Design Document

Michele Madaschi Lidia Moioli Luca Martinazzi

December 3, 2015

Contents

Introduction

1.1 Purpose

In this document we aim to provide a description for the architecture and design of MyTaxiService. This document is targeted towards the future developers of the system.

1.2 Scope

The application will be developed using a client-server paradigm. The server-side application must recognize an user (either an unregistered guest, a passenger, a taxi driver or an administrator), and accordingly signal the client the available actions.
The server-side application must manage the city-wide taxi deployment, by the means explained in the RASD document[1].
The client-side application must show a UI to which the users can interact.
The application must implement a report system, in order to incentive the good behavior of the users involved.

1.3 Definition, Acronyms, Abbreviation

- RASD: Requirements Analysis and System Design
- API: Application Programming Interface
- UI: User Interface
- MVC: Model View Controller
- Client-side: client application, front end
- Server-side: server application, back end

[1] see reference documents

1.4 Reference Documents

In order to write this, we referred to the following documnts:

1 : our RASD document

2 : the design document template received from the professor

1.5 Document Structure

Architectural design : here we describe every components of our system, and their functions.

Algorithm design : here we define which algorithms to outsource, which ones to develop ad-hoc, and we give a brief overview of the latter ones.

User interface design : here we write some guidlines in order to design a better user interface.

Requirements traceability :here we describe how our components interact, in order to meet the equirements.

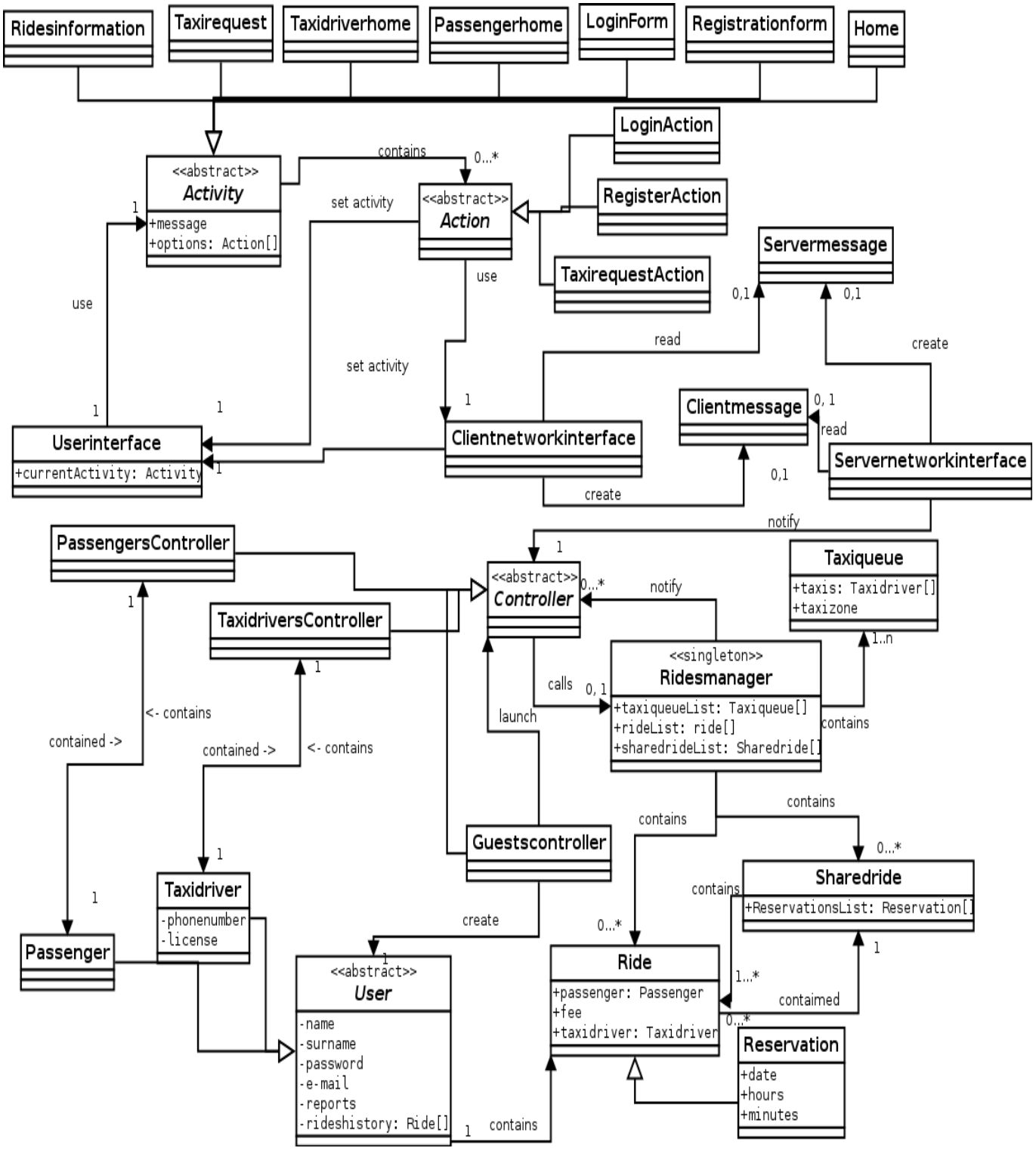
Architectural Design

2.1 Overview

The system is composed by 3 tier: database, application server, presentation(client in case of users or guests, terminal in case of master view). The system will also include a load balancer. The distributed application is composed by a server side, and a client one. The client side interacts with users (or guests), showing the correct activity, and sends requests to the server side, when needed. The client side must be able to interact with the GPS. The server side manages requests coming from the client side, and notifies the users (or the guests) involved in the requests. The server side must also interact with a map system, in order to retrieve information about the route and the length of the ride.

2.2 High level components and their interaction

In the following page you will see an UML diagram that show the main classes and their interaction. No methods are written in the diagram, but class's functions are described in the "Component view" section.


Ridesinformation
Taxirequest
Taxidriverhome
Passengerhome
LoginForm
Registrationform
Home
LoginAction
RegisterAction
TaxirequestAction
<<abstract>>
Activity
+message
+options: Action[]
contains
0...*
set activity
<<abstract>>
Action
use
Servermessage
0,1
read
create
Clientmessage
0, 1
Userinterface
+currentActivity: Activity
Clientnetworkinterface
Servernetworkinterface
notify
PassengersController
TaxidriversController
<<abstract>>
Controller
Taxiqueue
+taxis: Taxidriver[]
+taxizone
1..n
<<singleton>>
Ridesmanager
+taxiqueueList: Taxiqueue[]
+rideList: ride[]
+sharedrideList: Sharedride[]
calls
launch
contains
contained ->
<- contains
Guestscontroller
Sharedride
+ReservationsList: Reservation[]
Taxidriver
-phonenumber
-license
Passenger
create
<<abstract>>
User
-name
-surname
-password
-e-mail
-reports
-rideshistory: Ride[]
Ride
+passenger: Passenger
+fee
+taxidriver: Taxidriver
containes
Reservation
+date
+hours
+minutes

The client side is composed by a set of activities composed by one or more actions and displayed through the user interface. The client side has also an interface which manages the interaction with the server. The server side has a controller for each connected client, that manages the requests coming from the users (or guests), taking data from the ride manager. It also sends messages to the clients in order to resolve the requests. The controller interacts with the clients through a network interface.

