My Taxi Service

Design Document

Politecnico di Milano **A.A. 2015-2016**
Software Engineering 2 Raffaela Mirandola

Salvatore Perillo 853349 Claudio Sesto 841623

Table of content

1. Introduction

1.1 Purpose

This document aims to help all those who are in charge to develop the software and eventually those programmers who will have to make updates.
This document contains the architectural design of the system (including hardware and software views, runtime behavior) some algorithm and user interfaces as guideline for the developement of the system.

1.2 Scope

The main software that will be developed is “myTaxiService”; the aim of this software is to allow people to use better the taxi service already existing in the city both via web application or via smartphone.
The user access to the web server via browser or simple smartphone app; from there he can send a request that is forwarded to the application server that works in order to find a taxi in the zone from which the request was sent.
At the same time there are teams of technicians working on new features on a virtual machine that emulates the system.

1.3 Definitions

- Web Server: is the one that anyone can access to use the application myTaxiService and contains all functionalities that let the user to send the requests
- Application Server: is the one with a rule engine implemented and manages the requests and the dynamic queues saved in the “Dynamic Database”
- Virtual machine: is the virtual machine which technicians uses to work on the new features
- Dynamic Database: is the database which contains and manages queues and requests
- “Request”: we consider both simple request and reservation
- Static Database: is the one that stores all informations about taxi driver, taxi, technicians and registered use
- Q/R manager: Queue/Request manager

1.4 Reference document

- RASD document
- www.uml-diagrams.org

1.5 Document Structure

This document will contain a general description about how we thought the system can be implemented; next chapters will be about:

- Architectural design: so how different components of our system (both hardware and software) interact between them
- Algorithm design: some suggestions about how implement the most relevant algorithms of the system
- User interface design: is the graphic interface of our system
- Requirements traceability: this section will link the RASD document with this document in order to have an easy view about requirements definition-implementation

2. Architectural design

2.1 Overview

In this section we explain the architecture of our system focusing on how different subsystems interact between them; we use different UML diagrams to simplify the description.

The architecture, components and behavior described in this section are not considered mandatory but as a simple guideline for the developers.

2.2 High level components and their interaction

Here are listed some of the high level components:

- Web application
- Technician workstation
- GPS
- Q/R manager
- Database (static/dynamic)

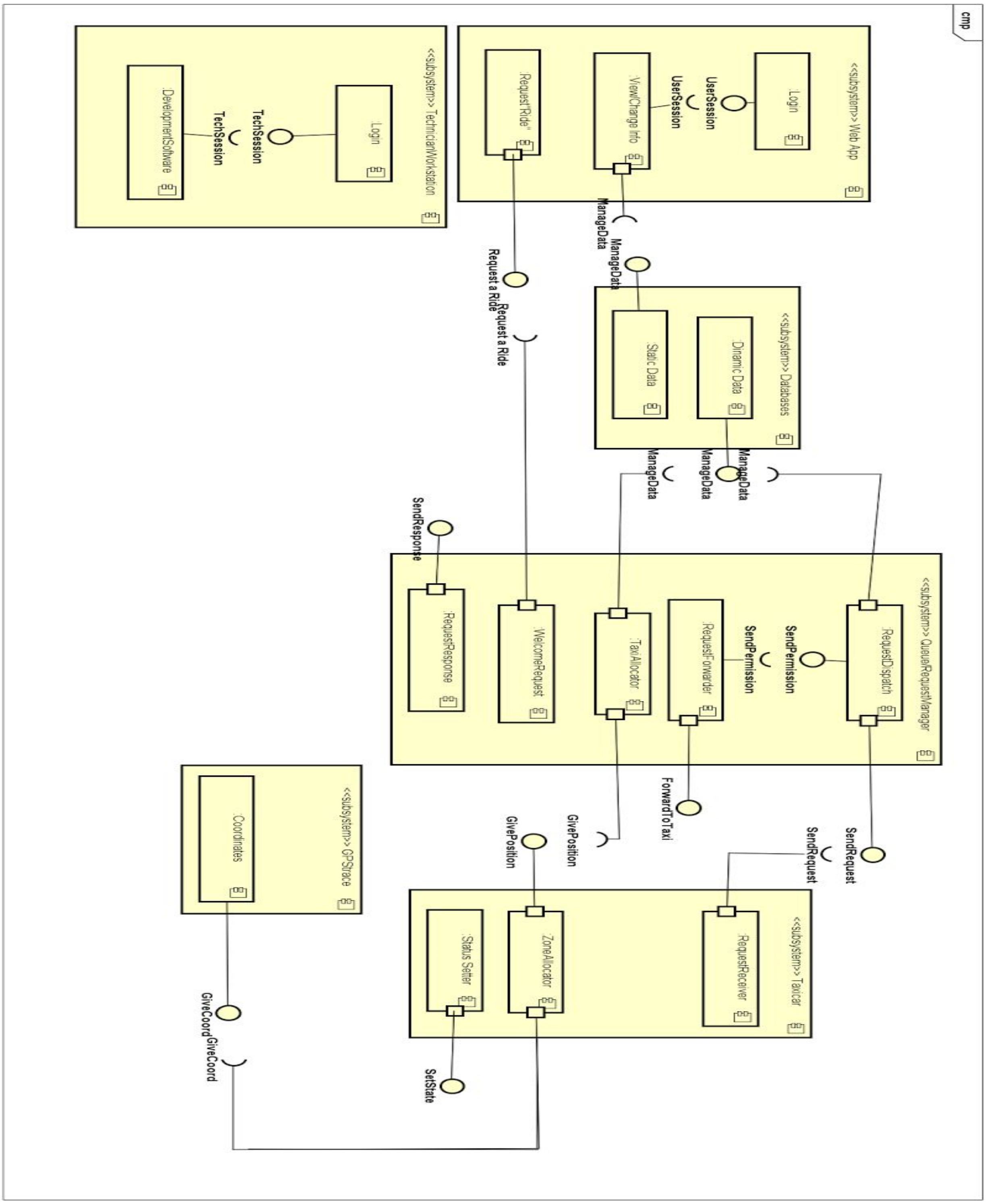
Simply, the web application is responsible of the registration/update profile of users (both clients, taxi drivers and technicians) and also forwards the requests to the Q/R manager.

