A.Y. 2015/2016

Software Engineering 2 : “My taxi service”

Design Document

Version 1.1

Ivan Antozzi(790962) , Riccardo Giambona(788904)

03 February 2015

1.Introduction 3
1.a Purpose 3
1.b Reference Documents 3
2. Architectural design 3
2.a Overview 3
2.b High Level Components and their interaction 3
2.c component view 4
2.c.1 Central System 5
2.c.2 Database 9
2.d Deployment view 10
2.e Run Time view 11
2.f Component Interfaces 19
2.g Selected Architectural design and patterns 20
2.g.1 Patterns 20
2.g.2 Architectural Styles 21
2.h Other Design Decisions 22
3. Algorithm design 22
Queues Management 22
Scheduler Management 23
4. User Interface Design 25
5. Requirements traceability 26
Changes in version 1.1 28

1.Introduction

1.a Purpose

The purpose of this document is to give an high livel view of the components that compose the system and to explain how they interact . The Design Document is also useful to explain some aspects that weren't explained in a well-detailed way in the RASD document and finally to put the basis to a low –level design and after that the implementation.

1.b Reference Documents

DESIGN(Part I and II).pdf

Architectural Styles(part I).pdf

Architectural Styles(part II).pdf

http://www.uml-diagrams.org/component-diagrams.html

http://www.cloudcomputingpatterns.org

2. Architectural design

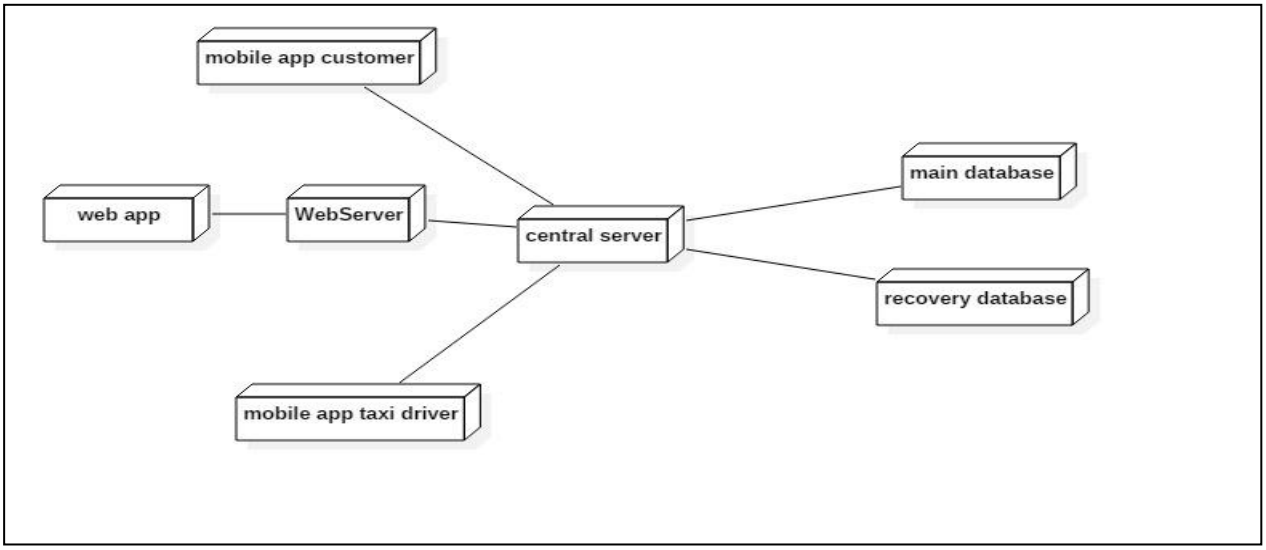
2.a Overview

In this section we will see how the high level system is made and how its components interact.

2.b High Level Components and their interaction

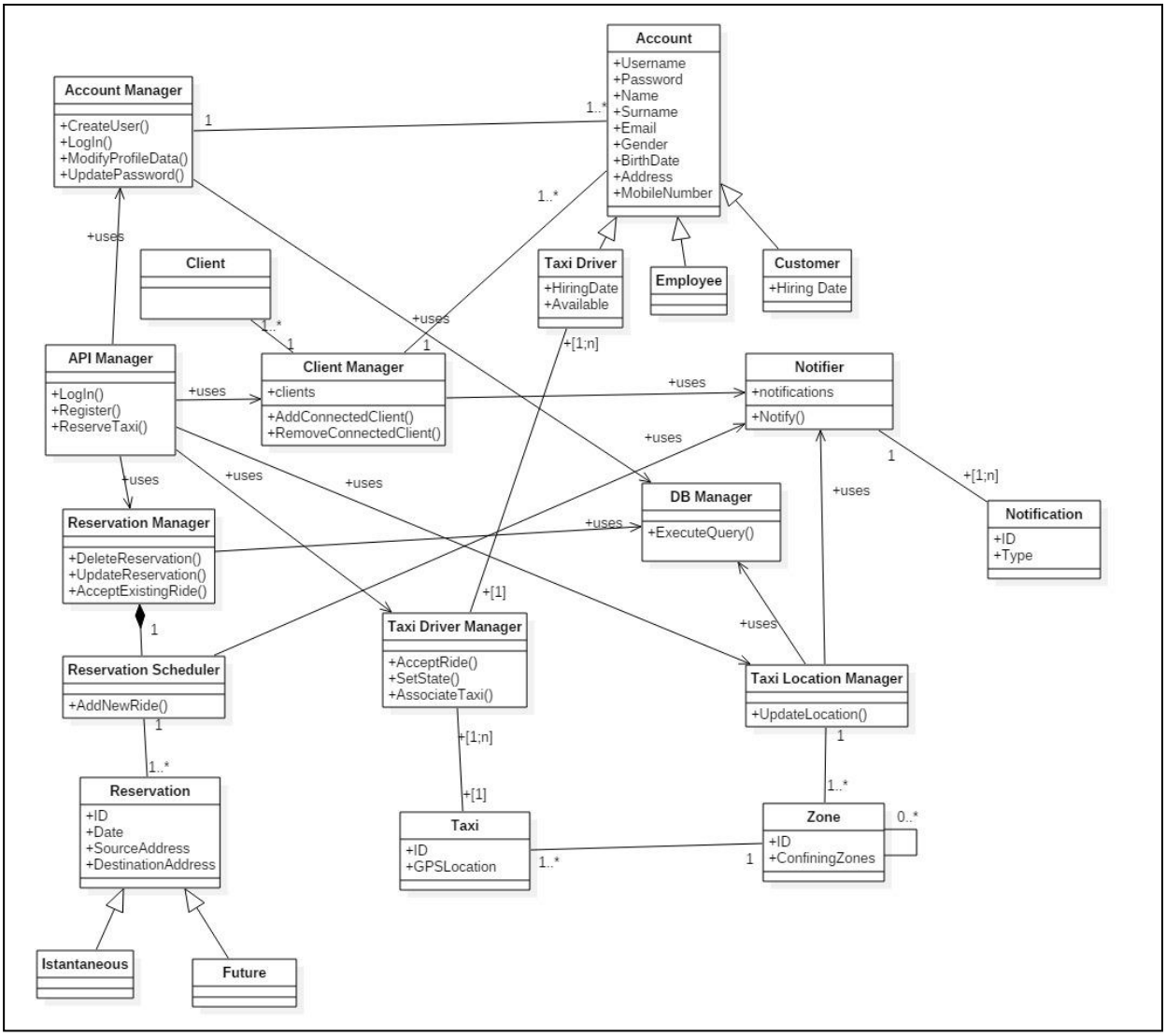
The central system offers a set of APIs, used by the clients to communicate with the central system.

The call to these APIs is synchronous, but, for istance, the reservation process has an asynchronous part(selection of taxi driver). For this reason, all the clients must be able to manage notifications from the central system.



2.c component view

Before showing the components of the system in detail we want to show here the logical objects and relations between them and the main methods and attributes of each one of them.



With the following sections of 4.c section we will show the components and subsystems of the central system. We won't specify the components of the clients because,as we will say in 4.G, they are thin clients and therefore they don't have complex logic or components to model.

2.c.1 Central System

The components are:

2.c.1.1 DB Manager

Overview

It manages all the operations with the DBs. This component exposes the execute query interface, which is used to perform querys to the DBs.

Interfaces

- Void ExecuteQuery(String query): Execute the specified query in the DB

2.c.1.2 Reservation System

Overview

This component manages all the reservations made by the client. To do so it exposes four interfaces: accept new ride, delete reservation, update reservation, accept existing ride(used to notify the reservation manager that a taxi driver has took care of an existing ride). It has three subcomponents: the reservation manager takes care of all the reservations and as soon as a reservation arrives, it adds its to the reservation scheduler. If the reservation is instantaneous, the reservation scheduler will take care immediately of the reservation. Otherwise, if it's a future reservation, this will be put in the third component which is the reservation list. The full detailed functions of this component are described in the algorithmic part.

Interfaces

- Void AnswerNewRide(Reservation newReservation): Accepts a new reservation in the system
- Void DeleteReservation(Reservation existingReservation): Deletes an existing reservation in the system
- Void UpdateReservation(Reservation existingReservation): Updates an existing reservation in the system
- Void AcceptExistingRide(Reservation existingReservation,Account taxiDriver): Associate the specified reservation with the specified taxi driver

2.c.1.3 Account Manager

Overview

This component allows to manage an account. He has the interface create user/logout user.

Interfaces

- Void CreateUser(Account newAccount): Creates the account in the database with the data contained in the specified newAccount
- Void LogInUser(String username,String password): LogsIn the user with the specified credentials.
- Void LogOutUser(String username): Logs out the user with the specified username
- Void ModifyProfileData(String username,Account newAccountData): Updates the profile data in the database of the account with the specified username,with the data contained in the new AccountData

- Void UpdatePassword(String username,String oldPassword,String newPassword): Updates the password of the account with the specified new password,checking that the old password is correct

2.c.1.4 Communication System

Overview

This component is used to communicate with the clients via socket. This component has a subcomponent (Client Manager) that has a list of connected clients.

Interfaces

- Void OpenConnection(): This opens a connection via socket and adds the connected client to the Connected Clients list.
- Void CloseConnection():This closes a connection via socket and removes the connected client to the Connected Clients list.
- Void AssociateClient(String username,Socket socketConnection): This method associate the username with the specified socketConnection
- Void DeassociateClient(String username,Socket socketConnection): This method deassociate the username with the specified socketConnection
- Void SendMessage(SerializableObject object, String username): this method send a message to the client.

2.c.1.5 Taxi driver system:

Overview

This system is used to notify the acceptance of the requests by the taxi driver(using interface acceptance) and to associate a new reservation a taxi(using associated taxi interface). These two operations are effectively made by the taxi driver manager, which has a taxi driver list. He has another interface called set state, that permits to the taxi driver to set the state as occupied, busy, available.

Interfaces

- Void AcceptRide(String username, Reservation reservation)
- Void SetState(String username, State status)
- Void AssociateTaxi(String username, Taxi taxi)

2.c.1.6 API Manager

Overview

This is a component that allows client to interact with the central system and to use all the functionality to have the system working. Clients that want to execute some functions must use this component (this is the public interface-API's)

Interfaces

All the APIs of the API Manager component will be fully explained in the component interfaces section.

2.c.1.7. Taxi Positioning System

Overview

This component is made to update the location of each taxi and to know where exactly they are. This is done by the taxi location manager, which has a list of zones, that have at their time a list of taxis inside. The only interface to do so is the update location interface.

