

REQUIREMENT ANALYSIS AND SPECIFICATION DOCUMENT

MyTaxiService

Authors:

M. Albanese, M. Bianchi, A. Carlucci



POLITECNICO
MILANO 1863

Contents

1	Introduction	2
1.1	Purpose	2
1.2	Scope	2
1.3	Stakeholders	2
1.4	Definitions, acronyms and abbreviations	2
1.4.1	Definitions	2
1.5	Product perspective	4
1.5.1	User interfaces	4
1.5.2	Hardware interfaces	12
1.5.3	Software interfaces	12
1.6	Assumptions	13
2	Specific requirements	14
2.1	Functional Requirements	14
2.2	Scenarios	18
2.3	Use cases and UML diagrams	21
2.3.1	Class Diagram	36
3	Alloy	37
3.1	Output	44
3.2	Generated Worlds	45
4	Appendix	48

1 Introduction

1.1 Purpose

This is the Requirement Analysis and Specification Document (RASD from now on). The aim of this document is to show the functional and non-functional requirements of the system-to-be, based on several important aspects: the needs expressed by the stakeholders, the constraints which it is subject to, the typical scenarios that will happen after its deployment. The targeted audience is mainly made of software engineers and developers who have to actually develop the service here described.

1.2 Scope

The system will be an optimization of a pre-existing, non-software solution for renting taxis already in use in the city. The new system will let users to rent or reserve a taxi through a mobile or a web application and will also let taxi drivers to take care of the users' requests in a more simple and effective way. In addition to a better user interface, the new system will focus on a smarter organization of the vehicles deployed in each city zone, resulting in a more efficient service for the citizens.

1.3 Stakeholders

MyTaxiService has several stakeholders. The prime stakeholder on the supply side is the **government**, whose aim is to expand and simplify its taxi service in order to make it more easy and immediate to use and also to save public money by tightening the reservation process.

Another supply-side stakeholders are **taxi drivers** whose right amount of work will be guaranteed and communication with citizens will be improved.

On the demand side, the prime stakeholders are **citizens**. Their taxi experience will be enhanced using all the platform's functionalities and services. Last but not least, also third-party developers are stakeholders because they will be constantly improving this service.

1.4 Definitions, acronyms and abbreviations

1.4.1 Definitions

- **Request** for a taxi: A customer asks for a taxi that he wants to use immediately.
- **Reserve** a taxi: A customer asks for a taxi for a specified date and hour.
- **Place**: the address made of street name and street number

- **Sharing:** The possibility for a customer to share a reserved taxi (does not apply to requested taxis).
- **Agents**
 - **Guest:** A person who has not logged in yet. He can only sign up or sign in if already registered in the system.
 - **User:** A registered person either as a Customer or as a Taxi driver.
 - **Customer / Passenger:** A person who has already registered as a client of the service via mobile or web app. He can call for a taxi, reserve one or modify his own profile info.
 - **Taxi driver:** This user is registered as a counterpart of an actual driver. The registration into the system is not made by the driver himself but by the City after verifying its necessary documents. He can notify his availability to the system and accept a request for a ride.
- **Taxi queue:** the internal data structure that handles the taxi for a certain zone in the city.
- **Zone:** area of approximately 2 mq². The city is partitioned into several zones, each of which is covered by at most a taxi.
- **Taxi Handler:** the object which handles each taxi allocation after a request or a reservation has been made by a customer.

1.5 Product perspective

The software-to-be is going to be made of several parts: two front-end applications for the customers, a different one for the taxi drivers and an API for developers.

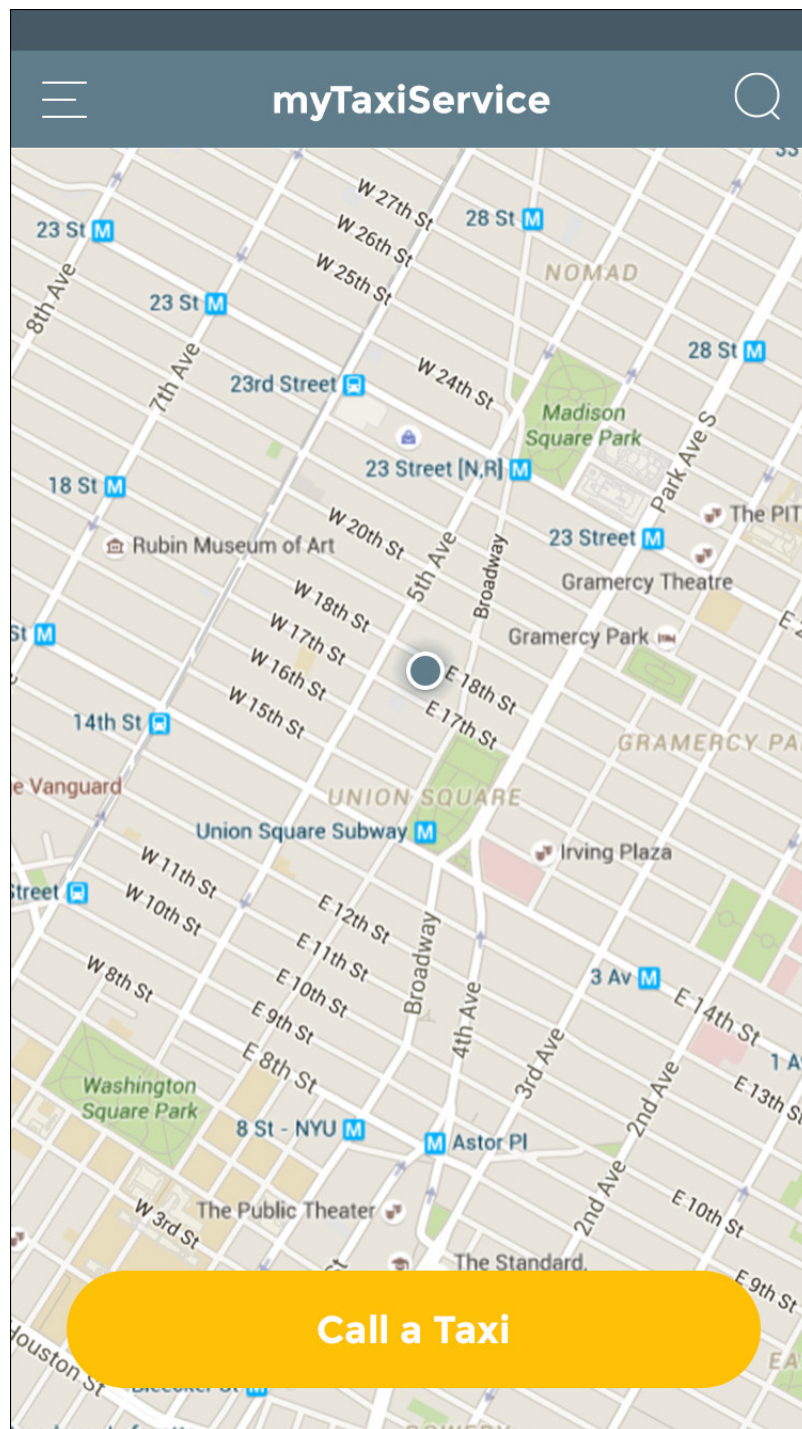
Concerning the customer interface, it will be pretty similar to other related apps (for example, Uber or MyTaxi), since it will supply similar functions. It will primarily let them reserve or request a taxi.

As for the taxi drivers, a simpler interface will be given, since only one functionality will be developed (accepting a request for a ride).

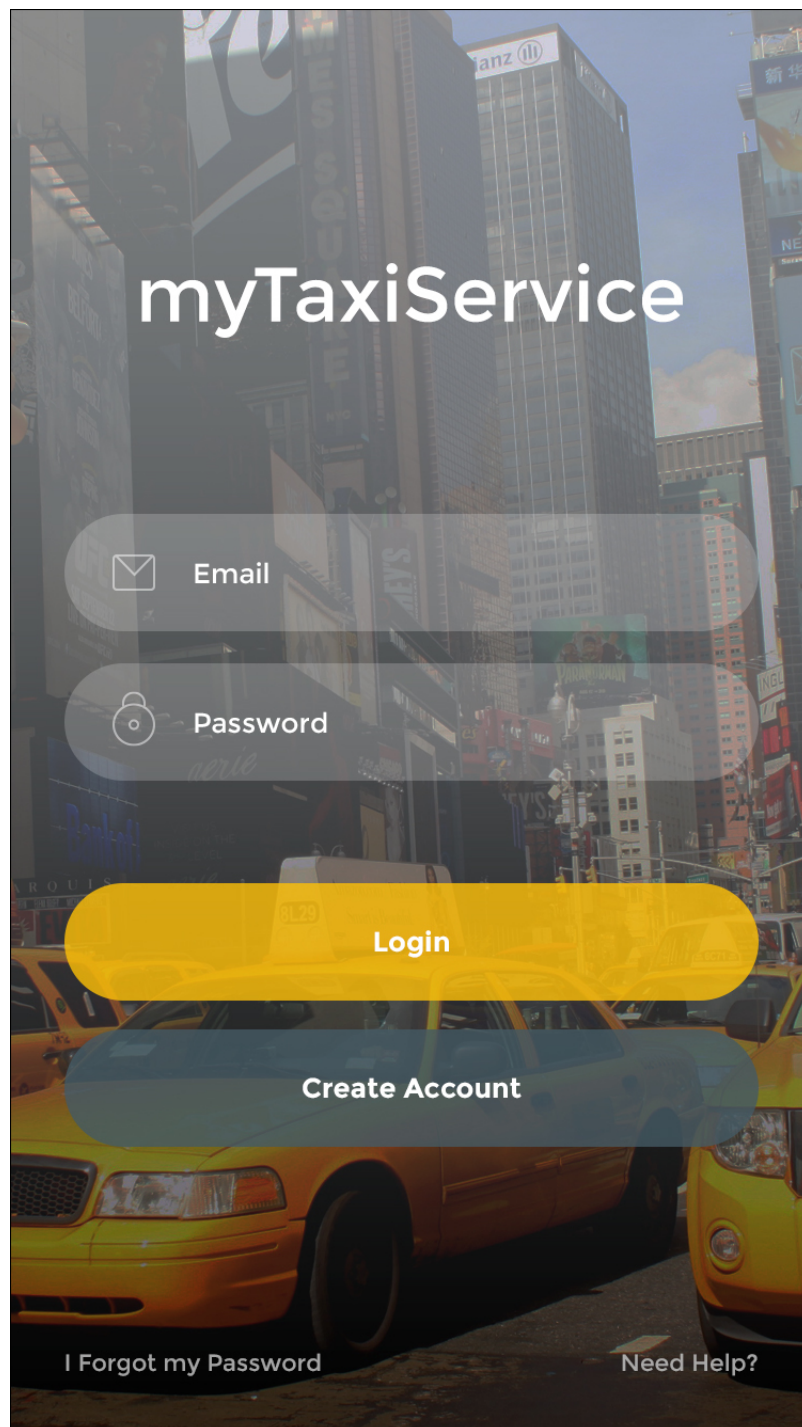
Finally, the API will provide all public methods described in the class diagram.

