MY TAXI SERVICE

Requirements Analysis and Specifications Document

Software Engineering 2

Amos Paribocci (854818); Lorenzo Pinosa (852231)

1. Contents

1. Introduction

1.1 Purpose

Purpose of this document is to illustrate and detail MyTaxiService platform enough to be developed and put in action.

1.2 Scope

With myTaxiService we aim to provide a reliable and extensible platform to manage and organize the taxi service in any large city.

The platform, in conjunction with mobile apps and a web service, will allow any user to reserve a taxi with just a few taps. It will also enable taxi drivers to save time and money while providing a better service. All the processing will be location-aware thanks to localization features.

Our product will focus on immediate feedback in both directions: taxi drivers will be notified of users' requests and will be able to provide an immediate reply as well as the user will receive a notification as soon as his reservation is confirmed. ETA will also be dispatched.

We care about the fairness of our implementation while keeping efficiency in mind. User requests will only be served by taxis within a specified range and when multiple taxi are available requests will be forwarded using a FIFO policy. We will ensure that user request will be handled within a maximum timespan enforcing a timeout for the call to be accepted.

1.3 Definitions, acronyms and abbreviations

- User: any person who interacts with the system;
- Customer/passenger: any user that will exploit the service in order to request a taxi to pick him up;
- Taxi driver: user to which customer requests are forwarded selectively by the system, and is expected to fulfill them in practice;
- Taxi zone: area of the logical city subdivision, to be used for taxi selecting and queuing management while serving a customer request;
- Busy (w.r.t. taxi): taxi that has been assigned a task and is still working on it (picking the customer up or moving towards the ride destination);
- Available (w.r.t. taxi): taxi that is ready to accept a new task;
- App: abbreviation for "mobile application";
- System/backend: the part of the platform that will hold all the data and do all the processing on it, as well as handle instant communication and notification tasks. Mobile apps and the web service will rely upon this to work;
- Platform: what all MyTaxiService components build up, including the Website, the mobile apps and the backend.

1.4 References

- Specification Document: myTaxiService RASD (this document);
- IEEE STD 830-1998: IEEE Recommended Practice for Software Requirements Specifications.

1.5 Overview

This document is organized according to IEEE STD 830-1998 document. In section 2 we will offer a general overview of MyTaxiService, its perspective, main features and domain assumption. Section 3, Specific requirements, will focus on features (both functional and nonfunctional) details. Section 4, lastly, will complete this document with a full domain model and its validation in Alloy.

2. Overall description

2.1 Product perspective

MyTaxiService is a self-contained service and won't interact with any pre-existent one. It will expose a set of APIs to easily allow extensions and additions of functionalities.

2.1.1 System interfaces

As stated in 2.1 section, myTaxiService will provide a set of APIs that will allow to:

- Reserve a taxi given a specific place and customer;
- Hook the customer reservation request event, enabling to extend and/or to overwrite the default behavior of the system;
- Override the parameters used by taxi selection process while serving a customer request;
- Hook and override the taxi queue management policy.

2.1.2 User interfaces

MyTaxiService should offer a very simple and easy to use yet complete interface to the users. Customers should be able to use both the mobile app and the web service at first sight. Taxi drivers as well should take no more than a 15-minutes training to understand the system workflow and be able to correctly use the mobile application.

2.1.3 Operations

Passengers and taxi drivers will both have access to the service by the following means: the former may choose between a mobile app and a web application, while the latter may use the provided mobile app.

The end users should manually perform the following operations:

- Passengers may both request a taxi "on the fly" or ask for a taxi to come on appointment;
- Taxi drivers may choose to accept or decline any request which has been assigned to them.

Regarding the System backend, Backup tasks should be automatically performed.

2.1.4 Site adaptation requirements

No particular adaptation is needed, since end users are expected to already own a smartphone capable of running myTaxiService mobile apps.

2.2 Product goals

Our product will provide several functions, summarized by the following lists of goals.

