Politecnico di Milano
A.A. 2015-2016
Software Engineering 2: "TAXInseconds"
Requirements **A**nalysis
and
Specifications **D**ocument

Roberto Clapis (841859), Erica Stella (854443)

November 4, 2015

Contents

1 Introduction

1.1 Purpose

This document represent the Requirement Analysis and Specification Document (RASD). It aims at explaining the domain of the system to be developed and the system itself in terms of functional requirements, nonfunctional requirements and constraints. It also provides several models of the system and typical use cases. It is intended for all the developers who will have to implement the system, the testers who will have to determine if the requirements have been met and the system analysts who will have to write specifications for other system that will relate to this one. It is also intended as a contractual basis thus being legally binding.

1.2 Actual System

The government of the city wants to optimise its taxi service with a completely new application. Therefore, we assume there are no previous systems to take into account.

1.3 Scope

The aim of the project TAXInseconds is to provide a new application to optimise the taxi service of the city that will be accessible via browser, mobile or public APIs.
The city managed by TAXInseconds is divided in zones of 2 km^2 each and every zone has its own queue of taxis. The queues are automatically computed by the system with the information it receives from the GPS of the phones of the taxi drivers.
Taxi drivers can be available or not. Only available taxi drivers can be in a queue. When a taxi driver changes her state from not available to available the system automatically adds her to the queue of the zone she is currently in, based on the information of the GPS of her phone.
Users that are not registered can only see the estimated time of arrival of the nearest taxi with TAXInseconds.
Registered users can also request a taxi or make a reservation for a taxi. Reservations can only be made at least two hours before the ride and must be done specifying the starting location, the destination and the meeting time. Requests, instead, only need the starting location and the destination.
When a request is made, the first taxi driver of the queue of the starting location's zone is prompted to accept or reject it. If the taxi driver rejects it her state is automatically put on unavailable by the system. If a taxi driver doesn't accept or reject the request within 1 minute, it will be passed on to the next taxi driver in the queue and the first one will be moved to the end of the queue. If there are no available taxis in the zone of the request the system will propagate the request to the closest available taxi.

When a request is accepted, the user that has made the request receives a notification from the system informing her of the code of the incoming taxi and the estimated time of arrival.
When a reservation is made, the system confirms it to the user and allocates a taxi 10 minutes before the meeting time. If a taxi for that zone is not available the closest available taxi will be notified. When a taxi driver accepts the reservation, the user receives from the system the code of the incoming taxi. If a taxi driver doesn't accept or reject the reservation within 1 minute, it will be passed on to the next taxi driver in the queue and the first one will be moved to the end of the queue.
Requests can be cancelled before they have been accepted by a taxi driver while reservation can be cancelled until 10 minutes before the meeting time.

1.4 Goals

- Provide an easy way to request a taxi.
- Provide an easy way to reserve a taxi.
- Guarantee a fair management of the taxi queues.
- Create an extensible system that allows expansion and interactions with other services.

1.5 Definition and Acronyms

1.5.1 Definitions

- *Guest:* a person that has to sign up or log in the system.
- *Secure Channel:* a communication channel to ensure privacy and authenticity for both the server and the clients
- *Logged in user:* a person that has already signed up and logged in the system.
- *Administrator:* a person authorised to modify the list of taxi drivers stored by the system.
- *Request:* a call from a registered user who needs a taxi immediately.
- *Meeting time:* the date and time in which the registered user needs the taxi in case of reservation.
- *Reservation:* a booking of a taxi at a certain meeting time.
- *State of a taxi driver:* the state the taxi driver is currently in. It can be available or not available. Taxi drivers are in a queue if and only if they're available.

- *Closest available taxi:* if there are no taxis in the zone of the request or of the reservation, the system automatically finds the closest available taxi choosing the one with the smallest estimated time of arrival from the taxi queues of the other zones.

1.5.2 Acronyms

- ETA: estimated time of arrival: the time, estimated by the system, that the closest available taxi will take to get to the starting location of the ride.
- CAT: closest available taxi (see definition in the previous period).
- API: application programming interface; it is a set of routines, protocols, and tools for building software applications on top of this one.
- MAD: maximum allowed delay; the maximum time, calculated by the system according to its information about distance and traffic, that a taxi driver has to get to the starting location of a request.

1.6 Actors

- *Guest*: guests are able to sign up, login or ask the system for an ETA.
- *Logged in users*: after successfully logging in, registered users can request or reserve taxis or ask the system for an ETA.
- *Taxi drivers*: after successfully logging in, taxi drivers are able to set their current state as available or not and to accept or refuse requests.
- *Administrator*: after successfully logging in, the administrator will be the only user allowed to edit the taxi drivers list stored by the system.

1.7 References

- The document with the assignment for the project
- The IEEE Standard for SRS

1.8 Overview

This document is structured in three parts:

- Introduction: gives an high-level description of the software purposes and context.
- Overall Description: gives a general description of the application, focusing on the context of the system, going in details about domain assumptions and constraints. The aim of this section is to provide a context to the whole project and show its integration with the real world.

- Specific Requirements: this section contains all of the software requirements to a level of detail aimed to be enough to design a system to satisfy said requirements, and testers to test that the system actually satisfies them. It also contains the detailed description of the possible interactions between the system and the world with a simulation and preview of the expected response of the system with given stimulation. (Details are given with Alloy specifications and UML diagrams)

2 Overall description

2.1 Product perspective

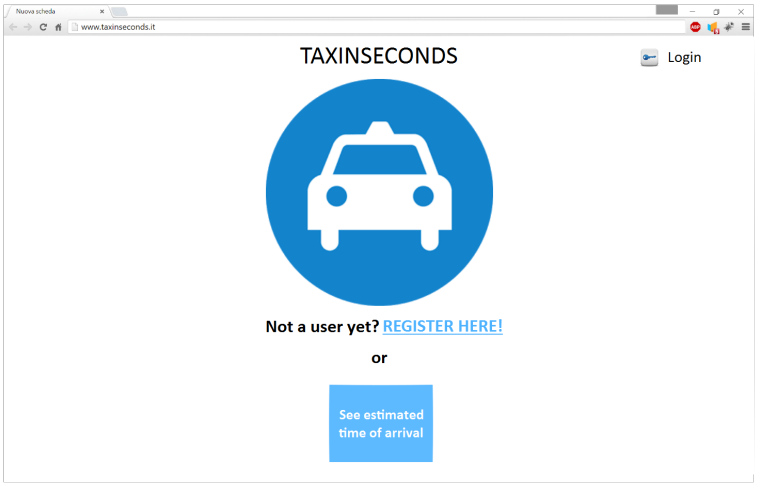
The TAXInseconds application will be released as a web application and as a mobile application. There are no existing systems to integrate it with.
It will provide a total of 4 main interfaces:

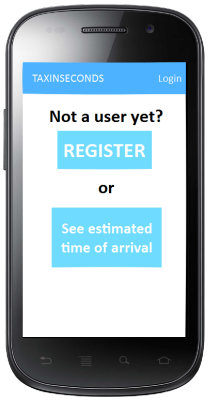
- For both type of users
  - Registered users
  - Guests
- For taxi drivers
- For administrators
- A non graphical interface for APIs

2.1.1 User Interfaces

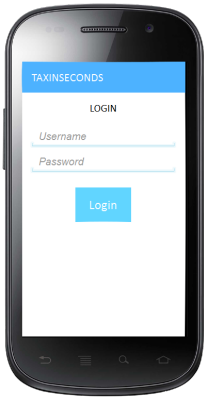
This section presents some mockups to provide an approximate idea of how the application's pages will be structured.

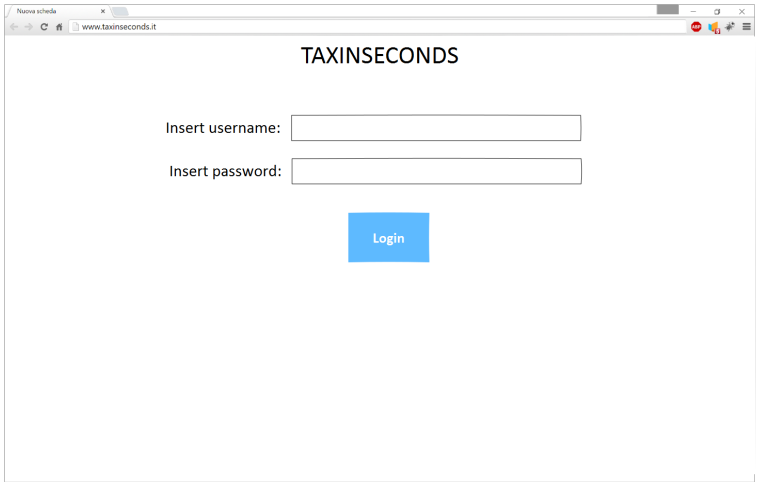
2.1.1.1 Guest interface This is the first page of the application where guests can choose to register, login or see the ETA.





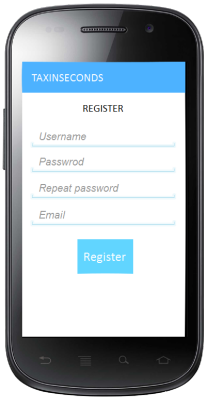
2.1.1.2 Login This is the login page where guests can log in the application and become either logged in users, administrators or taxi drivers.

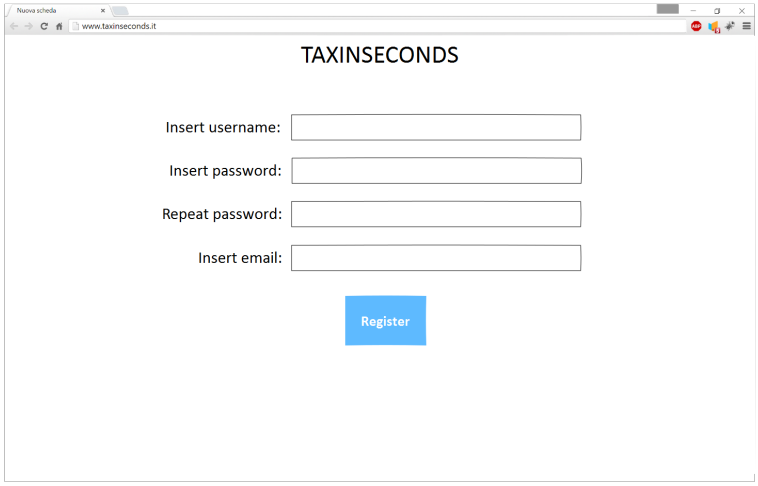




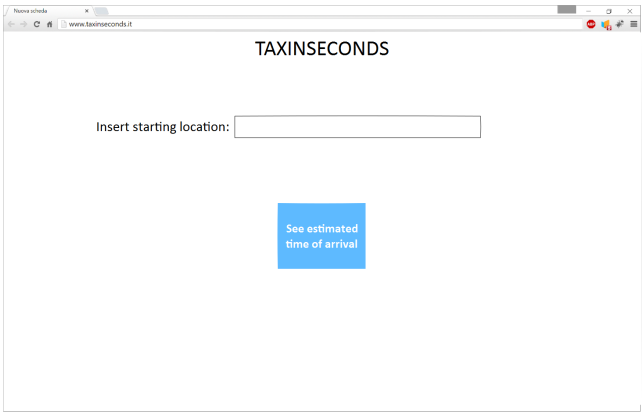
2.1.1.3 Register

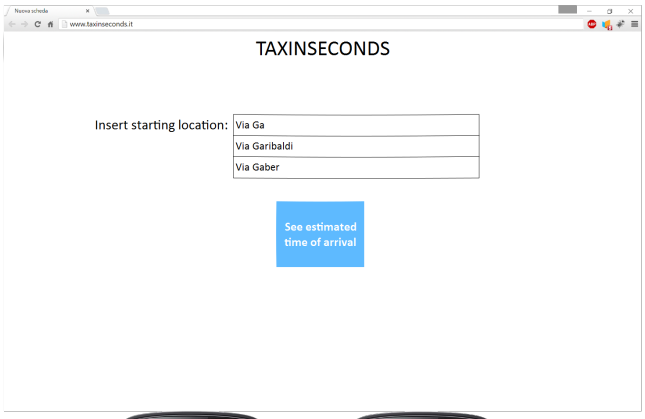
This is the mockup of registration form.