1.5.1 User interfaces

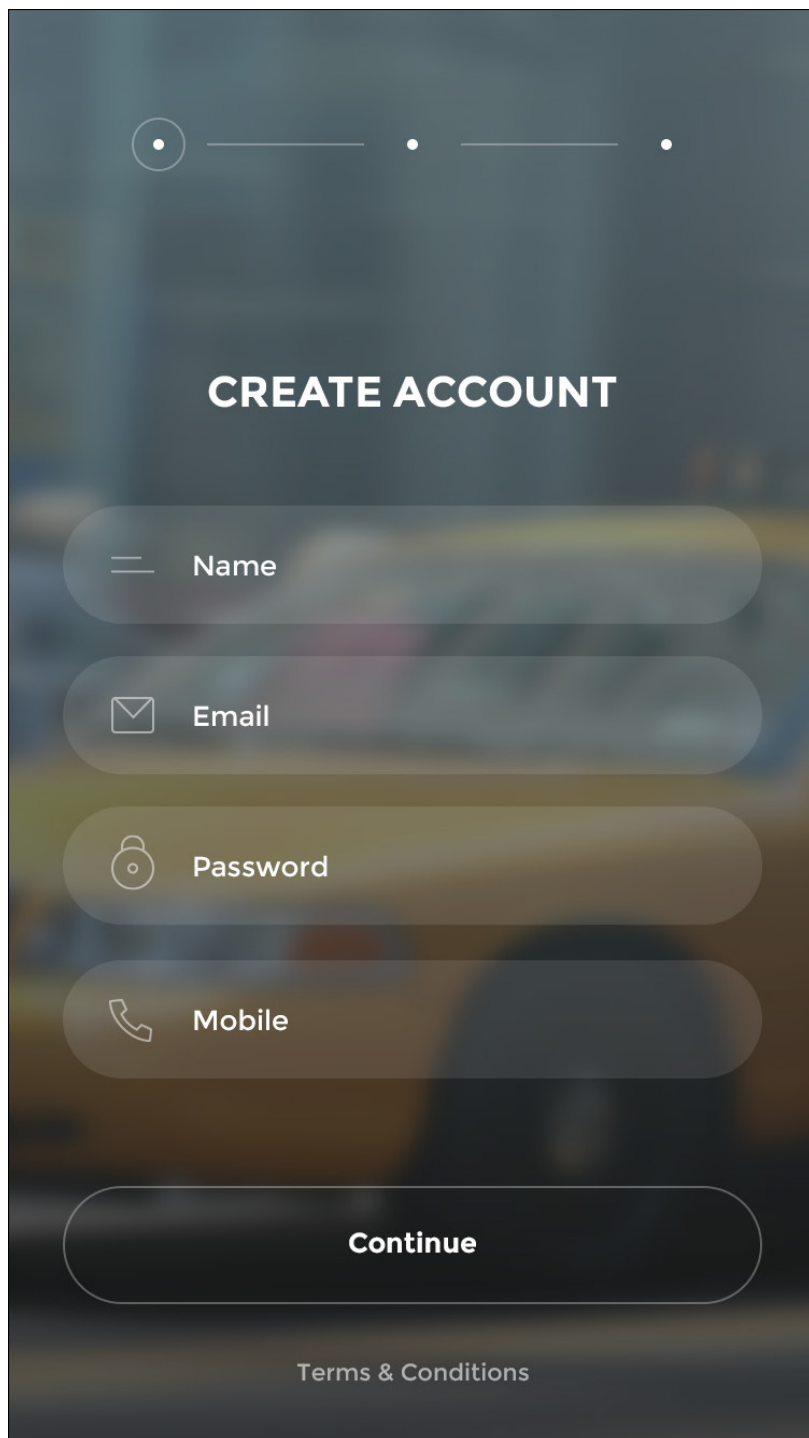
The application will have two client interfaces, a web app and a mobile app. For simplicity's sake, here are presented the mobile interfaces only.



Mockup 1: The home app page. Clicking on “Call a taxi” opens the “Request a taxi” screen.




Mockup 2: The login screen for a customer.





A registration page mockup with a dark background and a blurred image of a person. At the top, there is a progress indicator with three dots, the first of which is highlighted with a circle. Below this, the text "CREATE ACCOUNT" is centered in a bold, white, sans-serif font. Underneath the title are four input fields, each with a white icon on the left and a label on the right: a horizontal line icon for "Name", an envelope icon for "Email", a padlock icon for "Password", and a telephone handset icon for "Mobile". Each input field is a rounded rectangle with a light gray border. At the bottom of the form is a large, rounded "Continue" button with a white border and white text. Below the button is a link labeled "Terms & Conditions" in a smaller, lighter font.


• — • — •

CREATE ACCOUNT

 Name

 Email

 Password

 Mobile

Continue

[Terms & Conditions](#)

Mockup 3: The registration page for a guest willing to become a customer.