2.2.1 Web application

2.2.1.1 Allow a visitor to register;

2.2.1.2 Allow a user to edit his account parameters and preferences;

2.2.1.3 Allow a user to display his taxi ride history;

2.2.1.4 Allow a user to ask for assistance and support;

2.2.1.5 Allow a user to request a taxi immediately, specifying origin and destination of the ride;

2.2.1.6 Allow a user to book a taxi ride, specifying both time, origin and destination of the ride;

2.2.1.7 Allow a user to edit the parameters related to a scheduled taxi ride;

2.2.1.8 Allow a user to cancel a scheduled taxi ride;

2.2.1.9 Allow a user who is already registered to log in;

2.2.1.10 Allow a user to log out and end the session.

2.2.2 Mobile application (for customers) goals

2.2.2.1 Allow a new user to register;

2.2.2.2 Allow a registered user to log in;

2.2.2.3 Allow a user to log out and end the session;

2.2.2.4 Allow a user to edit his account parameters and preferences;

2.2.2.5 Allow a user to ask for assistance and support;

2.2.2.6 Allow a user to request a taxi immediately, specifying the destination of the ride;

2.2.2.7 Allow a user to book a taxi ride, specifying both time and destination of the ride;

2.2.2.8 Allow a user to display his taxi ride history;

2.2.2.9 Allow a user to edit the parameters related to a scheduled taxi ride;

2.2.2.10 Allow a user to cancel a scheduled taxi ride;

2.2.2.11 Allow a user to receive notifications about his taxi reservations;

2.2.3 Mobile application (for taxi drivers) goals

2.2.3.1 Allow a user to log in;

2.2.3.2 Allow a user to log out and end the session;

2.2.3.3 Allow a user to ask for assistance and support;

2.2.3.4 Allow a user to receive notifications;

2.2.3.5 Allow a user to reply to a notification;

2.2.3.6 Allow a user to display the ride history.

2.2.4 System backend

2.2.4.1 The backend will store all users accounts, logins and data

2.2.4.2 The system should be able to identify a set of Taxi suitable to fulfill a customer request;

2.2.4.3 The system should forward the customer request to only one taxi driver at once picked from the previously built list and wait for the response;

2.2.4.4 Once a taxi has accepted the ride request, the backend should send a confirm notification to the customer;

2.2.4.5 The system backend should record taxi booking requests from customers and issue a taxi request automatically

2.3 User characteristics

Our ideal user (passenger) is a person who is looking for a simple and immediate way to ask for taxi service, via web application or mobile app. That is why we expect him to be able to use a web application and/or perform basic interactions in the context of a mobile app.

Our ideal user (taxi driver) is a taxi agency employee who is asked to use a mobile application in order to improve the quality of service. He is expected to be able to use a smartphone and to launch apps on it.

2.4 Constraints

2.4.1 Regulatory policies

No particular regulatory policy is needed.

2.4.2 Hardware limitations

We want to ensure that our products (both the web application and the mobile apps) will run smoothly on the average end-user devices.

2.4.3 Interfaces to other applications

MyTaxiDriver does not need to be interfaced with other applications.

2.4.4 Parallel operation

Multithreading will be used both on mobile apps and the system backend, in order to ensure the best response time while being able to accomplish many tasks.

2.5 Assumptions and dependencies

2.5.1 The registered users of myTaxiService products may only be customers or taxi drivers: it is assumed that a user cannot be both a customer and a taxi driver at the same time;

2.5.2 Customers cannot use our services that are intended to be used by taxi drivers, and vice versa;

2.5.3 Taxi drivers are already registered by the company and given the login credentials (they do not need to manually register);

2.5.4 The city contains at least one taxi zone;

2.5.5 Taxi zones are unique and disjoint;

2.5.6 A taxi ride may be accomplished only if the relative request exists;

2.5.7 Given a task assignment, a taxi rider must alternatively accept or decline (not replying in a certain amount of time will be treated as declining);

2.5.8 Given a ride request, a taxi driver (and only one) will accomplish the task;

2.5.9 In every moment, every taxi is located in an unique taxi zone;

2.5.10 A taxi can be either available, busy or unattended – i.e. no taxi driver is in it.

2.6 Apportioning of requirements

No specific apportioning of requirements is needed. We are confident that the flexibility and extensibility of the software will allow an easy writing of further requirements.

3. Specific requirements

3.1 External interface requirements

As stated in 2.1 section, myTaxiService will provide a set of APIs.
It's needed that they will allow to:

3.1.1 Reserve a taxi given a specific place and system user;

3.1.2 Hook the user reservation request event, enabling to extend and/or to overwrite the default behavior of the system;

3.1.3 Override the parameters used for taxi selection process while serving a customer request;

3.1.4 Hook and override the taxi queue management policy.

3.2 Functional requirements

To reflect the structure used listing goals in section 2, functional requirements will be organized by resulting product.

3.2.1 Web application

3.2.1.1 Allow a visitor to register

- 3.2.1.1.R1 Visitor must not be already registered to perform registration process.
- 3.2.1.1.R2 Visitor must choose a username not already used by another user.
- 3.2.1.1.R3 Visitor is allowed to see only the login page and not the user area.
- 3.2.1.1.R4 Visitor must specify a valid email to register and must prove to own it.

