```
{
        "status": "ERROR",
        "description": "Some description"
}
```

4.6.2 Taxi client requests

4.6.2.1 Location update

This request is performed by the taxi client to update its position to the server when the taxi is free or on duty.

URL: /api/taxis/***idtaxi***/location
URL parameters:

- ***idtaxi***: it is the ID of the taxi. It is part of the URL

Request parameters	Mandatory	Description / possible values
latitude	Yes	String representing the latitude part of the position. I.e. 1.2
longitude	Yes	String representing the latitude part of the position. I.e. 3.6

In case of a positive response:

Response parameters	Mandatory	Description / possible values
status	Yes	The outcome of the request. The only possible value is OK
zoneId	Yes	String containing the zip code (or any other id) defining the zone
zoneName	Yes	String containing a friendly name of the zone
position	Yes	String containing the position of the taxi in the queue of that zone
queueLength	Yes	String containing the total number of taxis in the queue of the zone the taxi is currently in
order		Object containing information if there's a call for the taxi. It is defined below.

Format of the order object:

Fields	Mandatory	Description / possible values
orderId	Yes	The order ID
srcLatitude	Yes	String representing the latitude component of the starting position. I.e. 1.2
srcLongitude	Yes	String representing the longitude component of the starting position. I.e. 2.4
srcStreetAddress		String representing the starting address, in the form of street name, street number. It is assumed the starting point is in Milano
destination		String representing the destination address, in the form of street name, street number. It is assumed the starting point is in Milano
numPassengers	Yes	String representing the number of passengers to be carried
departureTime		String representing the time the taxi should be serving the client. The format is 2012-11-19T14:42:48.373
phoneNumber	Yes	The customer's phone number
name	Yes	The customer's name

In case of a negative response

Response parameters	Mandatory	Description / possible values
status	Yes	The outcome of the request. The only possible value is ERROR
description	Yes	String containing the description of the error. Some standard values have been defined: "Out of Milan"

Description values:

- "Out of Milan": when the taxi is out of the area of Milan

Other information:
This request should be repeated every 30 sec.
If there is an order for the taxi, the details are specified in the "order" field. Otherwise, the "order" field is not present.

When the taxi goes off duty, it stops sending the location.

Request example:
URL: /api/taxis/***123456789***/location

{
"Longitude":"9.2131949",
"Latitude":"45.5203422"
}

Response example:
{
"zoneId":"20126",
"zoneName":"126",
"position":1,
"queueLength":1,
"order":null,
"status":"ok"
}
Or:
{
"status": "ERROR"
"description":" Out of Milan"
}

4.6.2.2 Order confirmation

When a taxi driver receives an order, he has to confirm it, accepting or rejecting it. If the order is accepted, then the taxi driver has to update the status of the order.

URL: /api/orders/***orderId***/
URL parameters:

- ***orderId***: it is the order ID. It is part of the URL.

Request parameters	Mandatory	Description / possible values
status	Yes	String representing the taxi drivers decision. Possible values are: - accept: when the taxi driver accepts the order - reject: otherwise
TaxiId	Yes	The taxi ID
eta		String representing an estimation of the arrival time. It can be communicated back to the customer.

Response format: OK/ERROR

Other information:
If the taxi accepts the order, the server puts the taxi to busy, there is no need to perform a status update call. If the taxi driver rejects the order, the order is passed to the next available taxi.
After the taxi accepts the order, it has to inform the server about the evolution of the request performing the following calls.

4.6.2.3 Order pickup

When a taxi driver reaches the customers and picks him up, then he has to inform the server

URL: /api/orders/***orderId***/
URL parameters:

- ***orderId***: it is the order ID. It is part of the URL.

Request parameters	Mandatory	Description / possible values
status	Yes	The only value is “pickup”

Response format: OK/ERROR

Other information:
After taxi notifies server that customer is picked, customer is no longer available to track his order.

4.6.2.4 Order delivered

When a taxi driver reaches the destination and the clients leaves the taxi, the taxi has to inform the server about the completed operation.

URL: /api/orders/***orderId***/
URL parameters:

- ***orderId***: it is the order ID. It is part of the URL.
-

Request parameters	Mandatory	Description / possible values
status	Yes	The only value is “delivered”

Response format: OK/ERROR

Other information:
The server will set the taxi status as free and puts the taxi at the end of the queue. The taxi client can perform a location call to know its position in the queue.