This last one is a piece of software that takes informations of the Dynamic database and allocates the taxi for the client basing on which zone the taxi is; this information is given by the GPS installed inside the car.

The technician workstation is an independent part that uses a Virtual Machine that simulate the system in order to develop new features and test them easily

2.3 Component view

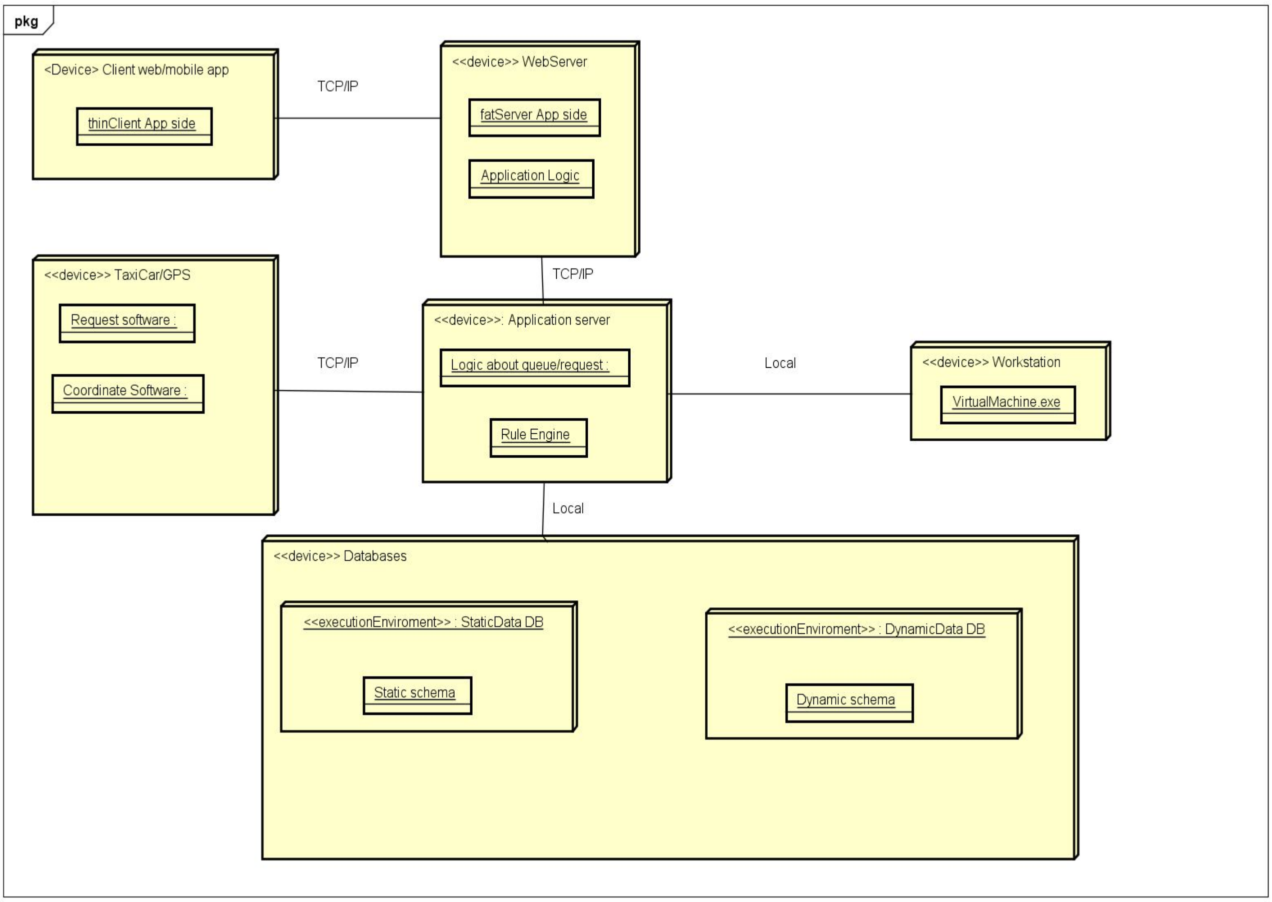
Here we are describing some basic components with interfaces that links the components one to each other.