2.3 Component view

2.3.1 Client

Activity : an activity is a set of messages and actions, that the application must display to the users. No more than one activity can be displayed at the same time; The default activity is the "guest home" activity. Each time the user taps a button, the application must execute the related action, and select the next activity.

Action : an action is something that a user can do, in order to interact with the application. If an action needs some data, the userinterface must display a field, for each input needed, that allows the human to provide the necessary incormations. Every Action class must also implement a "serverrun" method, that will be used by the controller, when the user (or the guest) tap on an action that need to communicate with the server. Some actions can also select the next activity that must be shown, or/and send informations to the server, in order to complete their job.

Userinterface : is the component that directly interacts with the human. It contains the activity that must be displayed, and read the components of the activity, in order to display them. It also launches the actions selected by the human.

Clientnetworkinterface : is the component that allows the other ones to exchange messages with the server side.

2.3.2 Application server

Controller : we have one controller for each client connected; initially it will be a guest controller, which allows login/register. After a successful login, the guest controller will create a taxi driver/passenger controller, and substitutes himself with the new controller. The controller must manage requests coming from the client, taking informations from the ridesmanager, and eventually adding new objects to it. In order to execute client's requests, the controller must launch the "serverrun"

method implemented in the corresponding action class In order to send and receive messages from the client, the controller communicates with a Servernetworkinterface. The controller also contains a User object, that allows him to retrieves information about the specific logged user.

Ridesmanager : it is a singleton, that contains informations about the queues, the active rides. This corresponds to the model in the MVC pattern and it is shared by all the controllers. The Ridesmanager must also implement some functions, that manage authomatic actions (like the allocation of a taxi driver for a reservation, 10 minutes before the reservation occur).

User : is the component that contains every possible information about a logged user, taken from the database. User is a part of the controller (obviously only in case of passenger/taxi driver controller). User is split in taxi driver and passenger, cause there are some differences between the two kind of user. Also user is a part of the model, but it is not shared.

Ride : this class allows to create object, containing all information about a specific ride. As soon as a passenger makes a taxi requests, the controller creates a ride containing information recevied from the client. In case of shared ride, a ride object will be created and added to the corresponding sharedride ones.

Sharedride : the sharedride object contains the set of the ride object, the total cost of the ride and the route. When a new passenger is added to the shared ride, the controller will adjust the path, the total cost, and the cost of each rides object, contained in the current sharedride.

Servernetworkinterface : is the component that allows the interaction with the client side. It converts messages coming from the other server component, in messages readable by the Clientnetwrokinterface, and vice versa.

2.3.3 Master view

(the whole master view component is singleton)

CLI: simple text based interface, receives input from stdin, sends output to stdout

Controller: interprets the cli commands, and invokes the relative methods on the specific server classes; the outcome of the operation is returned to the cli.

2.3.4 Database server

The database server will contain all informations about users, and, for each user, the list of unsent notifications. The client will periodically send notification request to the server, that must query the database, send the list of notification, and empty the unsent notification list.

