Politecnico di Milano

A.A. 2015-2016

Software Engineering 2 Project

My Taxi Service

Design Document (DD)

version 1.0

Giovanni Brena (858328), Andrea Canale (858638)

Content

1. Introduction

1.1. Purpose

Here is presented myTaxiService application Design Document. This documents provides a complete description of the required hardware components, interfaces, software modules, data management and guidelines to algorithms development. DD is targeted to stakeholders, developers and software analysts. Many structural diagrams will be provided in order to make decription as clearest as possible while UX design views will give information about application Look and Feel.

1.2. Scope

MyTaxiService main scope is to provide a fast and simple taxi booking service for one or more cities. The system has to assure quality, reliability and security for users. Scalability has been taken in to account so that application could be easily expanded in the future. During design process well know styles and pattern has been uses in order to make work easier for software developers.

1.3. Definitions, Acronyms, Abbreviations

1.3.1. Definitions

Guest: someone who open/visit myTaxiService and is not logged in.
User: someone who is registered and logged in to myTaxiService.
Taxi Driver: a type of user who can answer to passenger's call.
Passenger: a type of user who can request a taxi or make a reservation for a taxi.

1.3.2. Acronyms

RASD: Requirement Analysis and Specification Document
DD: Design Document
API: application programming interface
JEE: Java Enterprise Edition
JDBC: Java Database Connectivity
JSP: Java Server Pages
JMS: Java Message Service
HTTP: HyperText Text Protocol
HTML: HyperText Markup Language

UML: Unified Modeling Language
XML: eXtensible Markup Language
JSON: Javascript Object Notation
UX: User eXperience
MVC: Model View Controller
JPA: Java Persistence API
ACID:ATOMICITY,CONSINTENCY,ISOLATION,DURABILITY , set of database properties that guarantee that database transactions are processed reliably

1.3.3. Abbreviations

TD:taxi driver
CR:call request
DB:database

1.4. Reference Documents

- Specification document: Assignements 1 and 2 (RASD and DD).pdf
- DD guidelines: IEEE Standard For Design Document.pdf
- Given examples: examples of DD documents
- Software Engineering 2 (De Nitto) online material (slide, references)
- Java EE Oracle Documentation

1.5. Document Structure

Document has been structured following these points:

- description of the system main phisical components
- software sub-modules and their interaction
- sw modules interfaces
- runtime behavior
- other system properties (connection types, security modules ecc)
- main algorithms analysis
- UX design
- RASD properties traceability

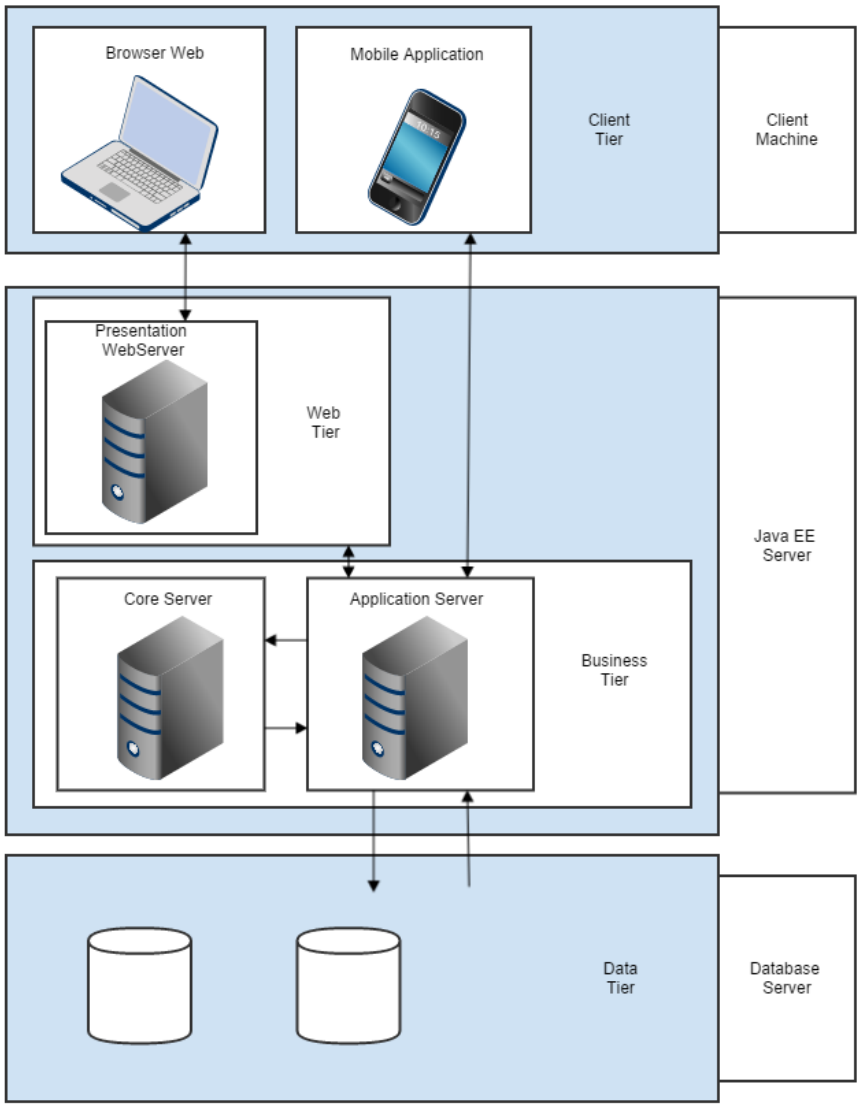
2. Architectural Design

2.1. High Level Components and their interactions

2.1.1. Overview

The project has been designed as a Three Tired System (Client Machine, Server Machine and Database Server). Each tier has been divided into few sub-components and all of them provide interfaces through which interchange data.

The System has to be developed as a Java Enterprise Edition application.



2.1.2. Client Tier

Clients implements front-end part of the system and are the only "user visible"

components. Through them users can interact with the system primarily performing actions, providing inputs and receiving data from the Server tier. System will offer two different type of client: Web Client and Application Client.

Web Client will give the possibility to make use of the system through any modern Web Browser connecting to a specific URL. Content will be provided as HTML documents by Server Tier using a JSP (Java Server Pages) system while data should be preferably exchanged through JSON objects. Communication protocol is based on HTTP Get and Post requests.

Application Client will be a mobile App designed as a thin client developed through IONIC framework, which let to make a cross-platform application without loosing functionality. While User Interface and communication interfaces will be embedded into application, logic implementation is delegated to the Server so that mobile App will act as a terminal through which send, receive requests and present data. Application Clients must be RESTful i.e. be able to interact with a REST Web Service. Mobile App should be available for iOs devices, Android devices and optionally for Windows Phone devices.

2.1.3. Java EE Server

Java EE Server is the larger and more complex component of the application. It is divided into two sub-systems: Web Tier and Business Tier. The first contains the Web Server while the second contains Application Server and Core Server.

Web Server is the component that provide JSP pages to Web Clients fulfilled with data provided by Business Tier. It uses HTTP protocol.

Application Server contains logic of the system. It receive requests, process data, interact with Core Server and Data Tier and send response. It also manage Java Beans and their lifecycle.