3.2.1.2 Allow a user to edit his account parameters and preferences

- 3.2.1.2.R1 User must already be registered and logged in;
- 3.2.1.2.R2 New values for parameters must be consistent w.r.t. the parameters' domain;

3.2.1.3 Allow a user to display his taxi ride history

- 3.2.1.3.R1 User must already be registered and logged in;
- 3.2.1.3.R2 If the taxi ride history is empty (the service has never been used), an appropriate message is displayed;

3.2.1.4 Allow a user to ask for assistance and support

- 3.2.1.4.R1 Users can ask for assistance anytime;
- 3.2.1.4.R2 When email support is requested, an appropriate form is provided;

3.2.1.5 Allow a user to request a taxi immediately, specifying origin and destination of the ride

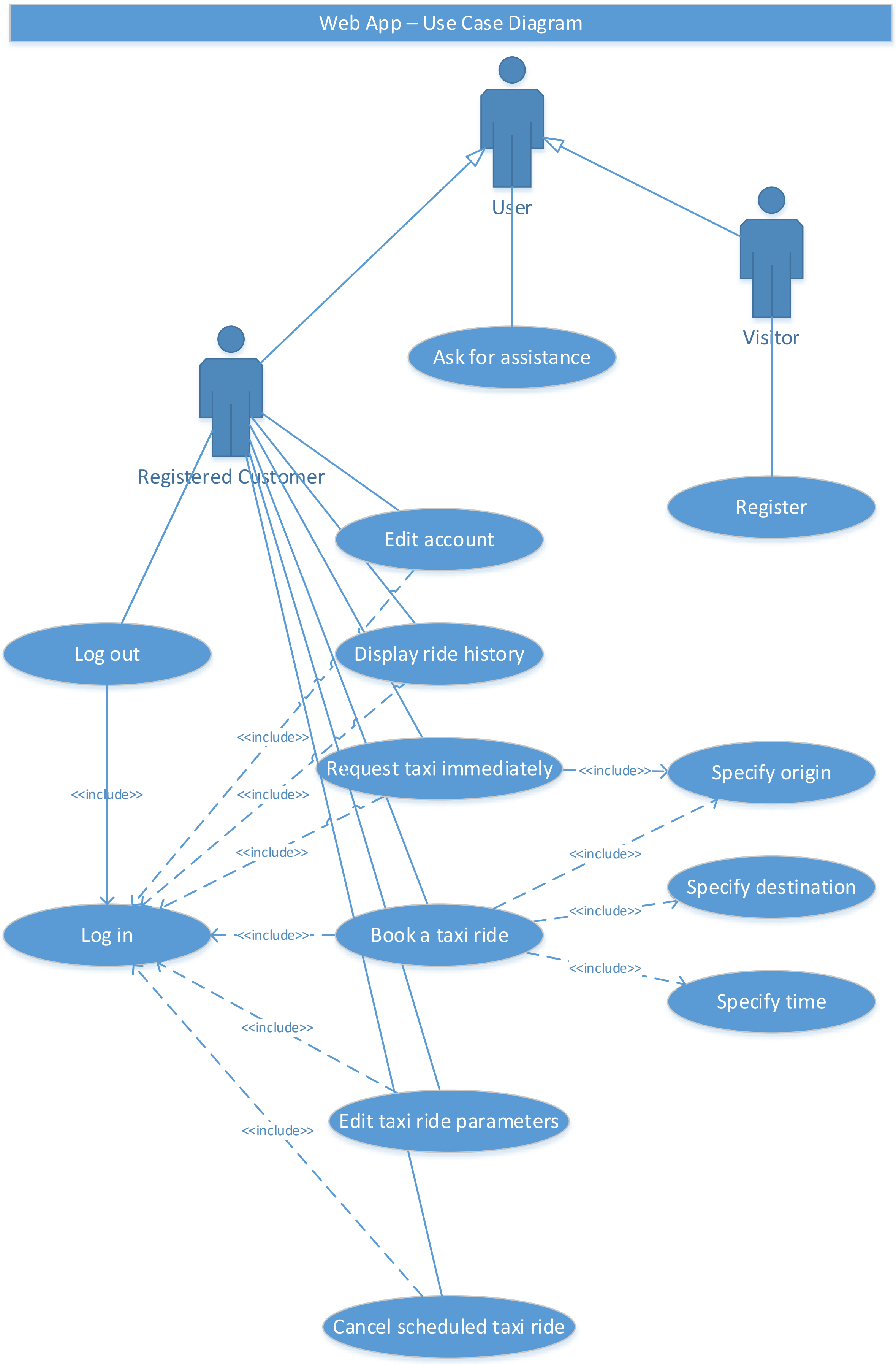
- 3.2.1.5.R1 User must already be registered and logged in;
- 3.2.1.5.R2 Locations should be validated against MyTaxiService working area;

3.2.1.6 Allow a user to book a taxi ride, specifying time, origin and destination of the ride

- 3.2.1.6.R1 User must already be registered and logged in;
- 3.2.1.6.R2 Locations should be validated against MyTaxiService working area;
- 3.2.1.6.R3 The request time/date may not be 1 week later than the current date;

3.2.1.7 Allow a user to edit the parameters related to a scheduled taxi ride

- 3.2.1.7.R1 User must already be registered and logged in;
- 3.2.1.7.R2 The taxi ride must have been requested by the user;


Web App – Use Case Diagram
User
Visitor
Registered Customer
Ask for assistance
Register
Edit account
Log out
Display ride history
Request taxi immediately
Specify origin
Specify destination
Specify time
Log in
Book a taxi ride
Edit taxi ride parameters
Cancel scheduled taxi ride
<<include>>