myTaxiService

RESERVE A TAXI

From

331 W 30th St, New York, NY

To

JFK Airport, New York, NY

Date

Tomorrow at 5.00 AM

TRIP OPTIONS

Share my Taxi

Number of people

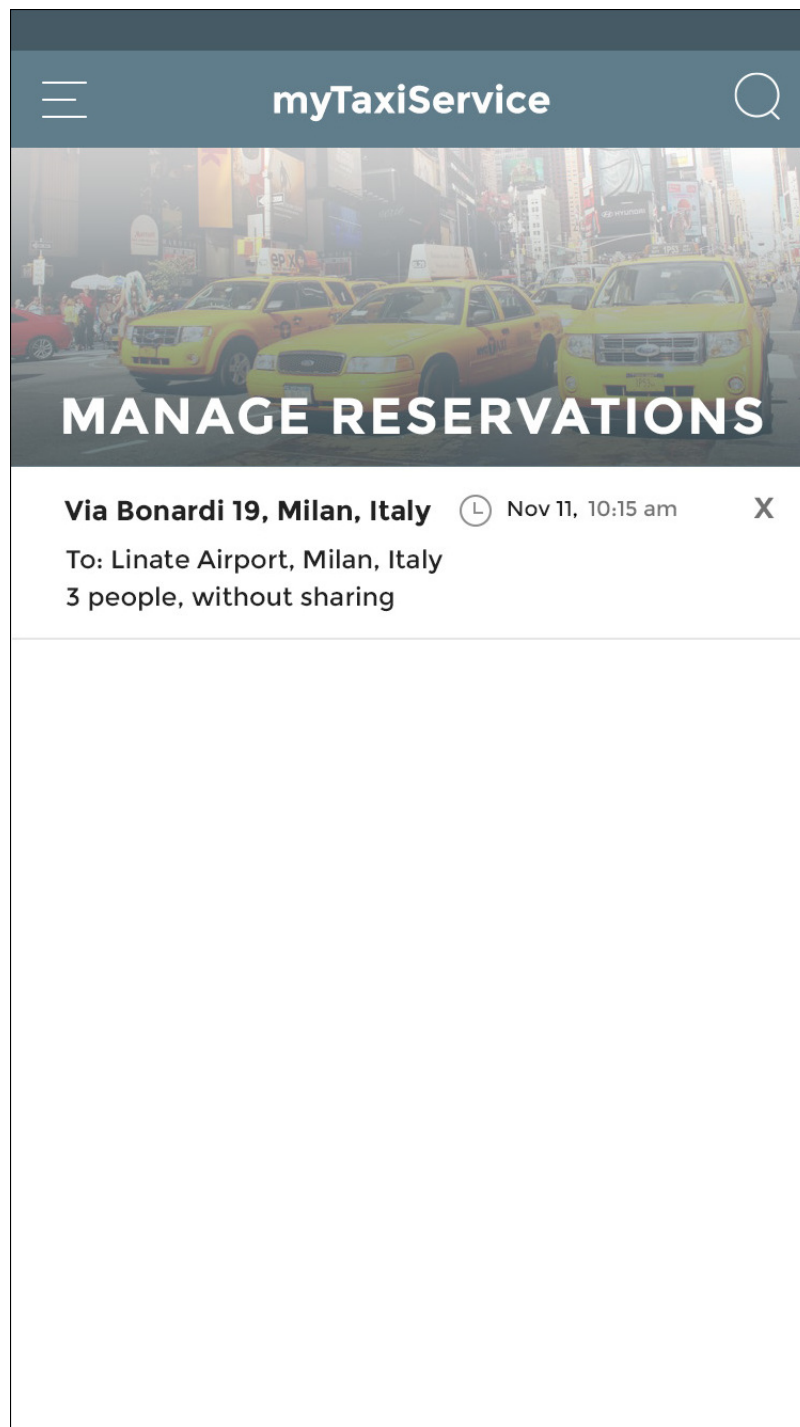
This taxi will be serving 2 people

Payment method

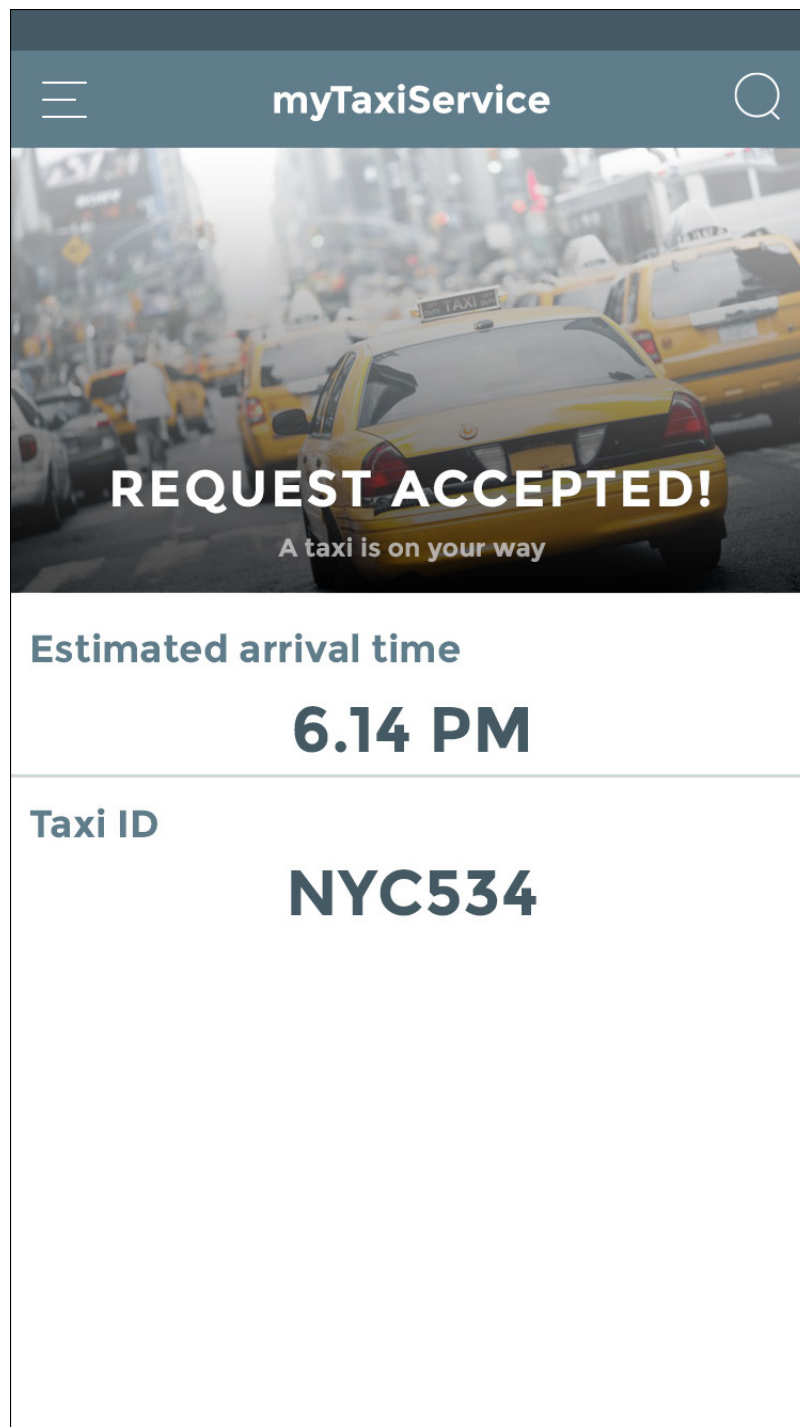
Current VISA **** * 1661

Reserve a Taxi

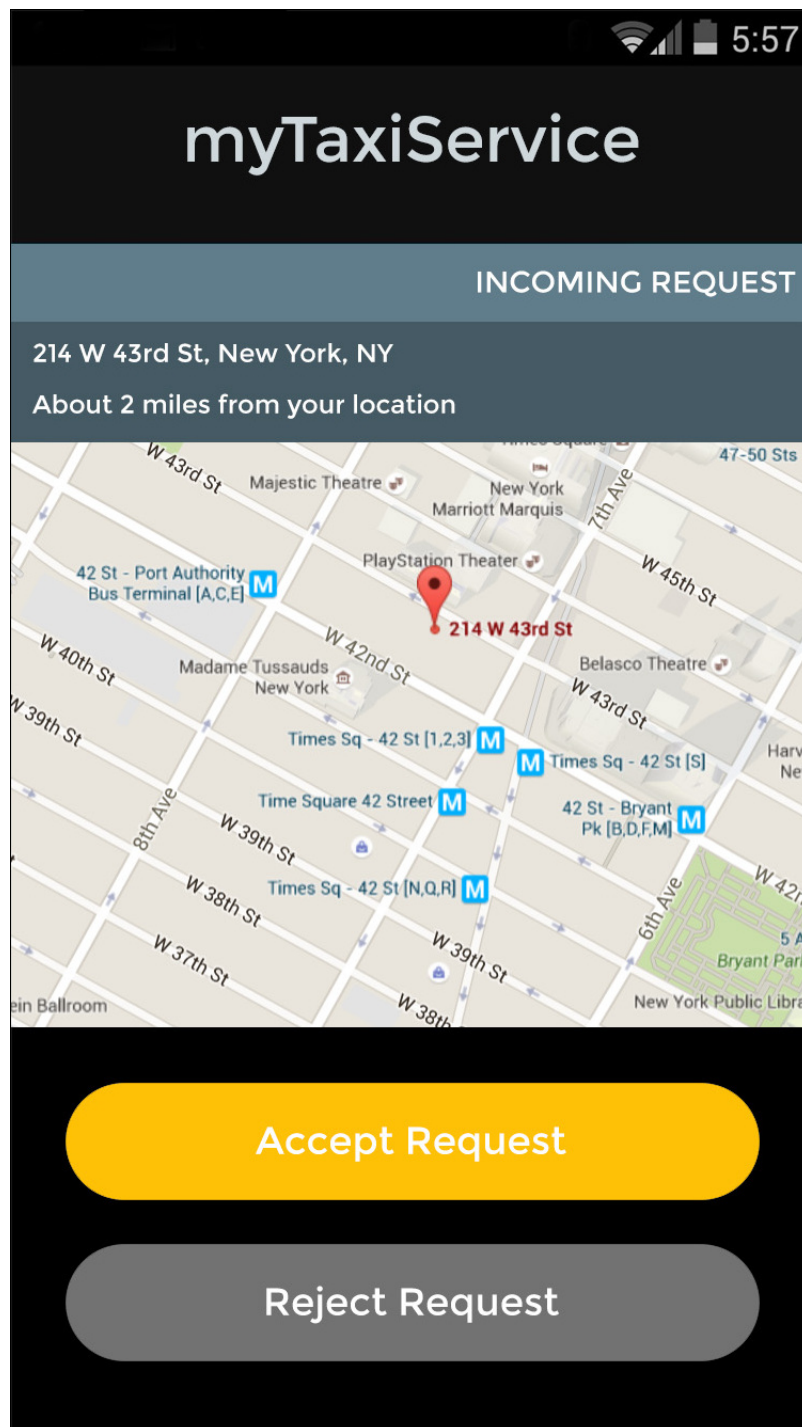
Mockup 4: Here is the page where a customer can reserve a taxi for a specified day. Starting point, destination and time are mandatory information. The mockup also illustrates the “Sharing” option enabled. The “Request a taxi” functionality uses a stripped-down version of this window (no sharing option, only current day available)



Mockup 5: Here is the page where a customer can delete one of his reservations.



Mockup 6: After requesting or reserving a taxi, the service shows you a confirmation screen.



Mockup 7: This shows how the notification appears on a taxi driver's smart-phone. No web app interfaces available for this functionality.

1.5.2 Hardware interfaces

In order to correctly use myTaxiService, a GPS system is required in order to track all vehicles and distribute them fairly in each zone. In addition to this, some vehicles have a point-of-sale terminal (POS), which is expressly indicated in the system; this way, if a user chooses to pay with a credit card, only POS-enabled cars are used to grant the service.

1.5.3 Software interfaces

- **Back-end**

- **DBMS:**

- * Name: MySQL
 - * Version: 5.7
 - * Source: <http://www.mysql.it/>

- **Programming language:**

- * Name: Java (EE framework)
 - * Version: 7
 - * Source: <http://www.oracle.com/technetwork/java/javasee/overview/index.html>

- **Operating System:**

- * Name: Linux Debian
 - * Version: 8.2
 - * Source: <https://www.debian.org/>

- **Map API:**

- * Name: Google Maps
 - * Source: <https://developers.google.com/maps/>

- **Front-end**

- **Operating Systems:**

- * Android (Client + Driver mobile app)
 - * iOS (client)
 - * Blackberry (client)

1.6 Assumptions

Here is a list of domain assumptions the writers of this document assume to hold in the real world:

- A taxi driver is available only when in his own vehicle
- A taxi driver notifies his availability if and only if he is actually available for a ride (not busy in another one nor not working at the moment)
- At least a driver in a queue is available to accept a ride
- Customers who decided to share a taxi always accept other customers without any discrimination
- Each request is sent to one taxi at a time
- Each taxi driver actually serves a request if he accepted it
- Given a place, the system can always detect the zone where it is
- If a customer reserves or requests a taxi, he will use it
- If a taxi driver wants to take care of a request, he accepts it
- The customer will actually show up at the given place, date and time when his request or reservation is made
- The number of drivers is greater or equal than the number of zones
- There already is a geolocalization system for each taxi, based on the GPS information
- There is always an available taxi driver in every queue
- There is a bijection between a taxi and its driver (aka a taxi belongs to just a driver and a driver uses just a vehicle)

2 Specific requirements

2.1 Functional Requirements

Goal 1: Allow guests to register to the platform.

Requirement 1: The service shall prevent guests from accessing any service before being registered or logged in.

Requirement 2: The system shall validate any input by the guest.

Requirement 3: The system shall send a verification link and an SMS with a code to the user who has just signed up in less than 5 minutes.

Requirement 4: The system shall expire the validation link if not used after 48 hours.

Goal 2: Allow users to access the platform by logging in.

Requirement 1: The system shall validate any input by the system before sending it.

Requirement 2: The system shall check in the DB if user and password are correct and grant access if so.

Requirement 3: The system shall prevent anyone from logging more than once at a time.

Goal 3: Guarantee a fair management of the zone queues.

Requirement 1: The system shall send a notification to the taxi driver on top of the queue when a request comes from a customer in the same zone.

Domain Assumption 1: There is always an available taxi driver in every queue.

Domain Assumption 2: At least a driver in a queue is available to accept a ride.

Goal 4: Allow taxi drivers to indicate their availability.

Requirement 1: The system shall provide two buttons on the taxi driver's app, "Notify Availability" and "Notify Unavailability".

Requirement 2: The system shall let a taxi driver to notify his availability only when unavailable and viceversa.

Domain Assumption 1: A taxi driver notifies his availability if and only if he is actually available for a ride (not busy in another one nor not working at the moment).

Goal 5: ~~Guarantee presence of taxi drivers in every zone.~~

Requirement 1: ~~The system shall assign each taxi to a zone only if the number of taxis in that zone is between a minimum and a maximum threshold, determined according to the expected traffic.~~

Domain Assumption 1: ~~The number of drivers is greater than the number of zones.~~

Goal 6: Allow developers to add functionalities to the system.

Requirement 1: The system must provide APIs in at least a programming language.

Requirement 2: The system shall provide all basic functionalities that can be found in the web/mobile app through these APIs.

Requirement 3: The system must check that every command submitted by APIs could be run at that user level of privileges.

Goal 7: Allow customers to request a taxi.

Requirement 1: The system shall let the user insert starting point, destination, payment method and number of people.

Requirement 2: The system shall check if the input is valid (aka the places actually exist).

Requirement 3: The system shall send the request as soon as the customer clicks on the Request button.

Requirement 4: The system shall show a confirmation screen with the Taxi ID and expected time of arrival.

Requirement 5: The system shall let the user choose a payment method between cash and credit card.

Domain Assumption 1: If a customer reserves or requests a taxi, he will use it.

Goal 8: Allow the system to efficiently allocate a taxi.

Requirement 1: When a customer calls for a taxi using his app, the system must deliver a request to the first taxi driver in the customer's zone queue that meets the payment prerequisites (e.g. only POS-enabled taxis must be used when the customer decided to use a credit card as payment method) and has enough seats.

Requirement 2: The system shall set the taxi driver status to Busy when a request is accepted by him.

