# Requirements Analysis and Specifications Document


Figure 1: Politecnico di Milano

**Version 0.1**


- Claudio Cardinale (mat. 849760)
- Gilles Dejaegere (mat. 853950)
- Massimo Dragano (mat. 775392)

# Contents

# Introduction

## Description of the given problem

We will project and implement myTaxiService, which is a service based on mobile application and web application, with two different targets of people:

- taxi driver
- clients

The system allows clients to reserve taxi via mobile or web app, using GPS position to identify client's zone (but the client can insert it manually) and find taxi in the same zone.
On the other side the mobile app allows taxi driver to accept or reject a ride request and to communicate automatically his position (so the zone).
The clients are not registered since the company wants a quickly system so if there is a registration a lot of clients won't use the app. So the clients must insert their name and phone number each time (this is faster than creating an account and logging each time).

The system includes extra services and functionalities such as taxi sharing

The main purpose of the system is to be more efficient and reliable than the existing one in order to decrease costs of the taxi management and offer a better service to the clients.

### Actual system

Until now the taxi company has a system where the clients have to call a call center communicating its position via voice (so it can be not correct), the call center's operator inserts the request into an internal information system and the taxi driver can accept or reject it via a dedicated hardware device.

The system sends automatically an SMS to the client with the estimated arrival time and the taxi name.

When the taxi driver join into the company he receives the login data that he cannot change.

This system stores taxi information into a Mysql database.

## Goals

**Taxi drivers:**

- [G1] Allows taxi drivers to log in the system.

- [G2] Allows taxi drivers to precise to the system if they are available or not.
- [G3] Taxi drivers should receive a push notification for incoming request.
- [G4] Taxi drivers should receive a push notification if they have to take care of another client (during a shared ride).
- [G5] Allows taxi drivers to accept or decline incoming requests for an immediate ride
- [G6] Allows taxi drivers to accept or decline incoming request for a later reservation.

**Clients:**

- [G7] Allows clients to request for an immediate taxi ride.
- [G8] Allows clients to request for the reservation of a taxi at least two hours in advance.
- [G9] Clients should receive a SMS notification with the ETA and code of the taxi that takes care of the client's request.
- [G10] Allows clients to require to share the taxi.
- [G11] Allow clients to identify themselves via phone number (and name) not login, they are not registered into the system.
- [G12] Allow clients to specify number of passengers.

## Domain properties

We suppose that these properties hold in the analyzed world :

- Actual drivers are already registered on the previous system
- A taxi driver accepting a ride of reservation will actually take care of the request.
- A client requiring a taxi will actually take it.
- All the GPS always give the right position.
- The GPS of the taxi drivers can not be switched off.
- Taxi drivers answer all types of demands in less than 5 minutes.
- The client pays the taxi driver directly for each commission.
- A taxi can be in only one zone at the same time and this is the real zone.
- Client make a reservation two hours before the ride **Here as domain or do we have to do the requirements into G7?**
- When a new taxi driver joins in the taxi company the taxi company registers him in the information system. Analogously when a taxi driver exits from the company, the company deletes him from the information system.
- The taxi arrives at start point with max 30 minutes of delay.

- Start zone may be different from end zone.
- The old system works properly without problems.
- If a queue is empty, a taxi joins in this queue in max 15 minutes.

## Glossary

- Client: he is a client of the service. He should insert each time he performs a request/reservation the following information
  - Name
  - Phone number
  - Position, it can be taken automatically from GPS (either via APP or Web browser)
  - Number of passenger
  - Sharing mode
  - Time (only for reservation)
- Taxi driver: he is a taxi driver registered on the taxi company, which grants to taxi driver the access to this information system
- Taxi queue: when more than one taxi are in the same zone, there is a FIFO queue. So in this way when there is a new client the oldest taxi can take it. There is a queue for each zone.
- Requests queue: when more than one requests are in the same zone, there is a FIFO queue. So the first taxi available serve the first request.
- Request: is the request of a taxi, it can be shared or not. When an user requests a taxi, the request is created and inserted in request queue. See client to fields details.
- Merged request: is a request associated to more than one user. When more that one request in the same queue respect the taxi sharing condition, a merge request is created (merging that requests). The merging is performed when a shared request is at head of the queue.
- Ride: it starts when the taxi receives the request and ends when it leaves the last client of the ride. The simple ride is specified by start ride, client and taxi; but other ride types (like reservation or taxi sharing) have other parameters.
- Taxi sharing: it is the possibility that if different people (it's not required that they know each other) of the same start zone go to the same direction, even if the end is not the same, to use the same taxi and to have a unique group fee. A sharing ride is identified by clients that use it and for each client the start and end point
- Reservation: it is the ability to reserve a taxi until two hours before time of ride, so when a reservation is done the system makes a taxi request 10 minutes before the ride. The reservation is identified by start point, end point, client and time. It can be sharing or not. See client to fields details.

- Taxi request: it is the request the system sends (automatically or after a client request) to taxi to specify a ride, specifying start point, client and other elements if they are available.
- Client request: it is the request for a taxi drive as soon as possible, it contains the client data and the start point that can be get by GPS (current position) or inserting manually
- Zone: it is a zone of approximately 2 km^2, the city is split into these zones. From taxi position the system gets his zone and inserts the taxi into the zone queue. So the system guarantees a fair management of taxi queues. A zone is specified by a list of bounds.
- Task: a task is an action done automatically by the server, for example "send request 10 minutes before ride" is a task
- Taxi: it is a means of transport that can bring only 4 passengers.
- System: it is the new system we will create with the database of the old system.
- Matching itineraries : Two itineraries (A and B) are matching if one of the two following conditions are fulfilled :

  1. B is included in A: The start point and the end point of the itinerary B are both close to the itinerary A and the start point of B is closer to the start point of A than the end point of B.
  2. The beginning of B is the end of A: The start point of B is close to the itinerary A and the end point of A is close to the itinerary B.
  3. A is included in B: see condition 1.
  4. The beginning of A is the end of B: see condition 2.

- ETA: estimated time available, is the time that taxi need to arrive to client start position.
- SMS: short message service, is a notification sent to a mobile phone, we need a SMS gateway to use it.
- SMS gateway: is a service that allow to send SMS via standard API.
- API: application programming interface, is a common way to communicate with another system.
- Push notification: is a notification sent to a smartphone using the mobile application, so it must be installed.
- Push service: is a service that allow to send push notification with own API

## Text assumptions

- There is an old system as described above.
- We should develop a mobile application for clients where clients can make a reservation using the GPS position or by inserting their position. **Keep or remove?**

- Shared requests are took into account until them don't get accepted by any driver.
- The clients are not registered in the system, because we need only their name and their position. **SEE REQUIREMENTS** A client is identified by his personal data: name and phone number
- There are only normal taxis for only 4 passengers.
- The registration/deletion by company of a taxi driver is done in the same way of the old system, so we don't have to do this part.
- We need information only about taxi driver, not about taxi vehicle. So we store information only about taxi driver.
- The system doesn't need client registration, since it requires only identification data and position and since it works like the old system (where every client must say identification data via call). The real applications of many cities run in this way. **See description**
- All taxi drivers of the city are regulated and use this system
- The client cannot cancel a request
- We assume that we need a requests queue

## Constrains

### Regulatory policies

The system must require to client/taxi driver the permission to get his position and he has to manage sensible data (position, phone number) respecting the privacy law. Furthermore the systems mustn't use notifications to send SPAM respecting the privacy law.