3.2.1.7.R3 The taxi ride must not have already been processed, i.e. no immediate taxi request has already been issued by the system to fulfill the booking;

3.2.1.7.R4 The new parameters must be consistent w.r.t. the parameters' domain;

3.2.1.8 Allow a user to cancel a scheduled taxi ride

3.2.1.8.R1 User must already be registered and logged in;

3.2.1.8.R2 The taxi ride must have been requested by the user;

3.2.1.8.R3 The taxi ride must not have already been processed;

3.2.1.9 Allow a user who is already registered to log in

3.2.1.9.R1 The user must be already registered;

3.2.1.9.R2 If the credentials are correct, a new session is created for the given user;

3.2.1.10 Allow a user to log out and end the session

3.2.1.10.R1 An user must be logged in to perform log out

3.2.1.10.R2 After the log out, the session ends and a new login or a new registration can be performed;

3.2.2 Mobile application (for customers)

3.2.2.1 Allow a new user to register;

3.2.2.1.R1 Visitor must not be already registered to perform registration process.

3.2.2.1.R2 Visitor must choose a username not already used by another user.

3.2.2.1.R3 Visitor is allowed to see only the login page and not the user area.

3.2.2.1.R4 Visitor must specify a valid email to register and must prove to own it.

3.2.2.2 Allow a registered user to log in;

3.2.2.2.R1 The user must be already registered;

3.2.2.2.R2 If the credentials are correct, a new session is created for the given user;

3.2.2.3 Allow a user to log out and end the session;

3.2.2.3.R1 An user must be logged in to perform log out

3.2.2.3.R2 After the log out, the session ends and a new login or a new registration can be performed;

3.2.2.4 Allow a user to edit his account parameters and preferences;

3.2.2.4.R1 User must already be registered and logged in;

3.2.2.4.R2 New values for parameters must be consistent w.r.t. the parameters' domain;

3.2.2.5 Allow a user to ask for assistance and support;

3.2.2.5.R1 Users can ask for assistance anytime;

3.2.2.6 Allow a user to request a taxi immediately, specifying the destination of the ride;

3.2.2.6.R1 User must already be registered and logged in;

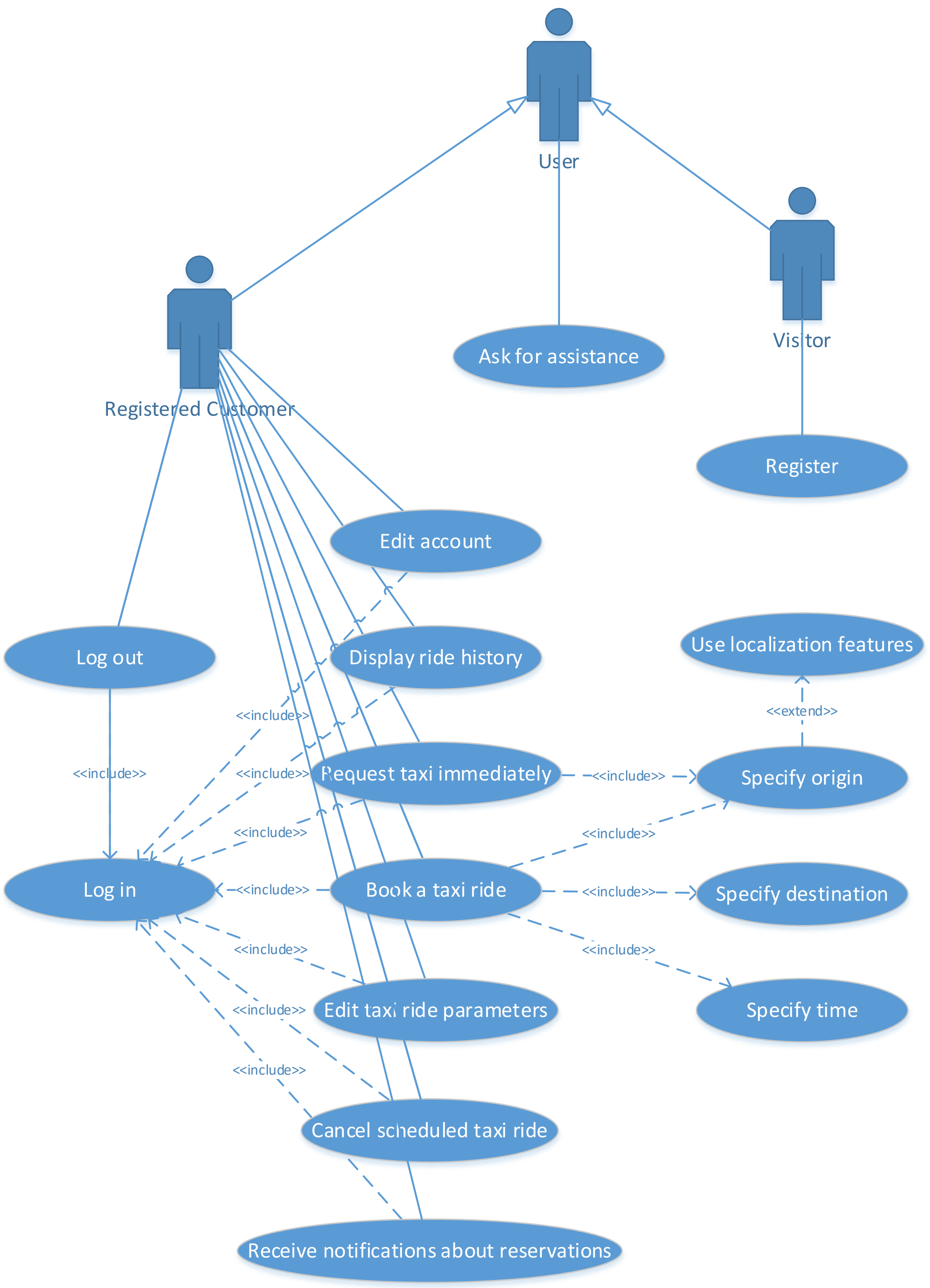
3.2.2.6.R2 If the device supports localization features, the user may choose whether to manually insert the ride origin or to retrieve it automatically;