Requirement 3: The system shall dequeue the taxi driver from the front and enqueue it in the bottom if he does not accept a request.

Domain Assumption 1: If a taxi driver wants to take care of a request, he accepts it.

Goal 9: Allow customers to reserve a taxi for a given date and time.

Requirement 1: The system shall let the user insert starting point, destination, payment method, number of people and date time.

Requirement 2: The system shall prevent the customer from reserving a taxi in less than 2 hours.

Requirement 3: The system shall start the allocation process only 10 minutes before the reservation time.

Requirement 4: The system shall check if the input is valid (aka the places actually exist).

Requirement 5: The system shall let the user choose a payment method between cash and credit card.

Domain Assumption 1: If a customer reserves or requests a taxi, he will use it.

Domain Assumption 2: The customer will actually show up at the given place, date and time when his request or reservation is made.

Goal 10: Allow customers to share a ride.

Requirement 1: The system shall provide a Sharing option when reserving a taxi.

Requirement 2: The system shall automatically insert the customer willing to share in the first shared-reserved taxi if the info provided coincide and at least one taxi meet these requirements.

Requirement 3: If a user wants to share a ride but no taxis are already shared-reserved, the system shall let the user reserve a new one as usual.

Requirement 4: The system shall automatically elaborate the shortest path to reach each customer for that ride.

Requirement 5: The system shall calculate each fee according to the actual distance run by each customer.

Domain Assumption 1: Given a place, the system can always detect the zone where it is.

Domain Assumption 2: Customers who decided to share a taxi always accept other customers without any discrimination.

Goal 11: Allow customers to manage their reservations.

Requirement 1: The system shall provide a list of personal reservations.

Requirement 2: The system shall let the possibility to remove a reservation if that is done at least 10 minutes before the reservation time.

2.2 Scenarios

Scenario 1

A shy student

Michael does not want to lose his favorite class, but unfortunately a strike has been announced for today. Since he lives not too far nor too near from his university, he decides to take a taxi; unfortunately, his shyness does not let him call for one. So he decides to use My-TaxiService, the application developed by his own city, in order to get a cab as soon as possible. As a registered user, the only thing he has to do is open the app, log in, click on the Get a taxi now button and just wait for the response to come.

Scenario 2

The german IT entrepreneur

Siegfried is a German IT entrepreneur who has just arrived to the city for a working meeting. Since he does not trust the public transport service, he wants to rely on the taxi service. After a quick search on Google, he finds myTaxiService and decides to download the app. After filling in his own personal data, including the cell phone number, he clicks on the Submit button; after few seconds, he receives an email with a link so that he can confirm his registration and start using the service.

Scenario 3

Taxi reservation

Norma has to fly to Denmark the next day early in the morning. Due to this, she cannot take any public transportation service. Being a very organized woman, she decides to reserve a taxi the night before. Since she has very little time, she wants a friendly and fast way to do so; she decides then to use the Reserve a taxi functionality provided by myTaxiService web app on her laptop. The only things she has to insert are her starting point, her destination and when to leave. After that, a confirmation message is given and a taxi is successfully reserved.

Scenario 4

The amusement park lover

Riccardo loves amusement parks and so he decides to organize a full day in the nearest one. He knows that the trains and busses will be pretty unusable; since he wants to spend as much as possible on his entertainment, he decides to reserve a taxi the night before by using the Sharing option. Luke and Christine decide to do the same too and find out that Riccardo had already reserved a vehicle for the same date and the same starting point. They accept to take the same taxi and equally divide the fees.

Scenario 5

The taxi driver

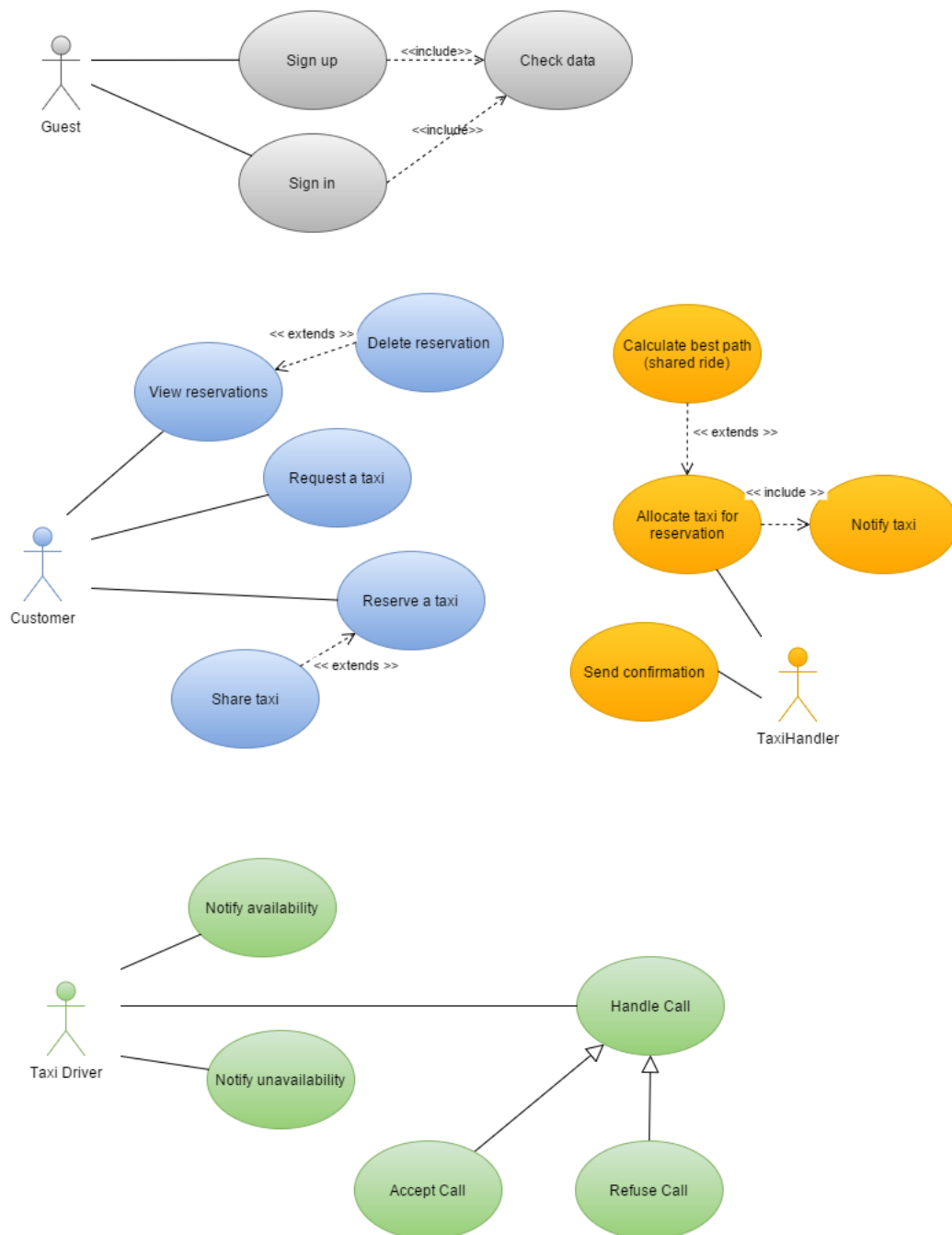
Max is a driver working for the public taxi service of his own city. During his daytime turn, his wife Marilyn calls from the city hospital saying that she is about to give birth to their child. During his frenetic race to the hospital a request comes from the system, but Max refuses to accept it by clicking on the Decline Request button. The system puts the driver in his zone queue tail. After that, Max realizes that he will be busy for a while so he decides to change his status to “unavailable”.

Scenario 6

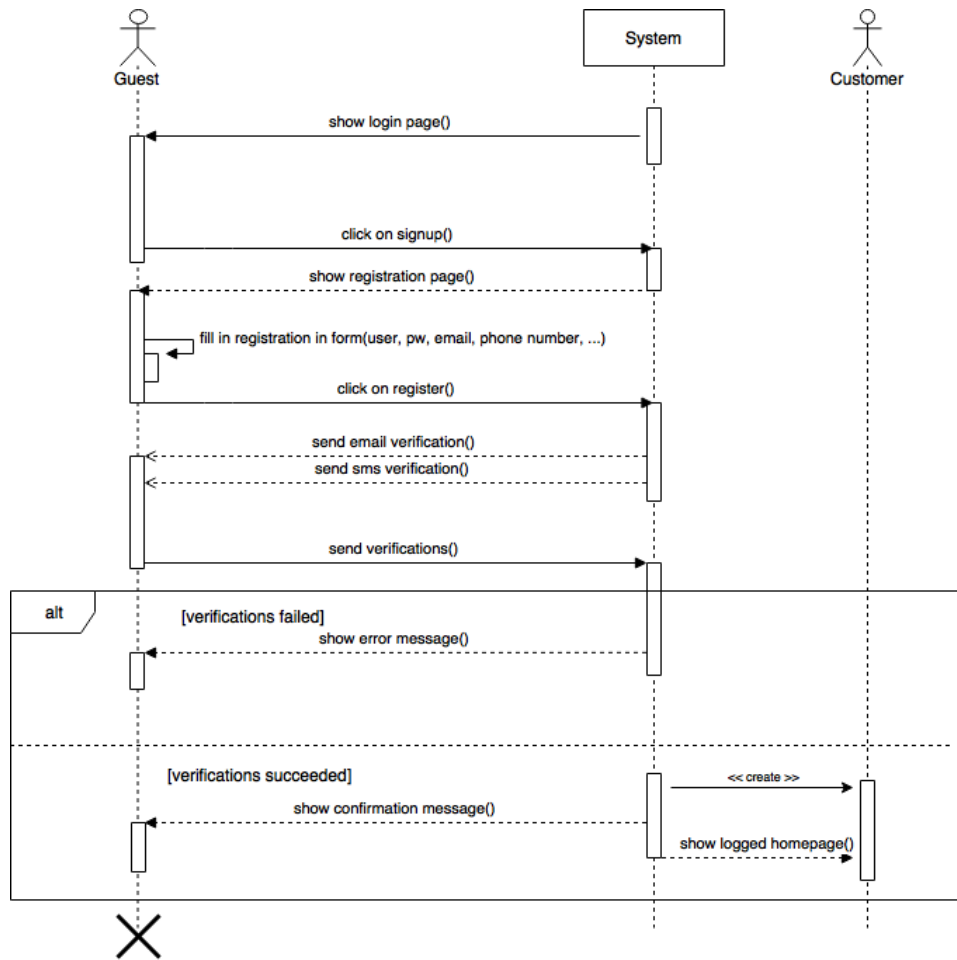
Cancel a reservation

Matthew is an engineering student who wanted to surprise his girlfriend, so he decided to reserve a taxi planning to show up in front of her house with a bouquet of roses. Unfortunately, few hours before the big event, her girlfriend broke up with him using a text message. With his heart broken, Matthew thinks that showing up at her place wouldn't be a good idea so he decides to go through his app in the “manage reservations” section and cancel his reservation for that day.