Core Server is an extension of Application Server that manage continuous runtime operations not directly related to Users. In the specific it manage taxi queues updates, taxi tracking and geocoding services.
Application Server and Core Server can run on the same machine or into different machines (or even on different Server Farms). This choice has been made to offer high scalability to the system, as well as higher reliability and security.

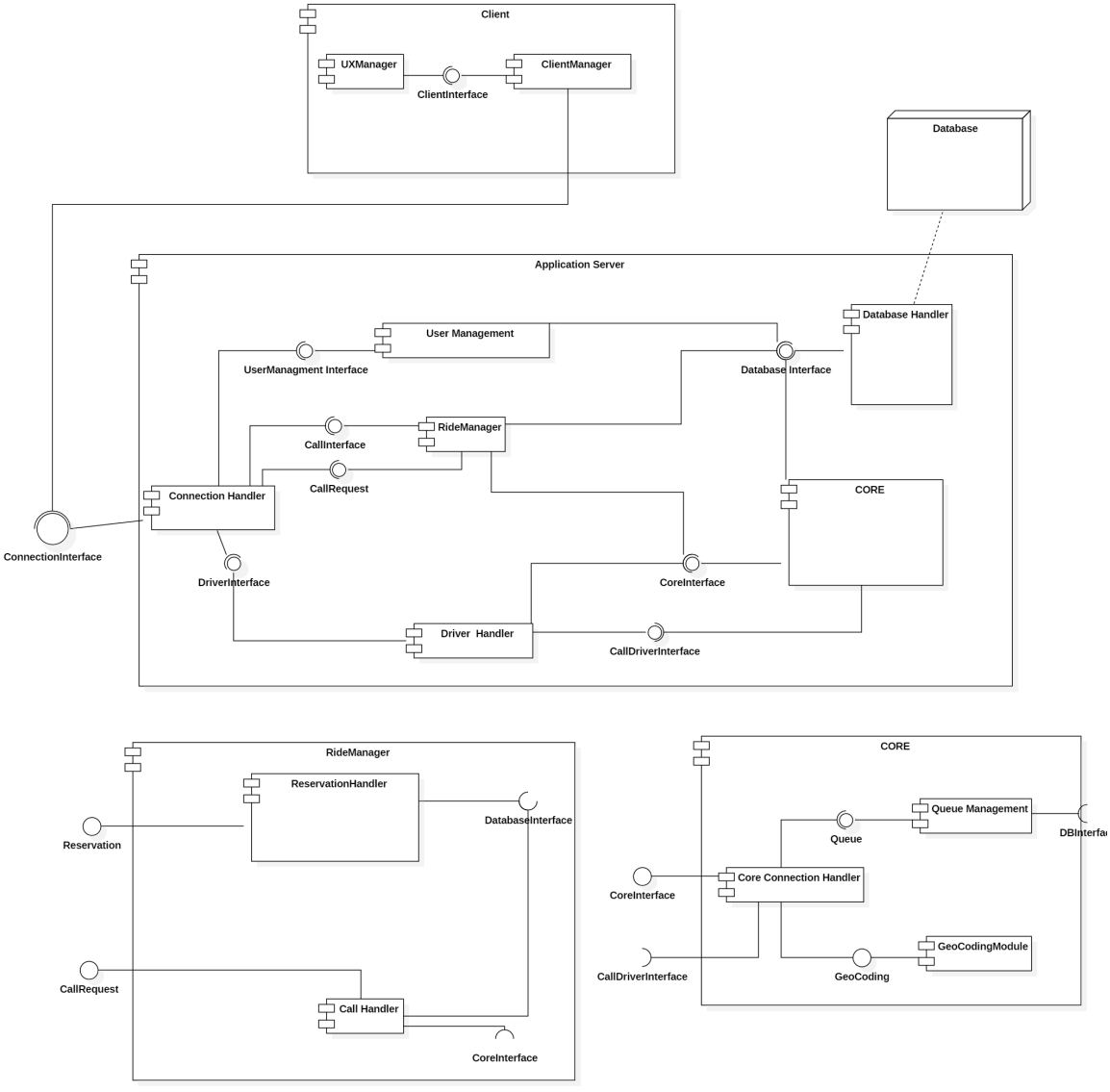
2.1.4. Data Tier

Data Tier provide persistent data storage to the system. It contains a Database Server and an SQL Database interfacing to the system through a JDBC protocol.

2.2. Architectural Modules and Components View

The system logic has been decomposed using an iterative top-down process into many different sub-systems. Each of these software modules offers communication interfaces to the rest of the system.

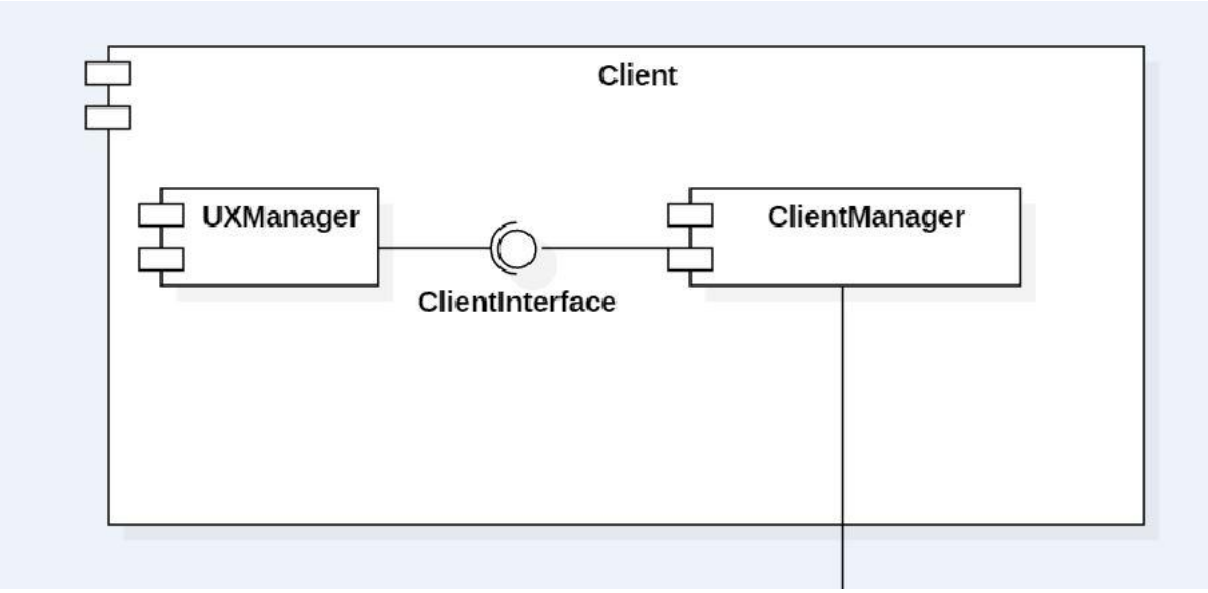
2.2.1. Component View



2.2.2. Client Manager

This component manages the connectivity through between CLIENT and SERVER allow the user to send all the request needed to server. A set of API is

provided to the "front-end" in order to divide the View from the rest of the System in relation of MVC pattern.