Interfaces

- Void UpdateLocation(Taxi taxi, String GPSLocation)
- Taxi AskNearestTaxi(Zone zona)

2.c.1.8 Notification System

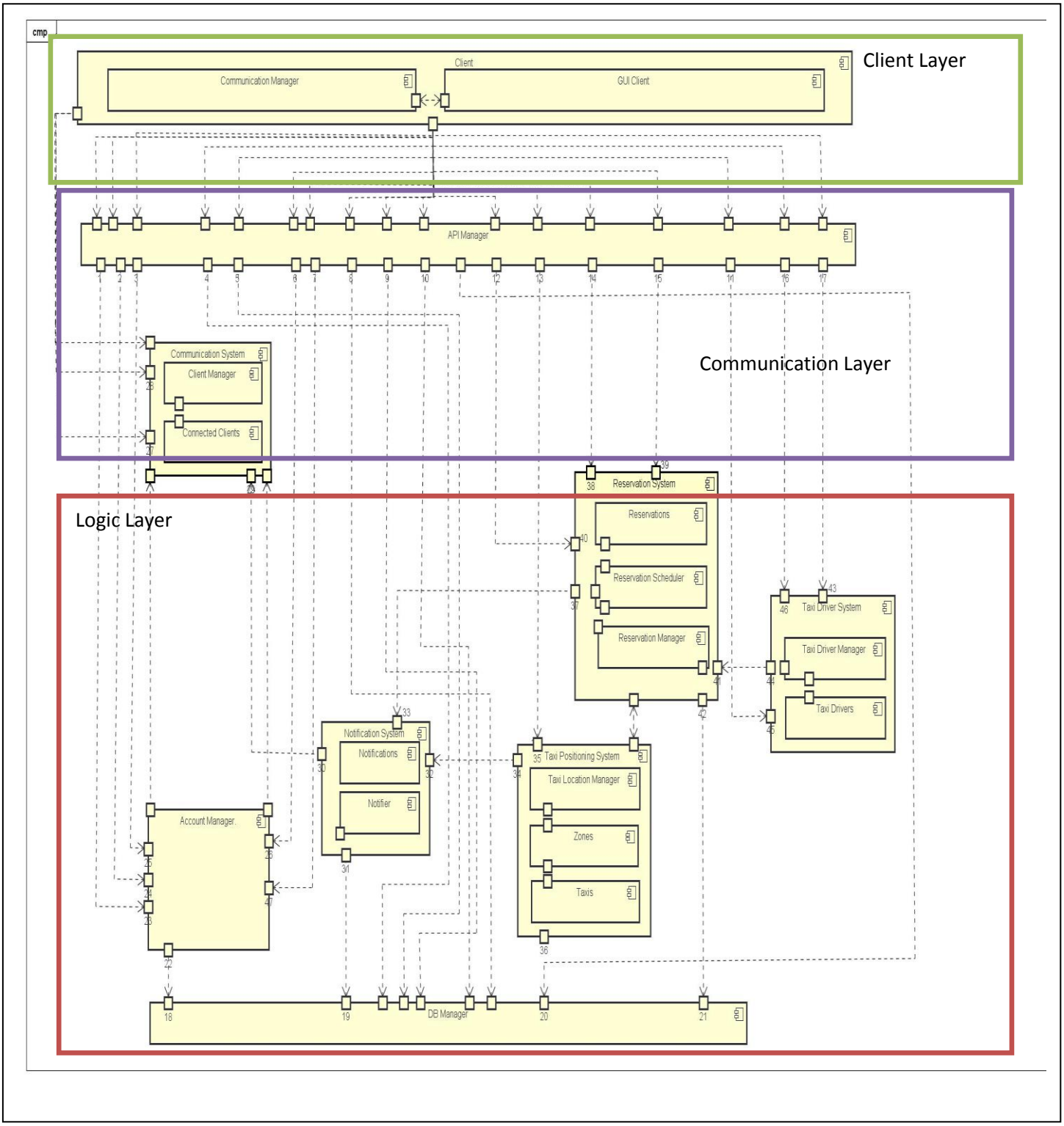
Overview

This component is made to notify to the client any kind of notification that the central system wants to send to the client. This is done by the subcomponent of the notifier that has a list of notifications to be sent. For each notification this component saves this notification in the db using the db manager to keep track of the notifications sent. (this will be useful later on -> see section 7 goal 15)

Interfaces

- Void Notify(Notification notification, String username);

Component view of the central system



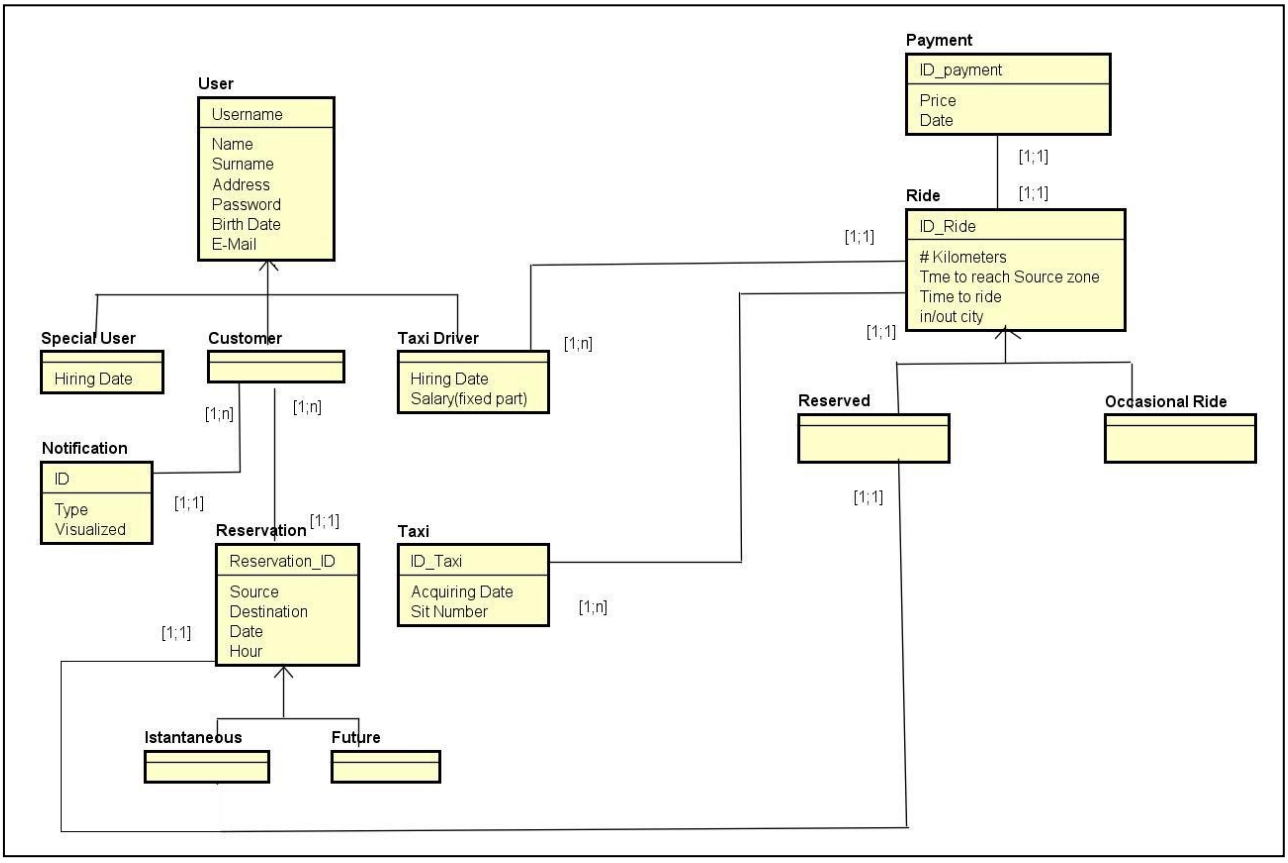
1 UPDATE PASSWORD	17 ANSWER RESERVATION	33 NOTIFY
2 LOGIN	18 EXECUTE QUERY	34 OUTPUT TO NOTIFICATION SYSTEM PORT
3 LOGOUT	19 EXECUTE QUERY	35 UPDATE LOCATION
4 PICK UP THE CLIENT	20 EXECUTE QUERY	36 NO LONGER USED
5 STORE RIDE DATA	21 EXECUTE QUERY	37 OUTPUT TO NOTIFICATION SYSTEM PORT
6 CREATE USER	22 OUTPUT TO DB PORT	38 DELETE RESERVATION
7 REGISTRATION	23 UPDATE PASSWORD	39 UPDATE RESERVATION
8 GET INFORMATION	24 LOGIN USER	40 ACCEPT NEW RIDE
9 ARRIVED CLIENT	25 LOGOUT	41 ACCEPT EXISTING RIDE
10 ARRIVED DESTINATION	26 CREATE USER	42 OUTPUT TO DB PORT
11 SET STATE	27 OPEN CONNECTION	43 ACCEPT RIDE
12 TAXI RESERVATION	28 CLOSE CONNECTION	44 OUTPUT TO RESERVATION SYSTEM PORT
13 UPDATE GPS LOCATION	29 SEND MESSAGE	45 SET STATE
14 DELETE RESERVATION	30 SEND MESSAGE	46 ASSOCIATE TAXI
15 UPDATE RESERVATION	31 OUTPUT TO DB PORT	47 REGISTRATION
16 CHOOSE TAXI	32 NOTIFY	

We decided to use simple arrows instead of the notational --)-- of Astah since using that symbol the diagram was more confusionary. The interfaces names are written in the table for an order matter.

So → is equivalent to --)--.

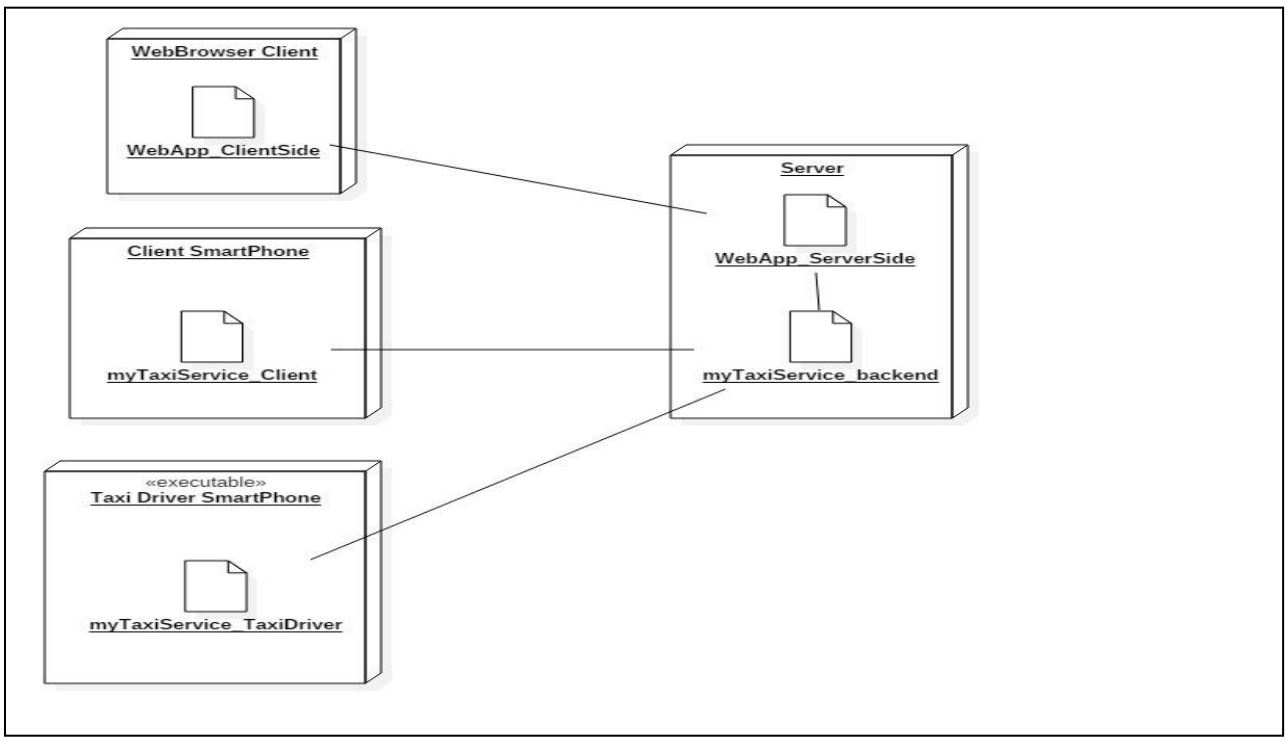
2.c.2 Database

In this section we want to give an high level ER design to give the idea of the main entities that will be present in the database once implemented. Of course these are not all the entities that there will be once the system will be effectively implemented.