### Hardware limitations

- Mobile app
  - taxi driver:
    - 3G connection
    - GPS
    - Space for app package
  - client:
    - 3G connection
    - Space for app package
- Web app
  - Modern browser with AJAX support
  - ...

#### Interfaces to other applications

- Interface with the old system. The new system will interface with the Mysql database of the old system.
- Interface with SMS gateway provider via standard SMS rest APIs, to send notification to clients
- Interface with the push service(s) via own APIs to send push notification to taxi drivers. Often we need an interface for each platform (android, iOS, and so on)

#### Parallel operation

The server supports parallel operations from different clients and different taxi drivers.

#### Reference documents

- Specification Document: Assignments 1 and 2 (RASD and DD).pdf
- IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications.
- Examples documents:
  - MeteoCal_RASD_example2.pdf
  - RASD Example SWIMv2.pdf
  - RASD_meteocal-example1.pdf

## Proposed system

We will implement a client-server architecture (Fig. 2) based on common REST API and MVC pattern, so with just one server application we manage both web application and mobile application, obviously we will have version for taxi driver and version for clients.

## Identifying stakeholders

We have only one main stakeholder: the government of the city, that wants to improve the current taxi service in terms of usability, efficiency and cost.
However we can adapt this system to other city (changing the interface with the old system)

## Other considerations about the system

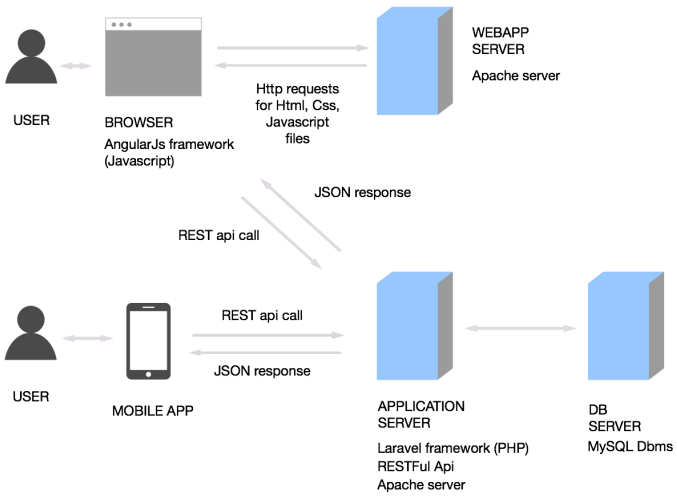



Figure 2: Architecture

## Actors identifying

The actors of our system are basically two:

- Taxi driver: it is a taxi driver registered automatically in the system by the taxi company
- Client: he doesn't need to register himself to the system, since he uses the system only to call a taxi (so he have to insert only name, phone number and location)

# Requirements

## Functional requirements

Assuming that the domain properties stipulated in the paragraph [1.3] hold, and, in order to fulfill the goals listed in paragraph [1.2], the following requirements can be derived.

The requirements are grouped under each goal from which it is derived. The goals are grouped following under the clients concerned.

**Taxi drivers:**

**Numbers will be adapted once we are sure about the Goals**

- [G1] Allows taxi drivers to log in the system:
  - The system must be able to check if the password provided is correct.
  - The system must only let the taxi drivers log in if the provided password is correct.
- [G2] Allows taxi drivers to precise to the system if they are available or not:
  - The system must be able to detect the taxi's location according to the taxi's GPS.
  - The system must be able to determine the appropriate queue for the taxi driver according to it's position.
  - The system must put the taxi driver in the appropriate queue when the taxi client becomes available.
  - The system must remove the taxi driver from the appropriate queue if he becomes unavailable.
  - The system must communicate to the driver it's position in the waiting queue.
- [G3] Taxi drivers should receive a notification for incoming request:
  - The system must be able to forward an incoming request to the appropriate taxi driver.
  - The system must be able to alert a taxi driver when a request is incoming.
- [G4] Taxi drivers should receive a notification if they have to take care of another client (during a shared ride):
  - The system must be able to match together matching request with the "shared ride" option activated.
  - The system must be able to inform the taxi when a ride is shared.

  - The system must be able to inform the taxi drivers of the different stops he has to do during his ride in order to take or drop the concerned passengers.

- [G5] Allows taxi drivers to accept or decline incoming requests for an immediate ride:
  - The system must ask to the taxi driver if he accepts to perform the ride.
  - The system must replace a taxi driver at the end of the queue if he declines the ride.
  - The system must ask to the next taxi driver if the former one declined the ride.
  - The system must notify the client with the code of the taxi driver who has accepted the ride.
  - The system must notify the client if no taxi driver in the queue accepts the ride.
- [G6] Allows taxi drivers to accept or decline incoming request for a later reservation:
  - The system must wait until ten minutes before the starting time of the reservation and manage it like a immediate ride.