2.4 Runtime view

2.4.1 Sequence diagrams

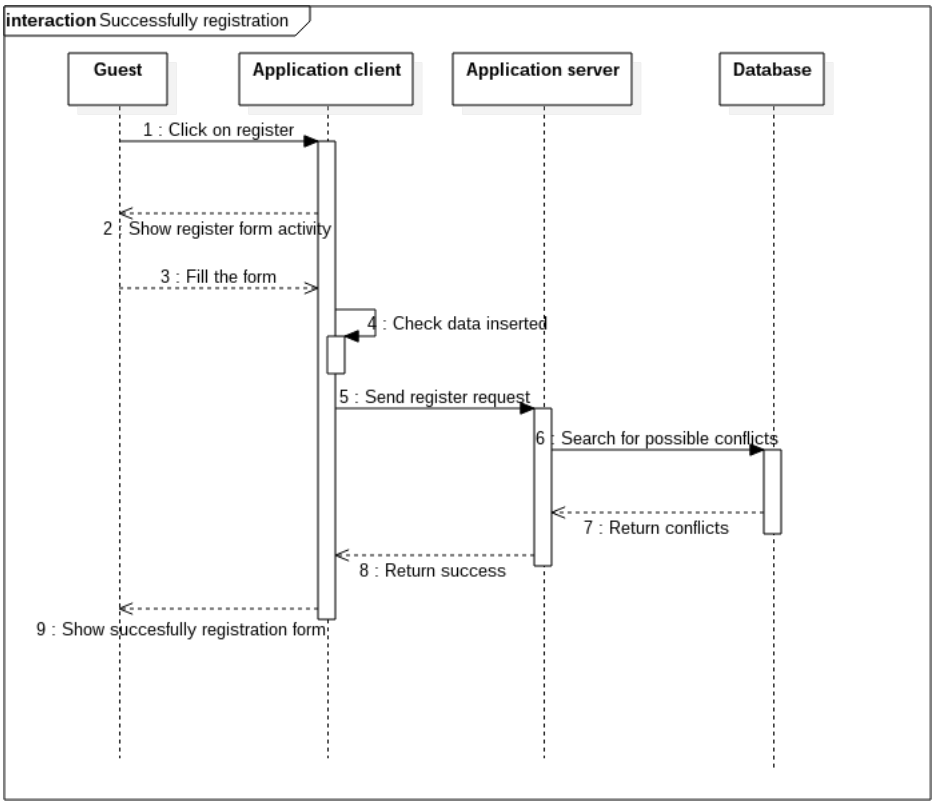


Figure 2.1: Registration

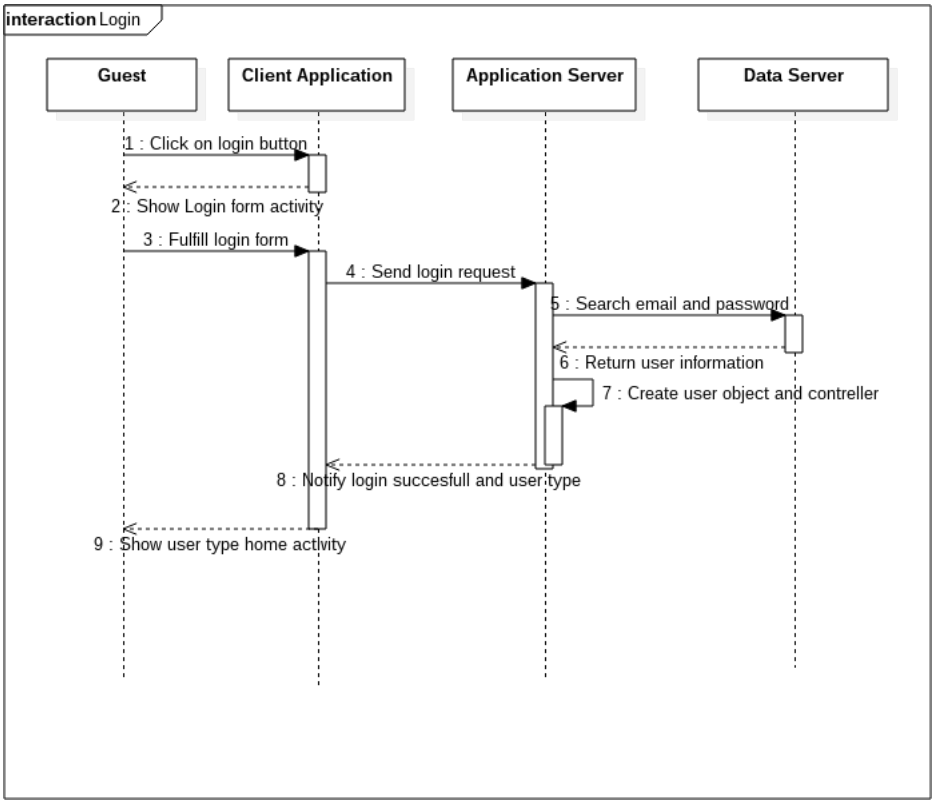


Figure 2.2: Login

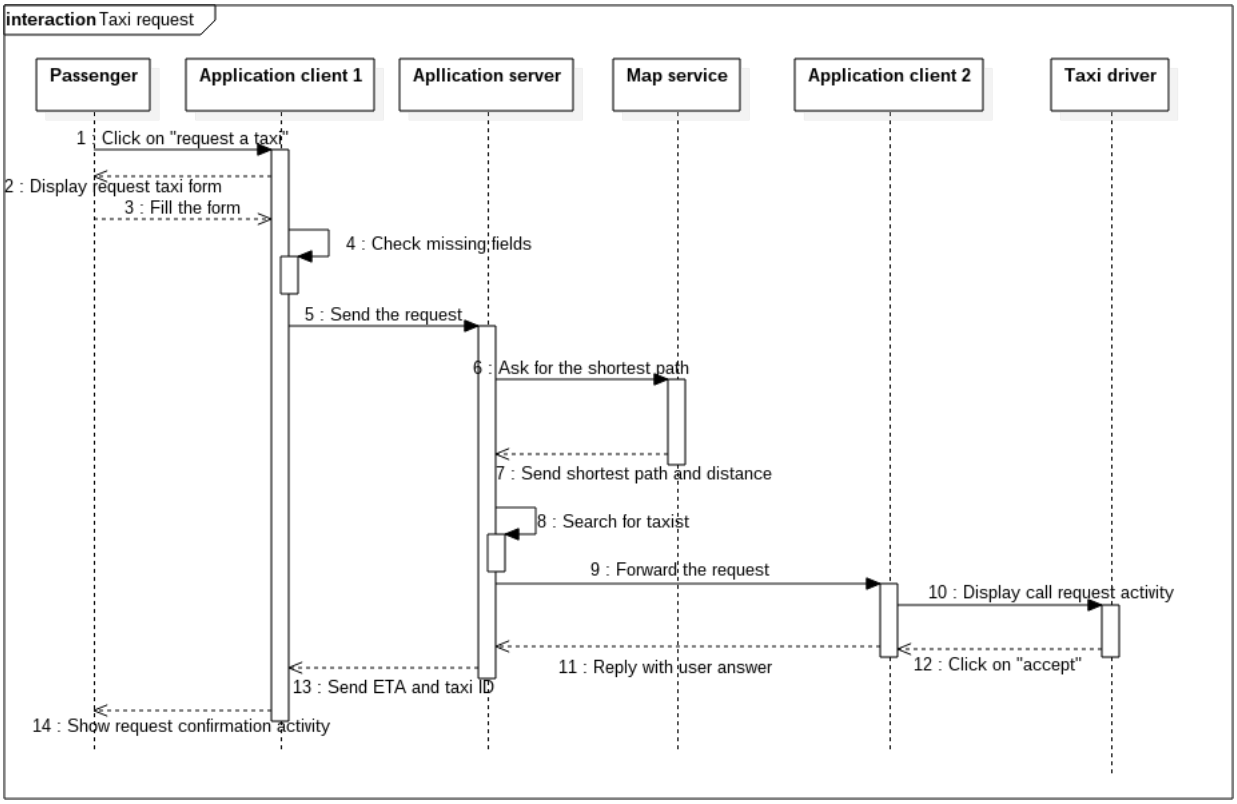


Figure 2.3: Taxi request

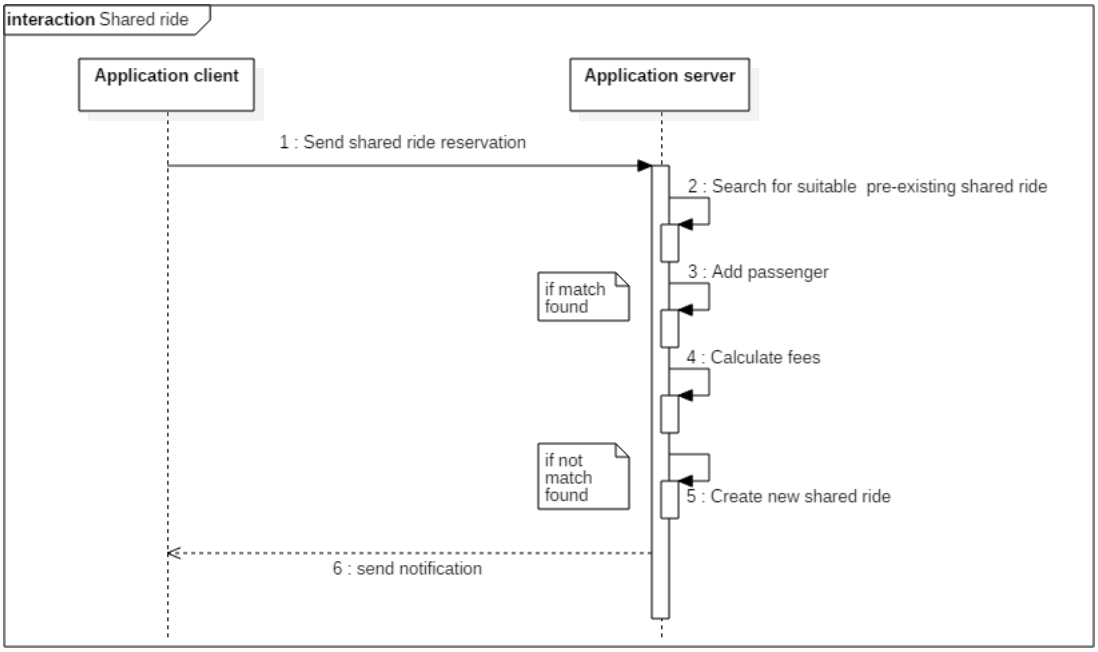


Figure 2.4: Request of a shared ride

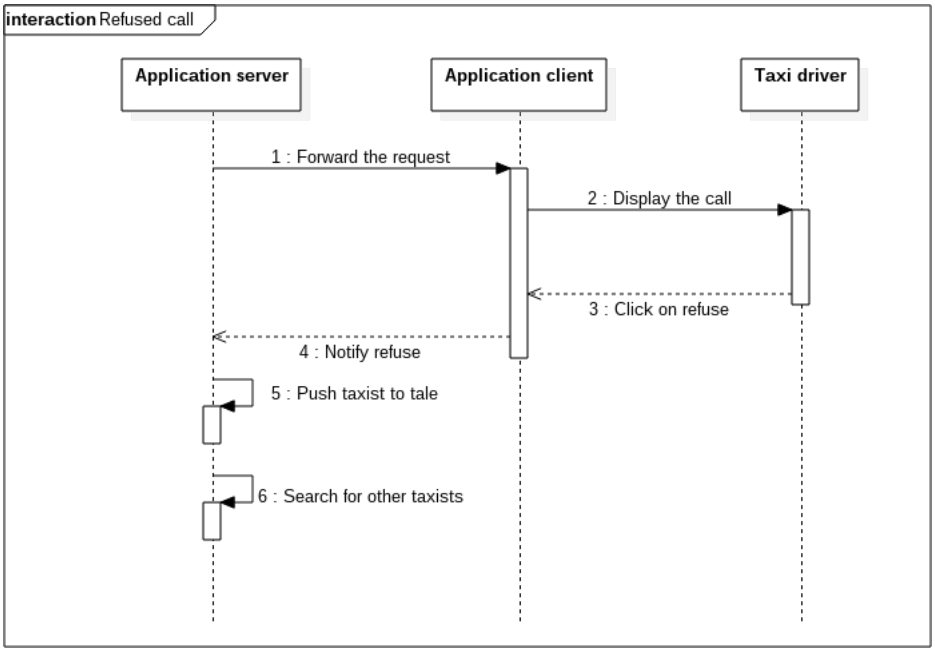


Figure 2.5: Refuse incoming call

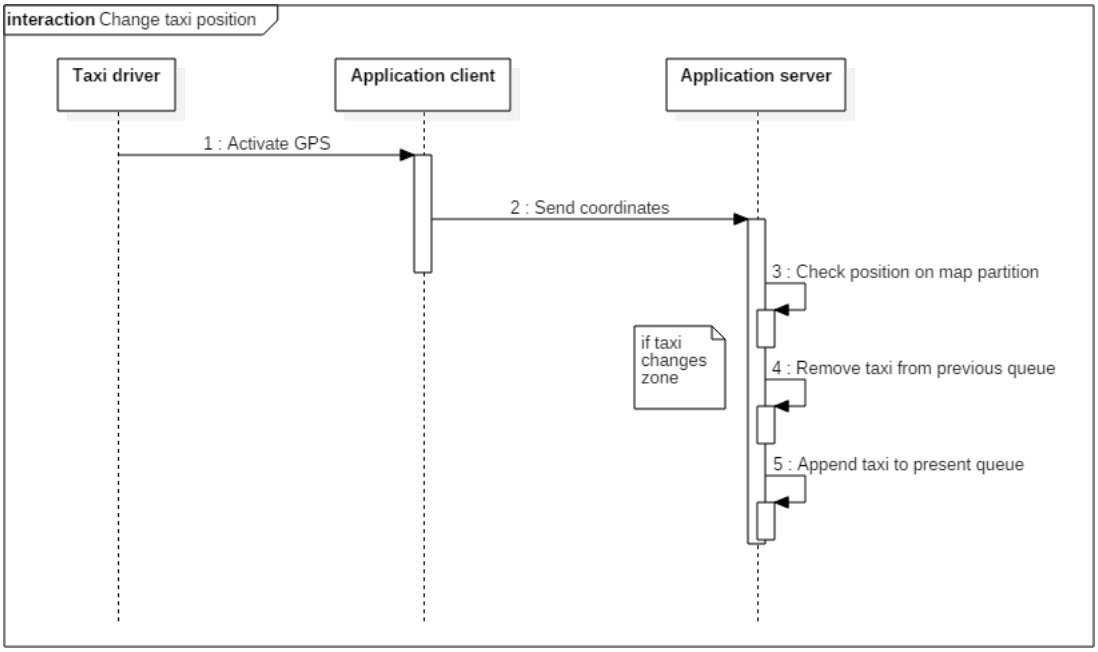


Figure 2.6: Change taxi position

2.4.2 MVC diagrams

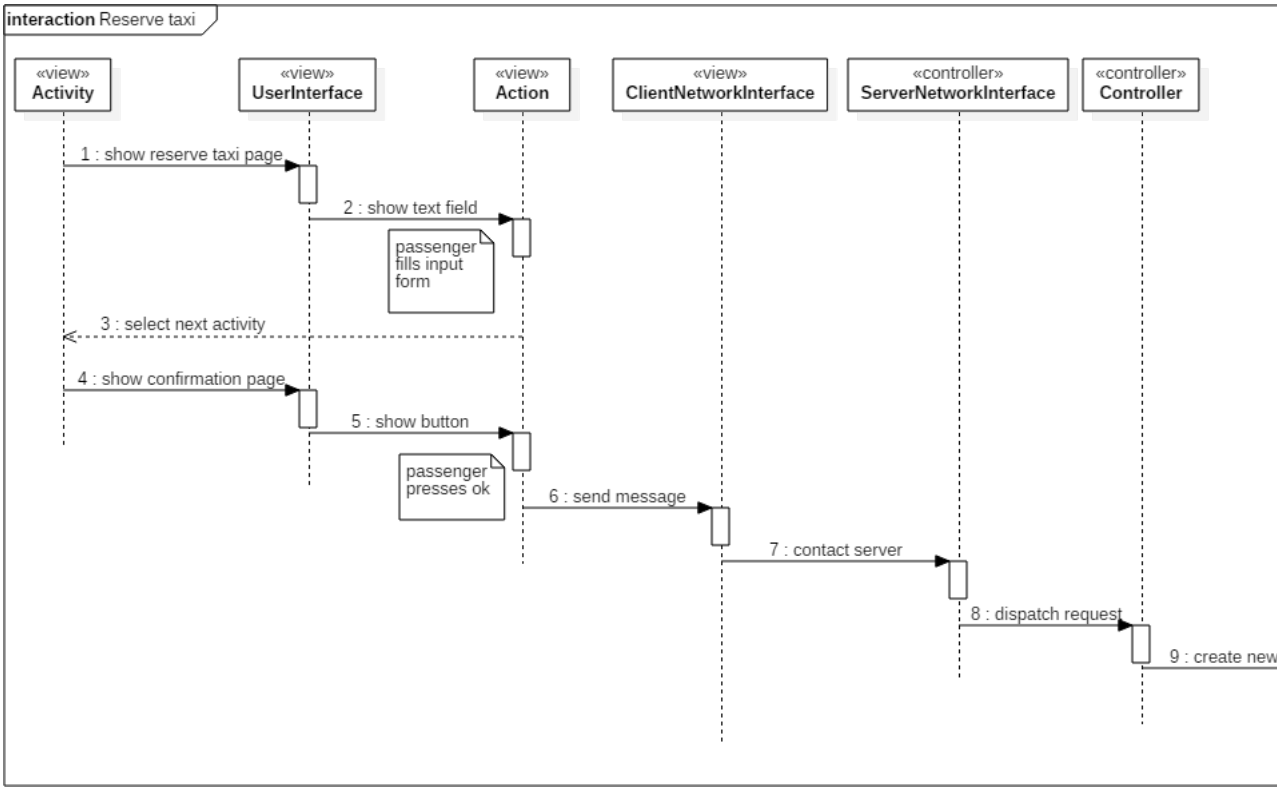


Figure 2.7: Reserve a taxi

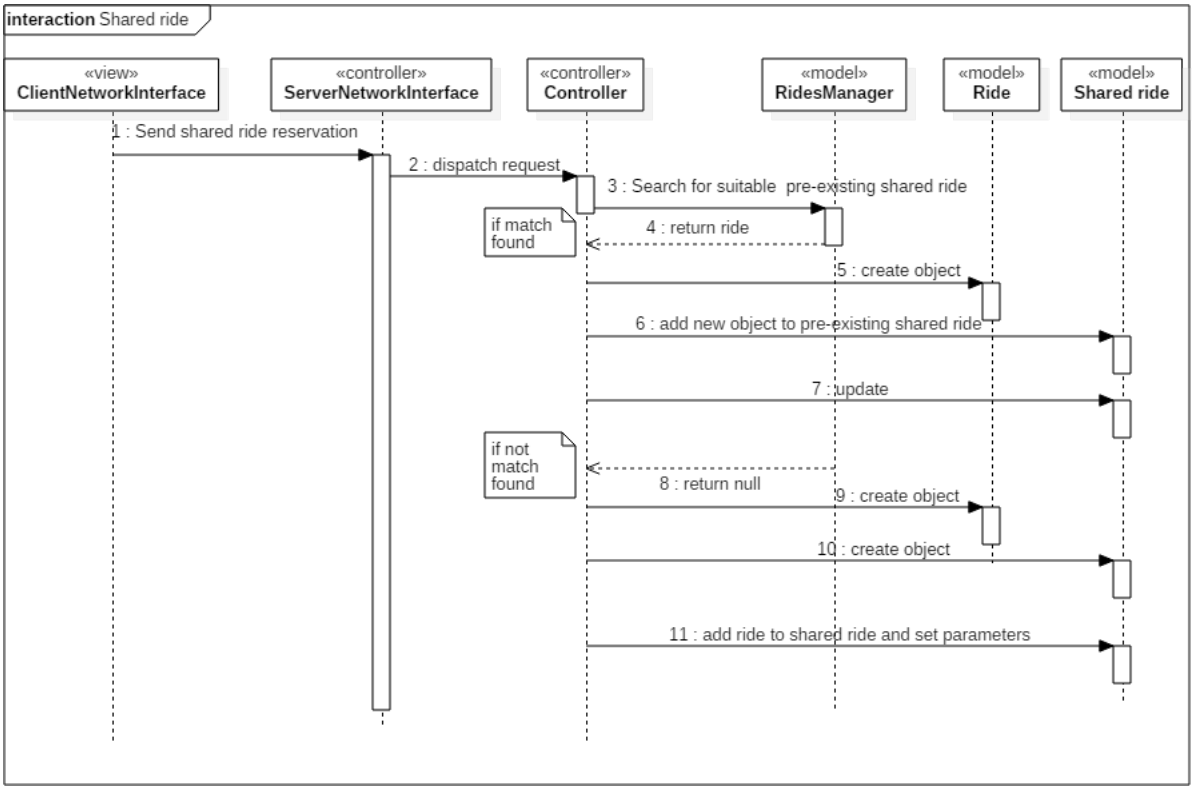


Figure 2.8: Shared ride request

2.5 Component interfaces

2.5.1 Usage of third-party API

The passenger controller class must interface with Google map sevice, using its API, to get the shortest path, the Eta to be sent to the user and the kilometers of the ride.

2.5.2 In-house developed API