2.d Deployment view

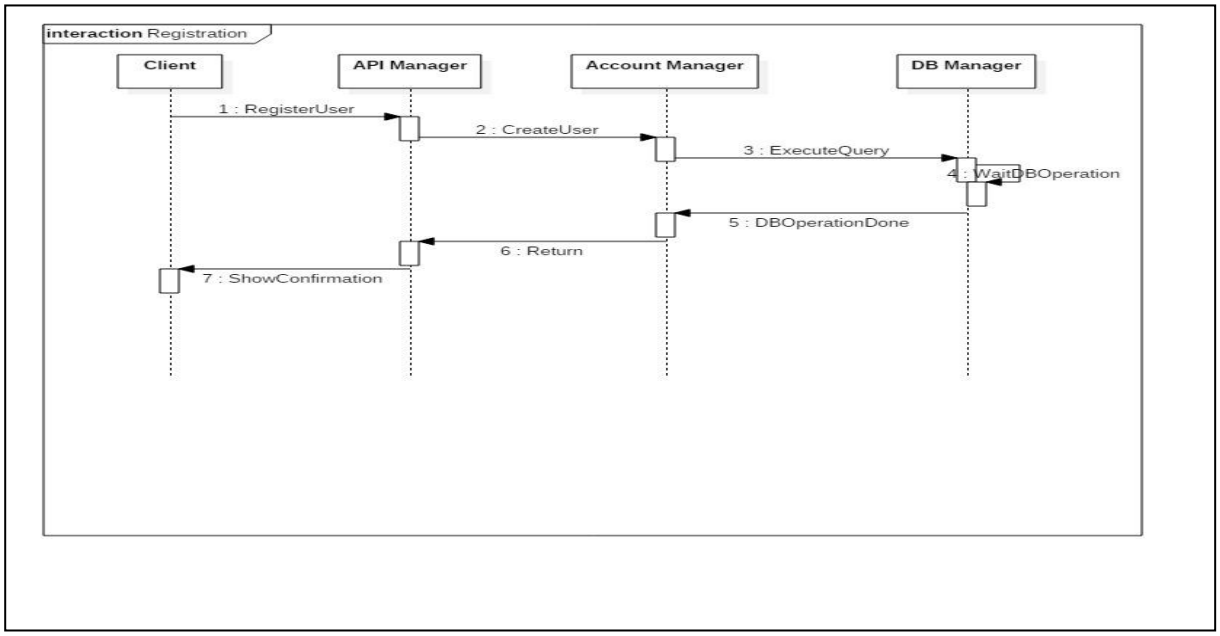
The system will be composed by five different artifacts. The first is the app of the taxi driver (myTaxiService_TaxiDriver) that will be deployed on his smartphone to allow him to do his work. The second is the client app installed on the client smartphone (myTaxiService_Client). The third and the fourth are two artifacts installed on the central system: the backend app (myTaxiService_backend), used to give services by the exposed APIs, and the backend web app (WebApp_ServerSide), used to define the logical part of the client and to elaborate the GUI server - side. The fifth and last one is the web app_ClientSide, used for the presentation client-side.



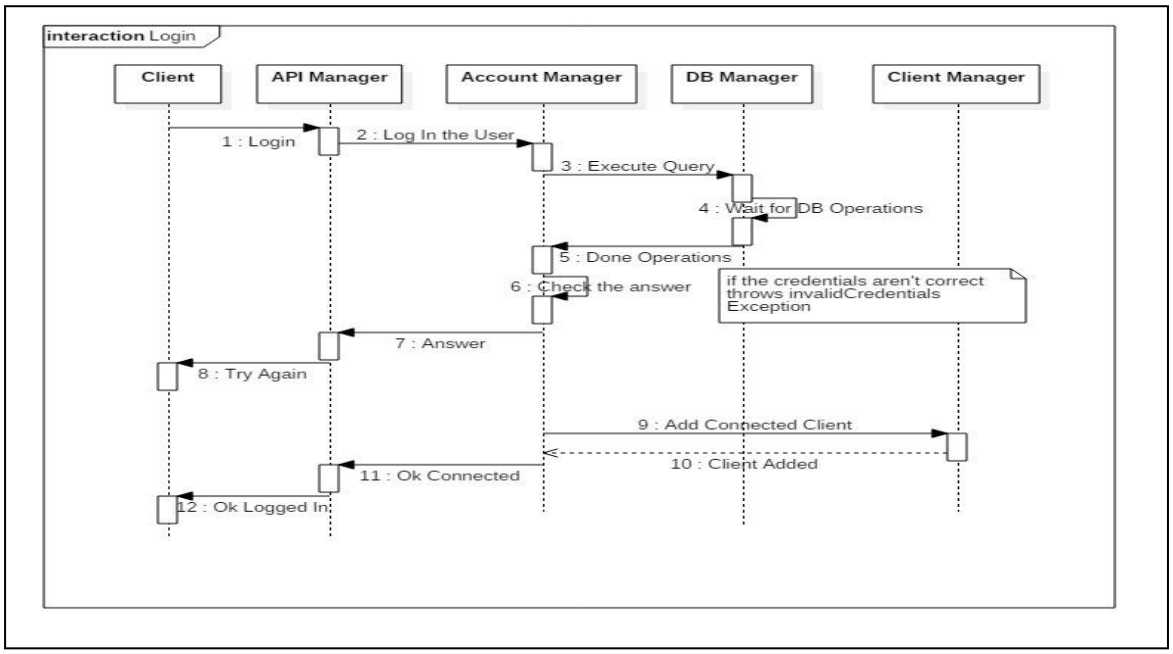
2.e Run Time view

In this section we want to give a more detailed vision about the sequence diagrams we already put in the RASD,showing how components interacts between each other. In each sequence diagram we suppose that all the operations go fine. In case of error the behavior was already descripted in the sequence diagrams of the RASD document.