**Clients:**

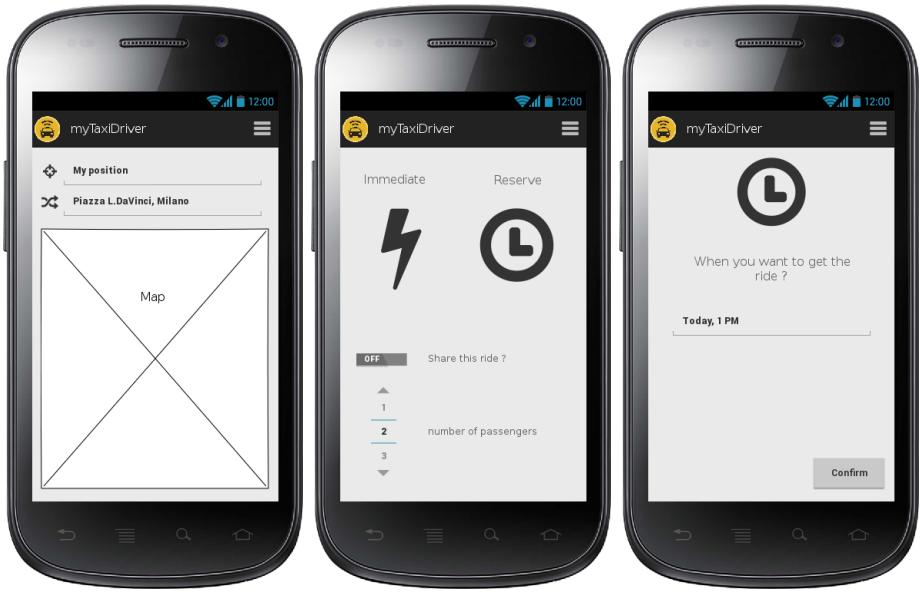
- [G7] Allows clients to request for an immediate taxi ride:
  - The system must be able to check the position of the client.
  - The system must not accept requests of clients outside the area of the city.
  - The system must transfer the request to the appropriate taxi driver.
  - The system must determine be able to determine the zone where the client is located according to the client's GPS position.
- [G8] Allows clients to request for the reservation of a taxi at least two hours in advance:
  - The system must be able to check the origin and the destination of reservation.
  - The system must not accept reservations with an origin outside the area of the city.
  - The system must transfer the reservation to the appropriate taxi driver.
- [G9] Clients should receive a notification with the code of the taxi that takes care of the client's request:
  - The system must be able to send an sms to the client with the code of the incoming taxi.

- [G10] Allows clients to require to share the taxi:
  - The system must be able to find if there are reservations or requests for the same time period and having matching itineraries.
  - The system must be able to merge together the reservations and request found above if the cumulated number of passengers of the corresponding requests or reservations does not exceed 4.
- [G11] Allow clients to identify themselves via phone number (and name) not login, they are not registered into the system:
  - The system must allow the clients to furnish their personal information to the system before making a request.
- [G12] Allow clients to specify number of passengers:
  - The system must allow the client to specify the number of passengers during the request or reservation of the ride.