2.3 Use cases and UML diagrams

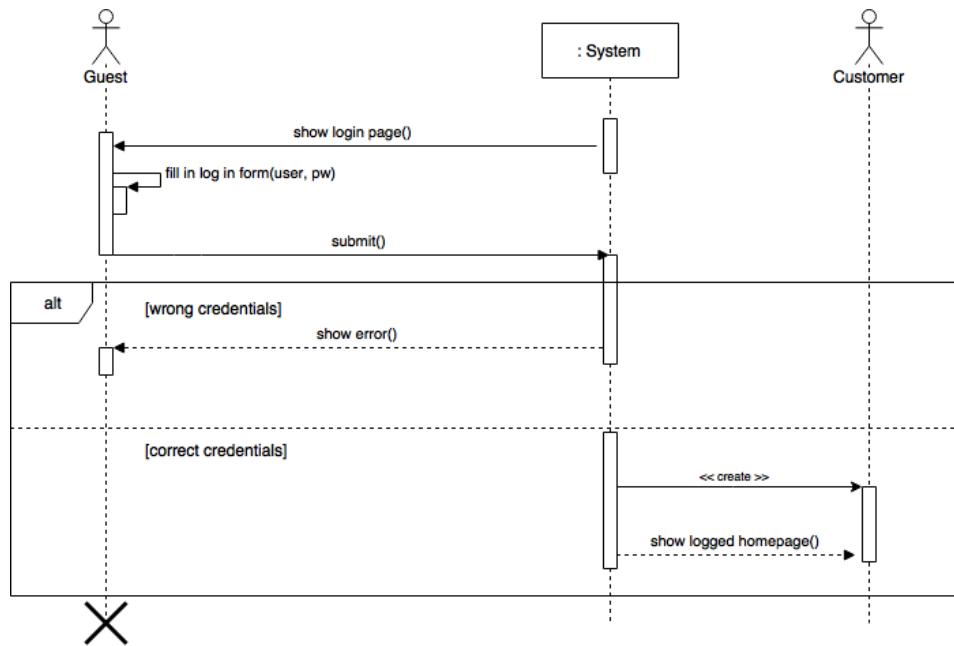


Use Case 1	
Name	Sign Up
Description	Registration to the system by a customer-to-be.
Primary Actor	Guest
Basic Flow	<ol style="list-style-type: none"> 1. Guests visit the service home page 2. The guest clicks on "Register" 3. The system outputs the form on screen 4. The guest inserts his name, mail, password and mobile phone number and then clicks on "Continue" 5. The system checks if the mail, password and phone number are syntactically valid; if so, it sends a mail and a sms with a verification link 6. The guest clicks on the link and verifies his account by inserting the code sent by sms 7. The guest is now a customer registered to the service
Alternate Flows	/
Exception	<ol style="list-style-type: none"> 1. The guest is already registered 2. One or more fields are not well-formed 3. Username already in use 4. The validation link is not more valid (after 48 hours)



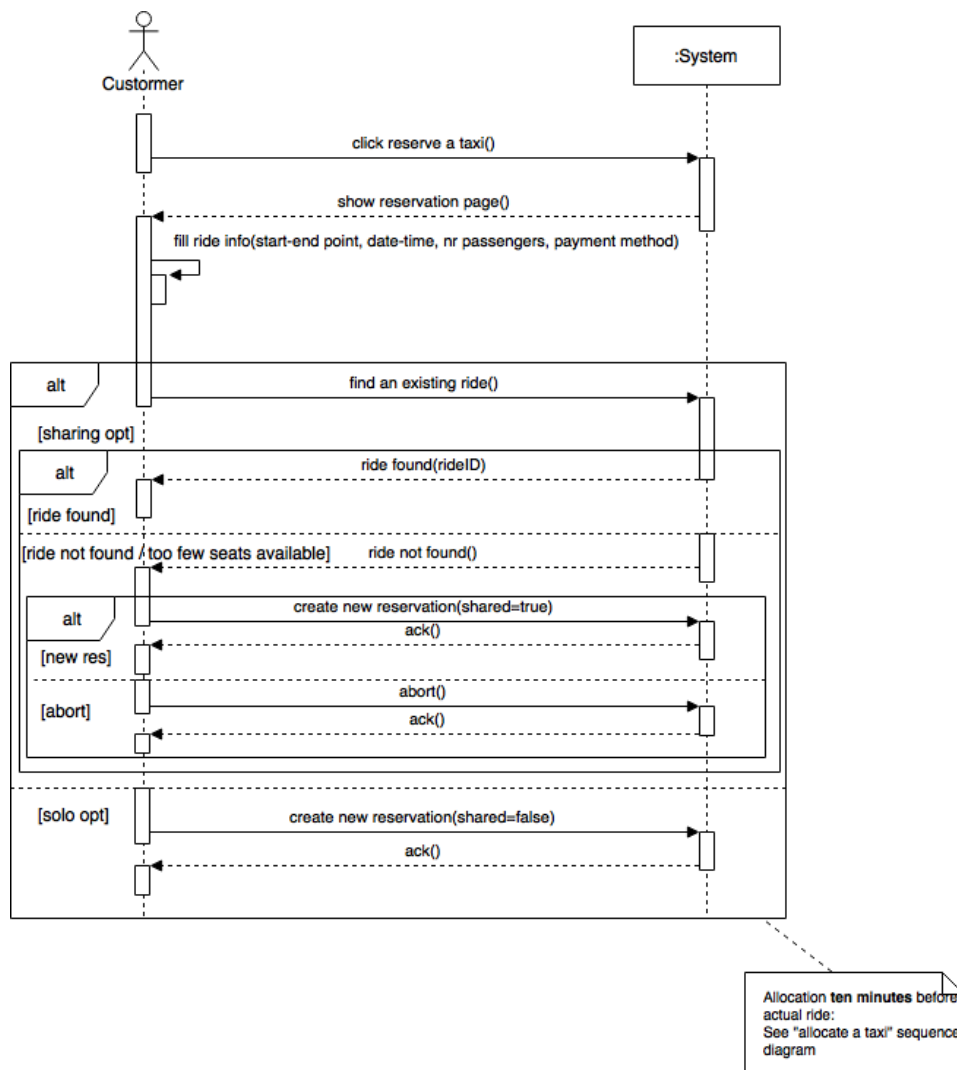
Sequence Diagram 1: Sign Up

Use Case 2	
Name	Sign in
Description	Login by a customer
Primary Actor	Guest, Customer
Basic Flow	<ol style="list-style-type: none"> 1. A guest visits the service home page 2. The guest clicks on "Log in" 3. The system outputs the form on screen 4. The guest inserts his name, password then clicks on "Continue" 5. The system checks if the mail, password and phone number are valid and present in the DB 6. The guest is redirected to the home page and recognized as a customer from now on.
Alternate Flows	/
Exception	<ol style="list-style-type: none"> 1. The user and/or password are incorrect 2. One or more fields are not well-formed 3. Username not registered

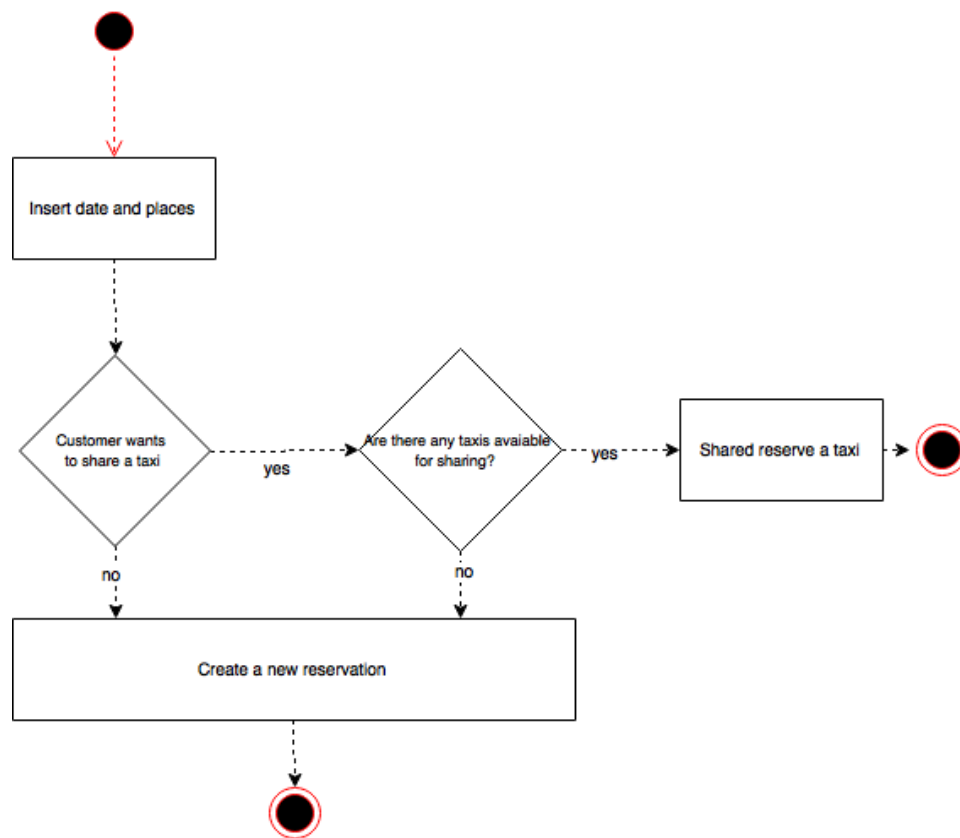


Sequence Diagram 2: Sign In

Use Case 3	
Name	Reserve a taxi
Description	Reservation of a taxi by a customer for a specified date, start and destination
Primary Actor	Customer
Basic Flow	<ol style="list-style-type: none"> 1. The customer clicks on "Reserve a taxi" 2. The system shows the reservation page 3. The customer fills in the mandatory information (starting point, destination, payment method, number of people and date-time) 4. The system registers the request and inserts it in the correct zone queue, according to the specified starting point. 5. The customer is informed that the operation did not fail thanks to a confirmation screen 6. TaxiHandler will evaluate it when needed (see "Allocate a taxi" use case)
Alternate Flows	<p>Sharing option</p> <ol style="list-style-type: none"> 1. Same operations as before till point 3 2. The user also specifies that he wants to share his taxi 3. The TaxiHandler assigns the customer to the first reserved taxi for that ride if present 4. If there are no available shared rides, a new one is created on the spot 5. Same operation as before from point 4 on
Exception	<ul style="list-style-type: none"> • The date and/or places are not valid