This diagram shows the major components of our software that are connected with interfaces. Some of them provide an interface so that others can use it to communicate with the desired one: more details on interfaces are in section 2.6.
The databases, as we ‘ve mentioned, contains static data and dynamic data: these are respectively modified by the web application (whenever a new registration is performed) and by the Q/R manager (whenever a taxi accept/refuse a request or the position of them is updated).
The GPS tracks the position of the taxi so the Q/R manager communicates with the taxi and updates the queue of the appropriate zone

2.4 Deployment view

This section describes the main harware components of our system with also their execution enviroment and communication protocols; again it’s not mandatory to use these protocols, they just can be a feasible solution.
Application server, workstation and databases are physically located inside the headquarter while the webserver is in a branch; taxi car/GPS and client web/mobile app are considered as normal client in the client-server paradigm: the taxi car/GPS represent the harware inside the car.
All is described by using the deployment diagram

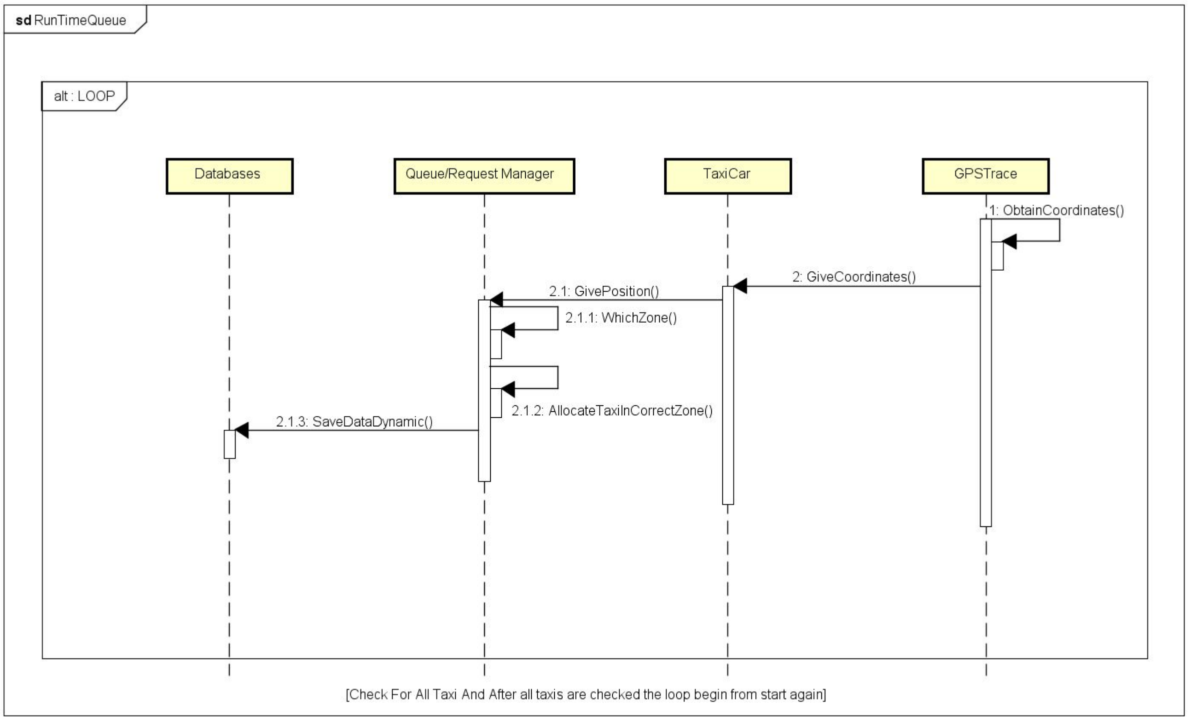


2.5 Runtime view

Below you can find some sequence diagrams that describes the behavior of the application, so how different components communicates each other with some function calls in order to provide a proper service.