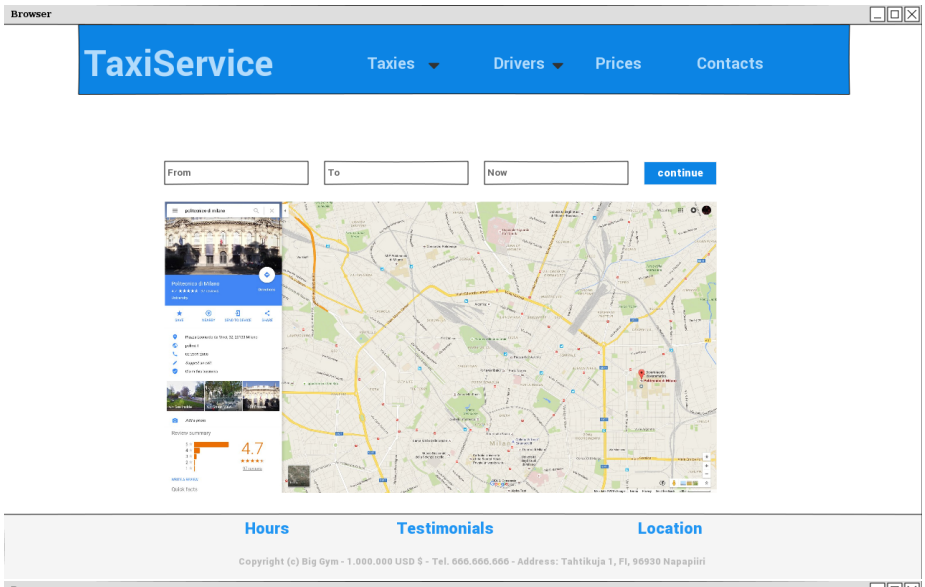
## Non-functional requirements

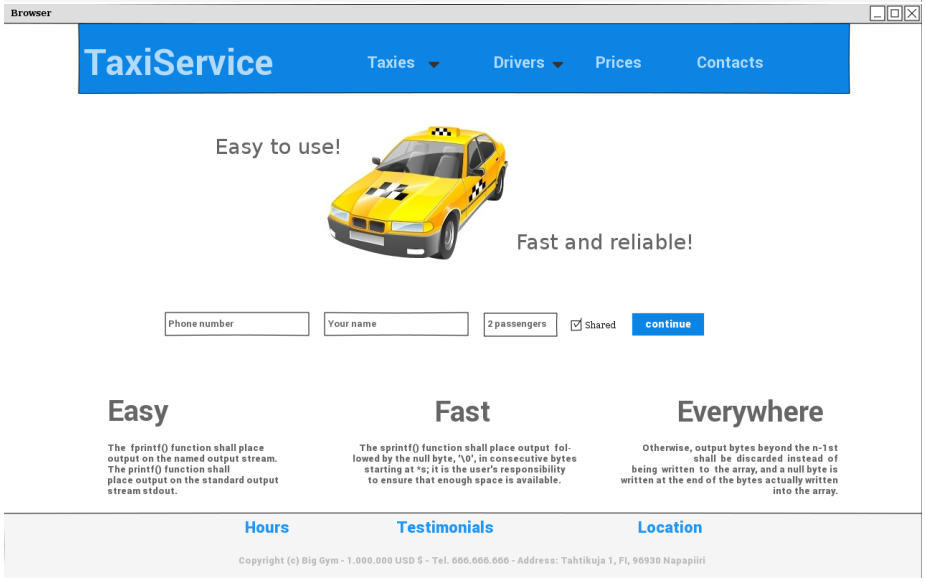
### Client interface

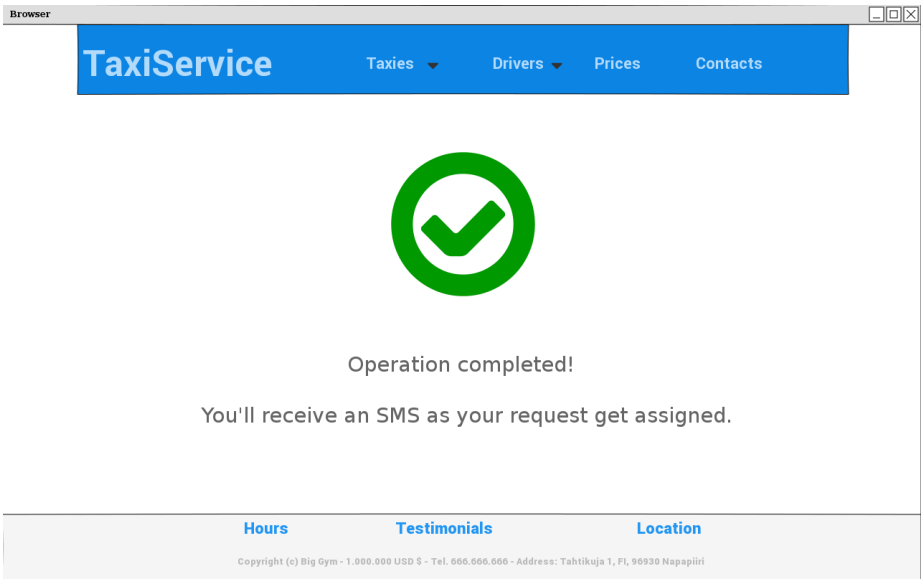
#### mobile



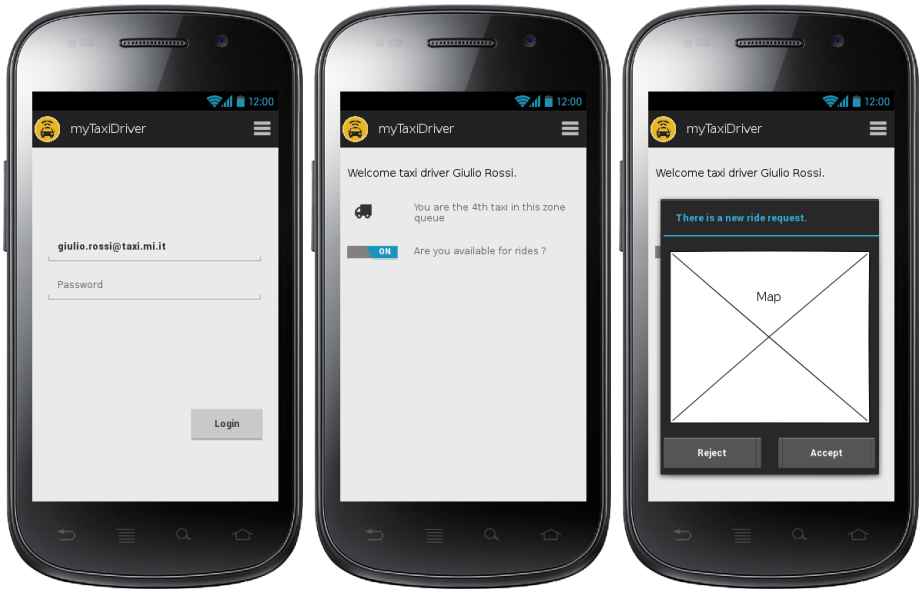

#### desktop


Browser
TaxiService
Taxies
Drivers
Prices
Contacts
From
To
Now
continue
4.7
Hours
Testimonials
Location
Copyright (c) Big Gym - 1.000.000 USD $ - Tel. 666.666.666 - Address: Tahtikuja 1, FI, 96930 Napapiiri


Browser
TaxiService
Taxies
Drivers
Prices
Contacts
Easy to use!
Fast and reliable!
Phone number
Your name
2 passengers
Shared
continue
Easy
The fprintf() function shall place output on the named output stream. The printf() function shall place output on the standard output stream stdout.
Fast
The sprintf() function shall place output followed by the null byte, '\0', in consecutive bytes starting at *s; it is the user's responsibility to ensure that enough space is available.
Everywhere
Otherwise, output bytes beyond the n-1st shall be discarded instead of being written to the array, and a null byte is written at the end of the bytes actually written into the array.
Hours
Testimonials
Location
Copyright (c) Big Gym - 1.000.000 USD $ - Tel. 666.666.666 - Address: Tahtikuja 1, FI, 96930 Napapiiri




### Taxi driver interface




### Documentation

We will draft these documents to well-organize our work in the way to do in a fewer time the best work as possible:

- RASD: Requirement Analysis and Specification Document, to well- understand the given problem and to analyze in a detailed way which our goals are and how to reach them defining requirements and specification.
- DD: Design Document, to define the real structure of our web application and its tiers.
- Testing Document: a report of our testing experience of another myTaxyService project.

**Architectural consideration**

We will use the following technologies:

- Apache with php (with laravel framework) as API server and task service
- Mysql as sql server to store data persistently, it is the same of the old system
- Apache server for static documents
- RESTFull and JSON for API communication over HTTP(S)
- Javascript (with angularJs framework), CSS and HTM to create responsive site that communicate to server using REST API. These files are got via HTTP(S)
- Modern browser with javascript and ajax support
- Java and swift respectively for android and iOS apps, using original SDK
- Internet connection for communication of data
- External rest APIs to send SMS

## Scenario identifying

Here some possible scenarios of usage of this application

### Scenario 1

John wants to go home as soon as possible after a day of work and he wants to do that as soon as possible. So in the morning (so he respects the constrains of two hours before) he reserves a taxi at the same time of the end of his job for a ride that starts from his office and ends at his home.
When he goes out from office he finds the taxi on the street that brings him to his home.

### Scenario 2

Some friends live in the same zone and want to go to the airport for a trip together, they want a cheap solution. So they choose the taxi sharing option to go to the airport. The morning of the trip's day all friends request a taxi with sharing option.
Since they are 6 and a taxi can bring only 4 passengers they need at least 2 taxis, so 4 friends are in the same taxi while two others are in other two taxis, each of them filled by other people that have chosen the taxi sharing option and start from the same zone and have to go in the same direction. **Update this inserting the possibility to reserve a taxi for more than one passengers, but keep this since start points are different**

### Scenario 3

John wants to visit the Duomo tomorrow. He decides to reserve a taxi. Therefore, he opens the MyTaxi **adapt name** website on his laptop and click on the "reserve a taxi button". He is the redirected on a form where he has to fill some information about his ride. After submitting his form, he is redirecting on a waiting page. A few minutes later, John is notified by the application of the confirmation of his reservation and of the code of the taxi taking care of the ride.

### Scenario 4

Julia is a taxi driver. She has just finished her last commission and has still plenty of time before the end of the day so she decides to make a new commission. She opens her MyTaxi **adapt Name** application and logs in her personal page. The she sets her availability to "Available". The application notifies her that she is the 3rd taxi on the waiting queue of her area. After waiting a small amount

of time Julia receives a request for an immediate ride very close of her location. She immediately accepts it and heads to the request location.

### Scenario 5

Bob wants to go and see the football match at the stadium tonight. He decides to reserve a taxi to bring him to the stadium, and to bring him back home after the game. To reduce the cost of the rides he decide to try to share a taxi. To do so, he enters the MyTaxi **adapt name** application on his smartphone, goes to the "Client home page", activate the slide button to enable the "sharing taxi" option and then clicks on the "Reserve" button. He then has to fill a form to indicates all the informations about the reservation. After doing so, he has to repeat the whole operation the reserve the return trip.