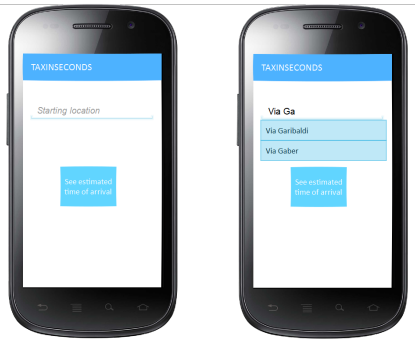




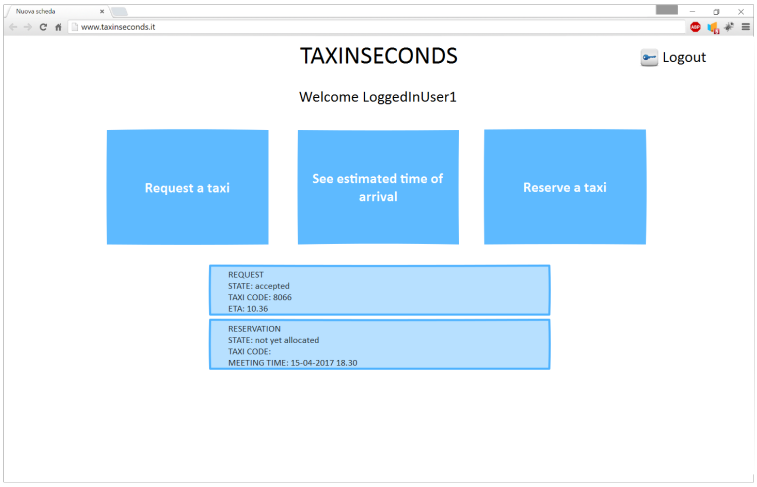
2.1.1.4 ETA This is the page where guests or logged in users can see the ETA to their starting location.

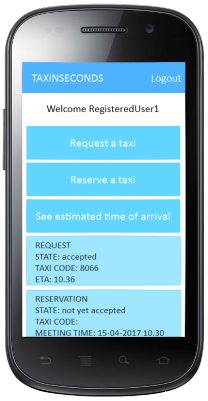




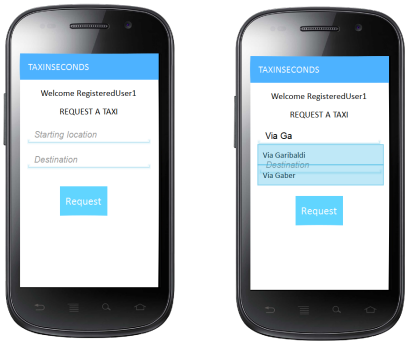


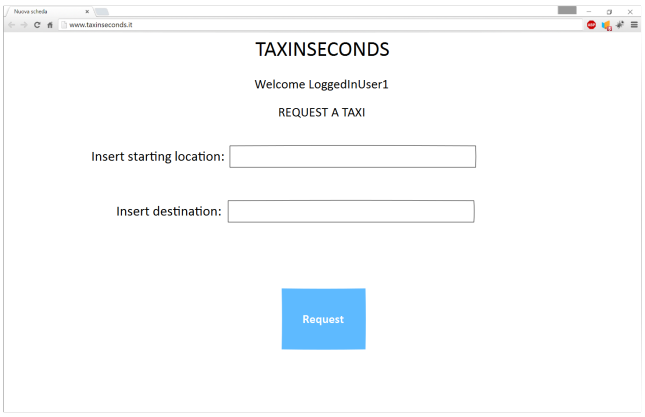
2.1.1.5 Logged in user interface This is the homepage of a logged in user where she can also see all the active requests and reservations.