Registration

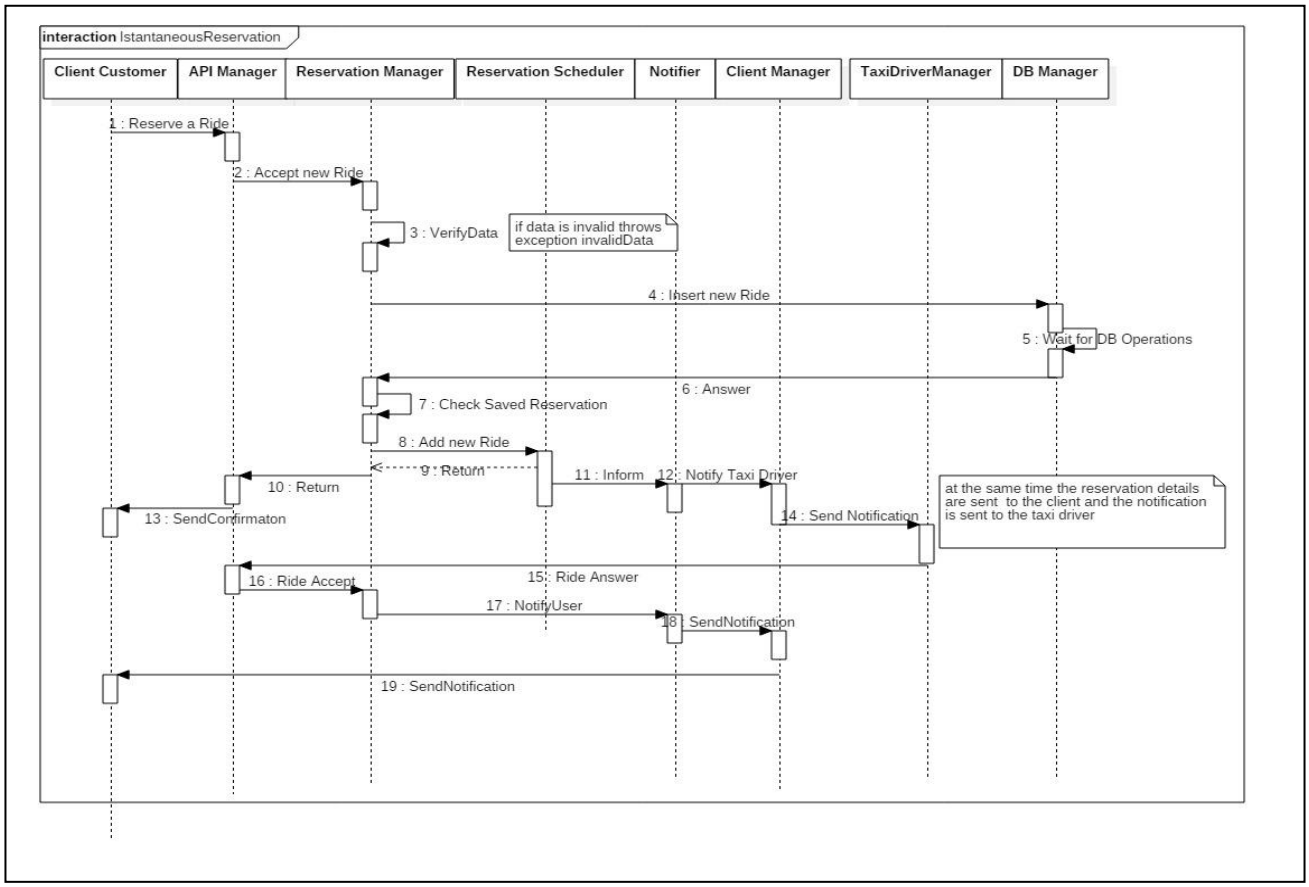


Login

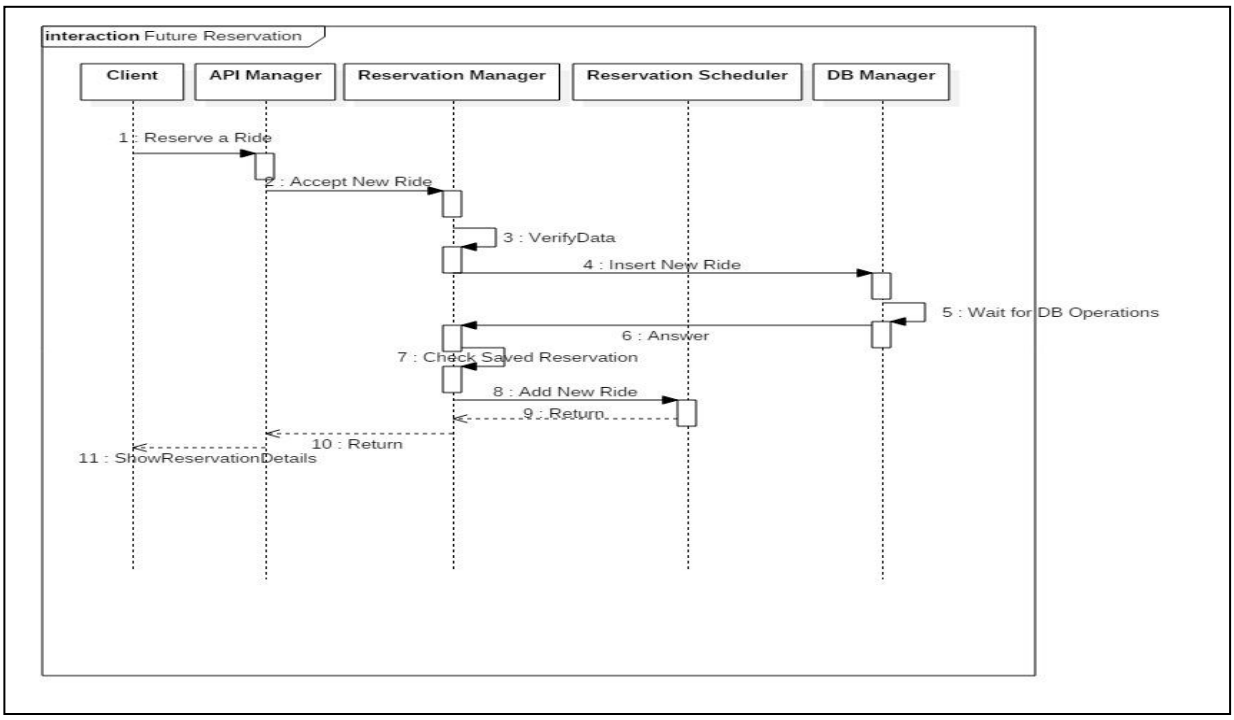


Istantaneous Reservation

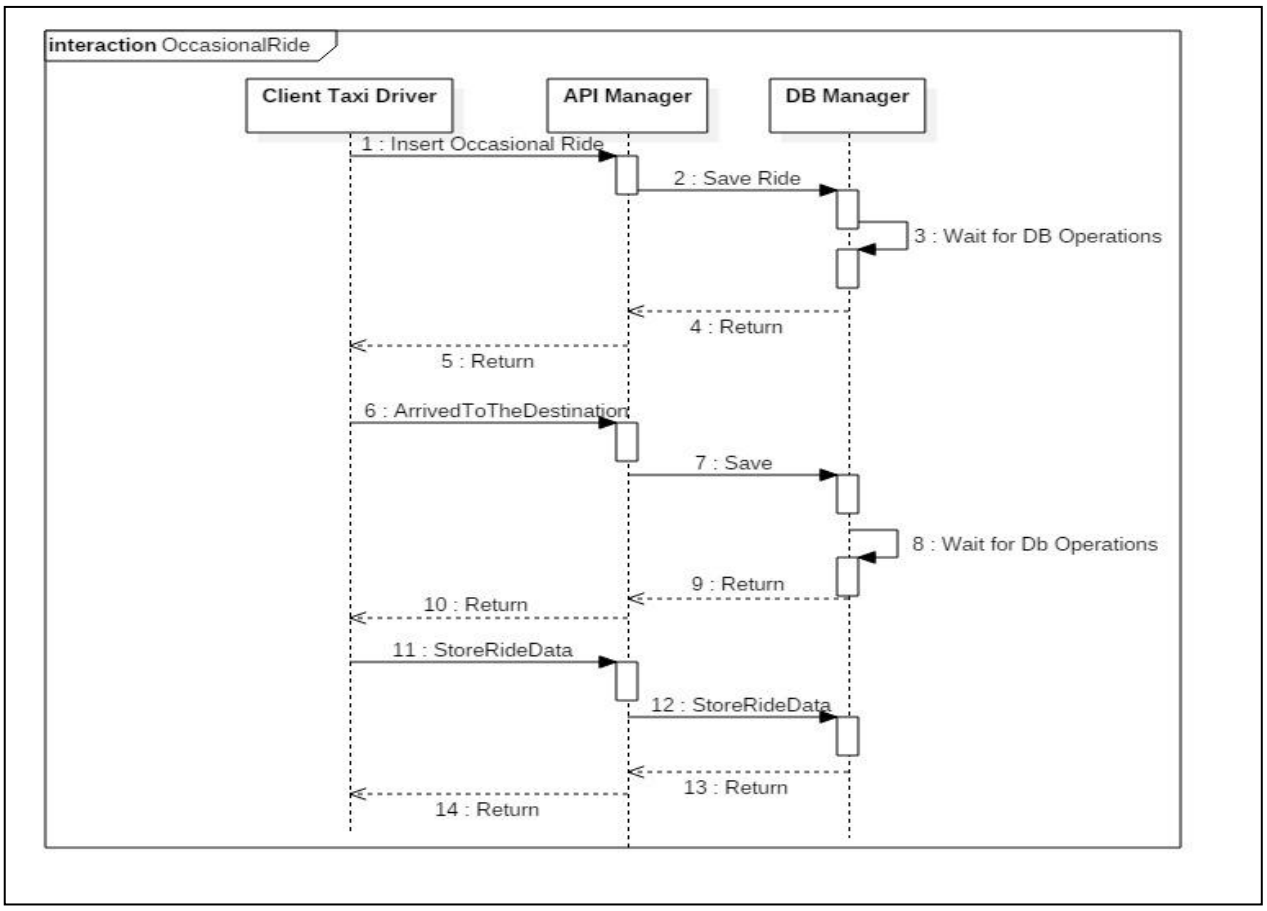
This sequence diagram describes how a client can make a reservation and it ends when the system has taken care correctly of the reservation.