### Scenario 6

Mark is a taxi driver. At the beggining of his workday, Marks opens his MyTaxi **adapt name** application and logs in. He then puts his status on "available" by operating the slide button. The application tells him that he is the 10th drivers in his queue. That is far too much for Mark. He decides to go to another area to check if the queue is smaller. To do so, he first changes his status to "Non available", then rides toward the desired area and finally sets his status to "Available" again. He is now 4th of his waiting queue.

### Scenario 7

Bob tells his cousin Alice that he is going to the stadium tonight. Alice is also interested in football so she decides to go with Bob to see the match. However, Alice do not know how to go to the stadium. Bob tells her about the MyTaxi **adapt name** application and convince Alice to give it a try. Alice tries to reserve a trip to the stadium, but she does not repect the "reserve two hours before the ride" condition and the reservation fails. She then tries to request for an immediate ride but no taxi driver accepts Alice's request. Alice will have to stay at home tonight.

# Uml models

## Use case diagram

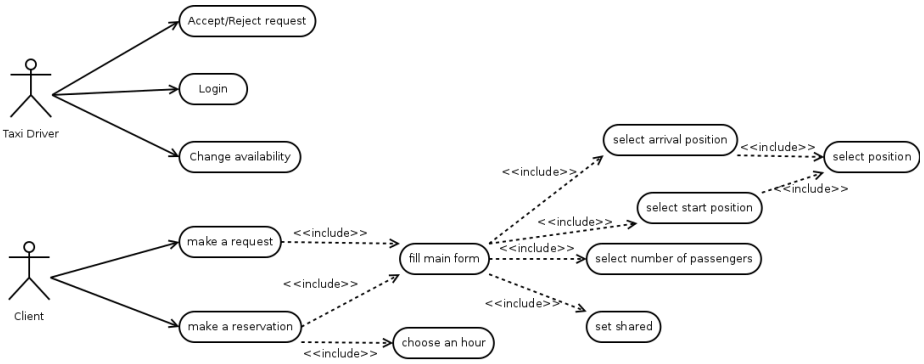



Figure 3: use case diagram

## Use case description

In this paragraph some use cases will be described. These use cases can be derived from the scenarios and the use case diagram.

#### Taxi driver log in

**Name :** Taxi driver log in
**Actors :** Taxi driver
**Entry conditions :** There are no entry conditions.
**Flow of events :**

- The taxi driver arrives on the homeActivity of the mobile application.
- The taxi driver inputs his taxi driver code (his ID) and his password.
- The taxi driver clicks on the log in button.
- The system redirects the taxi driver to his personal activity.

**Exit conditions :** The driver is successfully redirected to his personal page.
**Exceptions :** The code and password furnished by the taxi driver are not correct. In this case, the system does not redirect the taxi driver to his personal activity but notifies him that an error has been made and allows to input his code and password again.

**Taxi driver informs of his availability**

**Name :** Taxi driver informs of his availability
**Actors :** Taxi driver
**Entry conditions :** The taxi driver must be logged in.
**Flow of events :**

- The taxi driver activates the slide button on his personal activity.

**Exit conditions :** The system actualises the personal activity of the driver with the relevant informations. If the taxi driver selected the available button, he can now see in which waiting queue he is and his position.

**Exceptions :** There are no exceptions for this use case.

**Taxi driver responds to a request of immediate ride**

**Name :** Taxi driver responds to request.
**Actors :** Taxi driver
**Entry conditions :**

- The taxi driver has to be available.
- The taxi driver must be the first of his queue.

**Flow of events :**

- The system notifies the taxi driver of the request with an alert.
- The systems shows the taxi driver the information concerning the request and shows him two buttons: “Accept” and “Reject”.
- The taxi driver clicks on one of the two buttons.
- The system asks confirmation to the taxi driver concerning his choice.
- The taxi driver confirms.

**Exit conditions :**

- If the taxi driver has accepted the request, the taxi driver is removed from the waiting queue and the use case “notifying the client of incoming taxi” begins.
- If the taxi driver has declined the request, he is moved to the end of the queue and the use case “taxi driver respond to request of immediate ride” starts again.

**Exceptions :** There are no exceptions.

**Client requires a taxi for a ride**

**Name :** Client requires a taxi for a ride
**Actors :** Client
**Entry conditions :** The client has to be on the "Client Homepage".
**Flow of events :**

- The clients set the "Share this ride" slide button to the desired value if he is using the mobile application, or sets the "Share this ride" box to the desired value if he is using the website.
- The client set the "number of passengers" to the desired value.
- The client clicks on the "require immediate ride button".
- The systems redirects the client to a form where the client has to give some information like the start location of the ride.

**Exit conditions :** The system forwards the request to the appropriate taxi and the use case "taxi driver respond to a request of immediate ride" begins.
**Exceptions :** The client furnish invalid data (for example a negative or excessive number of passengers (see [1.3] Domain properties)). The request is not forwarded and the client is not redirected until he enters valid data.

**Client requires a taxi for a later reservation**

**Name :** Client requires a taxi for a later reservation
**Actors :** Client
**Entry conditions :** The client has to be on the "Client Homepage".
**Flow of events :**

- The clients set the "Share this ride" slide button or box to the desired value.
- The client clicks on the "require later reservation button".
- The client indicates the correct number of passengers.
- The system redirects the client to a form where he has to furnish the following information :
  - Departure place;
  - Departure time;
  - Destination;

- The client fills the form.
- The client clicks on the "Confirm" button.

**Exit conditions :** The system redirects the client to a waiting page and forwards the request for a reservation to the appropriate taxi driver. The use case "taxi

driver respond to a request of later reservation" begins.
**Exceptions:** The client furnish invalid data in the form (for example a negative or excessive number of passengers or not valid departure time (see [1.3] Domain properties)). The request is not forwarded and client is not redirected until he enters valid data.