The system must expose a set of public web based API. To access the API, an user must register first, using an unique key obtained from the API's maintainer. After successful registration, the system must return an authentication token to be used within a session. The key and the token must be sent using the HTTP POST method; when needed, user-specific passwords must also be sent via POST The server will send the required data encoded in JSON form. All the communications must be protected using the TLS protocol.

```
https://api.mytaxiservice.com/registerapikey
```

All the underlying functions (request taxis, list active calls, list taxis in a zone...) shall be made available using a simple, object-like interface:

```
https://api.mytaxiservice.com/request-ride/nonshared/<passengername>/
   <from_location>/<to_location>
https://api.mytaxiservice.com/list-taxis/by-zone/<zone_id>
https://api.mytaxiservice.com/list-calls/active/<passengername>/
https://api.mytaxiservice.com/list-calls/active/<drivername>/
```

A sample response (without the HTTP headers)

```
{
  type: "request_outcome",
  outcome: {
    taxi-id: "AZX934",
    eta: "12:30",
  }
}
```

2.6 Selected architectural styles and patterns

The application follows the MVC (Model-View-Controller) pattern. The model is entirely contained in the server side, and it's composed by a singleton class (Ridesmanager), that contains informations shared by all the controllers, and a user class (User), that contains informations about a specific

user, and is accessible only by the controller assigned to the corresponding user. Furthermore, the view is completely contained in the client side; it is composed by the "userinterface" class, that must display the activity's objects. The controller is mainly represented by the homonym class, in the server side. There are few controller's functions , that don't require data contained in the server, implemented in the action class.

2.7 Other design decisions

The server side structure is composed as shown in Figure 2.9: The load balancer is needed to equally divides requests coming from different clients, in order to increase performances. For the client side there's only one constraint, the obligation for the taxi driver to use a device with a working GPS.