2.2.3. *Connection Handler*

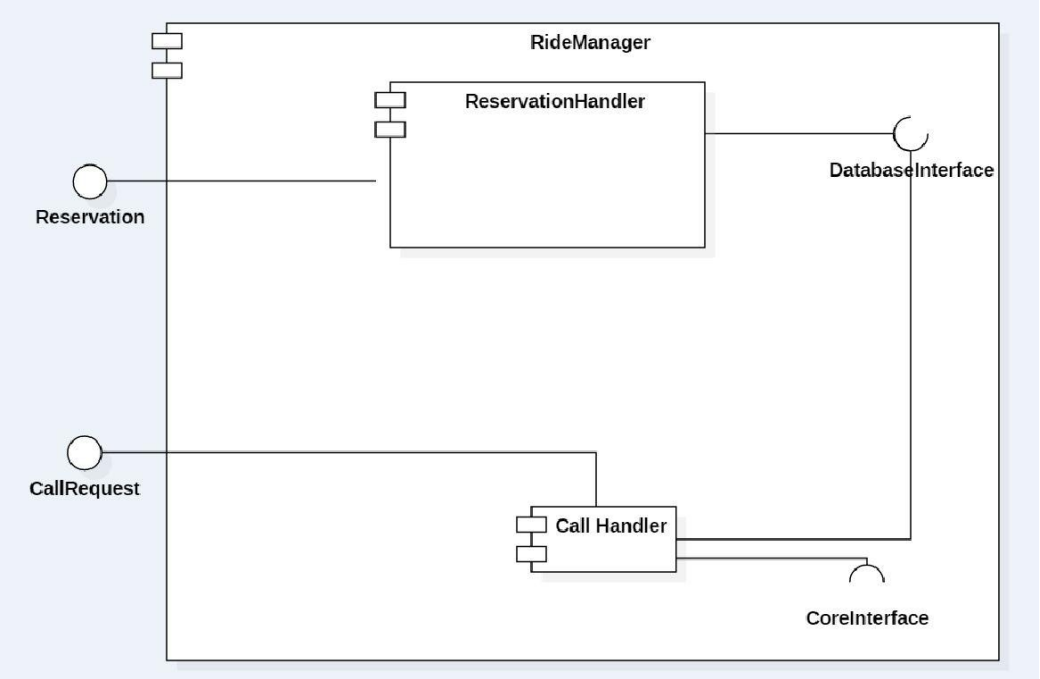
This component handles all the request that server receive forwarding them to the related Component. It provides stateless connection (State-Less Beans)for the functionality connected to the User Management Component and a stateful connection (State-Full Beans) for functionality connected to CallHandler and DriverHandler Components.

2.2.4. *User Management*

This component manages all typical request of a SOA (Service-Oriented Architecture) providing the functionality of login, register ,access to user personal data,modify personal data and retrieve access information.
The last two functionality are performed using JAVA MAIL API provided by ORACLE.

2.2.5. *Ride Manager*

This component manages the CallRequest and Reservation creation and evolution during the time in relation of the Passenger request.

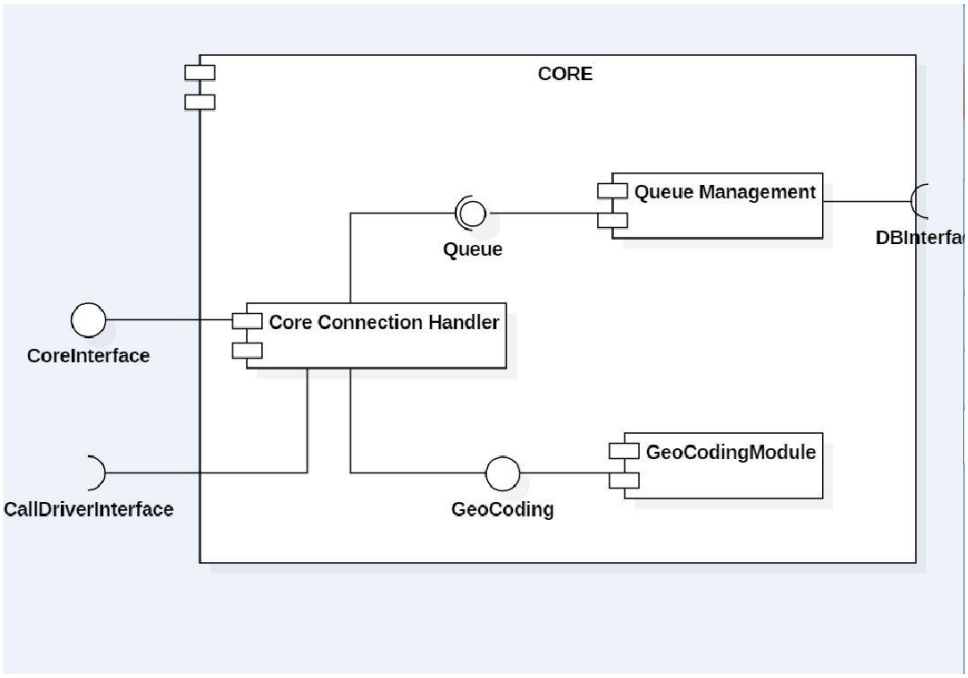


2.2.6. Driver Handler

This component manages the logical connectivity between TaxiDriver-Core. Through this the driver's position updating and the interaction with CallRequest by Driver View is handled.

2.2.7. Core

This component represents the core of the entire system in terms in computation view.



2.2.8. Core Connection Handler

This component manages the functionality of the other two elements handling the request of external component. To understand better his functionality see Algorithm Section and RunTime View Section. It ensure a correct concurrency access to the Queue Management.

2.2.9. Queue Management

This component manages the queues of the taxi driver of all zones, changing their structure in relation of event. This represent the dynamic MODEL of the SYSTEM so it's developed as a SINGLETON BEAN.

2.2.10. Geocoding Module

This component provide the functionality of encode the geographical position into the related Zone.

2.2.11. Database Handler

This component provide a set of API in order to query correctly the DB in relation of SQL rules providing in the meanwhile a correct concurrency access to the DB.