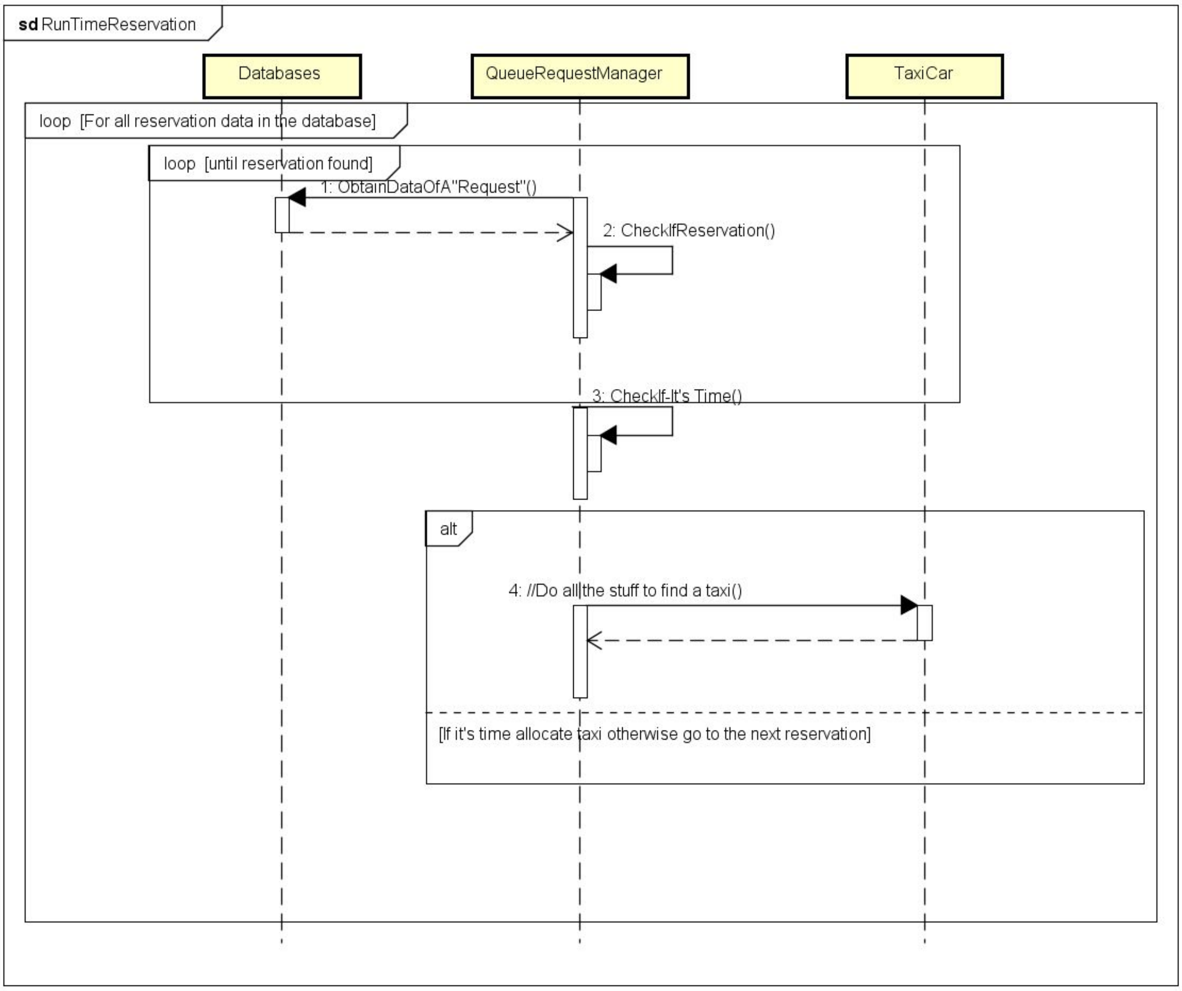
2.5.1 Taxi queue allocation

Here we are allocating a taxi in a specific queue basing on the coordinates obtained by the GPS