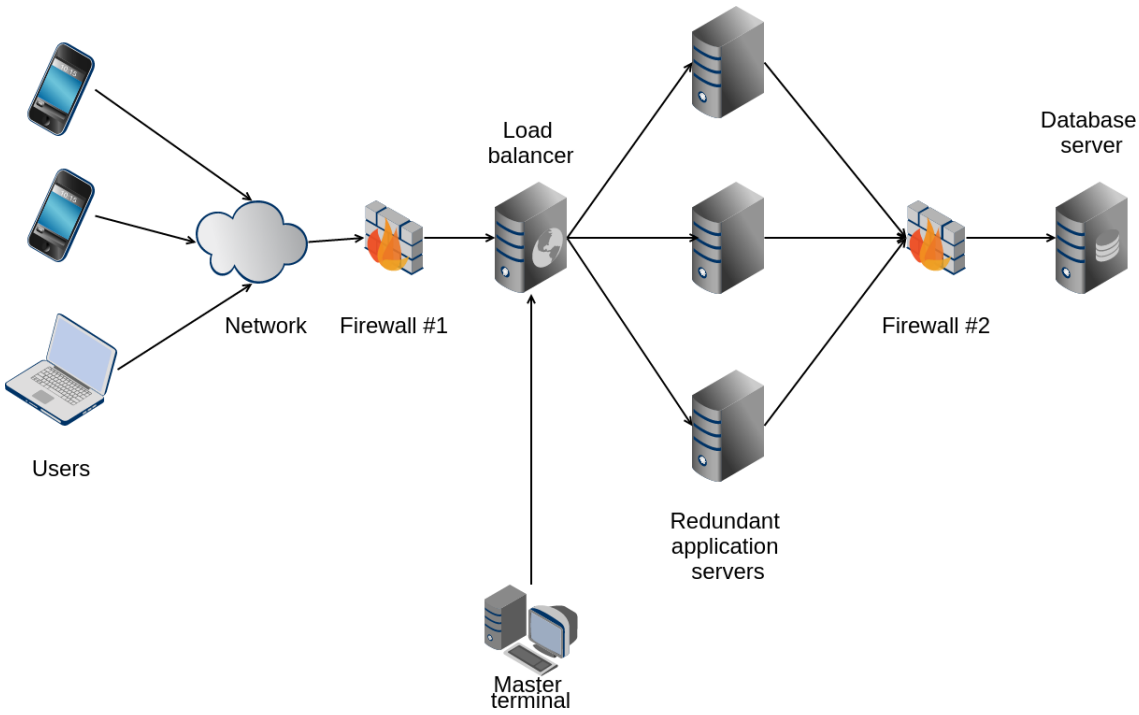


Figure 2.9: Server architecture

Algorithm Design

3.1 Outsourced algorithms

3.2 In-house developed algorithms

3.2.1 Shared ride

The following pseudocode describes how share taxi request should be handled

```
function sharedRequest(passenger)
    shared ← findSharedRideAvailable(fromZone, toZone, timeout)
    if exists shared then
        shared.addReservation()
    else
        taxi ← getAvailableTaxi(fromZone)
        if exists taxi then
            create new shared ride
        else
            error message
        end if
    end if
end function
```

The following function (dependency of *sharedRequest*), describes how the system should search for an available shared ride, compatible with the constraints provided via arguments

```
function findSharedRideAvailable(fromZone, toZone, timeout)
    for all ride in Rides do
        if ride is shared then
            if ride.fromZone == fromZone AND ride.toZone == toZone
then
                if not ride.isFull() AND not ride.isReserved() then
                    if now() − ride.allocationTime < timeout then
                        return ride
```

```
                end if
            end if
        end if
    end if
  end for
  return empty set
end function
```

The following function (dependency of *sharedRequest*), describes how the system should fetch an available taxi. Keep in mind that the process of issuing the call to a driver and managing his/her response happens inside this function (or its sub-routines).
Note: a timeout for the "accept/refuse call" action (executed by the taxi driver) must be implemented. However, it should be encapsulated inside taxi.sendRequest(), rather that the high level getAvailableTaxi() function.

```
function GETAVAILABLETAXI(fromZone)
  for all queue in Queues do
    if queue.getZone()==fromZone AND not queue.isEmpty() then
      taxi ← queue.getFirstTaxi()
      if taxi.sendRequest() == SUCCESS then
        manage queue return taxi
      else
        manage queue and retry
      end if
    end if
  end for
end function
```

3.2.2 Billing calculation

The following formula describes how the system should calculate the amount of money each passenger has to pay after a shared ride (to put it simply, how to "split the bill")

B_i = Amount paid by the i-th passenger

D_i = Distance traveled by the i-th passenger

D = Total traveled distance

C = Total amount calculated by the taximeter

$$B_i = \frac{D_i}{D} * C|_{\text{rounded to the nearest 0.1}}$$

$$T = \sum_{i=0}^{n} B_i = \text{Total cost, calculated as the sum of the n partial costs}$$

User Interface Design

4.1 Guidelines

The development of the UI must follow a series of general guidelines:

- Keep the UI as simple as possible (avoid bloat)
- The UI must be as intuitive as possible
- The user experience must be as consistent as possible, even across different devices

4.2 Mockup

Additionally, the following mock-ups are provided, in aid to the ui implementation.