2.3. Component Interfaces

2.3.1. Connection Interface

- login(username,password)
- register(registerForm)
- requestHistory(username)
- editProfile(username,editForm)
- makeTaxiCall(username,position)
- makeReservation(username,ReservationForm)
- updateTDposition(username,position)
- updateTDstatus(username,status)
- retrieveAccessInfoLink(username,code)
- requestRetrieveAccessInfo(email,username)

2.3.2. User Management Interface

- editProfile(editForm,username)
- requestHistory(editForm,username)
- login(username,password)
- register(registerForm)
- retrieveAccessInfo(email,username)

2.3.3. CallInterface

- makeTaxiCall(username,position)
- cancelTaxiCall(username)

2.3.4. ReservationInterface

- makeReservation(username,ReservationForm)
- cancelReservation(username,ReservationID)

2.3.5. Driver Handler Interface

- updateTDposition(username,position)
- updateTDstatus(username,status)

2.3.6. Call Driver Interface

- incomingCall(driver)

2.3.7. Database Interface

- queryHistory(username)
- queryLoginCredential(username)
- queryZone()

2.3.8. Core Interface

- findTaxi(passengerPosition)
- updateTDpositionCore(username,position,lastZoneAssigned)
- updateTDstatusCore(username,status)
- updateTDstatusCoreException(username,status,isException)

2.3.9. Geocoding Interface

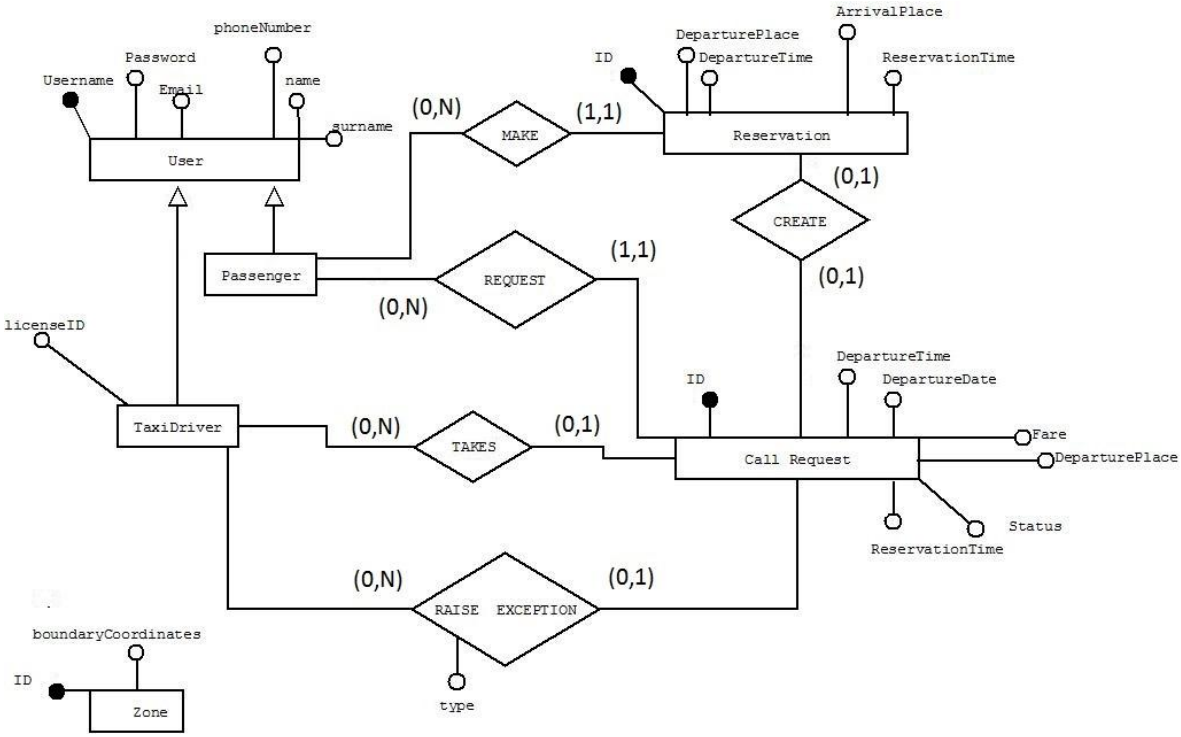
- findZone(position)
- getPathInfo(positionA, positionB)

2.3.10. Queue Management Interface

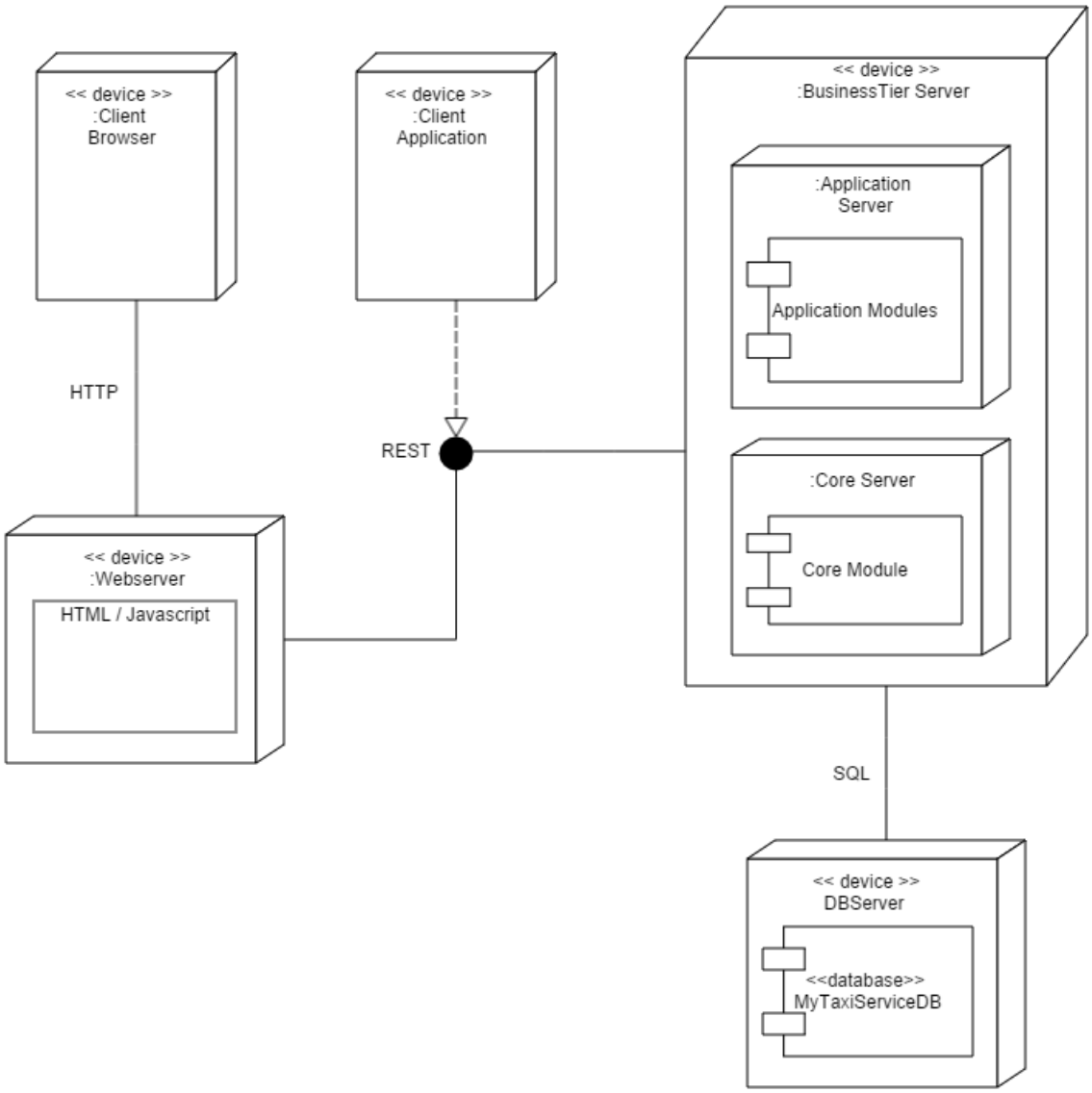
- findTaxi(zone)
- addDriver(zone)
- removeDriver(TaxiDriver)
- moveToLastPosition(Driver,Zone)
- changedZoneTD(TaxiDriver,oldZone,newZone)

2.4. Persistent Data Structures

In the following diagram are represented the conceptual model of the Database, note that that here are store only persistent data so all the dynamic behaviour of queue and callRequest are not tracked here.