3.2.2.6.R3 The locations must be compatible with the taxi service capabilities;

3.2.2.7 Allow a user to book a taxi ride, specifying both time and destination of the ride;

3.2.2.7.R1 User must already be registered and logged in;

Mobile Customer App – Use Case Diagram



3.2.2.7.R2 The locations must be compatible with the taxi service capabilities;

3.2.2.7.R3 The request time/date may not be 1 week later than the current date;

3.2.2.8 Allow a user to display his taxi ride history;

3.2.2.8.R1 User must already be registered and logged in;

3.2.2.8.R2 If the taxi ride history is empty (the service has never been used), an appropriate message is displayed;

3.2.2.9 Allow a user to edit the parameters related to a scheduled taxi ride;

3.2.2.9.R1 User must already be registered and logged in;

3.2.2.9.R2 The taxi ride must have been requested by the user;

3.2.2.9.R3 The taxi ride must not have already been processed;

3.2.2.9.R4 New values for parameters must be consistent w.r.t. the parameters' domain;

3.2.2.10 Allow a user to cancel a scheduled taxi ride;

3.2.2.10.R1 User must already be registered and logged in;

3.2.2.10.R2 The taxi ride must have been requested by the user;

3.2.2.10.R3 The taxi ride must not have already been processed;

3.2.2.11 Allow a user to receive notifications about his taxi reservations;

3.2.2.11.R1 User must already be registered and logged in;

3.2.3 Mobile application (for taxi drivers)

3.2.3.1 Allow a user to log in;

3.2.3.1.R1 The user must be already registered;

3.2.3.1.R2 If the credentials are correct, a new session is created for the given user;

3.2.3.2 Allow a user to log out and end the session;

3.2.3.2.R1 An user must be logged in to perform log out;

3.2.3.2.R2 After the log out, the session ends and a new login or a new registration can be performed;

3.2.3.3 Allow a user to ask for assistance and support;

3.2.3.3.R1 Users can ask for assistance anytime;

3.2.3.4 Allow a user to receive notifications;

3.2.3.4.R1 User must already be registered and logged in;