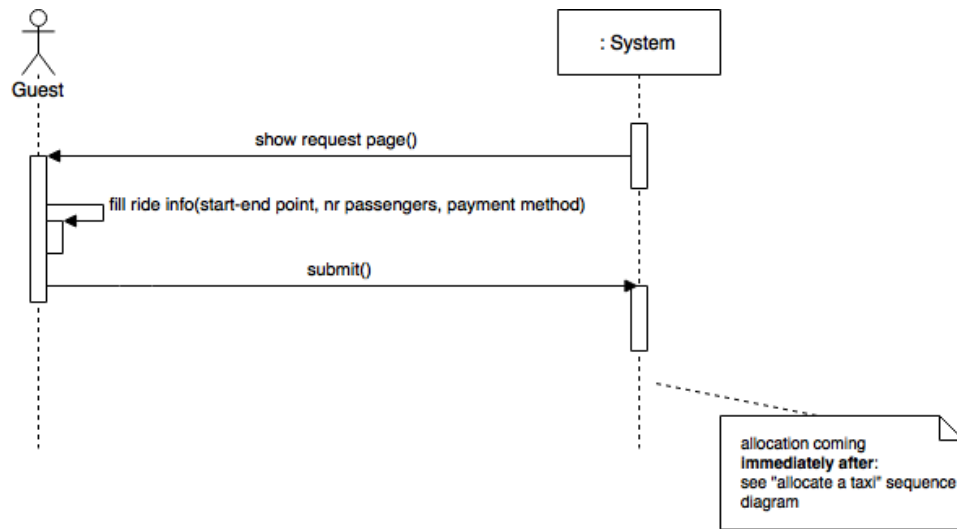


Sequence Diagram 3: Reservation



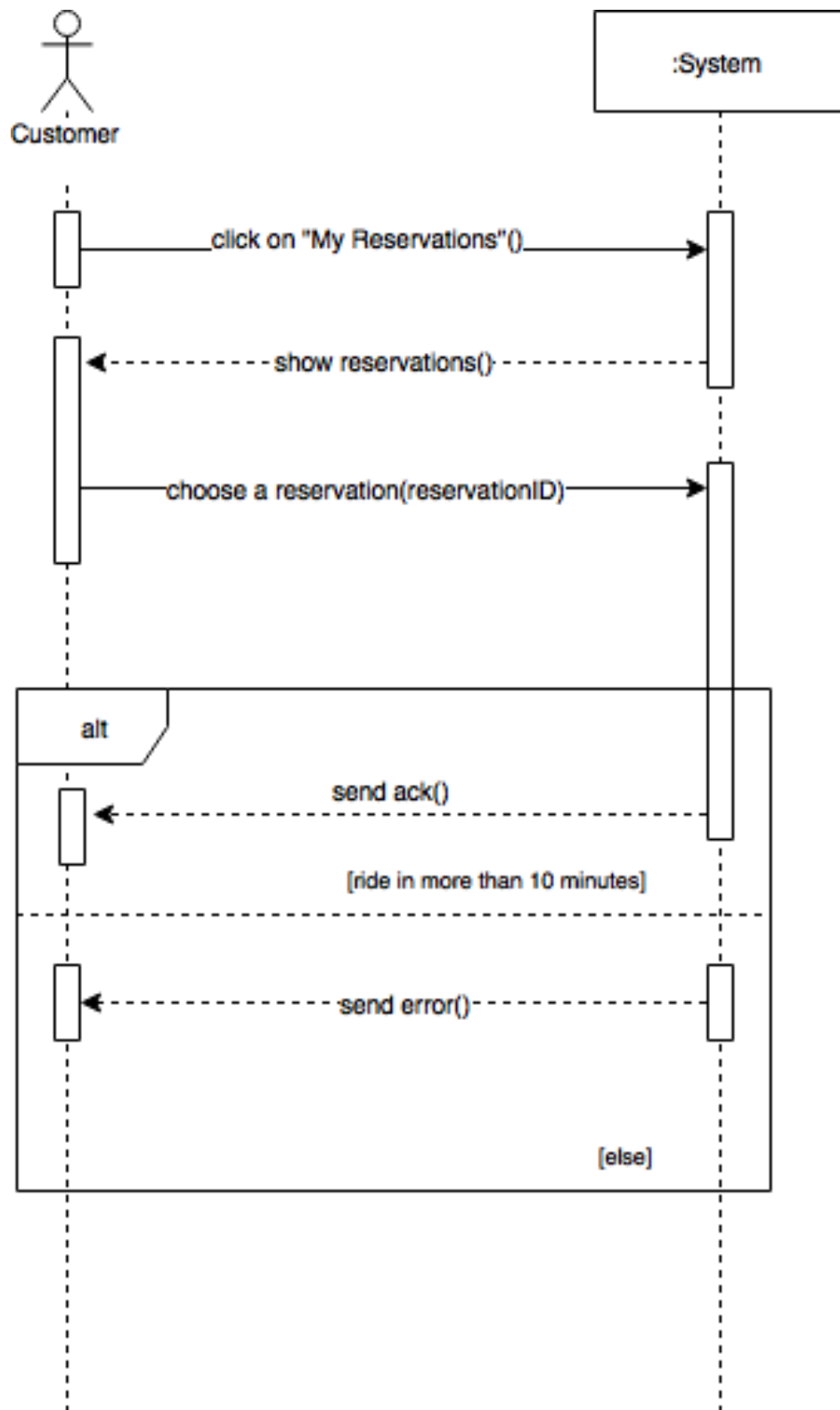
FSM 1: Reserve FSM

Use Case 4	
Name	Request a taxi
Description	Quick taxi request for a customer who needs a taxi for the immediate future at its location
Primary Actor	Customer, TaxiHandler
Basic Flow	<ol style="list-style-type: none"> 1. The customer clicks on "Request a taxi" 2. The customer fills in the mandatory information (starting point, destination, payment method and number of people) 3. The system registers the request and inserts it in the correct zone queue, according to the specified starting point. 4. TaxiHandler processes the call (see "Allocate a taxi" use case) 5. The customer is informed that the operation did not fail thanks to a confirmation screen
Alternate Flows	/
Exception	/



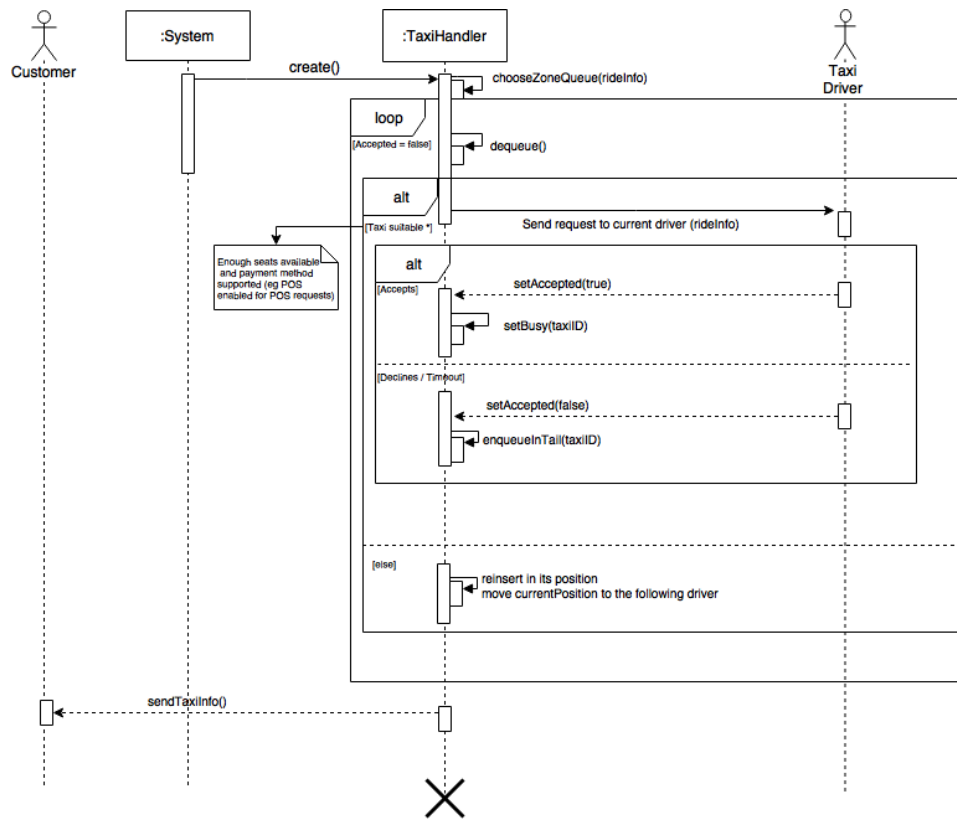
Sequence Diagram 4: Request a taxi

Use Case 5	
Name	Delete reservation
Description	Possibility of a customer to view and eventually delete one or more reservations made by him
Primary Actor	Customer, Taxi Handler
Basic Flow	<ol style="list-style-type: none"> 1. The customer clicks on "Manage reservations". 2. The customer sees the entire list of his reservations (with details for each one) and may delete them.
Alternate Flows	/
Exception	<ul style="list-style-type: none"> • The deleted reservation occurs in less than 10 minutes from the delete action.

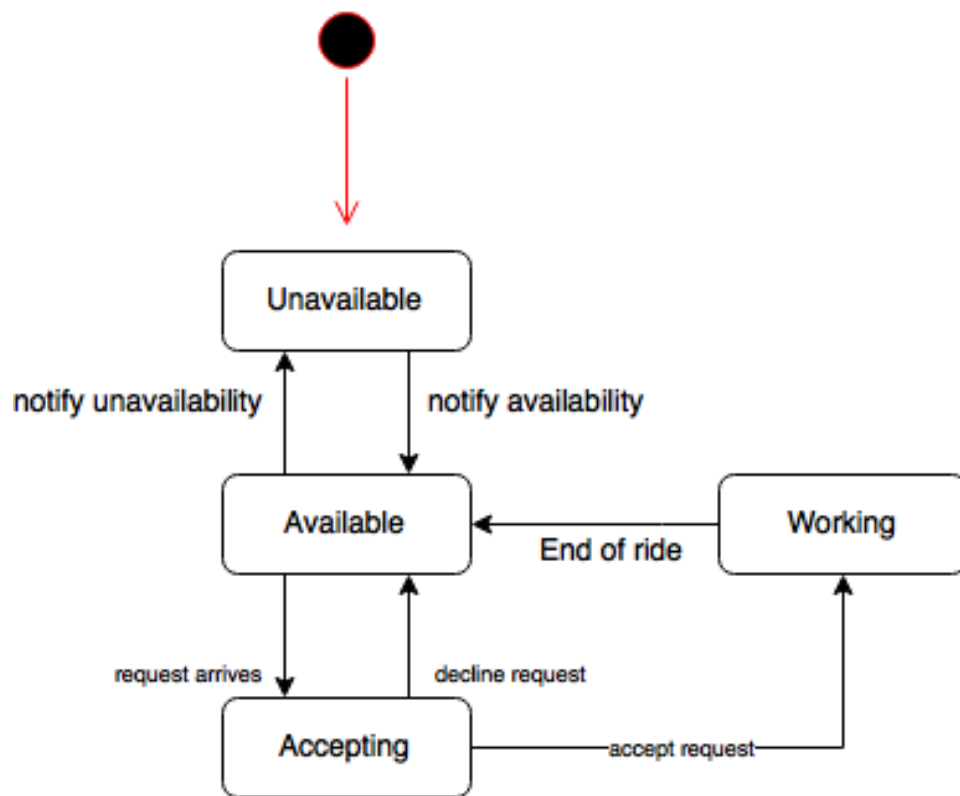


Sequence Diagram 5: Delete a reservation

Use Case 6	
Name	Allocate a taxi
Description	Selection and notification of a taxi driver to process a request or a reservation by the TaxiHandler
Primary Actor	Taxi Handler, Taxi Driver
Basic Flow	<ol style="list-style-type: none"> 1. The TaxiHandler sends a notification to the first taxi driver who meets the requirements in the given zone queue. 2. If a positive answer is given in 30 seconds the TaxiHandler provides to set the Driver as "busy". 3. Otherwise the driver is moved to the bottom of the queue. TaxiHandler restarts the seek from step 1.
Alternate Flows	/
Exception	/