4.6.2.5 Order cancel

When a taxi driver confirms an order, but then, for any reason, is not able to complete it, then he’s to inform the server to cancel the order.

URL: /api/orders/***orderId***/
URL parameters:

- ***orderId***: it is the order ID. It is part of the URL.

Request parameters	Mandatory	Description / possible values
status	Yes	The only value is “cancel”

Response format: OK/ERROR

Other information:
The server will set the taxi status as free and puts the taxi at the end of the queue. The taxi client can perform a location call to know its position in the queue.

4.6.2.6 Status update

This request is performed by the taxi client to inform the server about its state.

URL: /api/taxis/**idtaxi**/status
URL parameters:

- ***idtaxi***: it is the ID of the taxi. It is part of the URL.

Request parameters	Mandatory	Description / possible values
status	Yes	String representing the status of the taxi. Possible values are: - free - busy - off duty

Response format: OK/ERROR

Other information:
When the taxi ends its shift, it should send an "off duty" status to tell the system the taxi driver is no longer available.

Request example:
URL: /api/taxis/***123456789***/status

{
"status":"free"
}

Response example:
{
"status": "OK"
}

4.6.3 Customer client requests

4.6.3.1 Making Order

This request is performed by the customer client to request a taxi at a specific place to go to a specific destination. An optional time can be specified in case the client desires to book a taxi for a different time.

URL: /api/Orders

Request parameters	Mandatory	Description / possible values
srcLatitude	Yes	String representing the latitude component of the starting position. I.e. 1.2
srcLongitude	Yes	String representing the longitude component of the starting position. I.e. 2.4
dstLatitude		String representing the latitude component of the ending position. I.e. 1.2
dstLongitude		String representing the longitude component of the ending position. I.e. 1.2
departureTime		Time od departure
numPassengers	Yes	String representing the number of passengers to be carried
PhoneNumber	Yes	Customer's phone number

In case of a positive response:

Response parameters	Mandatory	Description / possible values
status	Yes	The outcome of the request. The only possible value is OK
address	Yes	String containing the resolved address where the taxi will be sent to.

In case of negative response, the format of the ERROR response is followed.

Request example:
URL: /api/Orders

{
"PhoneNumber":"123456789",
"NumPassengers":"3",
"SrcLongitude":"9.2131949",
"SrcLatitude":"45.5203422"
}

Response example:
{
"adress":"Viale Piero e Alberto Pirelli (Viale), 20126 Milan, Italy",
"status":"ok"
}

4.6.3.2 Customer inquiry about current order

Customer sends request to following URL: /api/Orders/id (id is phoneNumber of customer)

In case of positive response:

Request parameters	Mandatory	Description / possible values
status		ok
srcLongitude		Position of customer who ordered taxi - Longitude
srcLatitude		Position of customer who ordered taxi - Latitude
orderTime		Time when order was received
taxiLongitude		Position of the taxi assigned to the order - Longitude
taxiLatitude		Position of the taxi assigned to the order - Latitude
taxiName		Name of the taxi assigned to the order
orderStatus		Status of customer's pending order

In case of negative response: ERROR (along with description)

Response example:
{
"srcLongitude":15.973280946941296,
"srcLatitude":45.800669876235006,
"taxiLongitude":15.973280946941296,
"taxiLatitude":45.800669876235006,
"orderTime":"2012-12-11T00:36:22.19",
"taxiName":"koko",
"orderStatus":"Received",
"status":"ok"
}

4.7 Interfaces to External Systems

There are two external systems that our system will be interfacing to. These are:

- Google Maps
- Global Positioning System (GPS)

Maps will be embedded into client applications by using Google Maps API and JavaScript.
The classes of the Maps library offer built-in downloading, rendering, and caching of Maps tiles, as well as a variety of display options and controls.

GPS will be used to retrieve the current user location (in the mobile application). Using this location and the Google Maps API, we can localize the taxi position and calculate in which zone it is. Most mobile phones nowadays are equipped to use GPS. We can access the GPS features using the *android.location* package in the Android API.