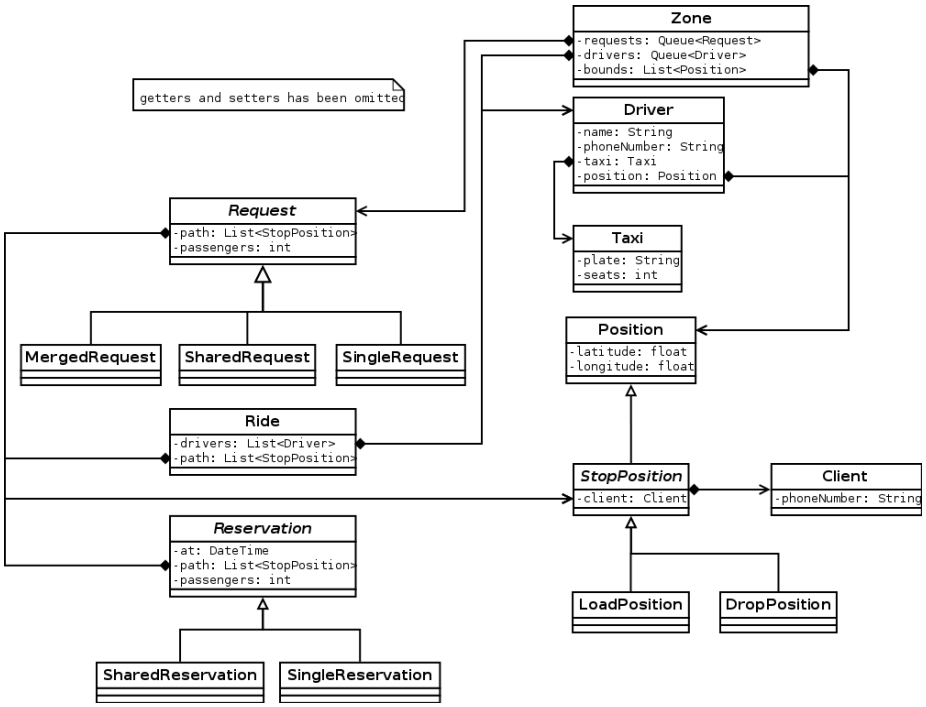
#### Notifying the client of incoming taxi

**Name :** Notifying the client of incoming taxi
**Actors :** Client
**Entry conditions :** The client must have required an immediate ride or a later reservation.
**Flow of events :**

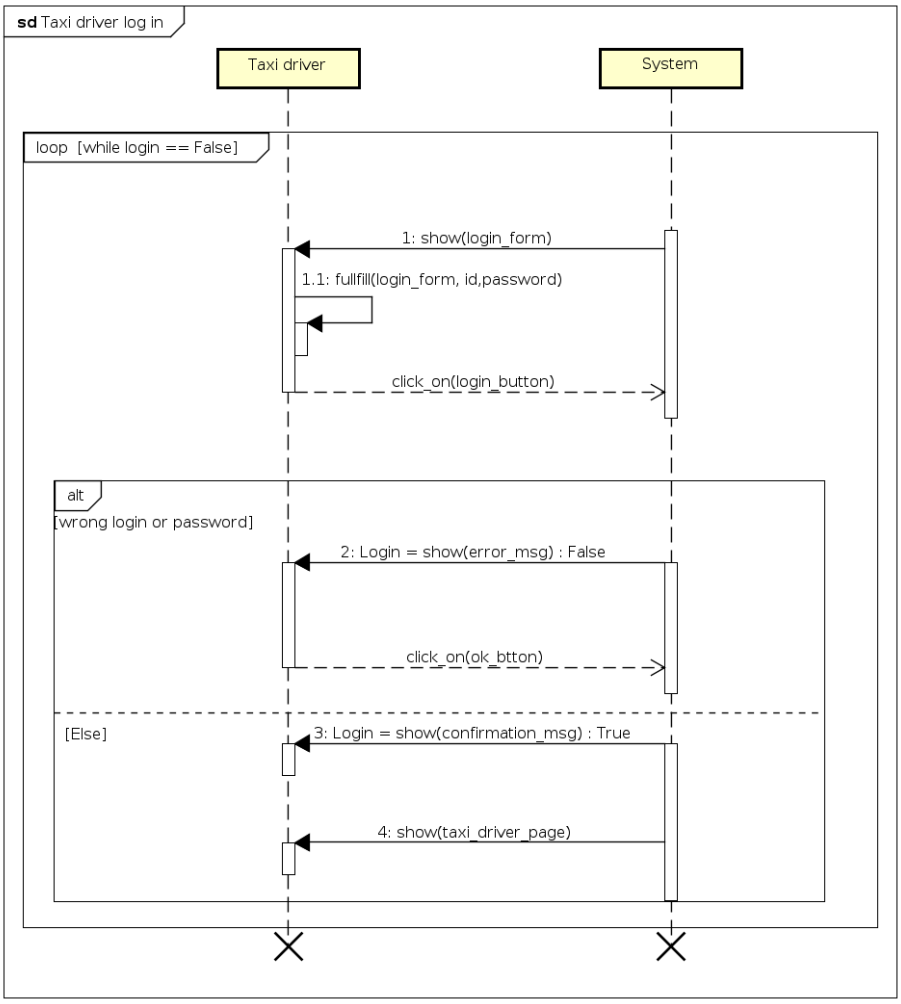
- The system sends an alert to the client.

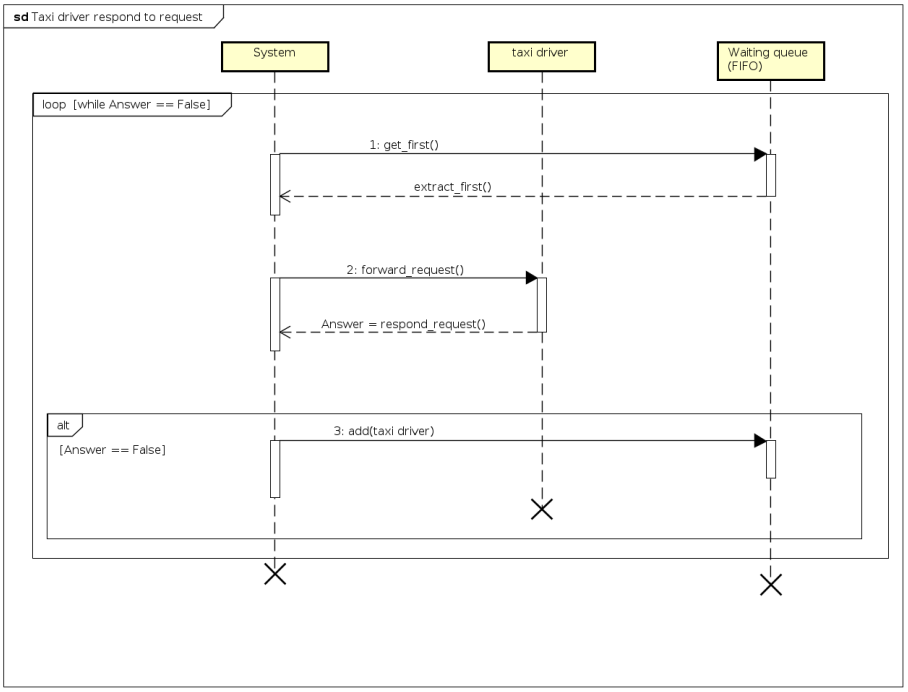
**Exit conditions :** The system redirects the client on a page containing the information about the incoming taxi.
**Exceptions :** No taxi driver has accepted the request of the client. The client is notified and redirected to the home page of the platform.