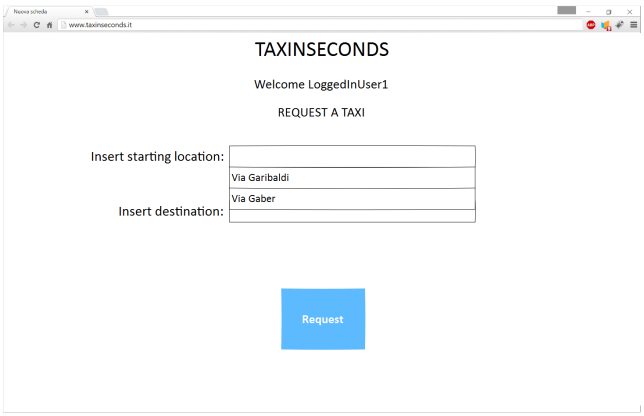




2.1.1.6 Request a taxi interface This is the mockup of the page where a logged in user can request a taxi

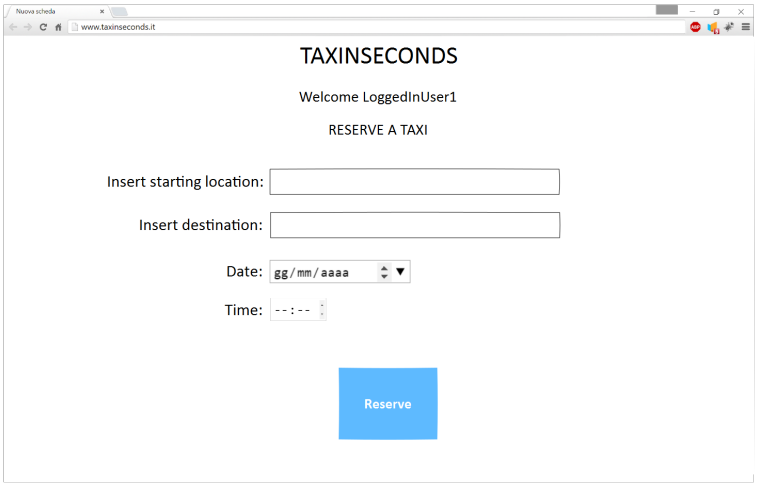


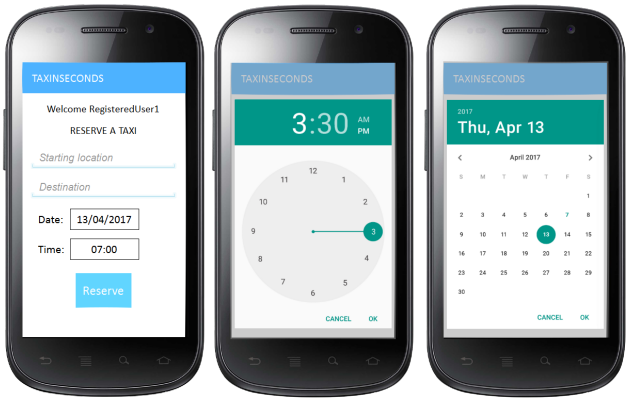




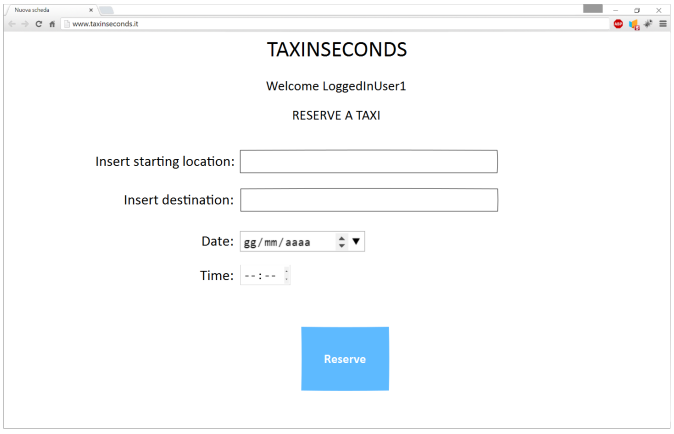
2.1.1.7 Reserve a taxi interface

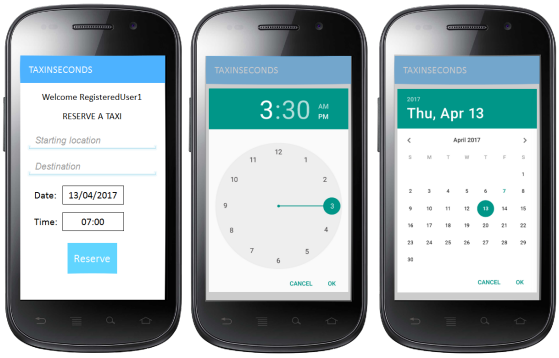
This is the mockup of the page where a logged in user can reserve a taxi





2.1.1.8 Administrator interface This is the page of an administrator for which there is no mobile





2.1.2 Hardware Interfaces

- Owned by the users and taxi drivers:
  - Any device running Android 4.0+ or iOS 6+ (GPS capability will make more functionalities available)
  - Any computer able to run an HTML5-compatible browser
- Owned by the company:
  - The server on which the core of the application will run, and to which the applications, the web UI and the API-related clients will connect to.
  - A machine with the DBMS

2.1.2.1 Software Interfaces

- Database Management System (DBMS):
  - Name: MySQL.
  - Version: 5.1.73
  - Source: https://www.mysql.com/
- HTTP/HTTPS server:
  - Name: Nginx
  - Version: 1.8.0
  - Source: http://nginx.org/
- PHP interpreter:
  - Name: PHP
  - Version: 5.3.3
  - Source: https://secure.php.net/
- Operating System:
  - Name: CentOS
  - Version: 7.1–1503
  - Source: https://www.centos.org/

2.1.2.2 Communication Interfaces

Protocol	Application layer Protocol	Port	Scope
TCP	HTTP	80	Upgrade to a secure connection over HTTPS
TCP	HTTPS	443	The web interface or the mobile apps
TCP	HTTPS/JSON	443	The APIs
TCP	HTTPS	443	The web interface or the mobile apps
TCP	DBMS over SSL	3306	Communication between the webserver and the DBMS

2.1.2.3 Memory constraints

- Primary memory:
  - for both taxi drivers' and clients' mobile devices at least 500MB
  - for the web application 1GB or more is suggested
  - for the server it is suggested to use a cloud service in order to resize memory according to traffic
- Secondary memory:
  - mobile devices will need to have 50MB of free space on the device
  - the web application requires no secondary memory
  - for the server it is suggested to use a cloud service in order to resize memory according to traffic

2.2 User characteristics

The TAXInseconds application is intended for all users who are at least 16 years old.

2.3 Assumptions and dependencies

2.3.1 Domain Assumptions

- All taxi drivers who intend to use the service will have a mobile phone with one of the supported mobile OSs
- All taxi drivers will have a phone with active GPS functionality
- The taxi drivers will grant the system the rights to handle their taxi codes
- Taxi drivers' phones will always have an internet connection while TAXInseconds is running
- Users will have access to the internet
- Users will enter a valid email address during registration
- Users who have requested or reserved a taxi will always be present when the taxi arrives.
- There is always at least an available taxi to fulfil a request or a reservation in the whole city

3 Specific requirements

3.1 Functional Requirements

On the user side:

- For Guests:
  - The system will be able to calculate and show the ETA
  - The system will be able to give suggestions to complete the partially inserted starting location.
  - The system will provide a registration functionality
  - The system will provide a login functionality that will also redirect to the right interface based on credentials.
- For logged in users:
  - The system will be able to calculate and show the ETA
  - The system will be able to give suggestions to complete the partially inserted starting location.
  - The system will store the username, password hash and email of every user
  - The system will provide a functionality to request a taxi at the given starting location
  - The system will provide a functionality to reserve a taxi at the given starting location and meeting time
  - The system will provide a functionality to modify the personal data of the user
  - The system will provide a functionality to see currently active reservations and requests
  - The system will provide a logout functionality
  - The system will provide a functionality to notify users of the code of the incoming taxi
  - The system will provide a functionality to notify users of the ETA of the incoming taxi
  - The system will provide a logout functionality
- Through the API:
  - The system will be able to calculate and show the ETA.
  - Using a Secure Channel and valid credentials:
    - Place a request for a given starting location
    - Place a reservation for given starting location and meeting time