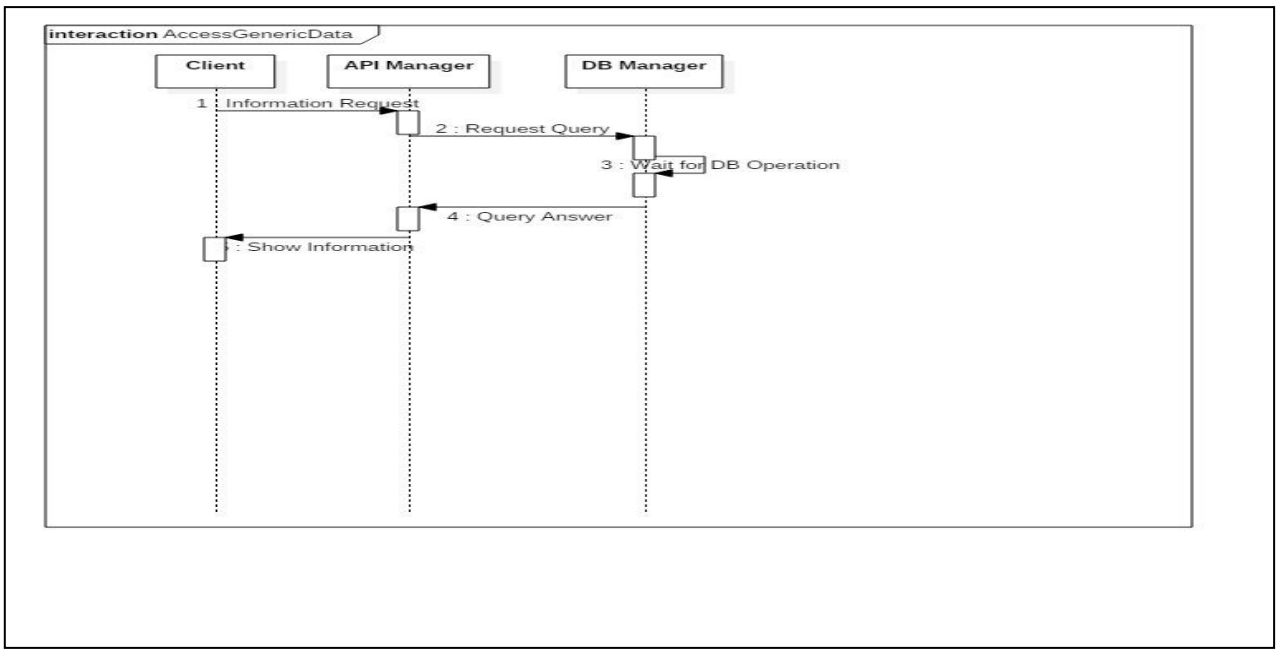
Future Reservation



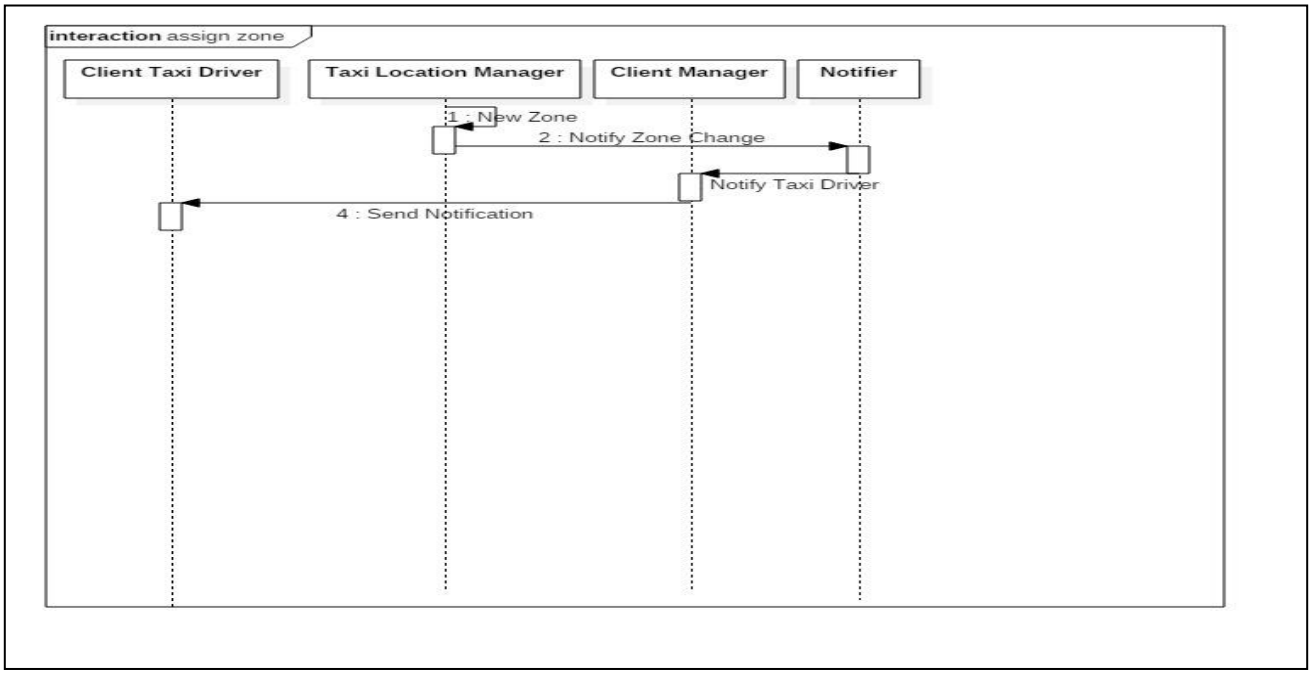
Occasional Ride



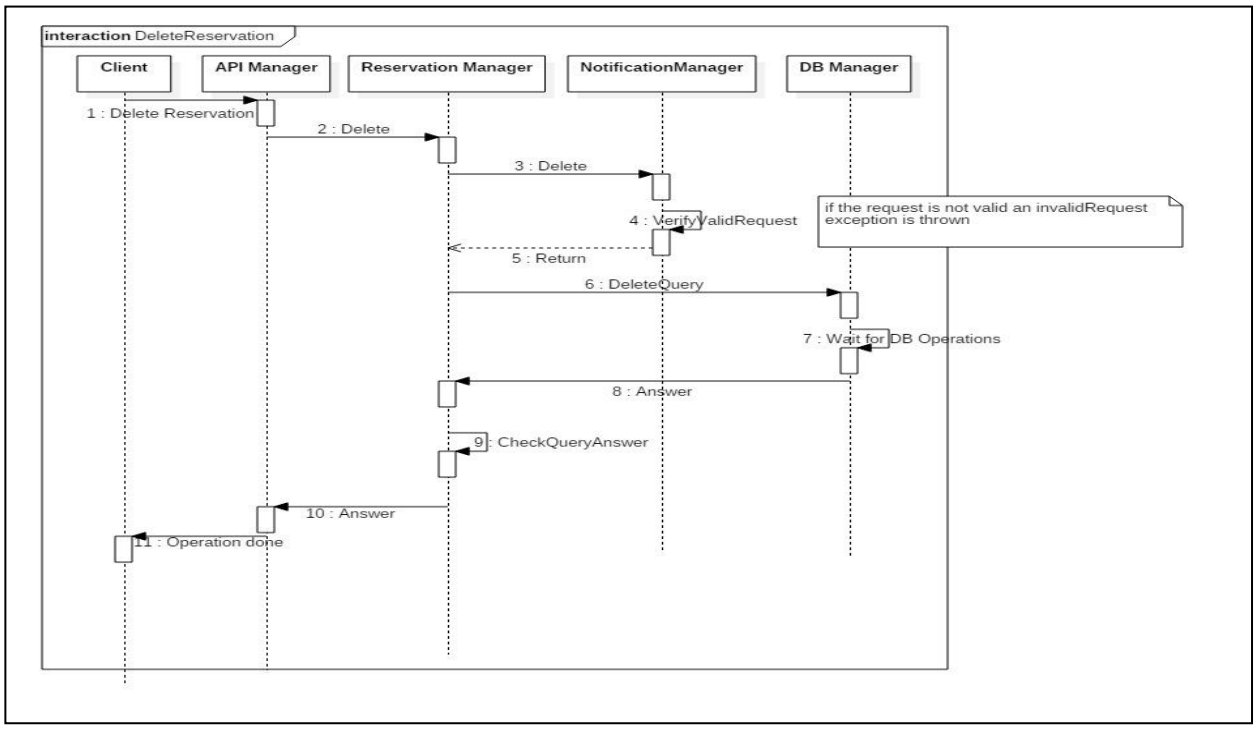
Access Generic Data



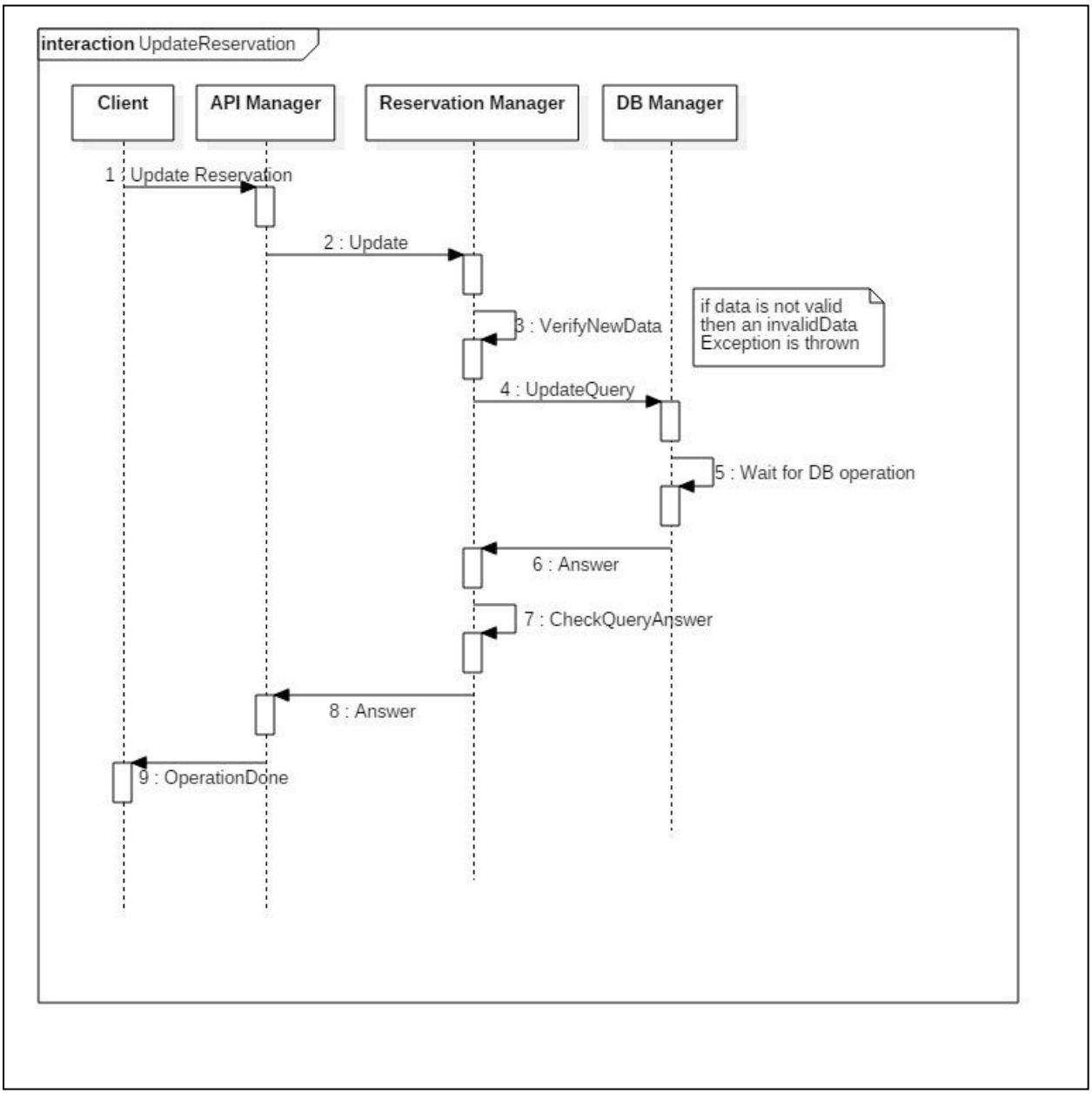
Assign Zone



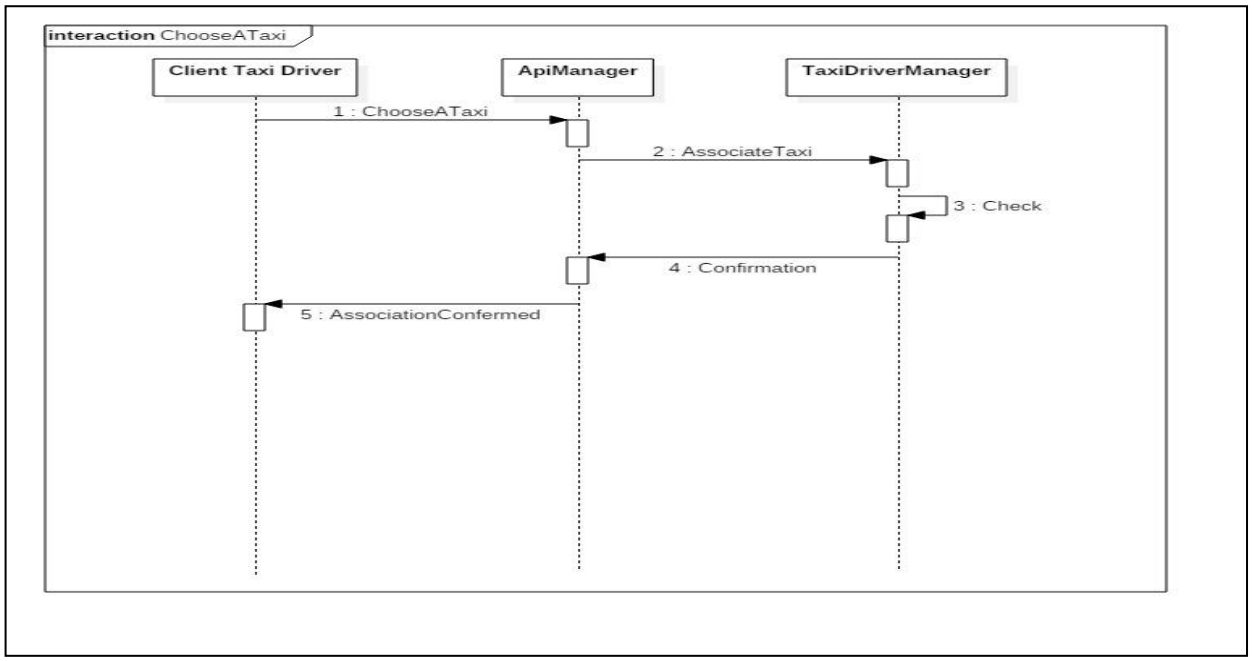
Delete Reservation



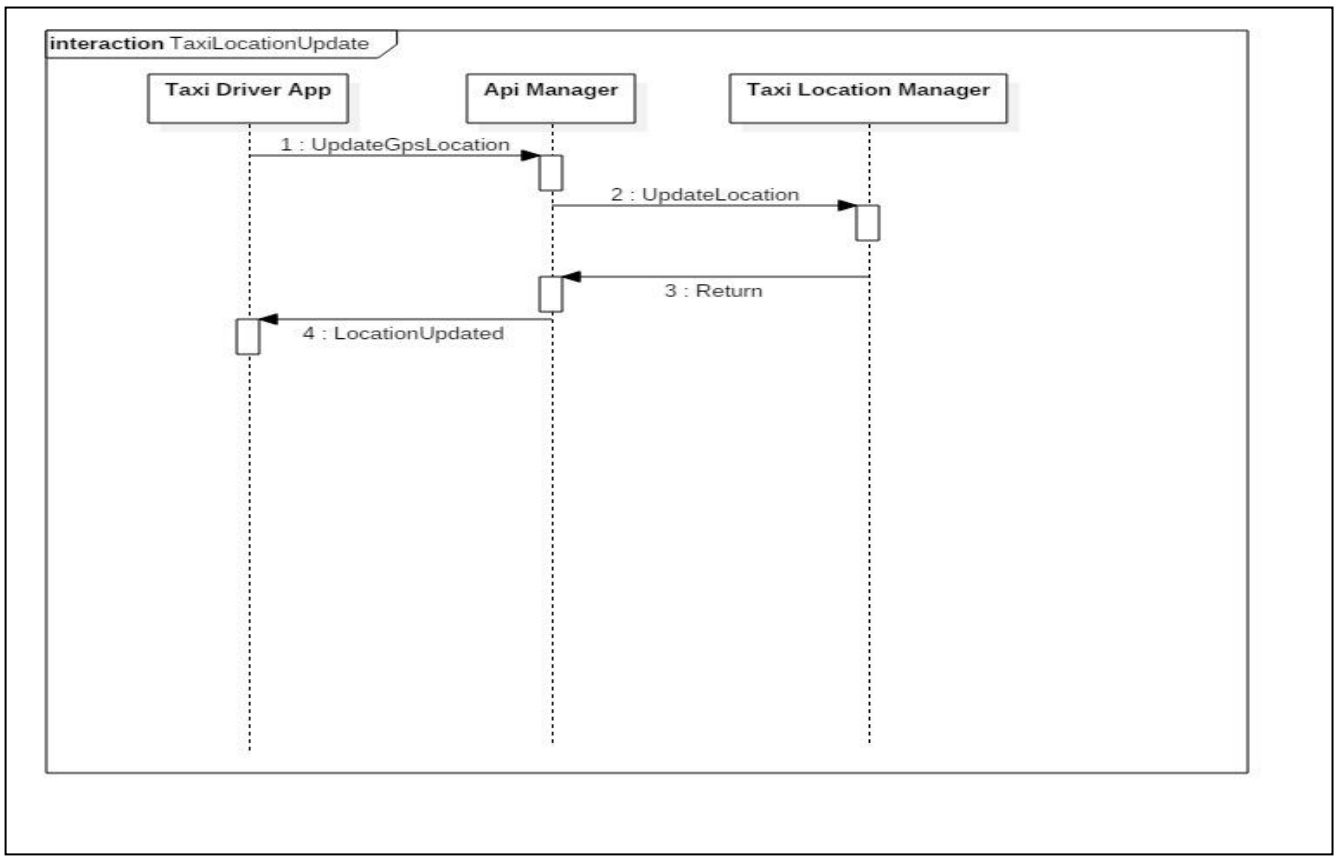
Update Reservation



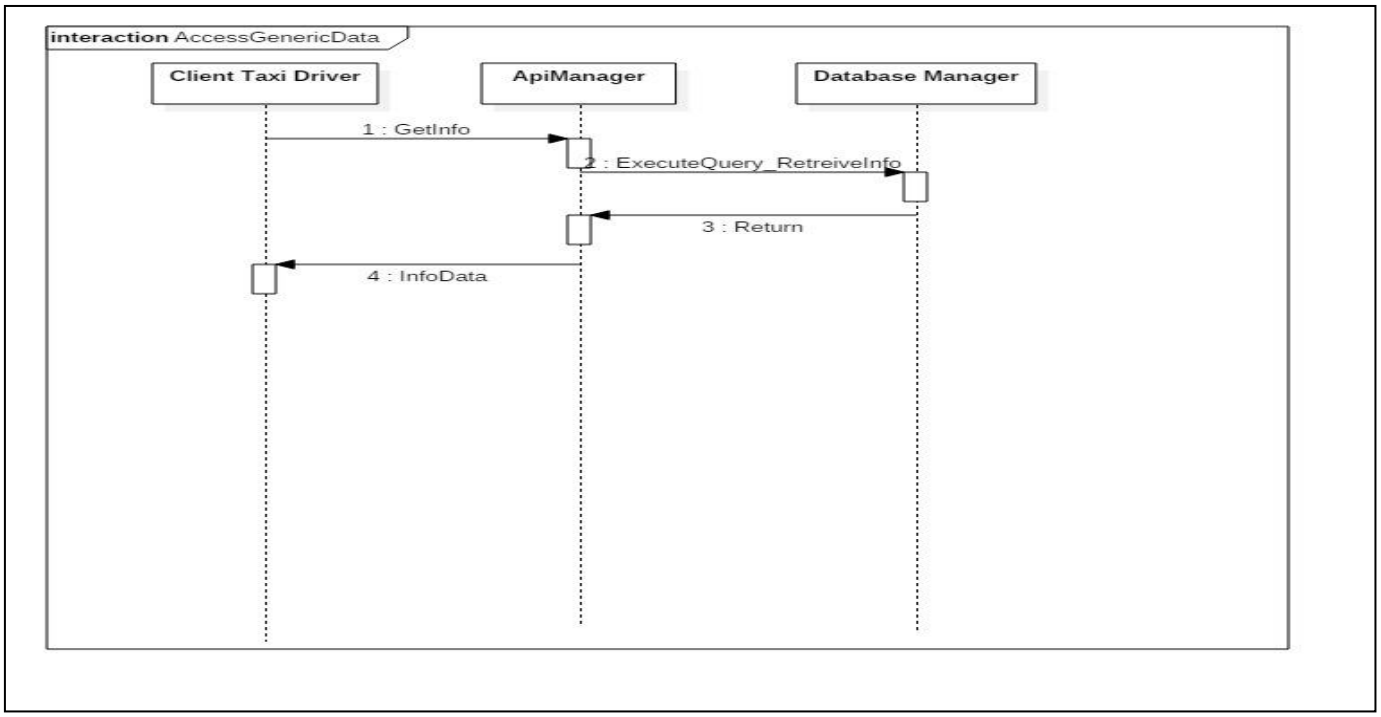