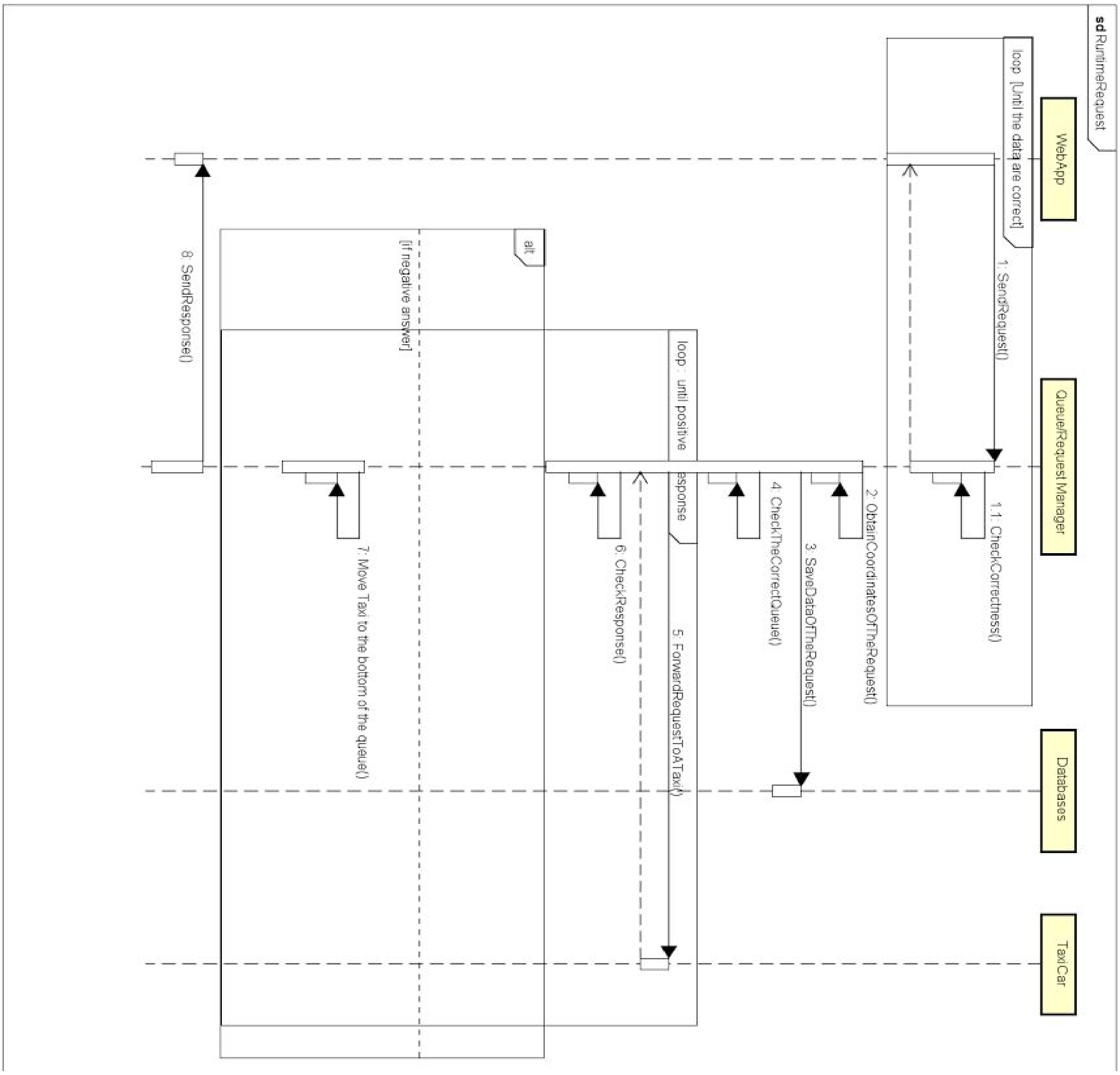
2.5.2 Reservation managing

This graph describes the process to ensure that if the request is a reservation, the taxi is allocated 10 minutes before the departure time.


sd RunTimeReservation
Databases
QueueRequestManager
TaxiCar
loop [For all reservation data in the database]
loop [until reservation found]
1: ObtainDataOfA"Request"()
2: CheckIfReservation()
3: CheckIf-It's Time()
alt
4: //Do all the stuff to find a taxi()
[If it's time allocate taxi otherwise go to the next reservation]

2.5.2 Taxi request

Whenever the client request the taxi, the procedure shown below starts.



2.6 Component interfaces

The most important interfaces are those between databases, web application, the Q/R manager and the taxi.

The interface “Manage data” of the database both take care about static data coming from web application and dynamic ones from Q/R manager (in particular from the request dispatcher and the taxi allocator); the web application has also another interface that communicate directly with the Q/R manager whenever a request is forwarded; the control of the address of the departure point is performed by the web application before forwarding the request.

Inside the Q/R manager there is the request dispatcher that links, with appropriate interfaces, the dynamic database with the taxi; that means the Q/R manager takes the correct queue and send the request to the taxi with the policy described in the RASD document.

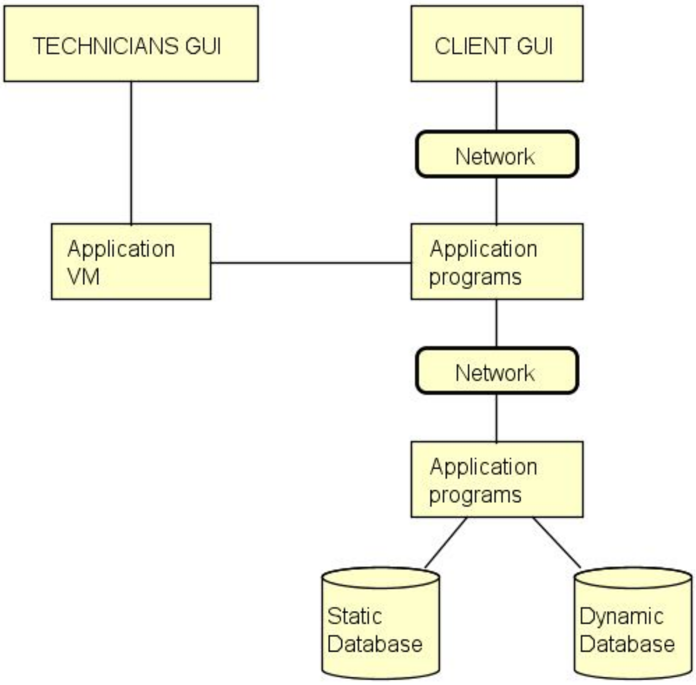
We can find interfaces also in the web app and technician workstation in order to take trace of the sessions.