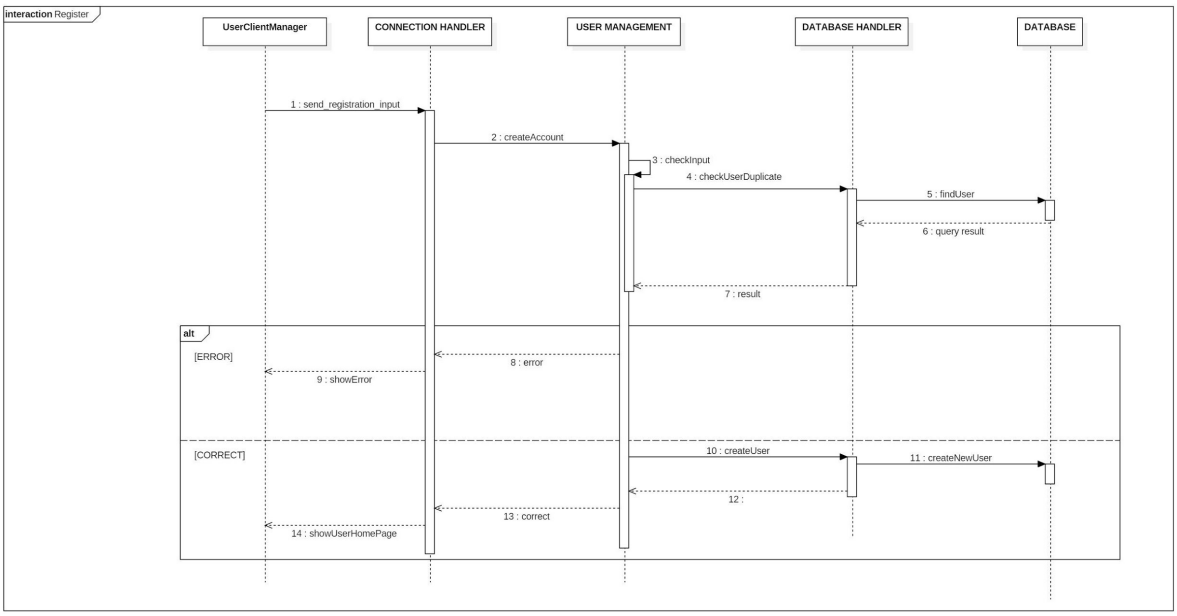


2.5. Deployment View

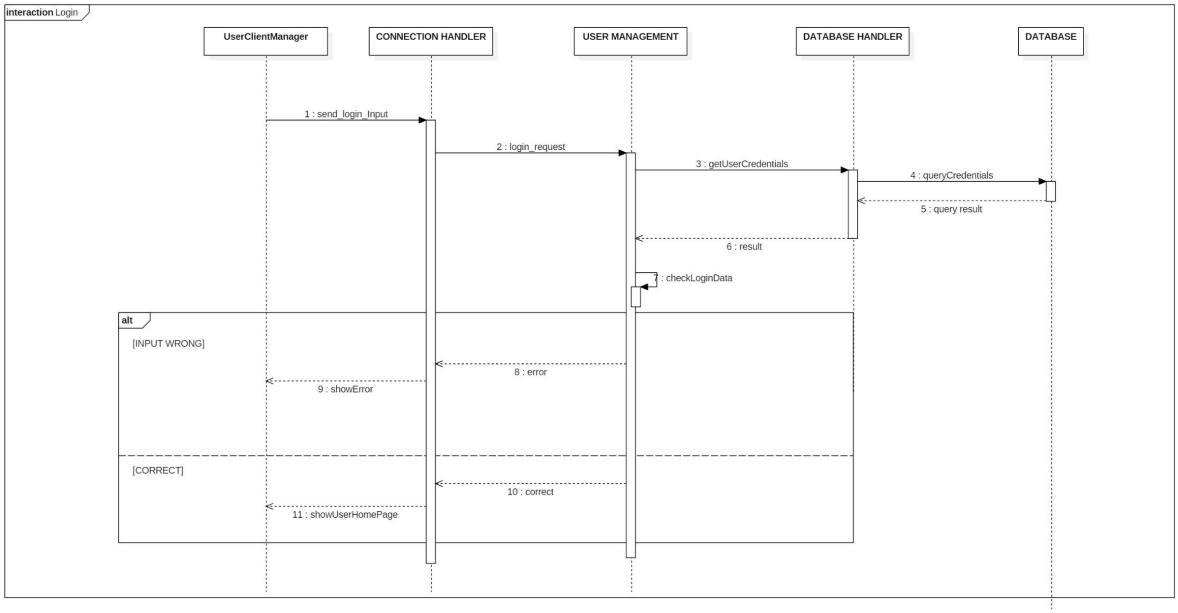


2.6. Runtime View

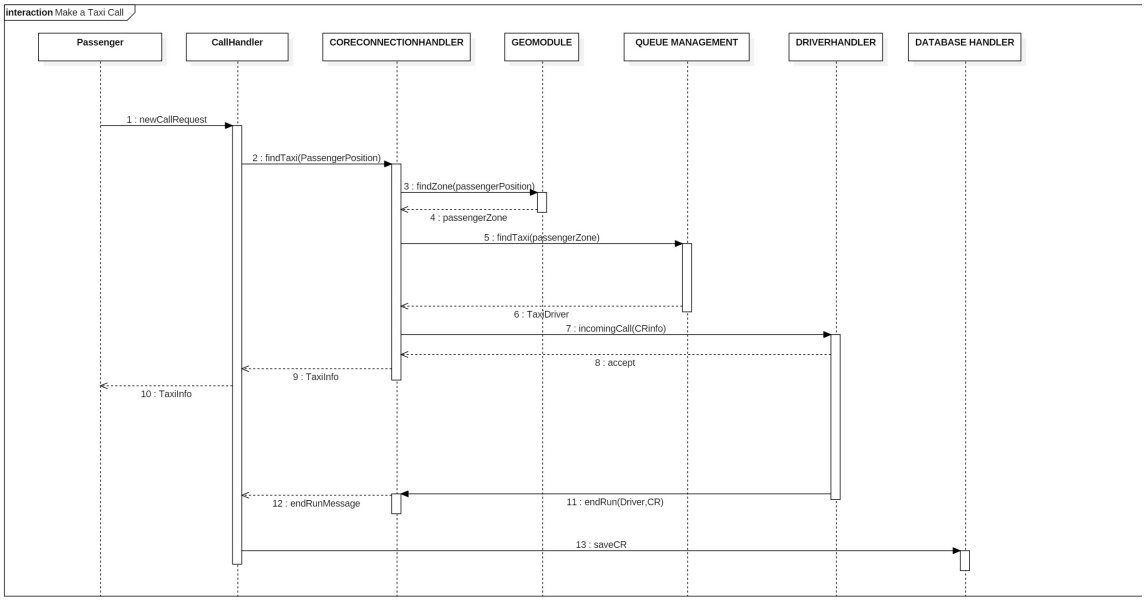
2.6.1. Registration



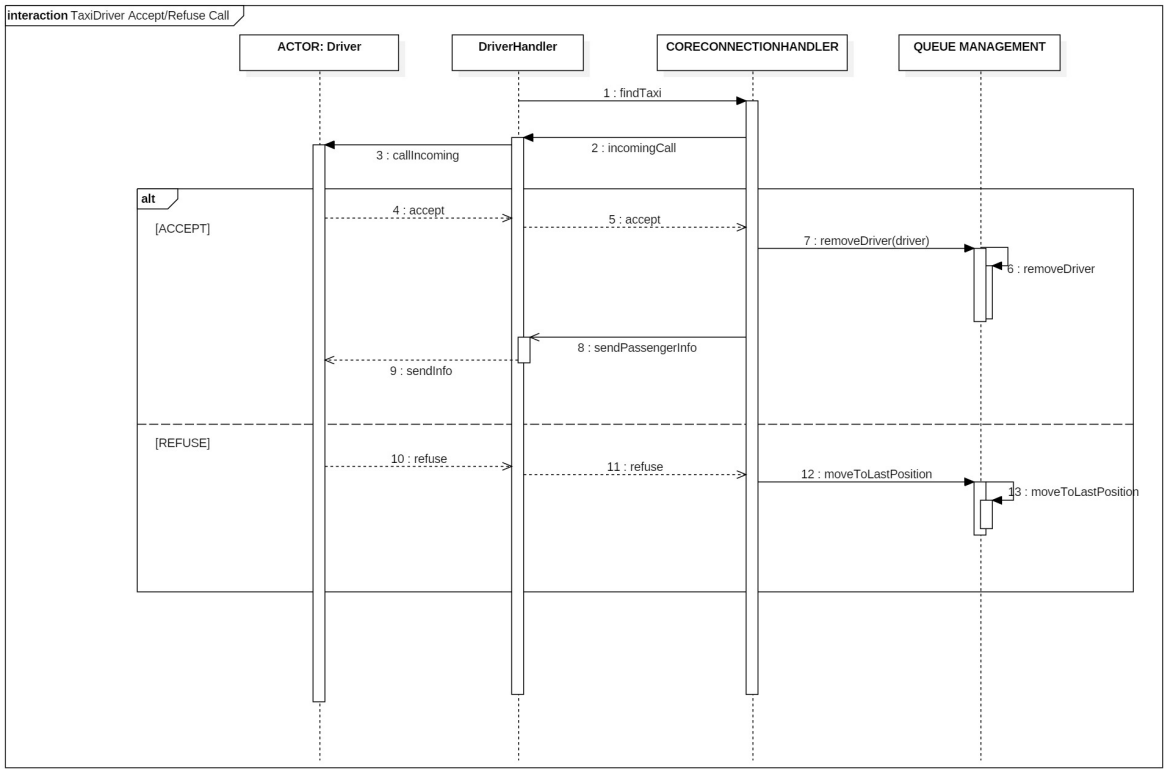
2.6.2. Login



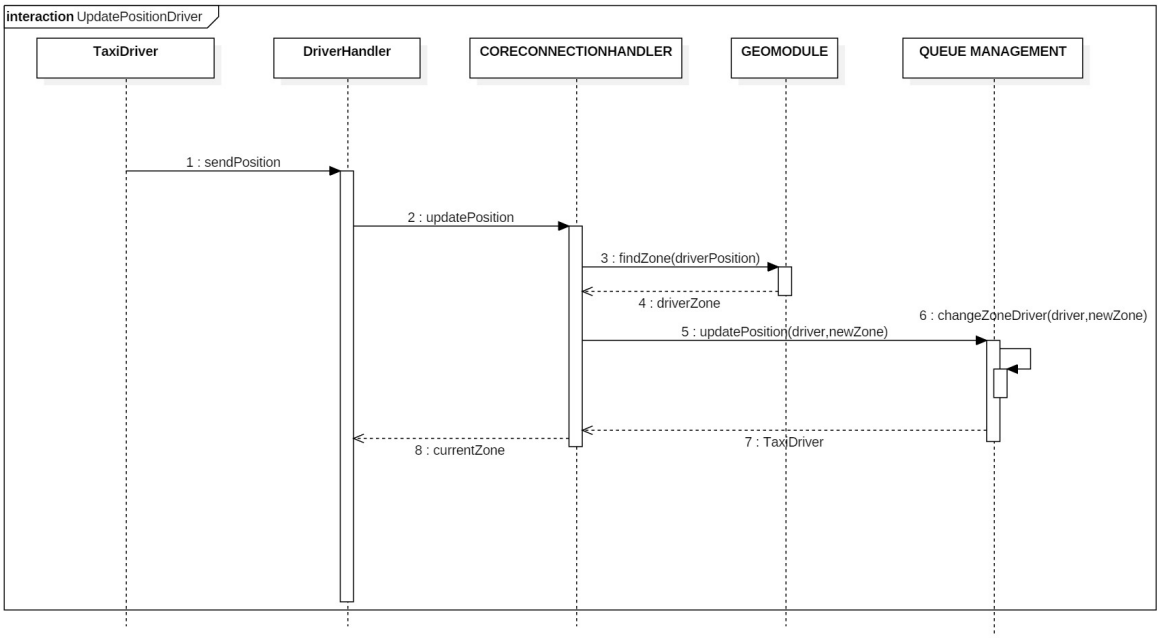
2.6.3. Make a Call



2.6.4. Accept & Refuse call

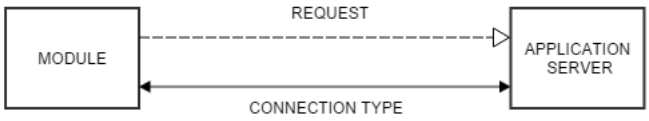


2.6.5. Update position



2.7. Communication Protocols and Connections

Connections between Clients and Web server will be performed through HTTP protocol requests incapsulating data into JSON objects. These connections will be mostly stateless, exception made for statefull connections related to Call Management (users) and Run Management – Driver Handler (drivers). Communication between Application Server and Database Server will use JDBC protocol.