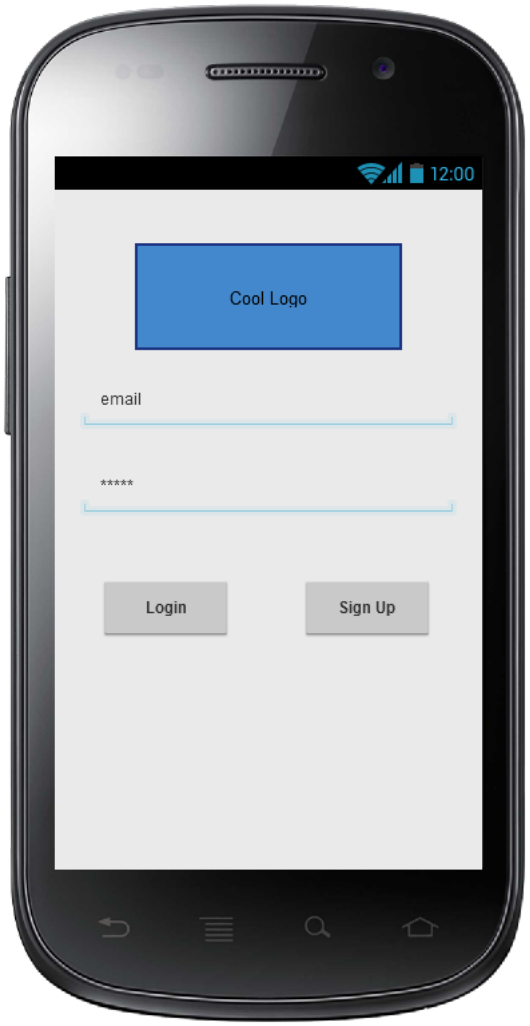


Figure 4.10: Login

Figure 4.11: Passenger map screen

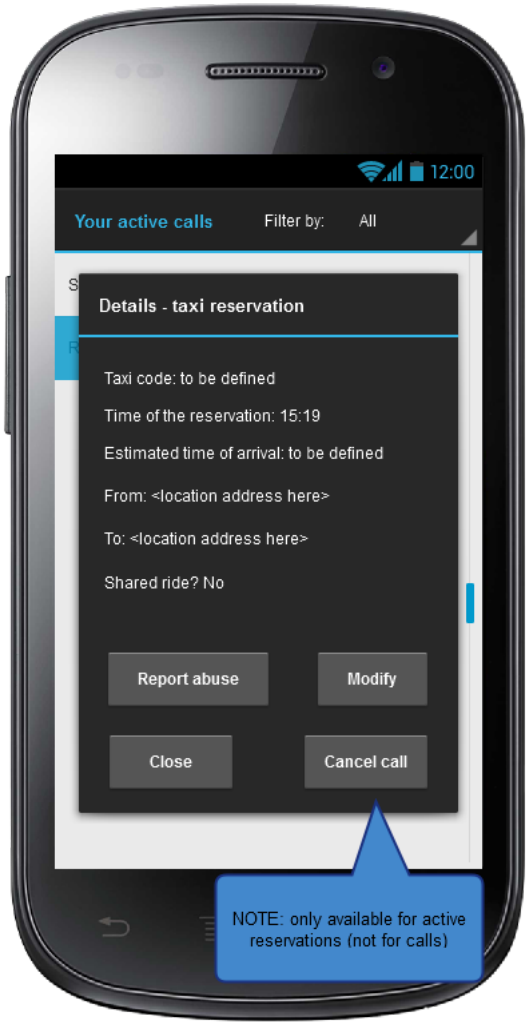


Figure 4.12: Passenger call list detail

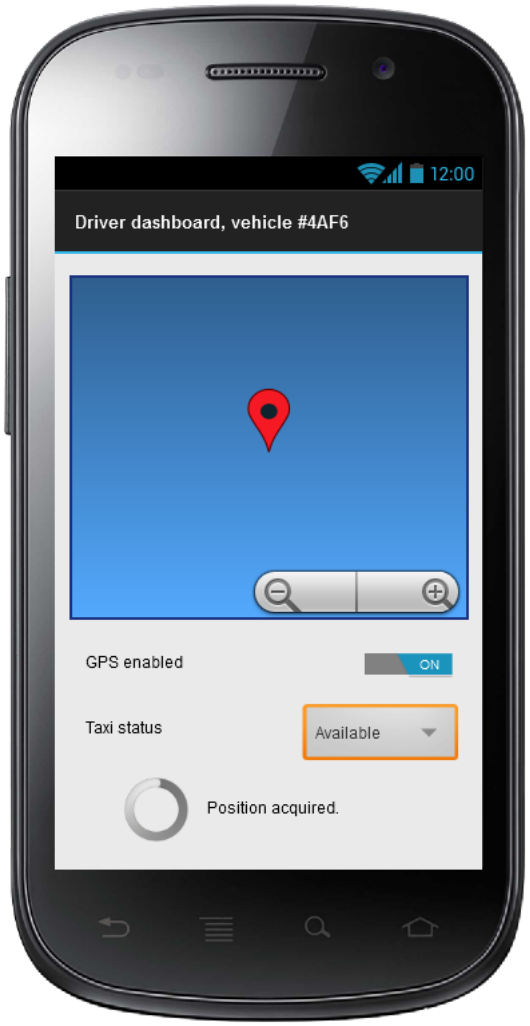


Figure 4.13: Driver Idle screen

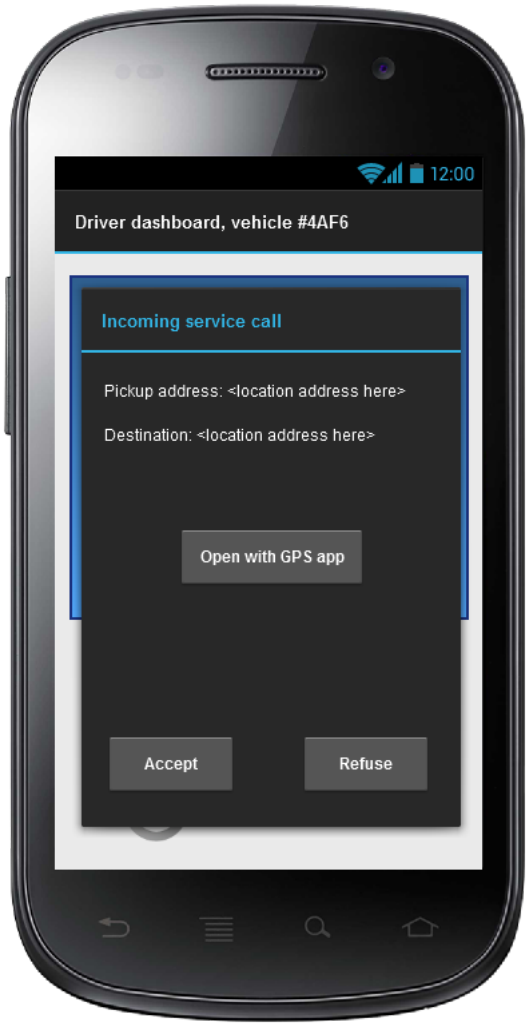


Figure 4.14: Driver new call dialog

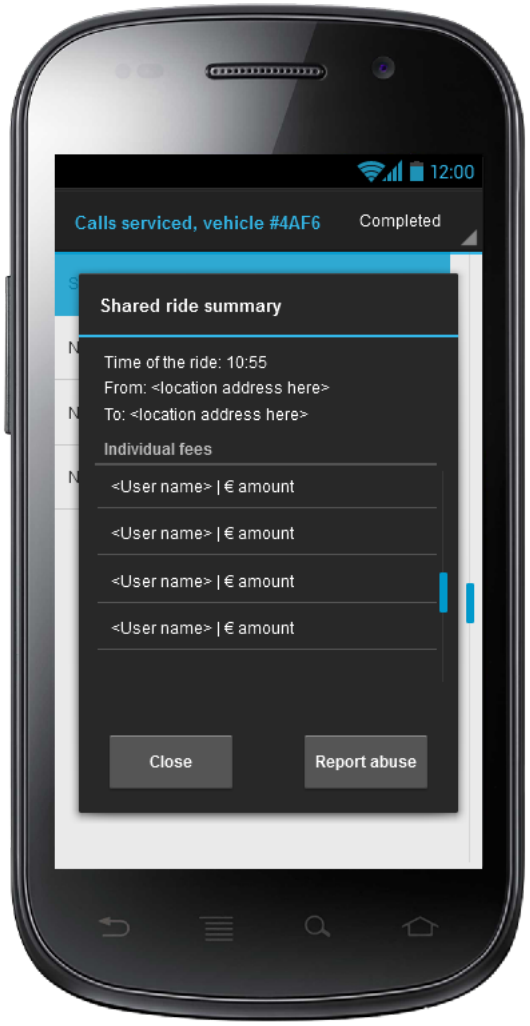


Figure 4.15: Driver shared fee dialog

Requirements Traceability

G1 : clicking on the button contained in the passenger home activity, a taxi request form will be shown. Filling the form allows the user to make a taxi request.

G2 : as soon as a taxist decides to take care about a call, the controller asks the map service to calculate an ETA. After that, the system will notify the passenger's client, that will display those information through an ad hoc activity.

G3 : when a taxi driver accepts to take care about a call, his controller will remove him from the queue.

G4 : in order to request a shared ride, a passenger must fill the form for a reservation/request, and add the shared option; The server will create a ride object, search for a sharedride that respect the constraints explained in the RASD document, and add the ride object to the shared ride; otherwise a new sharedride will be created.

G5 : when the taxi driver's client receive a call notification from the server, the application will select and display and ad hoc activity, that allow the taxi driver to click on "accept" or "refuse". After that, the client will send a message to the server, in order to inform him about the human's decision.

G6 : the taxi driver home activity provide a button that switch the taxist status; it will requests the controller to remove/ add himself to a taxi queue.

G7 : when a taxists accepts to take care about a requests (either shared or not), the controller will notify the client about the ETA and the fee amount. The client will display an ad hoc activity for these informations.