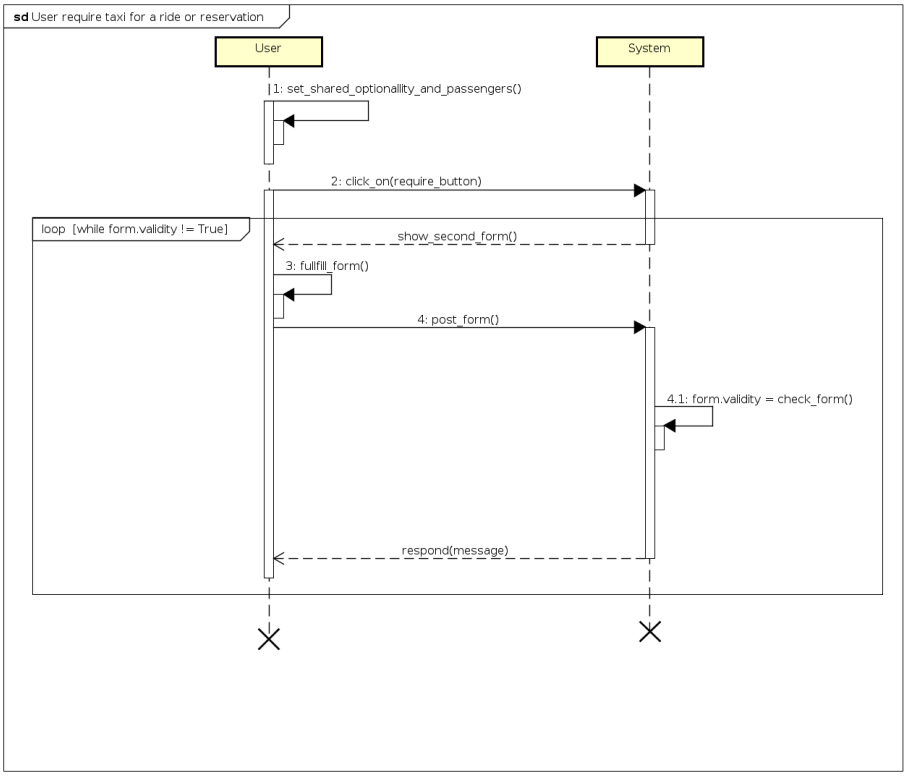
## Class diagram



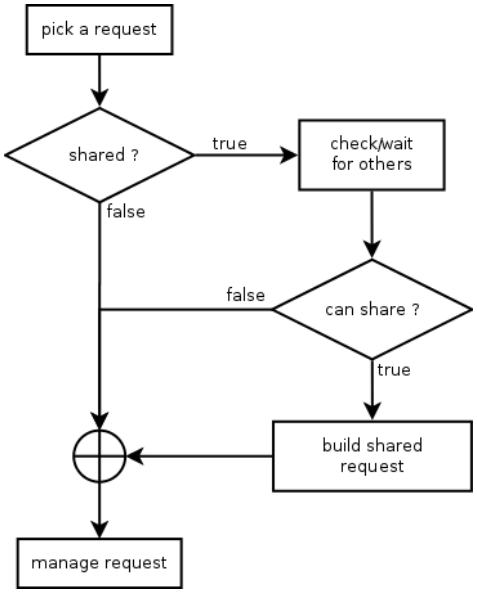
## Sequence diagrams



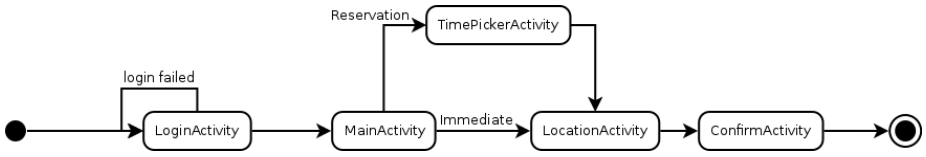

sd Taxi driver respond to request
System
taxi driver
Waiting queue (FIFO)
loop [while Answer == False]
1: get_first()
extract_first()
2: forward_request()
Answer = respond_request()
alt
[Answer == False]
3: add(taxi driver)


sd User require taxi for a ride or reservation
User
System
1: set_shared_optionality_and_passengers()
2: click_on(require_button)
loop [while form.validity != True]
show_second_form()
3: fullfill_form()
4: post_form()
4.1: form.validity = check_form()
respond(message)

## Activity diagrams




## State diagrams



# Alloy modeling

## Model

```
open util/boolean

//sig

one sig TaxiCentral {
hasTaxiDrivers: some TaxiDriver,
hasClients: some Client
}

sig TaxiDriver {
hasTaxi: one Taxi,
available: Bool,
belongsToQueue: lone QueueElement //if 0 the taxi is not in the queue
}

sig Queue {
}

sig QueueElement {
belongsToQueue: one Queue,
position: Int
}
{
position > 0
}

sig Taxi {

}

sig Client {

}

abstract sig Ride {
belongsToTaxiDriver: one TaxiDriver,
startZone: one Zone,
finished: Bool
}

sig NormalRide extends Ride{
```

```
belongsToClient: one Client,
passengers: Int,
endZone: one Zone
}
{
passengers<=4
passengers>=1
}

sig SharingRide extends Ride{
belongsToClients: some  Client,
endZones: Client -> one Zone
}

sig Zone {
hasQueue: one Queue
}


//If #hasRide = 0 the request is not completed yet
abstract sig Request {
belongsToClient: one Client,
startZone: one Zone
}

sig SharingRequest extends Request {
hasRide: lone SharingRide,
endZone: one Zone
}

sig NormalRequest  extends Request {
hasRide: lone NormalRide,
passengers: Int,
endZone: lone Zone
}

//If #hasRide = 0 the request is not created yet
abstract sig Reservation {
time: one Date,
belongsToClient: one Client,
startZone: one Zone,
endZone: one Zone
}

sig SharingReservation  extends Reservation {
hasRequest: lone SharingRequest
```