2.7 Selected architectural styles and patterns

As you can infer by the whole description of the system till here, we use an approach of “Fat server, thin client”: in fact all the computational work is made by the system of MyTaxiService while the client simply at most needs to download a small smartphone application that let him/her to access the service from anywhere.

We’ve also decided to use the 3-tier architecture in order to divide the logic of the system with data and presentation; in this way we aim to reach a better integration of the features developed by the technicians with the already existing system.

The subscription of a client to the system can be seen as a “Event-based paradigm” that is the registered client recieve a mail from the system whenever a new feature is available. Also the way we implement the new features is similar to the “Plug-in architecture”: we have the existing system that is the host application and the new features are the plug-ins; in this way we aim to reach high modularity of our system.

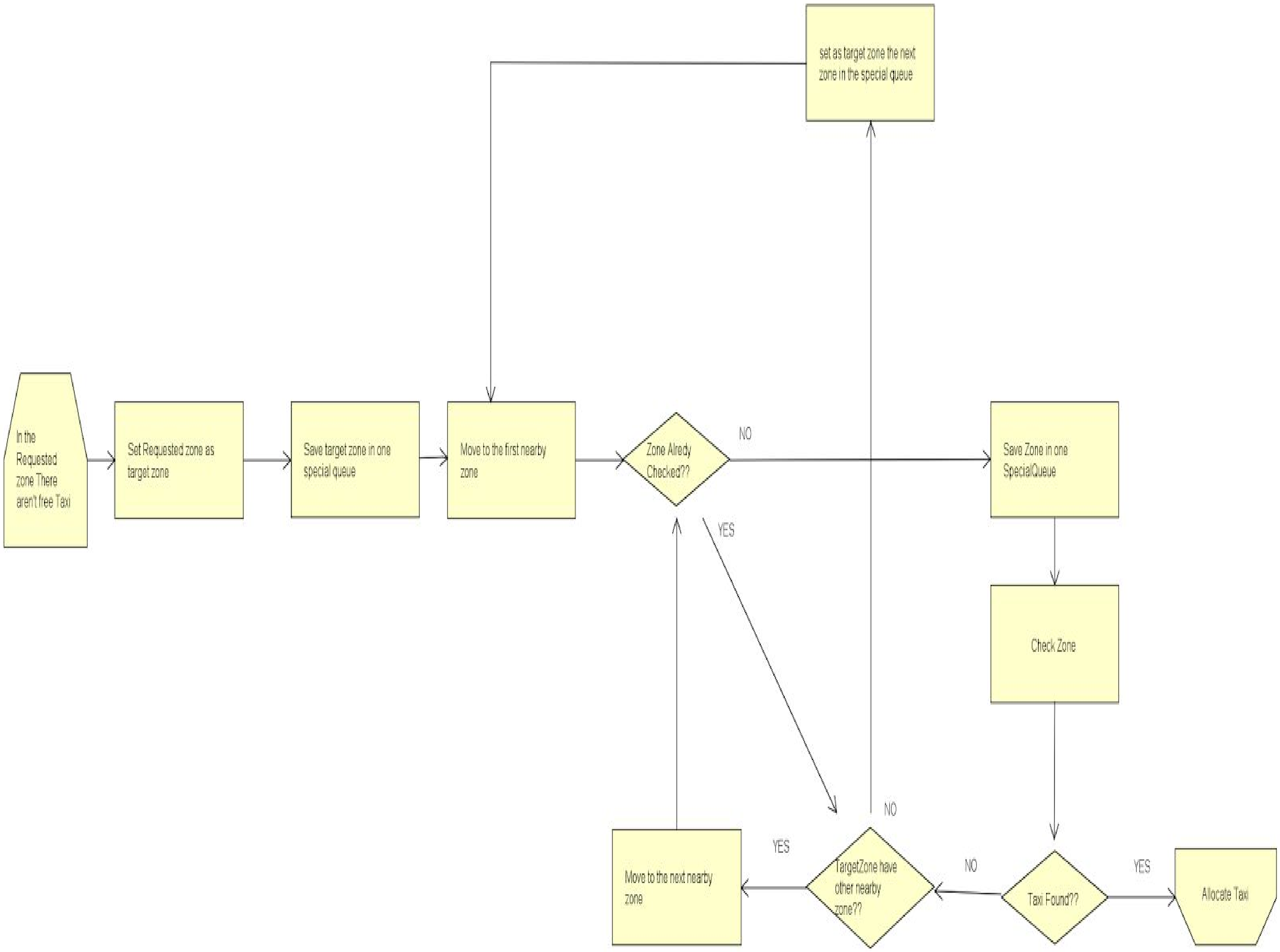


3. Algorithms design

In this section we are describing the algorithm used for the most significant functionalities implemented in our system:

1) Forward request to queue of adjacent zones
2) New request assigned to taxi

Important: to simplify the work of the developers and increase scalability of the system, we assign to each zone 4 coordinates and the relative queue; each zone has in common 2 and only 2 coordinates with the adjacent zone, in this way it’s easy to find the adjacent zone of a given one.
Morover we can assign easily the zone to a taxi given its coordinates, just check if the value of the GPS has latitude and longitude is between two appropriate pair of coordinates (for example the pair of the northern border and the pair of the eastern one).