Goal	Requirement	Design document sections involved
G1	Persons can create an account	2.3.1 Userinterface, Action, Activity, Clientnetworkinterface 2.3.2 Controller, User, Servernetworkinterface 2.3.4 Database server
G1	Persons can log into their account	2.3.1 Userinterface, Action, Activity, Clientnetworkinterface 2.3.2 Controller, User, Servernetworkinterface 2.3.4 Database server
G1	Passengers can select from a menu the option of requesting a taxi as soon as possible	2.3.1 Userinterface, Action, Activity
G1	Passengers can insert their position filling an input form and confirm it	2.3.1 Userinterface, Action, Activity, Clientnetworkinterface 2.1 GPS
G1	The system will receive the request and identify the zone in which the passenger is in	2.3.2 Servernetworkinterface,Ride, Controller
G1	The system will forward the request to the first taxi in the selected zone queue and wait for an answer	2.3.2 Controller, Ridesmanager, Servernetworkinterface
G1	If the taxist accepts, the system will remove him from the queue; otherwise it will append the taxist to the last position and scan the list for a taxist to accept	2.3.2 Controller, Ridesmanager, Ride
G2	As soon as a taxist accepts a request, the system invokes the support system to calculate the ETA giving the position of the taxi and the position of the passenger	2.3.2 Servernetworkinterface, Controller, Ridesmanager, Ride 2.5 Google map service
G2	he system will communicate the taxi code and the ETA	2.3.1 Userinterface, Activity, Clientnetworkinterface 2.3.2 Controller, Servernetworkinterface, Ridesmanager, Ride

Goal	Requirement	Design document sections involved
G3	Passengers can select from a menu the option of reserving a taxi for a chosen ride and date	2.3.1 Userinterface, Activity, Action
G3	Passengers can insert the initial and final position, time and date, their email and confirm it	2.3.1 Userinterface, Activity, Action 2.1 GPS
G3	The system will receive the reservation and if it respects the 2 hour constraint it will send a confirmation	2.3.2 Servernetworkinterface, Controller, Ridesmanager, Ride
G3	Ten minutes before the ride starts, the system allocates a taxi for it.	2.3.2 Passenger, Taxidriver, Ridesmanager, Ride
G4	The application must have a selectable option labled:"share your ride", that allows passengers to enable the shared ride service. In case of non reserved ride, the application will ask passengers the amount of time they can wait for others people	2.3.1 Userinterface, Activity, Action
G4	When the system receive a request of a shared ride, it will search for others shared ride requests starting from the same taxi zone, and going in the same direction	2.3.2 Servernetworkinterface, Controller, Ridesmanager, Sharedride, Ride 3.2.1 sharedRequest, findSharedRideAvailable
G4	When a new passenger is added to a shared ride, the system will interact with the map service, in order to retrieve a new route for the taxi driver, and to calculate new fees	2.3.2 Controller, Ridesmanager, Ride, Sharedride 2.5 Google map service 3.2.2 Billing calculation
G4	When the timeout of one passengers ,added to the current ride, occur, the system will procede with the allocation of the taxi	2.3.2 Ridesmanager, Sharedride, Ride 3.2.1 getAvailableTaxi
G4	After the taxi allocation, the passengers who requested the shared ride will receive, not only the taxi ID, but also the fee they have to pay	2.3.1 Userinterface, Activity, Clientnetworkinterface

Goal	Requirement	Design document sections involved
G5	The system must forward a taxi request in the following cases: 1: A passenger has requested a ride. 2: A taxi reservation is sheduled to begin in 10 minutes.	2.3.2 Controller, Ridesmanager, Ride, Servernetworkinterface
G5	If a taxi driver refuses to take care about a call, the system will move him at the end of the queue,and forward the request to the next taxi driver in the queue. If a queue is empty, the system will notify the passenger that there are no taxi available	2.3.2 Servernetworkinterface, Controller, Ridesmanager
G5	If a taxe driver accepts to take care of the call, the system shall remove him from the queue.	2.3.2 Servernetworkinterface, Controller, Ridesmanager
G6	A taxi driver logged in into the system can select the button " Ready ", then the application will notify the server that the logged user is ready to accept some passengr's call. The application also send the taxi driver's position detected with a GPS	2.3.1 Userinterface, Activity, Action, Clientnetworkinterface 2.3.2 Servernetworkinterface, Controller, Ridesmanager 2.1 GPS
G6	If the application needs to retrieve data from a GPS and this isn't available, it will remind the user to turn it on	2.3.1 Activity, Userinterface, Action
G6	When the system receive a notification , by a taxi driver, informing that he is ready to take care of some passengers, it will append the user in the queue corresponding to the taxi zone that include the position retrieved by the application	2.3.2 Servernetworkinterface, Controller,Ridesmanager

Goal	Requirement	Design document sections involved
G7	When a taxi driver is assigned to a shared ride, the system will send him the route he needs to follow, and the fee amount every passenger have to pay	2.3.2 Controller, Ridesmanager, Sharedride, Servernetworkinterface
G7	When a driver is assigned to a non-shared ride, the system will send him the route he needs to follow, and the fee amount the passenger has to pay	2.3.2 Controller, Ridesmanager, Ride, Servernetworkinterface
G8	It is also neccessary to develop programmatic interfaces that allow to customize the system, adding new features	2.5.3 In-house developed API
	Passengers can access a section, in which they be able to check the ID of the taxi assigned to their ride and manage (delete or modify) an active reservation	2.3.1 Userinterface, Action, Activity
	When a passenger delete a reservation , the system will remove it from the reservation scheduler and, if a taxi driver is already assigned, notify the taxist	2.3.1 Userinterface, Activity, Clientnetworkinterface 2.3.2 Controller, Servernetworkinterface, Ridesmanager, Ride
	A passenger can modify an active reservation changing position, date and time	2.3.1 Userinterface, Action, Activity, Clientnetworkinterface 2.3.2 Servernetworkinterface, Controller, Ridesmanager, Rid 2.1 GPS
	The system will accept modification only if sent before the taxi allocation	2.3.2 Controller, Ridesmanager, Ride
	The system will accept date and time modification if it occur at least two hours after the request or/and after the previous reservation	2.3.2 Controller, Ridesmanager, Ride

Goal	Requirement	Design document sections involved
	A taxi driver have the possibility to remove himself from the queue by clicking the:"Disable" button	2.3.1 Userinterface, Action, Activity, Clientnetworkinterface 2.3.2 Servernetworkinterface, Controller, Ridesmanager
	The system will remove a taxi from the list if receive the corresponding request by the taxi driver, or if the taxist logged out	2.3.2 Ridesmanager, Controller
	A master terminal interface must be implemented in order to allow, the stakeholder, to configure some parameters (the number of the taxi zones, the set of positions belonging to each zone, the number of reports (per day, month and year) needed to automatically ban a user and the maximum number of reports (per hour) a user can insert)	2.3.3 Master view
	Every time a report is added to a user, the system will check if the constraints inserted by the master terminal are satified, otherwise the system must automatically ban the user	2.3.2 User
	The master terminal interface allows to manually ban users, or enable banned users	2.3.3 Master view
	The system must refuse reports added by a user if the user has already reached the maximum number of reports (per hour) decided by the master terminal	2.3.2 User
	When the system refuse a report, a notification appear on the user screen, reminding him that he has already exceeded the maximum number of reports for that hour	2.3.1 Userinterface, Activity, Clientnetworkinterface 2.3.2 Servernetworkinterface, User, Controller

References