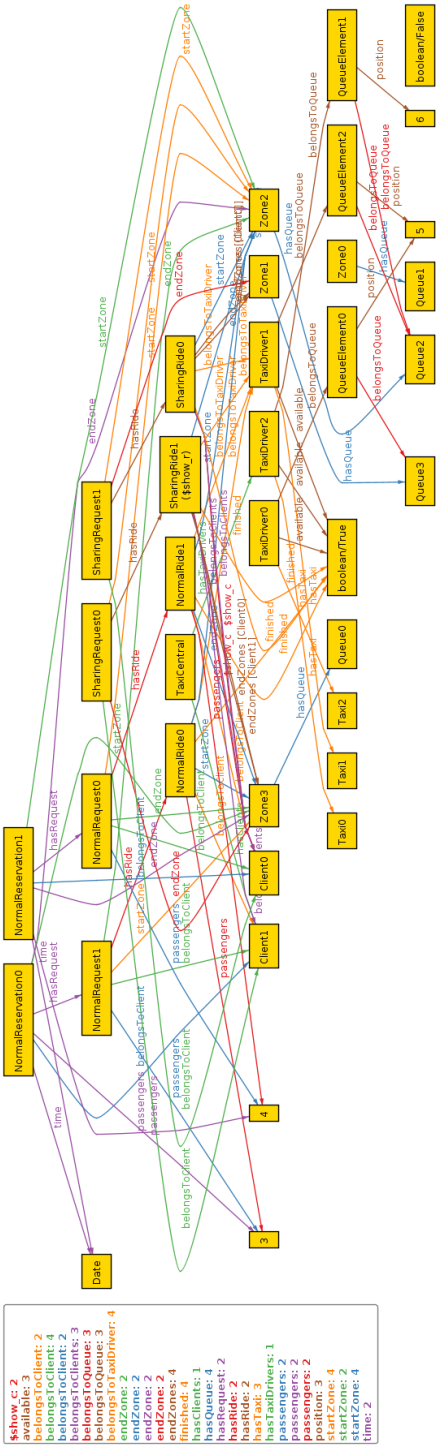
```
}

sig NormalReservation extends Reservation {
hasRequest: lone NormalRequest,
passengers: Int
}

sig Date {}

//fact

//A taxi can stay in the queue only if it is available
// and it doesn't run a ride and viceversa
fact taxiDriverConstrains {
all t: TaxiDriver | (#t.belongsToQueue = 1 <=> t.available.isTrue) and
(t.available.isTrue <=> no r: Ride | r.belongsToTaxiDriver = t and
 r.finished.isFalse) and
(t.available.isFalse <=> one r: Ride | r.belongsToTaxiDriver = t and
 r.finished.isFalse) and
//only one ride at the same time
(no r1, r2: Ride | r1 != r2 and r1.belongsToTaxiDriver = t and
r2.belongsToTaxiDriver = t and r1.finished.isFalse and r2.finished.isFalse)
}

fact TaxiCanBeAssociatedtoOnlyOneTaxiDriver {
no t: Taxi | some t1, t2:TaxiDriver |
t1!=t2 and (t in t1.hasTaxi) and
(t in t2.hasTaxi)
}

fact QueuesCanBeAssociatedtoOnlyOneZone {
no q: Queue | some z1, z2:Zone |
z1!=z2 and (q in z1.hasQueue) and
(q in z2.hasQueue)
}

fact SharingRequestRideDataCorrespondence {
all req: SharingRequest | req.belongsToClient in req.hasRide.belongsToClients and
req.hasRide.startZone = req.startZone and
req.hasRide.endZones[req.belongsToClient] = req.endZone
}

fact NormalRequestRideDataCorrespondence {
all req: NormalRequest | req.belongsToClient = req.hasRide.belongsToClient and
req.hasRide.startZone = req.startZone and
req.hasRide.endZone = req.endZone and
```

```
req.hasRide.passengers = req.passengers
}

fact SharingReservationRequestDataCorrespondence {
all res: SharingReservation | res.belongsToClient = res.hasRequest.belongsToClient and
res.hasRide.startZone = res.startZone and
res.hasRequest.endZone = res.endZone
}

fact NormalReservationRequestDataCorrespondence {
all res: NormalReservation | res.belongsToClient = res.hasRequest.belongsToClient and
res.hasRequest.startZone = res.startZone and
res.hasRequest.endZone = res.endZone and
res.hasRequest.passengers = res.passengers
}

fact NormalRequestCanBeAssociatedtoOnlyOneReservation {
no req: NormalRequest | some res1, res2: NormalReservation |
res1!=res2 and (req = res1.hasRequest) and
(req = res2.hasRequest)
}

fact SharingRequestCanBeAssociatedtoOnlyOneReservation {
no req: SharingRequest | some res1, res2: SharingReservation |
res1!=res2 and (req = res1.hasRequest) and
(req = res2.hasRequest)
}

fact NormalRideCanBeAssociatedtoOnlyOneRequest {
no ride: NormalRide | some req1, req2: NormalRequest |
req1!=req2 and (ride = req1.hasRide) and
(ride = req2.hasRide)
}

fact SharingAllRideMustBeAssociatedToARequest {
all r: SharingRide | one req: SharingRequest | req.hasRide = r
}

fact NormalAllRideMustBeAssociatedToARequest {
all r: NormalRide | one req: NormalRequest | req.hasRide = r
}

fact QueueElementCanBeAssociatedtoOnlyOneTaxiDriver{
no q: QueueElement | some t1, t2: TaxiDriver |
t1!=t2 and (q = t1.belongsToQueue) and
(q = t2.belongsToQueue)
```

```
}

fact AllQueueElementMustBeAssociatedToATaxiDriver {
all q: QueueElement | one t: TaxiDriver | t.belongsToQueue = q
}

fact SharingRideHasNoMoreThanFourClients {
all r: SharingRide | #r.belongsToClients <=4
//this implies no more than 4 requests
}

fact SharingStartDifferentFromEnd {
no r: SharingRide | all c: r.belongsToClients | r.startZone = r.endZones[c]
}

fact NormalStartDifferentFromEnd {
no r: NormalRide | r.startZone = r.endZone
}

fact OnlyOneRequestPerClientAtSameTime {
all c: Client |
(no r1, r2: SharingRequest | r1 != r2 and r1.belongsToClient = c and
r2.belongsToClient = c and #r1.hasRide = #r2.hasRide and #r2.hasRide = 1) and
(no r1, r2: NormalRequest | r1 != r2 and r1.belongsToClient = c and
r2.belongsToClient = c and #r1.hasRide = #r2.hasRide and #r2.hasRide = 1)
}

fact OnlyOneRidePerClientAtSameTime {
all c: Client |
(no r1, r2: SharingRide | r1 != r2 and r1.belongsToClient = c and
r2.belongsToClient = c and r1.finished.isFalse and r2.finished.isFalse) and
(no r1, r2: NormalRide | r1 != r2 and r1.belongsToClient = c and
r2.belongsToClient = c and r1.finished.isFalse and r2.finished.isFalse)
}


//pred

pred show(){
//at least one sharing ride with more than one clients
some r:SharingRide | #r.belongsToClients >1
#NormalRide>1
#TaxiDriver>1
#NormalReservation>=1
#SharingRequest>=1
}
```

```
//run

run show for 4
```

## World generated



## Used tools

The tools we used to create this RASD document are:

- DIA: for uml models
- Github: for version controller
- Pencil: for mockup
- Gedit and ReText: to write MarkDown with spell check
- Pandoc: to create pdf
- Alloy Analizer 4.2: to prove the consistency of our model.

## Hours of works

**Claudio Cardinale**

**Gilles Dejaegere**

**Massimo Dragano**