MODULE	REQUEST	CONNECTION TYPE
Sign Up	signUp()	HTTP Post
Login	login()	HTTP Post
Login	retrieveLoginData()	HTTP Post
User	performReservation() updateReservation()	HTTP Post
User	performCall()	Stateful Session
User / Taxi Driver	getProfileData() updateProfileData()	HTTP Post
Taxi Driver	setDriverState()	HTTP Post
Taxi Driver	performRun()	Stateful Session

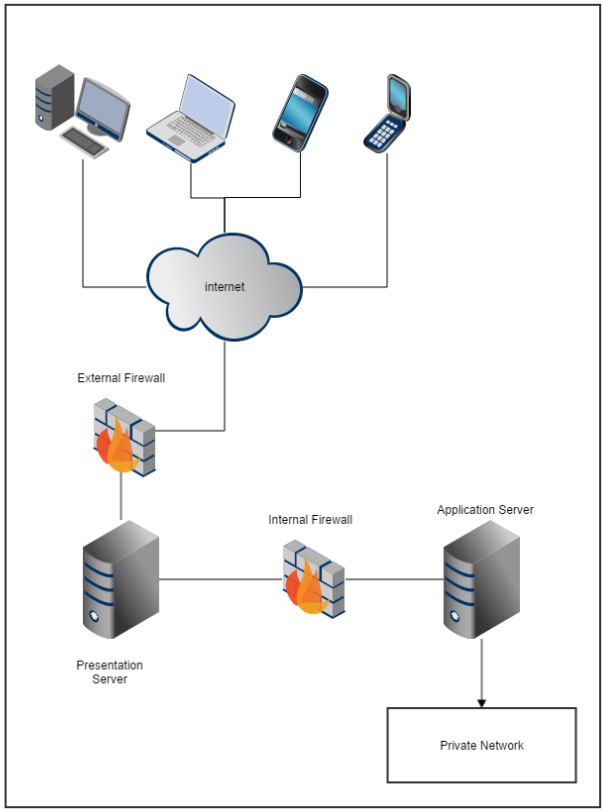
2.8. Security

Security protocols will protect system availability, data Integrity and Privacy.
Authentication is required to access any service of the system so that any actor can be uniquely recognized.
Authorization will be required to access any data or service. System will check if the figure requiring data or services own required access privileges.

Further protection against external thrusts (for example DOS attaks) will be also granted by Firewall protection. To this point thanks to the Three Tired division of the system we can use 2 differents approaches:

- EXTERNAL FIREWALL, Single Subnet: A single firewall protection will be deployed between system subnet and Internet.
- INTERNAL and EXTERNAL FIREWALLS, Two Subnet: in order to increase security we can deploy another firewall system between presentation layer and business layer.

It seems reasonable to deploy a ONE FIREWALL solution extendible to a double for safety reasons in the future.



2.9. Selected Architectural Styles and Patterns

Design of this architecture has been driven by many well known Styles and Pattern.

Syles

The system use a **Client-Server** approach and it has been designed as **three tired** system having both **distributed presentation** (web clients) and **remote presentation** (application clients). Inside this document system has been decomposed into modules using a **components & connectors** approach. MyTaxiService is a **SOA** (Service Oriented Architecture) **event-based-system**.

Pattern

Application client follows a **MVC** pattern.

Singleton pattern has been used defining components witch need to be persistent and accessible to many clients. We designed few beans (such as Connection Handler) to follow a **Factory** pattern. That's why system do not know in advance what kind of connection should be created (stateful or stateless).

3. Algorithm Design

3.1. Taxi Management

The city is divided into zone, every zone has:

- -a queue of taxi driver (TD) available
- -boundary coordinates of the zone

The queue is a FIFO queue, it is modified in these cases:

- When a CR occurs, findTaxi() is invoked on the queue and the first TD in the queue is returned to the function,after that can occurs one of these two events:
  - DRIVER ACCEPT A CALL: the selected TD is removed from the queue and his status is setted to UNAVAILABLE
  - DRIVER REFUSE A CALL: the selected TD is moved to the last position of the queue

- When TD ends a Run it's status is setted AVAILABLE and enqueue in the last position of the queue of the related zone.

- When a TD whose status is AVAILABLE changes zone he is removed from the starting Zone queue and enqueue in the last position of the final Zone.

3.2. Call Management

When a passenger request a Taxi his request is forwarded from the Client to the RideManagement component. The request has these data:

- -Position of the passenger
- -Username of the passenger
- -Telephone number of the passenger

The Ride Management data checks if the requesting User has already an ACTIVE

From that moment the sequence of event are:

1) A timer is setted to TIME_EXCEED value(see RASD) and forwarded to the CORE component

2) One of these three possible events occurs

2.1) TIME_EXCEEDED :the timer triggers, CR status is setted to CLOSED,the passenger is notified and the CR is stored on the DB

2.2) NO TAXI FOUND:CR status is setted to CLOSED,the passenger is notified and the CR is stored on the DB

2.2)TAXI_FOUND: -CR receives the data of the TD assigned and TD has already PassengerInfo

- -CR status setted to ACTIVE

3*)see Exception Management

3) The TD notify the start of the run,the CR status is setted to ON PROGRESS

4)CR receives an endRunMessage:

3.1)the CR data,including the ID of the TD assigned, are stored to the DB

3.2)the CR object is deleted

3.3. Exception Management

Before the TD reach the passenger and CR status is setted to ON_PROGRESS,the TD can raises two type of exception:

- NO_PASSENGER_FOUND:the TD can't find the passenger at the POSITION displayed
- GENERIC_EXCEPTION:anything else like a car breakdown

In the other way a passenger can cancel a CR before the TD arrive(CR status setted to ON_PROGRESS) raising a PASSENGER_CANCEL_CALL.

The algorithm manages these exception in these ways:

-NO_PASSENGER_FOUND:

1) An EXCEPTION with type NO_PASSENGER_FOUND is added to CR
2) CR status is setted to CLOSED and stored on the DB
3) CR object is deleted
4) the TD status is setted to AVAILABLE
5) TD is enqueue in the queue of the zone in which he is located

-GENERIC_EXCEPTION:

1) An EXCEPTION with type GENERIC_EXCEPTION is added to CR
2) The passenger is notified
3) The CR is resetted and follow the 1) point of the previous section

-PASSENGER_CANCEL_CALL:

1)After CR receives cancelCallRequest() a cancellCall(thisCR,TD) are send to the Core

2)An EXCEPTION with type PASSENGER_CANCEL_CALL is added to CR and stored in the DB

3)The Core notify the DriverHandler which notify the canceled call

4)The status of the TD is setted to AVAILABLE and handle like an update status with an exception :TD will be moved to the first position on the queue of the zone in which he is.

3.4. Reservation Management

ReservationHandler is a singleton which handle the request incoming and manage the awakening of all the stored one.

The input of a makeReservation(username,reservationForm) call are checked by ReservationHandler through a call to DB that check if the time of the reservation is almost 120 min later in the future than actual time.If yes a reservation is stored on the DB,otherwise return an ERROR_MESSAGE.

A request of deleteReservation(username,reservationID)is handled in the same way of the previous one except for the constraint of no-deleting:the request of delete must be sent 20 minutes before the reservation departure time.

A ReservationHandler queries the DB every hours to check which Reservation are going to start in the next hour.

For each of them an object ReservationTimer with reservation data is linked to the ReservationHandler.(OBSERVER_LISTENER pattern).

When one of the timers trigger ReservationHandler create a CR object with the RE data,from that moment CR is handled as in the Call Management section.

3.5. Core

This is the Core of the System,it manages position of the driver and passenger,the queue management in relation of the first paragraph of this section.