3.2.3.5 Allow a user to reply to a notification;

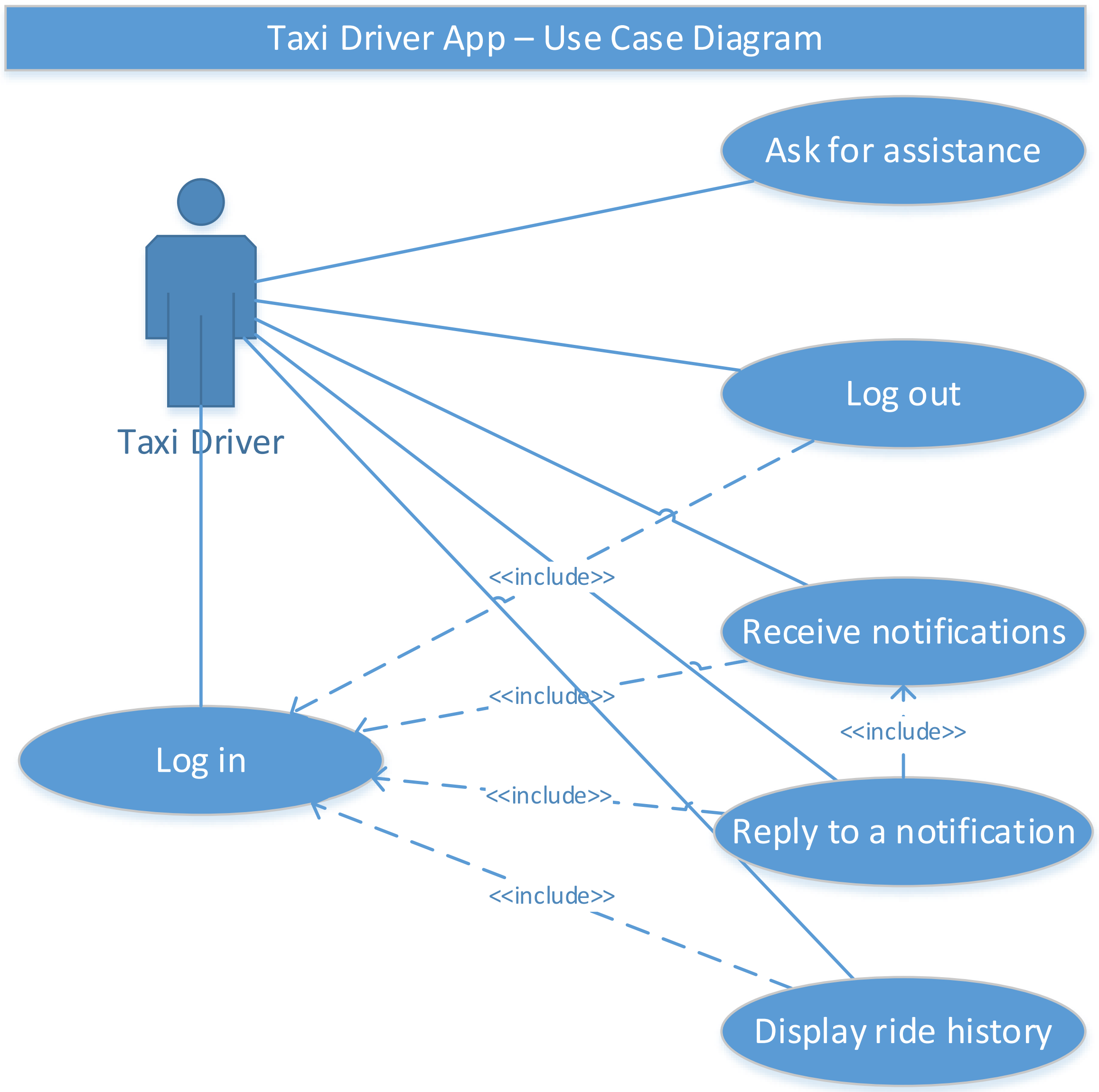
3.2.3.5.R1 User must already be registered and logged in;

3.2.3.5.R2 User must have already received a notification;

3.2.3.6 Allow a user to display the ride history.

3.2.3.6.R1 User must already be registered and logged in;

3.2.3.6.R2 If the ride history is empty, an appropriate message is displayed;



3.2.4 System Backend

3.2.4.1 The backend will store all users accounts, logins and data

3.2.4.1.R1 Password should be stored as hashes with a minimum length of 512bits.

The system should be able to identify a set of Taxi suitable to fulfill a customer request.

3.2.4.1.R2 The search criteria should be easily interchangeable by the administrator by CLI or a GUI;