In the Requested zone There aren't free Taxi
Set Requested zone as target zone
Save target zone in one special queue
Move to the first nearby zone
Zone Alredy Checked??
NO
YES
Save Zone in one SpecialQueue
Check Zone
Taxi Found??
YES
Allocate Taxi
NO
TargetZone have other nearby zone??
YES
NO
Move to the next nearby zone
set as target zone the next zone in the special queue

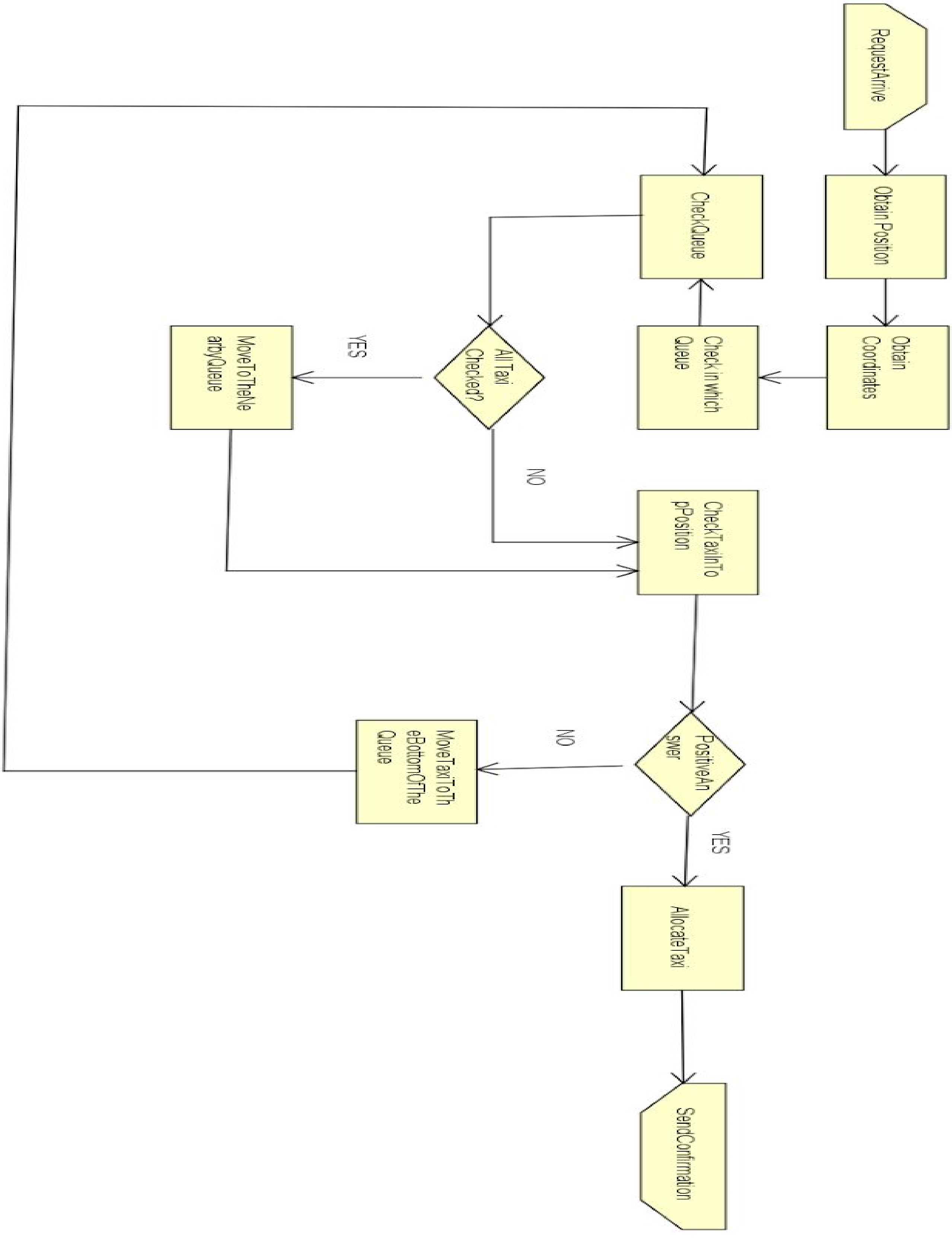
3.1 Forward request to the adjacent zones

This algorithm aim to manage the corner case in which in the request of a taxi has for starting point a zone in which the queue of taxi is empty.

The algorithm is quite simple: starting from a target zone x (the interested one), it checks (if there exist) in this order the northern zone, the eastern one, southern one and western one: if it finds an available taxi, the system forwards the request, otherwise the zone is saved in a queue named "specialQueue"(access policy FIFO of course), marks it as already checked, and moves to the next zone: already checked means that I've controlled all adjacent zones.