Choose a Taxi



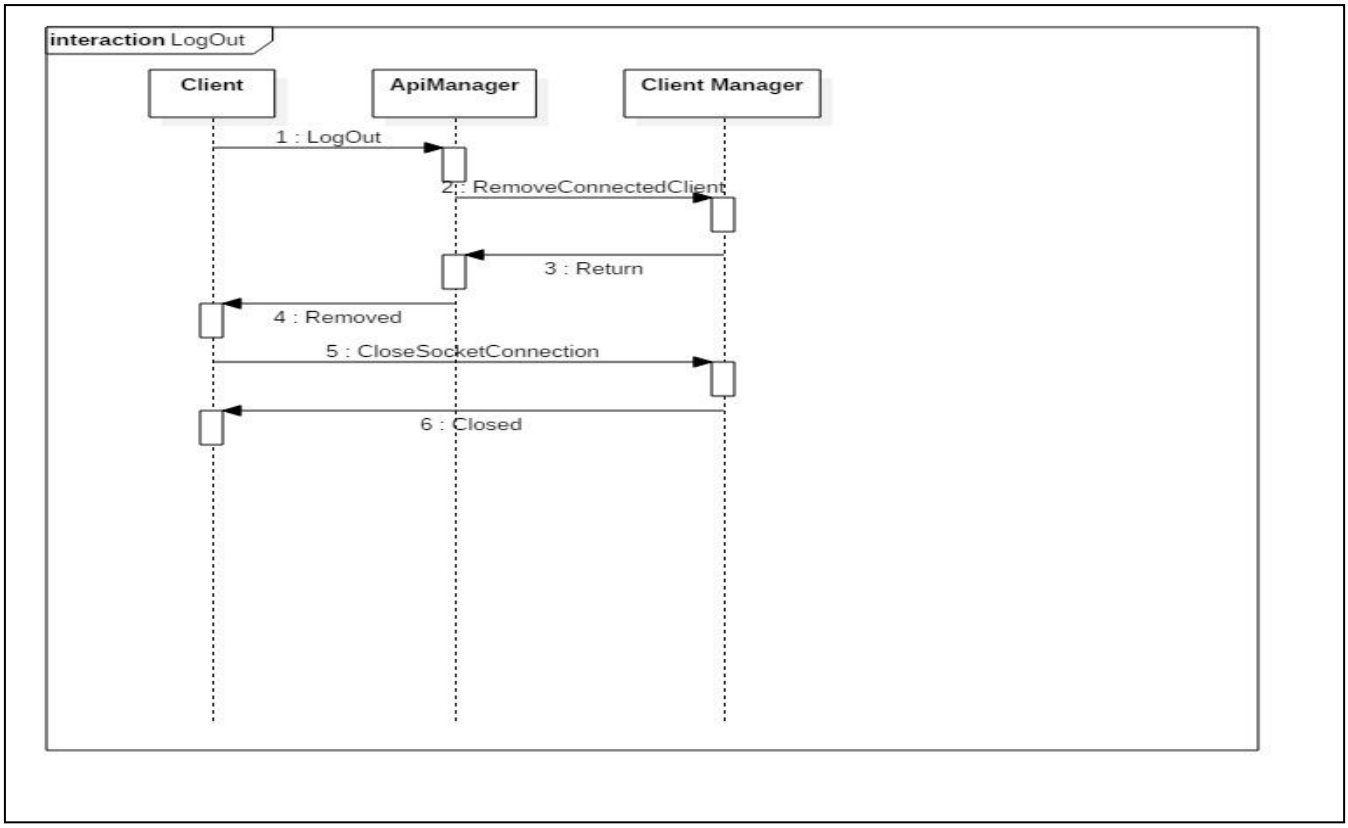
Taxi Location Update



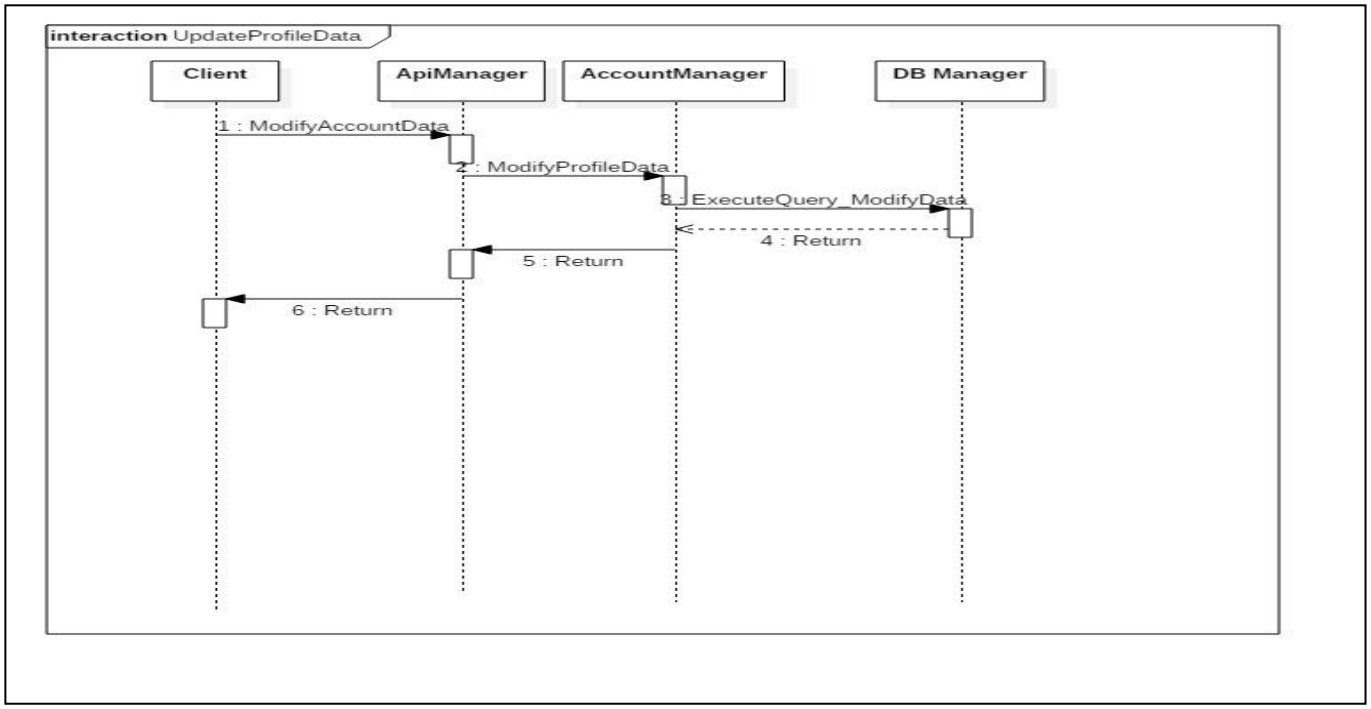
Access Generic Data



Logout

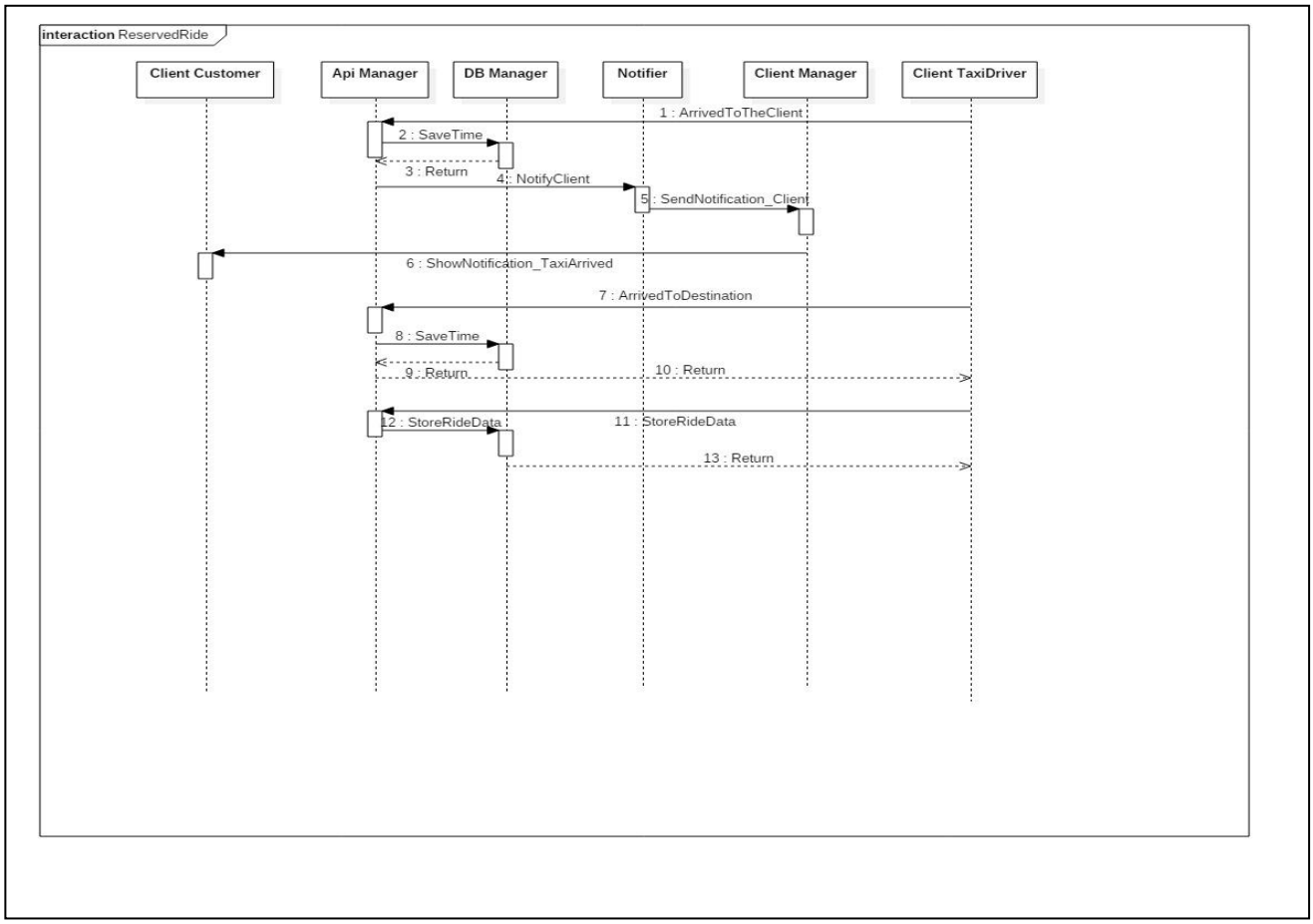


Update Profile Data



Reserved Ride

This is the sequence diagram that describes what happens when a taxi driver has accepted a ride. It comes after the instantaneous reservation.