    - Require the system to send a push message for updates about the status of a previous request

On the taxi driver side:

- The system will store the taxi code, username, password hash and email of every taxi driver
- The system will be able to calculate which taxi is the CAT
- The system will provide a functionality to switch the state of the taxi driver from available to not available or vice versa
- The system will notify the taxi driver of an incoming request showing the starting location
- The system will notify the taxi driver of a reservation showing the starting location and the meeting time
- The system will provide a functionality to allow taxi drivers to accept or reject reservations and requests
- The system will be able to always know the location of every available taxi driver
- The system will be able to compute the MAD and check whether the taxi driver gets to the starting location in time
- The system will provide a logout functionality

Through the API:

- Through a Secure Channel and with valid credentials:
    - Toggle the driver state
    - Send a notification when a call is made for the driver
    - Accept the answer to the call, whether the answer is Acceptance or Refusal
    - Compute the MAD and check whether the taxi driver gets to the starting location in time
-

On the admin side:

- The system will store the username, password hash and email of the admin.
- Add a new taxi driver into the system
- Remove a taxi driver from the system
- Modify registration data
- Log out

3.2 Non functional requirements

- *Availability — Responsiveness*
  - The server of the application must always be available
  - The app must never freeze
  - The system must store all of its data in an always-reachable database
  - Regular backups will be made in order to reduce or prevent data loss
- *Security*
  - In no situation sensible data will pass through an insecure channel

3.3 Scenarios

3.3.1 Scenario 1

Blair The Witch had to take her magical broom to the mechanic for the annual revision but she needs to go shopping to refill her stockpile of frog's tails. Her friend Mizune has told her about TAXInseconds, so she decides to give it a try. After downloading it on her smartphone, she signs up compiling the registration form with her username, password and email. Now she can complain about how slow car-based transports are!

3.3.2 Scenario 2

Suzuka is having a date tonight but, unfortunately, her car doesn't want to start. After several failures, she decides to use TAXInseconds. After logging in, she requests a taxi specifying her home as the starting location and the restaurant's address as the destination. The system notifies Takeshi, the first taxi driver of the queue in Suzuka's zone, of the request. Takeshi accepts the request and the system sends Suzuka the code of Takeshi's taxi and the ETA so she can finally get to her date.

3.3.3 Scenario 3

Ash Ketchum has to start his new adventure in Hoenn tomorrow but his mother is busy cleaning the house with Mr. Mime, so she can't take him to the airport to meet Prof. Oak. Ash decides to use once again TAXInseconds to reserve a taxi. After logging in, he makes a reservation for a taxi specifying the starting location as Pallet town, the destination and the meeting time. The morning after, 10 minutes before the meeting time, the system allocates Brock's taxi for the reservation and sends his code to Ash so that he knows who he'll meet to start his new adventure.

3.3.4 Scenario 4

Donald Duck is ready to start his first day as a taxi driver in Paperopoli. He jumps on his car, logs in TAXInseconds and changes his state to available. Unfortunately he doesn't know that, in his zone, there are 15 taxis in the queue before him, so he'll have to be patient for some time.

3.3.5 Scenario 5

After quite some time, Daisy requests a taxi and it's finally Donald's turn! The system notifies him but... it's breakfast time and Donald's having a cappuccino in a very crowded bar. He doesn't hear the notification popping on his phone so, after 1 minute, the system forwards Daisy's request to the next taxi driver in the queue and changes Donald's state to not available.

3.3.6 Scenario 6

Mickey Mouse is planning to go fishing with Pluto tomorrow. After logging in TAXINSECONDS, he makes two reservations: one for the morning and one for the evening. But, while he's enjoying his evening watching a movie, the phone rings. It's Minnie and she's reminding Mickey that they he had promised to go shopping with her the day after. Sadly, he has to say goodbye to his fishing trip and cancels the taxi reservations.

3.4 Use cases

3.4.1 Request a taxi

Actors: Logged in user, taxi driver *Preconditions:* The user is on the homepage of the application *Flow of Events:*

- The user clicks the "Request a taxi" button
- The system shows the user a page where she can enter the starting location and destination of the request
- The user inserts the starting location and the destination and then clicks the "Request" button
- The system notifies the first taxi driver of the starting location's zone's queue. If there are no taxis in that zone, the system notifies the CAT. If a taxi driver rejects the request the system passes the request on to the next taxi driver in the queue and changes the first one's state to not available. If a taxi driver doesn't accept or reject the request within 1 minute, the system passes the request on to the next taxi driver in the queue and moves the first one to the end of the queue
- The taxi driver accepts the request and goes to the starting location

- The system changes the state of the taxi driver to not available and notifies the user of the code of the incoming taxi and the ETA
- The taxi driver goes to the starting location to pick up the user

Postconditions: The taxi driver is not available anymore and has been removed from the queue.
Exceptions:

- If a taxi driver doesn't get to the starting location of the request within the MAD, the system will pass on the request to the next taxi driver in the queue and will notify the user of the change with the new ETA and taxi code. If the queue is empty, the system will notify the CAT.

3.4.2 Reserve a taxi

Actors: Registered user, taxi driver
Preconditions: The user is on the homepage of the application
Flow of Events:

- The user clicks the "Reserve a taxi" button
- The system shows the user a page where she can enter the starting location, destination and the meeting time of the reservation.
- The user inserts the starting location, destination and meeting time and then clicks the "Reserve" button
- Ten minutes before the meeting time, the system notifies the first taxi driver of the starting location's zone's queue. If there are no taxis in that zone, the system notifies the CAT. If a taxi driver rejects the reservation the system passes it on to the next taxi driver in the queue and changes the first one's state to not available. If a taxi driver doesn't accept or reject the reservation within 1 minute, the system passes it on to the next taxi driver in the queue and moves the first one to the end of the queue
- The taxi driver accepts the reservation and goes to the starting location
- The system changes the state of the taxi driver to not available and notifies the user of the code of the incoming taxi and the ETA
- The taxi driver goes to the starting location to pick up the user

Postconditions: The taxi driver is not available anymore and has been removed from the queue.
Exceptions:

- If a taxi driver doesn't get to the starting location of the request within the MAD after the meeting time, the system will pass on the reservation to the next taxi driver in the queue and will notify the user of the change with the new ETA and taxi code.