Once all the adjacent zones have been checked, it uses the next zone in the queue as target zone and restart the algorithm until all zones are marked as "already checked": due to the assumption that there is at least one taxi available in the whole city, at most in the last zone we must find a taxi.

We assume that the algorithm is fast enough to avoid that a taxi moves inside a zone already checked


RequestArrive
Obtain Position
Obtain Coordinates
Check in which Queue
CheckQueue
All Taxi Checked?
YES
MoveToTheNearbyQueue
NO
CheckTaxiInTopPosition
PositiveAnswer
NO
MoveTaxiToTheBottomOfTheQueue
YES
AllocateTaxi
SendConfirmation

3.2 New request assigned to a taxi

The system receive the coordinates of the zone and send the request to the right queue; after this, the procedure of forwarding the request to the taxi of the queue starts.

If the queue is empty or all the taxi refuse the request, the procedure described in the previous section begins.

The coordinates are computed from the web server basing on the address inserted by the client

4. User interface design

The different user interfaces are already described in the RASD document, so all the information concerning these interfaces are listed in the correct section of the RASD.

5. Requirements traceability

In this section we map the functional requirements of the RASD with a specific design elements that comes from our choices about the architecture and the system.
Below a list of functional requirements for each goal are reported from the RASD.

1. Allow guest to register the application in order to receive upcoming features
    - [R1]The system must have a database to store all registred users and provide a sign up functionality
    - [R2]A person who already registered, cannot register again unless previous unregistration

2. Allow user/guest to make a reservation
    - [R3]The system provide a page to make a reservation to any user
    - [R4]The guest have to insert the information about the trip: starting point, destination, meeting time
    - [R5]All these informations must be inserted 2 hours before the meeting time; the system send a message either the reservation is done or not.

3. Allow user/guest to delete a reservation
    - [R6]The system also have to provide the possibility to cancel an existing reservation from web or app; in this last case, no taxi will be allocated

4. Allow user/guest to take a ride
    - [R7]The system allow any user (from web site or smartphone app) to send a request for a taxi
    - [R8]After a request is made, the system have to send a message to the taxi at the top of the queue

5. Allow a technicians to implement additional features
    - [R9]The technician must be logged in the correct section
    - [R10]After the login a technician can choose one of the feature that want to modify
    - [R11]If a technician isn't allowed to work on some features the system doesn't allow him to make any modification
    - [R12]When the new feature is completed, the system integrate it automatically.

6. Allow the system to allocate correctly a taxi based on the currently location
    - [R13]The system automatically gets GPS informations of all taxis and add them to the queue of the correct zone

7. Allow user to change his information
    - [R14]In the personal page there will be a link to change informations
    - [R15]Updated informations are stored in the database

Requirement	Design Choice
[R1]	We store the data about a user on the database that contains only the “static data”.
[R2]	In the static database there one and only one profile for each e-mail address
[R3]	In the homepage of the web/mobile app a user/guest can have the possibility to make a reservation.
[R4]	There is a simple form in the web page that must be correctly filled
[R5]	A record about a reservation is invalid if the user/client try to make a reservation less than two hours before the requested time. A record is also invalid if any field of the form is empty: in both cases the request is not forwarded and the user/client is invited to refill correclty all fields.
[R6]	If a user want to cancel a reservation ,the data of the corresponding reservation are deleted from the “dynamic database”.
[R7]	In this case after a user request a ride ,the data about that ride are stored in the dynamic database. The Q/R manager will take care about this request.
[R8]	The Q/R manager through specific process manage all the required steps to take care of a request. This manager can find the specific queue of taxi that correspond to the correct zone of a ride and forward the request to all taxi until a positive answer.
[R9]	A technician is allowed to be logged only in the relative workstation.They cannot work through the web/mobile app.
[R10]	Through the worstation a technician can choose the feature that want to develop,but only if he is allowed to develop that specific feature.
[R11]	For [R11] we provide the same solution of [R10].
[R12]	From a workstation after the technicians check the correctness of a feature ,there is the possibility to integrate a feature into the main system.
[R13]	Each taxi have a GPS that send the current coordinates of the taxi to the Q/R manager through specific software. Once the manager has received the coordinates of one taxi ,it is able to allocate the taxi to the correct zone.
[R14]	In the personal page of the web/mobile app,we give to all users the possibility to change their information about the profile with a link

[R15]	After a change in the user information the system overwrite automatically the old information in the static database.

for each requirement we can “choose” the main components that fulfill it.

Requirement	Component
[R1]	Databases
[R2]	Databases
[R3]	Web App
[R4]	Web App
[R5]	Web App
[R6]	Databases
[R7]	Databases,Q/R manager
[R8]	Q/R manager
[R9]	Technician WorkStation
[R10]	Technician WorkStation
[R11]	Technician WorkStation
[R12]	Technician WorkStation
[R13]	GPS,Q/R manager,TaxiCar
[R14]	Web App
[R15]	Databases,Web App

6. Hours of work

Perillo Salvatore	20
Sesto Claudio	20