2.f Component Interfaces

These are all the APIs exposed by the logic server in the cloud (and referring to the component view by the API manager component):

- TAXI RESERVATION < username **string**;gps source position **string**;destination **string**;[reservation time **date time**]> (reservation time is an optional parameter)
- LOGIN<username **string**; password **string**; [facebook_name **string**];[facebook_password **string**]>
- REGISTRATION<name **string**; surname **string**; gender **enum(m,f,ns)**; e_mail **string**; birth_date **date time**; username **string**; password **string**; [address **string**]; [mobile phone **string**]>
- LOGOUT<username **string** >
- DELETE_PRENOTATION<username **string**, password **string**, id_prenot **int**>
- UPDATE _PRENOTATION<username **string**, password **string**, id_prenot **int** ; data **date time**; time **date time**; source **string**;destionation **string** >
- UPDATE_GPS_LOCATION<username **string**; gps_location **string** >
- UPDATE_PASSWORD<username **string**; old_password **string**; new_password **string** >
- TAXI_SHARING_ADD_USER_TO_RIDE<username **string**; id_ride **int**>
- GET_INFO<type **int** , username **string**, password **string** >
- OCCASIONAL_TAXI_RESERVATION<gps_source_position **string**, destination **string** >
- CHOOSE_TAXI<username **string**,taxi_Id **int**>
- MODIFY_PROFILE_DATA<username **string**, name **string**, surname **string**, address... **string** >
- PICK_UP_THE_CLIENT<username **string**, id_ride **int**>
- ARRIVED_TO_THE_CLIENT<username **string**, id_ride **int** , time **int**>
- ARRIVED_TO_THE_DESTINATION<username **string**, id_ride **int**, time **int**>
- STORE_RIDE_DATA<username **string**, id_ride **int**, km_done **float**, price **float**>
- ANSWER_RESERVATION<username **string**, id_ride **int** , answer **boolean**>
- SET_STATE<username **string**, availability **enum(available, occupied, busy)**>

Here there's the explanation of the previouses APIs:

- **Taxi reservation**: it's used to make a new reservation specifying the source position, the destination and, if the destination is future, even the reservation time. Otherwise, if the reservation is instantaneous, this parameter is not required.
- **Login**: it's used to log the client in the system. The facebook name and the facebook password are parameters used for the future possible implementation when facebook login will be available.
- **Registration**: this allows a client to be registered in the system specifying all the non optional parameters(those not closed in squared brackets)
- **Logout**: this allows the user to log out.
- **Delete reservation**: This allows user to delete a reservation, specifying the reservation ID.

- **Update reservation**: this allows user to update reservation, specifying the reservation ID, the new data, time, source and destination.
- **Update GPS location**: this is used by the taxi driver app in background to update the taxi location.
- **Update password**: this is used by the client to update the password specifying the old and the new password.
- **Taxi sharing add user**: this is for future possible implementations and it won't be at first stage implemented in our system, but it will be used to share a taxi with other users.
- **Get Info**: this is used to retrieve all personal information by the client such as account data, employee information, stats… Each kind of information is identified by an unique type specified by the type value (is an integer). For example Type = 0 means to see notifications in my account,Type=1 means to see the past reserved rides and so on.
- **Occasional taxi reservation**: this is used by the taxi driver to register an occasional ride specifying source and destination.
- **Choose taxi**: this is used by the taxi driver to specify the taxi he selected to work with at the start of the day.
- **Modify profile data**: this is used by the client to update his profile information.
- **Pick up the client**: this is useful for the future possible implementation of taxi sharing. It permits to the taxi driver to sign in the app a customer he picked up on his car.
- **Arrived to the client**: this is used by the taxi driver to inform the central system that he has arrived to the client and consequently notify the client about this.
- **Arrived to the destination**: this is used by the taxi driver to inform the central system that he has arrived to destination and consequently notify the client about this. In this API it's added also the time of the ride.
- **Store ride date**: this is used to store information of a ride.
- **Answer reservation:** this is used by the taxi driver to accept or deny a reservation. The answer parameter is a Boolean parameter: if true means that the taxi driver has accepted the reservation, false the opposite.
- **Set state:** this is used by the taxi driver to update his availability state. This is done with the availability parameter, that could be busy, occupied, available. These parameter is an enum.

2.g Selected Architectural design and patterns

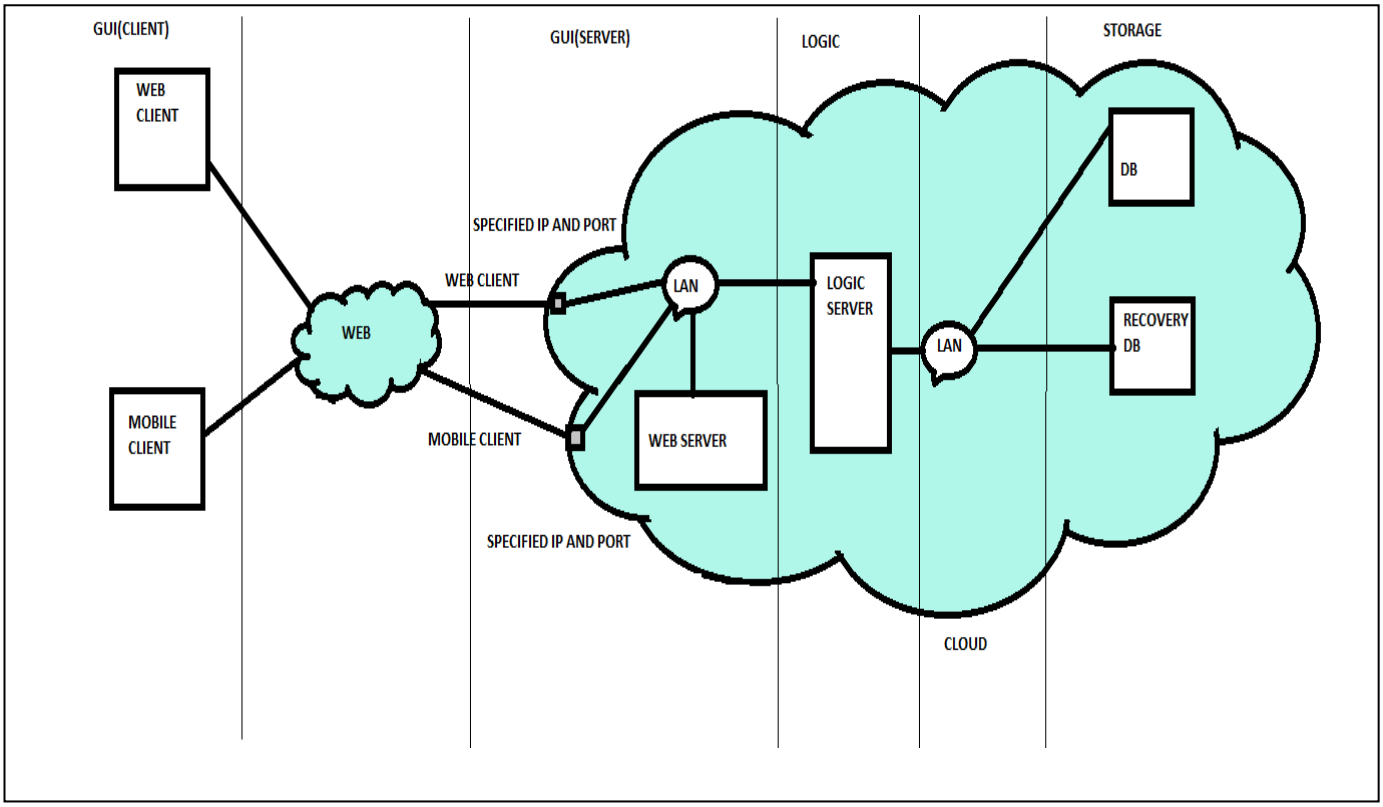
2.g.1 Patterns

For the logic class diagram we decided to use singleton pattern in order to have a single instance of this object and it will be easier to other classes to use them. The notifier class instead has some static methods that guarantee the notification services.

2.g.2 Architectural Styles

Since this system will be used by a large city we suppose that there will be an intense traffic by the clients, but also from the taxi drivers that must periodically update their location with the mobile app. Therefore, to guarantee a high quality and scalable service, we decided to use the cloud style in IAAS configuration to allow us to build the system as we wanted. All the client's applications are thin clients because all the logic will be in the central system developed in the cloud. Also the data will be stored in the cloud. For the APIs communication we use the SOA model with the REST protocol. Doing so the answers of the APIs will be in XML, which is a common Standard and will allow other applications to interact with our system(interoperability). To allow the notifications from the server all the clients are connected via socket and they can easily interact by sending some strings. This will also allow interoperability and extensibility.

Here we provide the graphical representation of the system:



With this representation we have a two tier application since the clients connect to the cloud and therefore there are two layers: the clients and the cloud. There is a different cloud structure between the web client and the mobile client caused to the fact that they work differently. The web client need a web server in the cloud that elaborate some part of the presentation(for example when the user logs in we wants to give a

personalized log in page, showing his personal information). The mobile client instead has already the presentation layer implemented inside the app and therefore it needs only the data to elaborate the interface to show to the user. For this reason we have two different public ports in the cloud, one for the mobile and one for the web. The mobile app connects directly to the central system(logic part), instead web client connects first to the web server that will connect themselves to the logic server. So the cloud structure for the mobile app is a two tier cloud structure(logic+storage), instead, for the web app, it is a three tier cloud structure representation(presentation+logic+storage).

2.h Other Design Decisions

Due to the fact that this is an high level design we only give some possible softwares and operating systems that can be installed in the system to let the system work. On the web server and the central server we decided to install windows server and to give to the web client the presentations page to show we have an ASP.net application and in the central system we have a web service also written in ASP.net to expose the apis. All these decisions were taken to the fact that windows server is reliable and the .net platform can run even on other operating system and therefore we can change operating system in a second time.
We decided also that the mobile version of the mobile app must be available to all the principal mobile stores (Apple,Microsoft,Google) to guarantee that most of the people can use our system.