- If the meeting time is not at least two hours after the reservation an error message will be displayed and the reservation won't be made.

3.4.3 Cancel a reservation

Actors: Registered user
Preconditions: The user is on the homepage of the application and has at least one reservation for which a taxi has not yet been allocated
Flow of Events:

- The user clicks on the reservation she wants to cancel
- The system shows the user a page with the details of the reservation
- The user clicks the "Cancel reservation" button
- The system shows a "successful deletion" message and brings the user back to her homepage

Postconditions: The reservation has been cancelled.
Exceptions:

- If a taxi is allocated while the user is cancelling the reservation the system will show an error message and won't cancel the reservation

3.4.4 Cancel a request

Actors: Registered user
Preconditions: The user is on the homepage of the application and has at least one request that hasn't been accepted by a taxi driver yet
Flow of Events:

- The user clicks on the request she wants to cancel
- The system shows the user a page with the details of the request
- The user clicks the "Cancel request" button
- The system shows a "successful deletion" message and brings the user back to her homepage

Postconditions: The request has been cancelled.
Exceptions:

- If a taxi driver accepts the request while the user is cancelling it the system will show an error message and won't cancel the request

3.4.5 Registration

Actors: Guest
Preconditions: The guest is on the homepage of the application
Flow of Events:

- The guest clicks on the "Register here" button
- The system shows a page where the guest can enter her username, password and email
- The guest inserts the requested data and clicks the "Register" button
- The system shows a "successful registration" message

Postconditions: The guest can now log in the system whenever she wants with the inserted credentials
Exceptions:

3.4.6 Login

Actors: Guest
Preconditions: The guest is on the homepage of the application
Flow of Events:

- The guest clicks on the "Login" button
- The system shows a page where the guest can enter her username and password
- The guest clicks the "Login" button
- The system logs in the guest and shows her her page

Postconditions: The guest is now logged in the system, thus becoming a taxi driver or a logged in user or an administrator depending on the inserted credentials.
Exceptions: The inserted credentials are not valid so the guest is not logged in and an error message is shown

The following use case is described using logged in users as actors but it also valid for taxi drivers or administrators

3.4.7 Logout

Actors: Logged in user
Preconditions: The user is on the homepage of the application
Flow of Events:

- The user clicks on the "Logout" button
- The system logs the user out and brings her back to the homepage of the application for guests

Postconditions: The user is now logged out of the system thus becoming a guest
Exceptions:

The following use case is described using logged in users as actor but is also valid for guests

3.4.8 Search for ETA

Actors: Logged in user
Preconditions: The user is on the homepage of the application
Flow of Events:

- The user clicks on the "See estimated time of arrival" button
- The system shows a page where the user is prompted to insert the starting location from which the ETA must be calculated
- The user inserts the starting location and clicks the "See estimated time of arrival" button
- The system shows the ETA

Postconditions: The user has been shown the ETA
Exceptions:

3.4.9 Modify taxi driver

Actors: Administrator
Preconditions: The administrator is on the homepage of the application
Flow of Events:

- The administrator clicks on the "Modify taxi driver" button
- The system shows the administrator the list of taxi drivers retrieved from the database
- The administrator clicks on the taxi driver she wants to modify
- The system shows the administrator a page where she can change the values of the fields of the taxi driver
- The administrator changes the needed fields and then clicks the "Save changes" button
- The system update the taxi driver's information with the new ones that have been inserted

Postconditions: The taxi driver's information have been updated and she can no longer access the application with the old credentials but she must use the new ones
Exceptions:

3.4.10 Add taxi driver

Actors: Administrator
Preconditions: The administrator is on the homepage of the application
Flow of Events:

- The administrator clicks on the "Add taxi driver" button
- The system shows the administrator a form where she can enter all the information of the taxi driver
- The administrator inserts the information then clicks on the "Add taxi driver" button
- The system adds the taxi driver to the database

Postconditions: The taxi driver's information have been added to the database and she can now log in the application as a taxi driver
Exceptions:

3.4.11 Delete taxi driver

Actors: Administrator
Preconditions: The administrator is on the homepage of the application
Flow of Events:

- The administrator clicks on the "Delete taxi driver" button
- The system shows the administrator the list of taxi drivers retrieved from the database
- The administrator clicks on the taxi driver she wants to delete
- The system shows the administrator a page with the details of the taxi driver
- The administrator clicks the "Delete taxi driver" button
- The system deletes the taxi driver from the database

Postconditions: The taxi driver's information have been deleted from the database and she can no longer access the application with those credentials
Exceptions:

4 Alloy

4.1 Model

In order to adapt a pure logical model to a system that changes with time and, in the mean time, not lose meaningfulness the following main adaptations were made:

- The "serve" relation (AvailableTaxi →ActiveClient) is used to represent the association the system made before a taxi was removed from the queue in order to serve a client.
- The "refused" relation (InactiveTaxi →ActiveClient) is used to represent a refusal of a request by a taxi that was active.

```
1     module TAXInseconds
2     //TAXIES
3     abstract sig Taxi{}
4     sig AvailableTaxi extends Taxi{
5         //For the queues
6         nextTaxi:lone AvailableTaxi,
7         //For the service
8         serve:lone ActiveClient
9     }
10    sig InactiveTaxi extends Taxi{
11        refused:lone ActiveClient
12    }
13    sig TaxiQueue{root:AvailableTaxi}
14
15    //A Taxi can't be his next
16    fact nextTaxiNotReflexive {
17        no t:AvailableTaxi| t = t.nextTaxi
18    }
19    //A Taxi can't be one of his followers in the queue
20    fact nextTaxiNotCyclic {
21        no t:AvailableTaxi | t in t.^nextTaxi
22    }
23    //If a taxi is active he must be in exactly one queue
24    fact allAvailableTaxiesBelongToOneQueue {
25        all t:AvailableTaxi | one q:TaxiQueue | t in q.root.*nextTaxi
26    }
27
28    //Domain Assumption
29    fact thereIsAtLeastOneFreeTaxi{
30        some t:AvailableTaxi | no c:ActiveClient |
31        t.serve = c
32    }
```

```
33
34      //CLIENTS
35      abstract sig Client{}
36      sig ActiveClient extends Client{
37          //For the queue
38          nextClient:lone ActiveClient
39      }
40      sig nonActiveClient extends Client{
41          //For reservations
42          reserved:  lone Area
43      }
44      sig ClientQueue{root:ActiveClient}
45
46      //A Client can't be his next
47      fact nextClientNotReflexive {
48          no c:ActiveClient| c= c.nextClient
49      }
50      //A Client can't be one of his followers in the queue
51      fact nextClientNotCyclic {
52          no c:ActiveClient| c in c.^nextClient
53      }
54      //If a client is Waiting for a Taxi
55      //he must be in exactly one queue
56      fact allActiveClientsBelongToOneQueue {
57          all c:ActiveClient| one q:ClientQueue| c in q.root.*nextClient
58      }
59
60      //All clients must have a taxi serving them
61      fact allClientsAreServed{
62          all c:ActiveClient |
63          some t:AvailableTaxi |
64          c=t.serve
65      }
66
67      //AREAS
68      sig Area{
69          taxis:  one TaxiQueue,
70          clients:  one ClientQueue
71      }
72      fact oneQueueoneArea{
73          no c:ClientQueue | some disjoint a,a':Area |
74              c=a.clients and c=a'.clients
75          no t:TaxiQueue | some disjoint a,a':Area |
76              t=a.taxis and t=a'.taxis
77      }
78
```

```
79      //All queues must be connected to an Area
80      fact allQueuesInAreas{
81          all c:ClientQueue | some a:Area | c=a.clients
82          all t:TaxiQueue | some a:Area | t=a.taxis
83      }
84
85      //INTERACTIONS
86      //Clients are served only by one taxi
87      fact noClientsObiquity{
88          no disjoint t,t':AvailableTaxi | t'.serve=t.serve
89      }
90
91      //Clients are served in order
92      fact ClientsRespectQueues{
93          no c:ActiveClient | some t:AvailableTaxi |
94          c=t.serve and no t':AvailableTaxi | t'.serve=c. nextClient
95      }
96
97      //Taxies are serving in order
98      fact TaxisServeInOrder{
99          no t,t':AvailableTaxi | some c:ActiveClient |
100         t'=t. nextTaxi and c=t.serve and no c':ActiveClient| c'=t'.serve
101     }
102
103     //If an area has more clients than taxies
104     //all taxies must be serving
105     fact TaxiServeIfNeeded{
106         no t:AvailableTaxi | some a:Area |
107         t in a.taxis.root.*nextTaxi and
108         #a.taxis.root.*nextTaxi <= #a.clients.root.*nextClient
109         and    #t.serve=0
110     }
111
112     //Serving local clients is preferrable
113     fact TaxiStayIfNeeded{
114         no t:AvailableTaxi | some a:Area |
115         t in a.taxis.root.*nextTaxi and
116         #a.taxis.root.*nextTaxi <= #a.clients.root.*nextClient
117         and    t.serve not in a.clients.root.*nextClient
118     }
119
120     //FUNCTIONS
121     //Get who a taxi is serving
122     fun getTaxiClient[t:AvailableTaxi]:  lone ActiveClient{
123         t.serve
124     }
```

```
125
126     //Get who serves a client
127     fun getClientServer[c:ActiveClient]:  lone AvailableTaxi{
128         c. serve
129     }
130     //Get Queues for an area
131     fun getTaxisInArea[a:Area]:  set AvailableTaxi{
132         a.taxis.root.*nextTaxi
133     }
134     fun getActiveClientsInArea[a:Area]:  set ActiveClient{
135         a.clients.root.*nextClient
136     }
137
138     //ASSERTIONS
139     //Taxies cross areas only if the client they are serving
140     //is in an area without taxies
141     assert TaxisRespectAreas1     {
142         all t:AvailableTaxi | some ca,ta:Area |
143         some c:ActiveClient |
144         c in ca.clients.root.*nextClient and
145         t in ta.taxis.root.*nextTaxi and
146         c=t.serve and
147         ca!=ta implies
148         #ca.taxis.root.*nextTaxi = 0
149     }
150     check TaxisRespectAreas1     for 3
151
152     //Taxies cross areas only if the client they are serving
153     //is in an area with less taxies than clients
154     assert TaxisRespectAreas2     {
155         all t:AvailableTaxi | some ca,ta:Area |
156         some c:ActiveClient |
157         c in ca.clients.root.*nextClient and
158         t in ta.taxis.root.*nextTaxi and
159         c=t.serve and
160         ca!=ta implies
161         #ca.taxis.root.*nextTaxi < #ca.clients.root.*nextClient
162     }
163
164     check TaxisRespectAreas2     for 3
165
166     assert ActiveClientsMustBeInOneArea {
167         all c:ActiveClient | some t:AvailableTaxi |
168         some a:Area |
169         c=t.serve implies c in a.clients.root.*nextClient
170     }
```