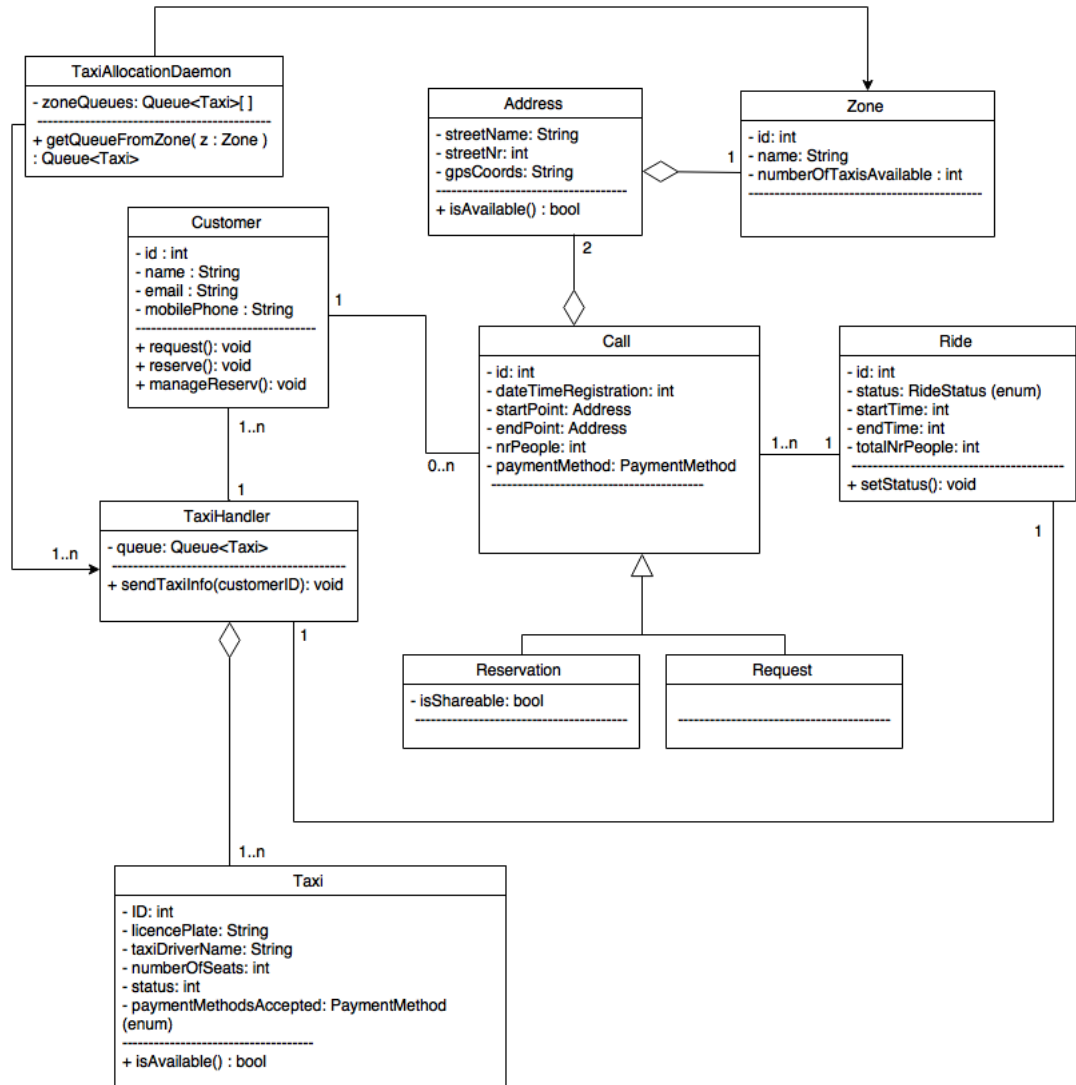


Sequence Diagram 6: Allocate a taxi



FSM 2: Taxi FSM

2.3.1 Class Diagram



3 Alloy

```
////////// SIGS //////////

sig Stringa { }

sig Address {
    streetName: one Stringa,
    streetNr: one Int,
    gpsCoords: one Stringa,
    zone: one Zone
} {
    streetNr > 0
}

sig Zone {
    id: one Int,
    name: one Stringa
} {
    id > 0
}

sig Ride {
    id: one Int,
    status: one RideStatus,
    startTime: one Int,
    endTime: lone Int,
    totalNrPeople: one Int
} {
    id > 0
    startTime > 0
    endTime = none or endTime > 0
    totalNrPeople > 0
}

abstract sig Call {
    id: one Int,
    timeRegistration: one Int,
    startPoint: one Address,
    endPoint: one Address,
    nrPeople: one Int,
    paymentMethod: set PaymentMethod,
    ride: one Ride
} {
    id > 0
    timeRegistration > 0
    nrPeople > 0
}

sig Reservation extends Call {
    isShareable: one Bool,
}

sig Request extends Call {}

sig Customer {
    id : one Int,
    name : one Stringa,
    email : one Stringa,
    mobilePhone : one Stringa,
    calls: set Call
}
```

```

}

sig TaxiAllocationDaemon {
    queues: set TaxiQueue
}

sig TaxiHandler {
    ride: one Ride,
    serve: set Customer,
    taxiQueue: one TaxiQueue,
    allocate: lone Taxi // zero if no taxis are assigned yet
} {
    allocate in taxiQueue.taxis
}

sig Taxi {
    id: one Int,
    licencePlate: one Stringa,
    taxiDriverName: one Stringa,
    numberOfSeats: one Int,
    status: one TaxiStatus,
    paymentMethodsAccepted: set PaymentMethod
} {
    id > 0
    numberOfSeats > 0
}

sig TaxiQueue {
    taxis: set Taxi
}

//////////////// ENUMS //////////////////

enum Bool {
    TRUE,
    FALSE
}

enum TaxiStatus {
    AVAILABLE,
    UNAVAILABLE,
    ACCEPTING,
    BUSY
}

enum RideStatus {
    PENDING,
    ASSIGNED,
    INRIDE,
    COMPLETED
}

enum PaymentMethod {
    CASH,
    POS
}

// note on dates: UNIX timestamps are used for convenience purposes

//////////////// FACTS //////////////////
// at least a zone in the city
fact atLeastAZone {

```

```

    #Zone ≥ 1
}

// there is always an allocation daemon running
fact taxiAllocationDaemonRunning {
    #TaxiAllocationDaemon = 1
}

// an UNAVAILABLE taxi driver must not appear in any queue
fact unavailableTaxiDriversNoQueue {
    all t : Taxi | t.status = UNAVAILABLE implies
        (no q : TaxiQueue | t in q.taxis)
}

// if there is no taxiHandler pointing to a taxi driver, that driver
// must not be BUSY and ACCEPTING
fact noBusyDriversWithoutTaxiHandlers {
    all t : Taxi |
        (no th : TaxiHandler | th.allocate = t) implies (t.
            status ≠ BUSY)
}

// bijection between taxiHandlers and rides
fact bijectionTaxiHandlerRide {
    (all th: TaxiHandler | one r : Ride | th.ride = r) ∧ // each
        ↪ taxiHandler has just a ride
    (all r: Ride | lone th: TaxiHandler | th.ride = r) // each
        ↪ ride belongs to just a taxiHandler if it still exists
}

// each TaxiHandler has just a reference to one of the queues of the
// TaxiAllocationDaemon
fact consistentZoneQueues {
    all th: TaxiHandler |
        one tad : TaxiAllocationDaemon |
            th.taxiQueue in tad.queues
}

// there are no queues outside the ones in the TaxiAllocationDaemon
// set
fact noExternalQueues {
    all q : TaxiQueue |
        one tad : TaxiAllocationDaemon |
            q in tad.queues
}

// number of zones = number of queues
fact equalZoneQueues {
    #{ Zone } = #TaxiAllocationDaemon.queues
}

// if a ride is in PENDING status, no taxi must be allocated yet
fact pendingRideNoAllocation {
    all r : Ride | r.status = PENDING implies
        ( (no th: TaxiHandler | th.ride = r) ∨ (one th :
            ↪ TaxiHandler | th.ride = r ∧
                #th.allocate = 0) )
    and
    all r : Ride | (r.status = INRIDE ∨ r.status = ASSIGNED)
}

```



```

    ⇨ implies
      ( one th : TaxiHandler | th.ride = r ∧ #th.allocate =
        ⇨ 1)
}

// if a ride is in COMPLETE status, then the corresponding taxiHandler
⇨ must not exist anymore
fact noTaxiHandlerForCompleteRides {
  all r : Ride | r.status = COMPLETED implies
    (no th : TaxiHandler | th.ride = r)
}

// if a ride is in INRIDE status, then the taxi driver must be busy
fact inRideTaxiDriver {
  all r : Ride | (r.status = INRIDE ∨ r.status = ASSIGNED)
    ⇨ implies
      one th : TaxiHandler | th.ride = r ∧
        #th.allocate = 1 ∧
        th.allocate.status = BUSY
}

// each ride must end after it started
fact noNegativeTimeRides {
  all r : Ride | (r.status = COMPLETED implies r.startTime < r.
    ⇨ endTime) ∧
    (r.status ≠ COMPLETED implies #r.endTime = 0)
}

// if there's a PENDING call, there must not be 2 taxis in ACCEPTING
⇨ status
fact oneAcceptingTaxiForACall {
  all r : Ride | r.status = PENDING implies
    ( (no th : TaxiHandler | th.ride = r) or (one th :
      ⇨ TaxiHandler | th.ride = r ∧ some t : Taxi | t
      ⇨ in th.taxiQueue.taxis ∧ t.status = ACCEPTING) )
}

fact acceptingTaxiRelatedToAPendingRide {
  all r : Taxi | t.status = ACCEPTING implies
    one th : TaxiHandler | th.ride.status = PENDING ∧ t in
      ⇨ th.taxiQueue.taxis
}

// a driver must be in just a queue
fact driverInOneQueueOnly {
  (no q1, q2 : TaxiQueue |
    q1 ≠ q2 ∧
    some t : Taxi | t in q1.taxis ∧ t in q2.taxis) ∧
  ( all t2 : Taxi | some q : TaxiQueue | t2 in q.taxis )
}

// there is at least a taxi available in each queue
fact atLeastOneTaxiAvailableInEveryQueue {
  all queue : TaxiQueue |
    some t : Taxi |
      t in queue.taxis ∧ t.status = AVAILABLE // but
      ⇨ at least one of them is available
}

// each customer is not duplicated