When an updatePosition(taxiDriver,position,lastZoneAssigned) is called by a TD:

1)the new position of the TD is passed to the GeoModule Module which returns newZone

IF (newZone != oldZone)

{it calls removeDriver(oldZone) and addDriver (newZone) on Queue Management module}

return newZone to TD

When a findTaxi(CRInfo,Passengerposition) is called by a CR on CORE:

1) the passengerPosition is passed to GeoModule Module which returns the Zone
2) it calls findTaxi(Zone) on Queue Management

2.1) TAXI_FOUND_CORE:forward the request info to the found driver through incomingCall(CRInfo)

2.1.1)DRIVER_REFUSE_CORE:call moveToLastPosition(Driver,Zone) on Queue Management and restart from step 2)

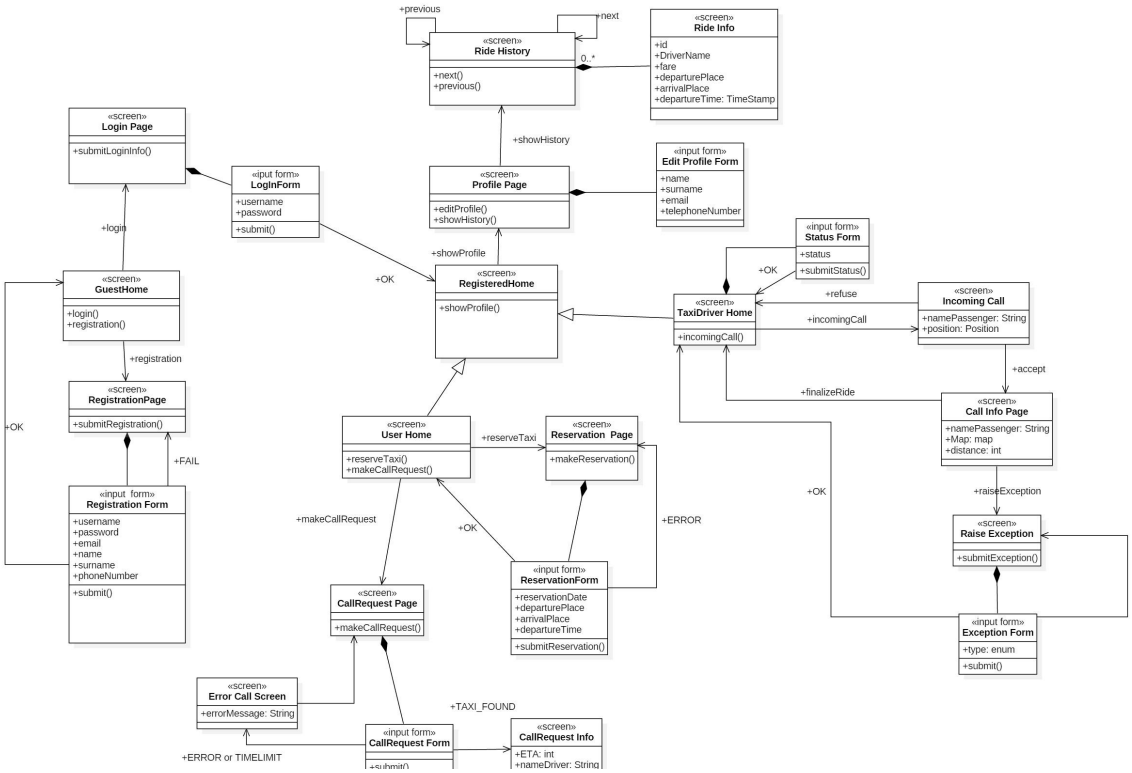
2.1.2)DRIVER_ACCEPT_CORE:manage the queue as explained in first section and return DRIVER data to the CR

2.2)NO_TAXI_FOUND_CORE:notify to CR the NO_TAXI_FOUND message

/*OPTIONAL 2.2.1*)ONCE TIME ONLY calls nearestZone(passengerPosition) and restart from step 2) with the new Zone

4. User Interface Design

For UX previews and mockups we refer to MyTaxiService RASD. Here has been designed a Screen Flow diagram.



5. Requirements Traceability

In this section are listed all Goals with related Requirements followed by an explanation or a direct reference on Design Document about how they are solved.

[G-V1] Visitors should be able to become registered users through

a registration form.

[R1] Visitor can only access to login screen and sign up screen

[DD] The UXMananger (User Interface) is designed to provide to VISITOR only ways for login and registration request

[R2] Visitor must provide a valid email address not already present

in the system.

[R3] Visitor must choose a username not already present in the

system.

[R4] Visitor must choose a password according to the minimum

security rules

[R5] Visitor must confirm registration through a link sent to the

provided email address

[R6] Visitor is asked to provide electronic payments information

(credit card information or PayPal account). In case of no info

provided the user will be able to pay only through cash

[R7] Visitor must accept Privacy Policies

[DD]In the RUN TIME VIEW in the section Register are shown how [R2][R3]

are satisfied by the call of CheckUserDuplicate() to the DB while the other one through a check on sent form in CheckUserInput() call.

3.1.2 [G-UD1] Registered user/driver should be able to log into the

application through his username and password.

[R1] Username and password couple must be present in the system

[R2] System must provide a password retrieving service

[R3] No login attempts limit will be implemented

[DD] The [R1] is resolved by the ACID properties that DB is supposed to have through the implementation of the DatabaseHandler.The [R2] Component View is explained in Component View under User Management implements typical service of SaaS

3.1.3 [G-UD2] Registered user/driver should be able to see and update personal information

[R2] Modify username is not allowed

[DD]The editProfile() method does not include username as an input

[R3] Modify email address will require new email verification

[DD]After the request of editProfile(email) the flow follow the same of the registration one

3.1.4 [G-U1] Registered user should be able to perform an immediate taxi call, cancel a pending call, know request status and reserved taxi position

[R2] Users are allowed to perform a call only if they do not have any other pending or active calls associated to their profile

[DD]Explained in Algorithm Section under RideManagement

[R3] User can cancel a taxi call if the status is “pending” or “active”

[DD]RideManagement

[DD]Explained in Algorithm Section under Call Handler

3.1.5 [G-U2] Registered user should be able to request a taxi reservation, cancel a reservation and see reservation info

[R2] Users can perform a reservation correctly providing these information: date and hour of departure, departure position, arrival

position

[R3] Date and time must be in the future and almost 120 min later than actual time.

[R4] Reservation can be canceled if departure time is at least 15 minutes later than actual time

[DD] [R2][R3] are explained in Algorithm Section under Reservation Handler

3.1.7 [G-D1] Registered drivers should be able to inform the system about his availability or unvailability

[R1] Driver must be logged

[R2] Driver must not be currently associated with an active taxi call

[DD]Explained in Algorithm Section under RideManagement

3.1.8 [G-D2] Registered drivers should be able to accept or reject a taxi

[R1] Driver must be logged

[R2] Driver state must be “available”

[R3] Driver must have received a taxi call

3.1.9 [G-D3] Registered driver should be able to know information about accepted calls (customer position and personal info)

[R1] Driver must be logged

[R2] Driver must have an active taxi call associated

[DD]As said in the Algorithm Section these info are send after he accepts a call.