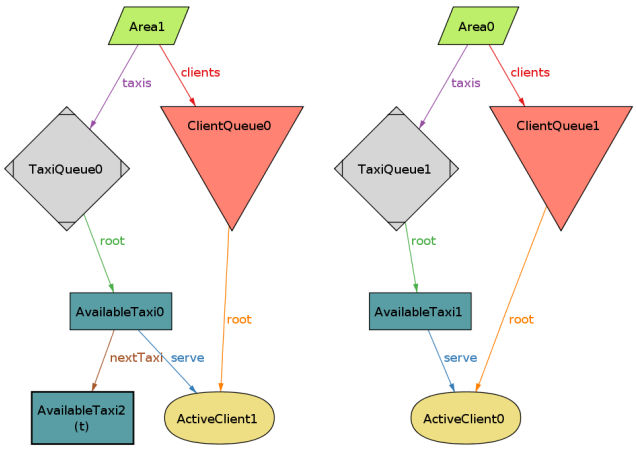
```
171     check ActiveClientsMustBeInOneArea for 3
172
173     assert TaxiQueuesAreRespected{
174         no t:AvailableTaxi | some t':AvailableTaxi |
175         t' in t.*nextTaxi and
176         #t'.serve = 1     and
177         #t.serve = 0
178     }
179
180     check TaxiQueuesAreRespected for 3
181
182     assert ClientQueuesAreRespected{
183         all c,c':ActiveClient | some t,t':AvailableTaxi |
184         c' in c.*nextClient and
185         c' in t.serve implies
186         c in t'.serve
187     }
188
189     check ClientQueuesAreRespected for 3
190
191     //PREDICATES
192
193     pred OneAreaFewAgents{
194         #AvailableTaxi = 2
195         #Area = 1
196         #ActiveClient = 1
197     }
198     run OneAreaFewAgents{} for 2
199
200
201     pred ALotOfTaxis{
202         #AvailableTaxi = 5
203         #Area = 1
204         #ActiveClient=1
205     }
206
207     run ALotOfTaxis{} for 2
208
209     pred TwoAreas{
210         #Area=2
211     }
212
213     run TwoAreas{} for 2
214
215     pred show{}
216     run show{} for 3
```

4.2 Result

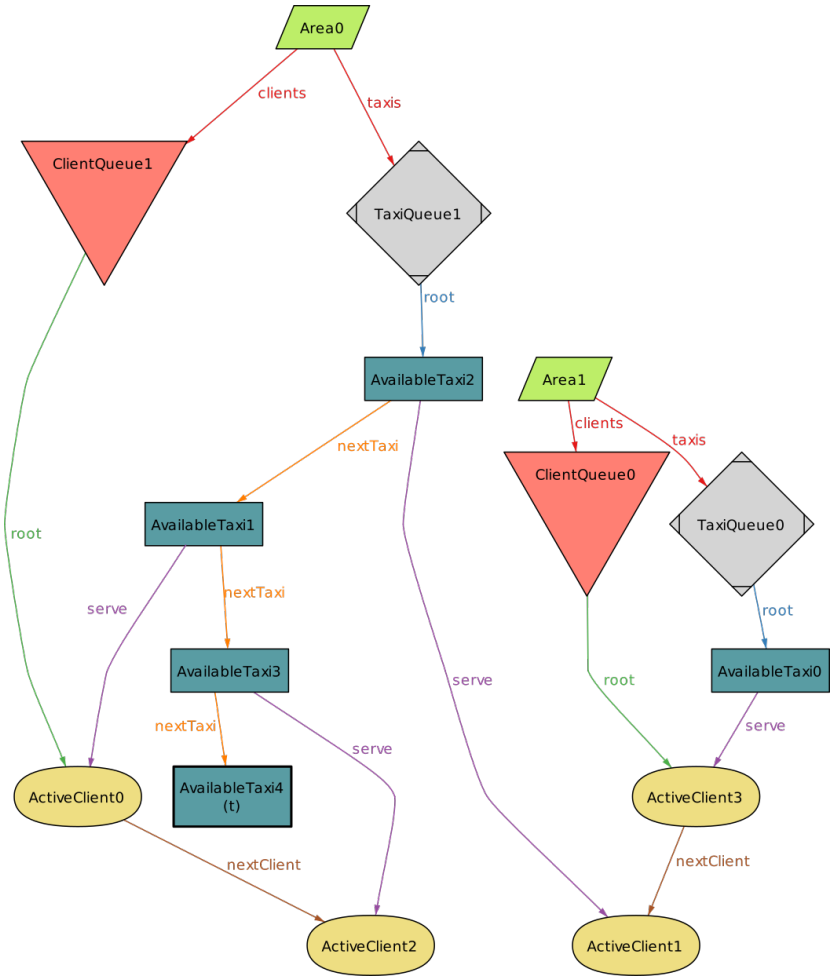
```
9 commands were executed. The results are:
   #1: No counterexample found. TaxisRespectAreas1 may be valid.
   #2: No counterexample found. TaxisRespectAreas2 may be valid.
   #3: No counterexample found. ActiveClientsMustBeInOneArea may be valid.
   #4: No counterexample found. TaxiQueuesAreRespected may be valid.
   #5: No counterexample found. ClientQueuesAreRespected may be valid.
   #6: Instance found. OneAreaFewAgents is consistent.
   #7: Instance found. ALotOfTaxis is consistent.
   #8: Instance found. TwoAreas is consistent.
   #9: Instance found. show is consistent.
```

4.3 Worlds Generated

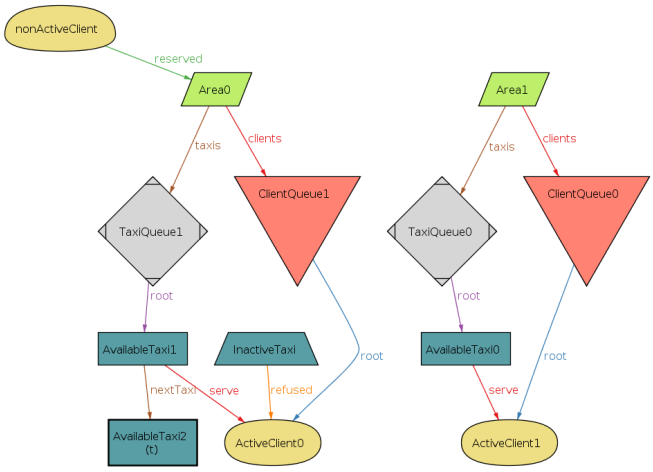
The following worlds were generated changing parameters about arity of present signatures in the show{} predicate.



This is a clean example of two areas, each one with a single client, no unusual interaction occour.



This is a more complex example, here we can see how the system behaves and the model adjusts if there aren't enough taxies in an area in order to serve all the clients.



In the following scenario we can see both a reservation of a client for a determined area and a refusal of a taxi for a client (now being served by an other taxi)