3. Algorithm design

Queues Management

Each zone has associated a taxi list and these lists have a different size depending on the number of taxis in each zone.
These zones have also a desired number of taxis that can change in these time slots:
6:00 – 10:00 ;

10:00 – 14:00;

14:00 – 17:00;

17:00 -20:00;

20:00 – 00:00

At the start of each time slot the TaxiLocationManager reads the reservations made in the past from the db and basing on this data ,with probablistic calculations,sets the desidered number of taxis for each zone.
The following are the most important methods to manage this logic written in pseudo-code:

```
//this method is called when the specified taxi has changed his postion
private void TaxiLocationChanged(Taxi taxi,GPSPosition newPosition)
{
    //Searches the zone where the taxi is currently
    Zone oldZone = FindZone(taxi);

    //Given the specified position it returns the zone that contains that position
    Zone newZone = FindZone(newPosition);

    //DA METTERE GESTIONE LISTA FUORI CITTA
    //IN case of external zone it returns a zone with id -1

    if(oldZone.ID !=newZone.ID)
    {
        //Removes the taxi from the old zone
        oldZone.Queues.Remove(taxi);

        //Insert the taxi in the new zone
        newZone.Queues.Insert(taxi);
    }


}

//This is the method used to update the desired taxi number at the start of a
//time slot of the day
private void UpdateDesiredTaxiNumber()
{
    //Read past reservations
    PastReservations= ReadPastReservationsFromDB();

    //This sets the desired number for each zone based on the probabilistic function
    SetDesideredNumberPerZone(ProbabilisticFunction(PastReservations));

}
```

Scheduler Management

Taxis are all inside of a taxi queue.

When a request arrives the algorithm identifies the zone from where the reservation has made. After this i scan the list of that zone selecting the first available taxi. If all the taxis in that zone are occupied, i go on with the BFS algorithm to search in the nearest zones from the source one (each zone a list of confining zones).A taxi driver can deny a client reservation and in this case his taxi goes in the bottom of the list of the taxi's zone,instead if he accepts his taxi will stay in the same position of the list. When a taxi changes its zone,that taxi is removed from the previous zone's queue and it is inserted in the last position of the new zone's queue.

The following are the main methods that implement in pseudo-code this logic:

```
// this is the method call each minute by the scheduler
private void SchedulerTick()
{

    //Checks if the reservation list is not empty
    if(Reservations.Lenght>0)
    {
```

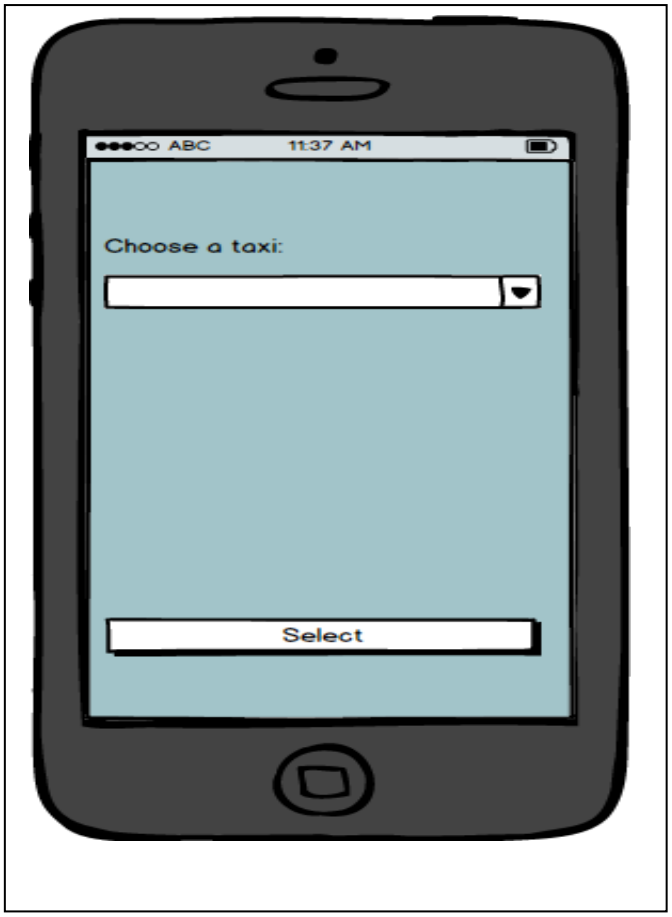
```
        //Checks if the currentDateTime of the first is 10 minutes or less before
      the reservation specified date and time
        //Note1: DateSubtract(DATE1,DATE2,TYPEOFDIFFERENCE)-> TYPEOFDIFFERENCE can
        be minutes,hours,month..
        //NOTE2: The Reservation list is ordered in a crescent way by the date time
        of the reservation
        if(DateSubtract(Reservations[0].Date,CurrentDateTime,minutes)<=10)
        {
            //This method starts the thread that manages an instantaneous
            reservation
            StartThreadInstReservation(new InstantaneousTask(Reservation));

            //Removes the reservation from the ReservationsList
            ReservationList.Remove(Reservation);

        }
    }

}


 //Manage the current reservation --> this method is in the instantaneousTask class
and we suppose that in this class we have an attribute that has the reservation to
manage
 private void ManageInstReserv()
 {

    //Finds the zone from where the request comes from
    Zone sourceZone = FindSourceZone(req);

    //All The taxi driver that have been contacted and denied
    List<TaxiDriver> AlreadyContactedTaxiDrivers;

    bool accepted = false;
    TaxiDriver selectedTaxiDriver = null;

    foreach(TaxiDriver currentTaxiDriver in TaxiDrivers)
    {
      //It finds the nearest taxi driver starting from the first available driver in
      the source zone using BFS,excluding the already contacted taxi drivers
      TaxiDriver taxiAvailable=
      FindNearestFreeTaxiDriver(AlreadyContactedTaxiDrivers,sourceZone);

        //Notifies the taxi driver to take care of the request
        Notifier.Notify(req);

        //Waits to let the taxi driver accept or deny the request
        //Returns true if he accepts,false if timeout or deny
        accepted = WaitForAcceptanceOrTimeout();

        if (accepted)
        {
            selectedTaxiDriver = taxiDriver;
            break;
        }
        else
        {
            //I move the taxi associated to the taxi driver to the last position of
            the queue
            Taxi taxi = FindTaxiAssociated(currentTaxiDriver);
            MoveTaxiToSpecifiedPosition(taxi, sourceZone.Queues.Lenght - 1);
        }
    }
```

```
        //Notifies the reservation manager the selected driver (if null,it means that
        //no taxi driver accepted the reservation)
        NotifyReservationManager(selectedTaxiDriver);



    }
```

4. User Interface Design

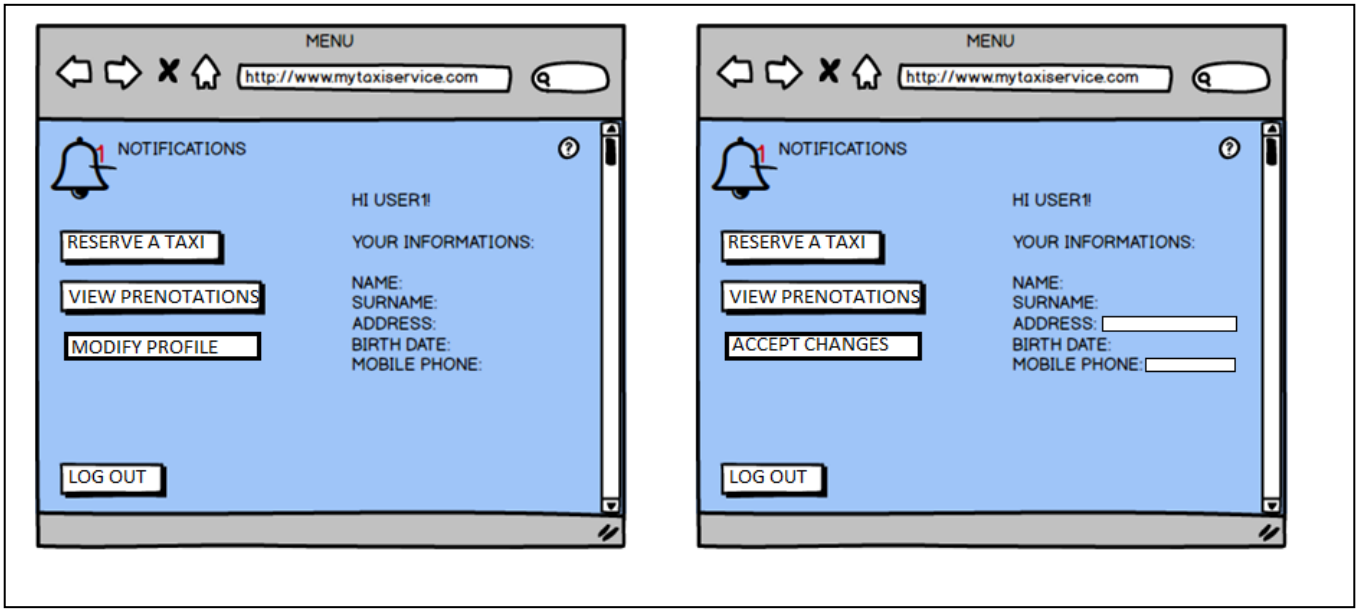
All the mockups have been already added in the RASD document in the user interfaces section. These other interfaces came up with a deeper analysis of the system.

Choose Taxi



Error

Modify Account Information



5. Requirements traceability

In this section we will explain how and the goals and their relative requirements are satisfied with the interfaces components we defined in this document.

1. **A user can login the system by writing his credentials(username and password):** this is accomplished with the login interface exposed by the API manager.
2. **A user can register himself to the system if he hasn't an account:** this is accomplished with the registration interface exposed by the API manager.
3. **A user can book an instantaneous taxi ride:** this is accomplished with the taxi reservation interface exposed by the API manager. Also this will be accomplished even by the reservation system in the central system.
4. **A user can book a future taxi ride:** this is accomplished with the taxi reservation interface exposed by the API manager. Also this will be accomplished even by the reservation system in the central system.
5. **A user can cancel a previous reserved taxi ride:** this is accomplished with the delete reservation interface exposed by the API manager.

6. **A user can view his previous reservations+A user can see his personal info+The taxi driver can see his past served rides+The taxi driver can view how much he has already earned this month as**

variable month salary+The employee of the government can view the service stats+The employee of the government can view the worker stats: All these six goals, that require to retrieve some personal information, are accomplished by using the get information API manager interface. Each one of them is identified by a different type value of the parameter.Of course the information will be shown in different interfaces.

7. **A user can view his position and the incoming taxi driver position:** this is accomplished by the client manager and taxi location manager in this way: whenever the taxi position changes the mobile application of the taxi driver sends a message that will be processed by the taxi location manager and the client manager sends the updated position to the customer.
8. **When the system has assigned to the customer a taxi, the mobile app must notify this fact to the user, so he can view how much time he will have for the taxi:** this is accomplished by the notifier component and the taxi driver manager system. As soon as the taxi driver accepts a reservation, the taxi driver manager uses the notifier to notify the customer.
9. **A user can update a previous made reservation:** this is accomplished by the update reservation interface of the API manager.
10. **A user sees the notifications of incoming taxi when he opens the web page:** When the central system wants to notify a client,the notifier saves this notification in the database so when a client needs to see all his notifications it just call the GetInformations API Manager's interface with type =0 as parameter.
11. **A taxi driver can accept or deny the customer requests:** this is accomplished by the answer reservation interface exposed by the API manager.
12. **The application must periodically communicate the taxi driver position:** this is accomplished by the updategpslocation interface exposed by the API manager.
13. **The taxi driver can set the occupied/available state:** this is accomplished by the setstate interface exposed by the API manager and by the taxi driver system.
14. **The taxi driver can view the assigned zone by the system where he will have to go once a ride is over:** this is accomplished by the notifier, that sends him a notification if the client is disconnected. Otherwise the client manager will send him a message directly.
15. **The taxi driver can accept an occasional user ride:** this is accomplished by the occ taxi reserved interface exposed by the API manager.

Changes in version 1.1

- Restructured component view for better readability
- Added the parameters and return types for each interface of the components in the component view

WORKING HOURS

Riccardo Giambona 30 hours

Ivan Antozzi 30 hours