```

```

fact noDuplicatedCustomers {
  no c1, c2 : Customer |
    c1.id = c2.id ∧
    c1 ≠ c2
}

// each ride is not duplicated
fact noDuplicatedRides {
  no r1, r2 : Ride |
    r1.id = r2.id ∧
    r1 ≠ r2
}

// each zone is not duplicated
fact noDuplicatedZones {
  no z1, z2 : Zone |
    z1.id = z2.id ∧
    z1 ≠ z2
}

// each taxi is not duplicated
fact noDuplicatedTaxis {
  no t1, t2 : Taxi |
    (t1.id = t2.id ∨ t1.licencePlate = t2.licencePlate) ∧
    t1 ≠ t2
}

// Each street must be in the same zone
// NOTE: can fail if a street is very long
fact addressConsistentZone {
  all a1 : Address | no a2 : Address |
    a1.streetName = a2.streetName
    // a1 a1.streetNr = a2.streetNr
    ∧ a1.zone ≠ a2.zone
}

// there are no more than one reservation for a not sharable ride
fact notSharableRideNumberOfPassengers {
  no r1, r2 : Reservation |
    r1.isShareable = FALSE ∧ r2.isShareable = FALSE ∧
    r1.ride = r2.ride ∧
    r1 ≠ r2
}

// the number of people for a certain ride must equal the sum of all
//   ↪ people that booked that ride through several reservations
fact numberOfSeatsForARide {
  all r : Ride |
    all c : Call |
      c.ride = r ∧ // for all calls relative to a ride
      (sum p : Call | #p.nrPeople) = #r.totalNrPeople // the
      ↪ sum of people in all calls ...
}

// number of people in a ride must be <= than the taxi availability
fact taxiCanActuallyTakeRide {
  all th : TaxiHandler |
    th.allocate.status = ACCEPTING or th.allocate.status =
      ↪ BUSY implies
      (#th.ride.totalNrPeople ≤ #th.allocate.
      ↪ numberOfSeats)
}

```

```

// all requests have been made before the ride actually started
fact noRequestsAfterRide {
  all c : Call |
    c.timeRegistration < c.ride.startTime
}

// the allocated taxi is compatible with the chosen payment method
fact POSEnabled {
  all c : Call |
    one th : TaxiHandler | th.ride = c.ride ^
      th.allocate.paymentMethodsAccepted & c.paymentMethod ≠ none
}

// all taxis must accept cash as a valid payment method
fact cashTaxi {
  all t : Taxi |
    CASH in t.paymentMethodsAccepted
}

fact oneRideForATaxi {
  all t : Taxi | all th1, th2 : TaxiHandler | (th1 ≠ th2 ^ t =
    ↪ th1.allocate implies t ≠ th2.allocate)
}

fact pendingEqualAccepting {
  #{r : Ride | r.status = PENDING} ≥ #{ t : Taxi | t.status =
    ↪ ACCEPTING }
}

////////// ASSERTIONS //////////

// ##### A1 #####
// There are no queues anywhere with no available taxis (it also means
  ↪ that there is always a taxi in each queue)
assert noAllUnavailableTaxisInAQueue {
  all q : TaxiQueue |
    some t : Taxi | t in q.taxis ^ t.status = AVAILABLE
}

// WORKING
//check noAllUnavailableTaxisInAQueue for 7

// ##### A2 #####
// for every ride with an allocated taxi, there must be an equal
  ↪ number of busy taxi driver
assert equalNumberAllocationsBusyDrivers {
  #{th : TaxiHandler | #th.allocate = 1} = #{t : Taxi | t.status
    ↪ = BUSY}
}

// WORKING
//check equalNumberAllocationsBusyDrivers for 7

// ##### A3 #####
// it there are more than call for a ride, then it must be a shared
  ↪ reservation
assert sharedReservations {
  all c1, c2 : Call | (c1 ≠ c2 ^ c1.ride = c2.ride implies
    (c1 in Reservation and c2 in Reservation and c1.
      ↪ isShareable = TRUE and c2.isShareable = TRUE) )
}

```

```

// WORKING
//check sharedReservations for 15

// ##### A4 #####
// if there is an endTime set for a ride, that must be after the start
// ↪ time
assert timeConsistency {
  all r : Ride | #{r.endTime} > 0 implies
    r.startTime < r.endTime
}

// WORKING
//check timeConsistency for 10

// ##### A5 #####

////////// PREDICATES //////////

pred show(){
  #TaxiAllocationDaemon = 1
  #Address = 3
  #Zone = 2
  #Ride = 5
  #TaxiQueue = #Zone
  #TaxiHandler = #{r : Ride | r.status ≠ COMPLETED }
  #Taxi ≥ #Zone + 1
  #{ r : Ride | r.status = INRIDE } ≥ 1
  #{ r : Ride | r.status = PENDING } = 2
  #{ t : Taxi | t.status = ACCEPTING } ≥ 1
  #Customer ≥ 2
}

//ok
//run show for 15

pred shareable {
  #TaxiAllocationDaemon = 1
  #Customer ≥ 2
  all c: Customer | some x : Call | x in c.calls
}

// ok
//run shareable for 10

```

3.1 Output

Here is the result of Alloy Analyzer for each assert and predicate.

```
Executing "Check noAllUnavailableTaxisInAQueue for 7"
  Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
  43821 vars. 2555 primary vars. 119574 clauses. 573ms.
  No counterexample found. Assertion may be valid. 73ms.

Executing "Check equalNumberAllocationsBusyDrivers for 7"
  Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
  43760 vars. 2548 primary vars. 119696 clauses. 605ms.
  No counterexample found. Assertion may be valid. 2247ms.

Executing "Check sharedReservations for 15"
  Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
  162731 vars. 8010 primary vars. 430470 clauses. 565ms.
  No counterexample found. Assertion may be valid. 1443ms.

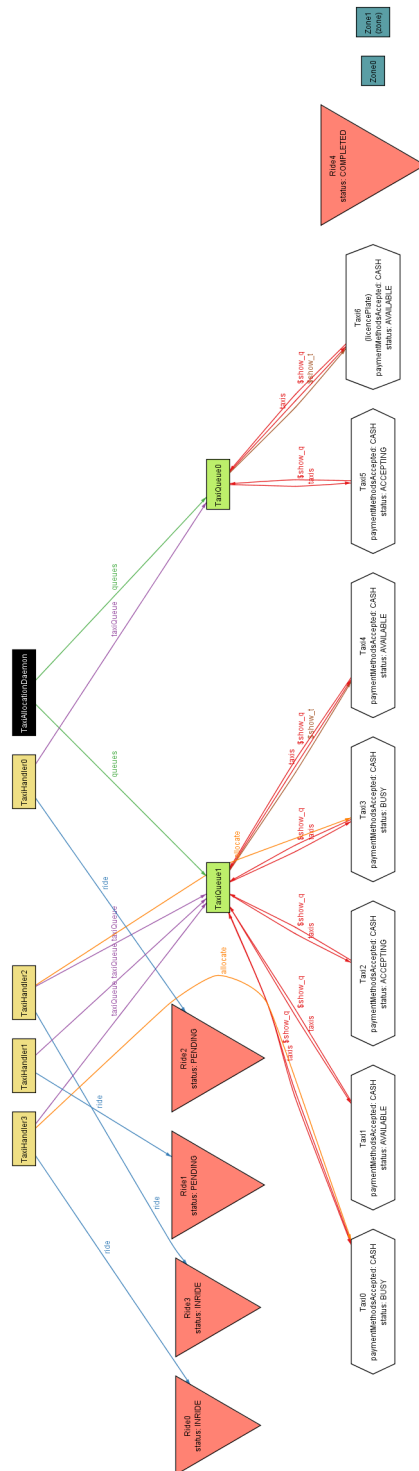
Executing "Check timeConsistency for 10"
  Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
  77524 vars. 4280 primary vars. 210321 clauses. 184ms.
  No counterexample found. Assertion may be valid. 83ms.

Executing "Run show for 15"
  Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
  162757 vars. 7980 primary vars. 431605 clauses. 490ms.
  Instance found. Predicate is consistent. 883ms.

Executing "Run shareable for 10"
  Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
  77135 vars. 4370 primary vars. 208331 clauses. 181ms.
  Instance found. Predicate is consistent. 923ms.
```

3.2 Generated Worlds

Here are presented two generated worlds, according to the model specified in Alloy. The first one shows a general case with some pending rides and some in use; the second one stresses the possibility to share a ride and to use alternative payment methods when possible.





4 Appendix

References

The following documents were used to produce this one:

- [1] Assignments 1 and 2 (RASD and DD).pdf
- [2] IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications.

Revision notes

- Some minor changes in layout
- **Goal 5** in Section 2 removed (as written later in the *DD*)
- Some minor corrections in the Use Case Diagram

Tools used

The following tools were used to produce this document:

- **Draw.io** to draw all the diagrams - <https://www.draw.io>
- **Photoshop** to make all mockups - <https://www.adobe.com>
- **Alloy Analyzer 4.0** to verify alloy models - <https://alloy.mit.edu>
- **LaTeX** as typesetting system to write this document
- **VIM** as editor - <https://www.vim.org>
- **TeXMaker** as editor - <http://www.xmlmath.net/texmaker>

Hours spent

The following hours were spent to produce this document:

Elenco ore SW 2					
Data	Alain	Mattia	Michele		
22/10/2015	5	2	2		
23/10/2015	0	1	1		
27/10/2015	3	3	3		
28/10/2015	3.5	4	5		
29/10/2015	4.5	5.5	4		
30/10/2015	0	2	0		
01/11/2015	6.5	4	6		
02/11/2015	6	5	6		
03/11/2015	3.5	2	3.5		
04/11/2015	1.5	3	3		
05/11/2015	3	6	8		
06/11/2015	9	7	12		
					Media:
	45.5	44.5	53.5		47.83333333