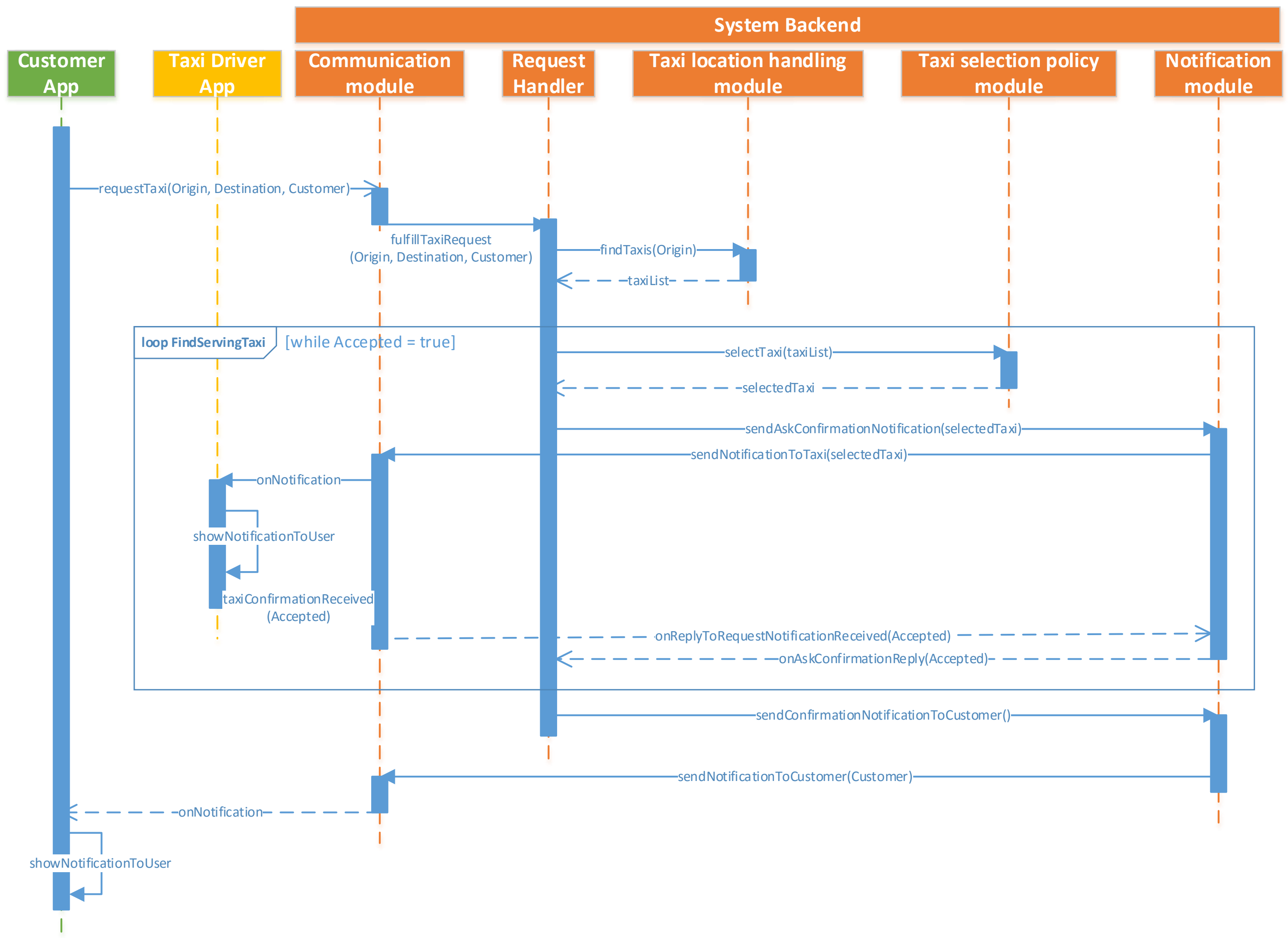
3.2.4.1.R3 Busy or unattended taxis are not suitable to serve a customer request

3.2.4.1.R4 The default criteria will be: all taxis present in the taxi zone where the user request comes from;

3.2.4.2 The system should forward the customer request to only one taxi driver at once picked from the previously built list and wait for the response;

- 3.2.4.2.R1 The timeout for a taxi driver to answer is 15 sec;
- 3.2.4.2.R2 In case of negative or no answer, the system will fall back to another taxi in the list;
- 3.2.4.2.R3 The taxi selection policy from the list must be easily interchangeable;
- 3.2.4.2.R4 The default policy will order the list by distance with the user. In case of negative answer, the taxi will be moved at the end of the list;
- 3.2.4.2.R5 To ensure responsiveness on customer end, the taxi searching process may last at maximum 2 minutes and a taxi may be asked for a job at maximum 2 times;
- 3.2.4.2.R6 All details about this process should be persisted in order to be able to create statistics regarding punctuality and accuracy of Taxi Driver. This data might be use difficult in future for more complex taxi selection policies;

Requirements 3.2.4.2 and 3.2.4.3 are exemplified in the following sequence diagram:

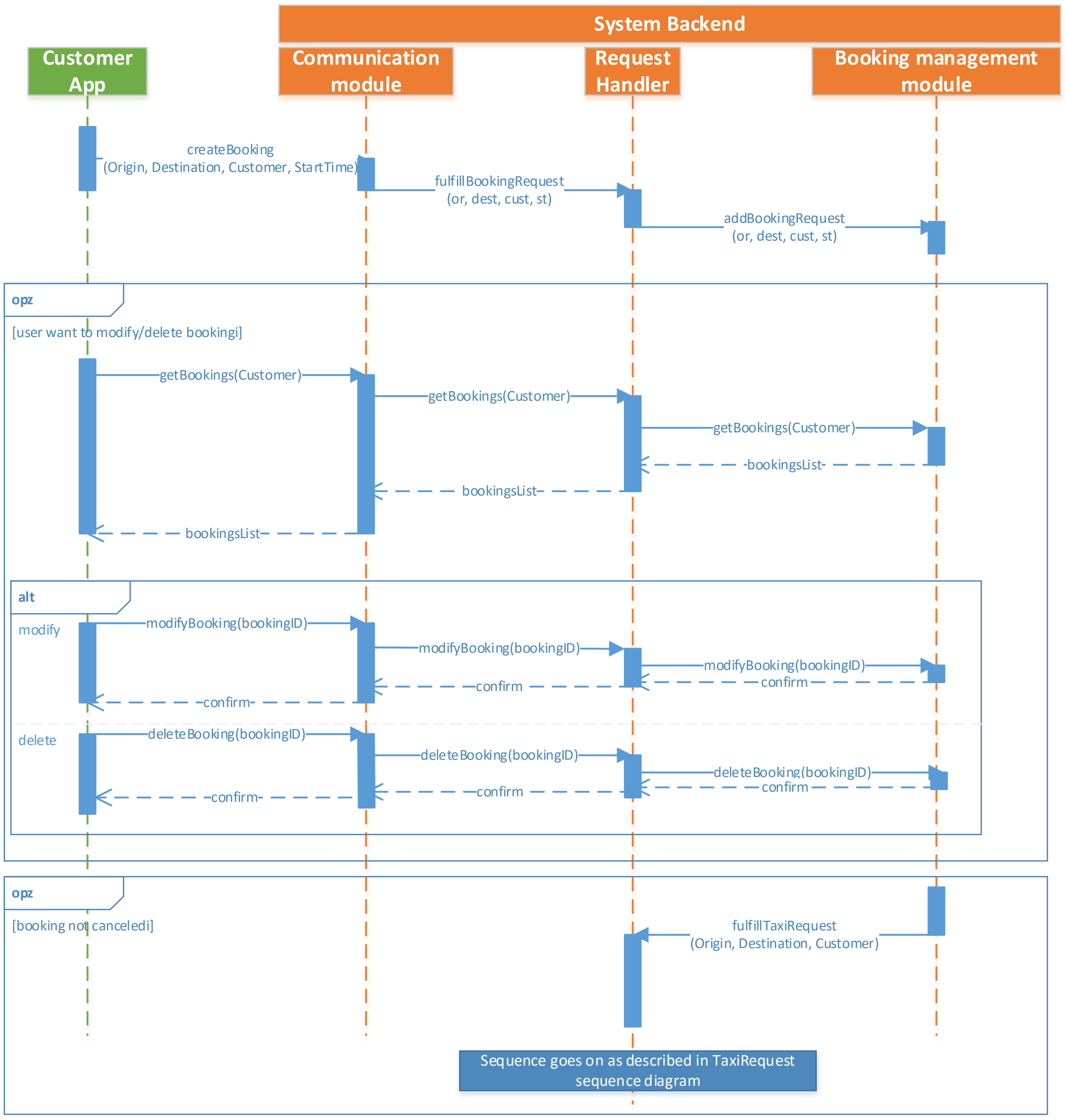

System Backend
Customer App
Taxi Driver App
Communication module
Request Handler
Taxi location handling module
Taxi selection policy module
Notification module
requestTaxi(Origin, Destination, Customer)
fulfillTaxiRequest (Origin, Destination, Customer)
findTaxis(Origin)
taxiList
loop FindServingTaxi
[while Accepted = true]
selectTaxi(taxiList)
selectedTaxi
sendAskConfirmationNotification(selectedTaxi)
sendNotificationToTaxi(selectedTaxi)
onNotification
showNotificationToUser
taxiConfirmationReceived (Accepted)
onReplyToRequestNotificationReceived(Accepted)
onAskConfirmationReply(Accepted)
sendConfirmationNotificationToCustomer()
sendNotificationToCustomer(Customer)
onNotification
showNotificationToUser

3.2.4.3 Once a taxi driver has accepted the ride request, the backend should send a confirm notification to the customer;

3.2.4.3.R1 The confirm notification should include details about the incoming taxi and estimated waiting time;

The system backend should record taxi booking requests from customers and issue a taxi request automatically

3.2.4.3.R2 The booking request can be modified or deleted by the user after being recorded, unless a taxi request has already been issued



3.2.4.3.R3 The taxi request will be issued 10 minutes before the booking goes off

Requirement 3.2.4.5 is detailed in the sequence diagram in the previous page.

3.3 Performance requirements

Immediate feedback to all users of myTaxiService is essential. The system should serve the 99% of the requests within 5sec, considering even subsequent message dispatching to other users.

For example, if a customer requests a taxi, once the request is received by the system it should be processed and dispatched to the selected taxi within 5sec.

3.4 Logical database requirements

The DBMS on which the system relays should perform enough well to allow the section 3.3 Performance Requirements to be fulfilled.

The main purpose of the DBMS will be to store the information about real time taxi positions, so it must be able to deal with geographic data and to issue geographical queries – for example but not limited to retrieve taxis located in a specified area that may not be rectangular-shaped.

Referring to section 4.1 UML Classes Diagram, the following data entities and their relationships should be persisted to the database:

Users, Taxis, TaxiZone, TaxiRequest, TaxiRide, TaxiBooking, TaxiRequestNotification, Reply.

It's not required to maintain a Taxi Location history. Customer request history instead should be retained for at least 2 years.

3.5 Design constraints

3.5.1 Customer mobile app and web service

Ease of use and interface responsiveness is fundamental. 98% of the customers should be able to place a taxi request after at maximum 60 sec after they opened the mobile app or the web service for the first time.

3.5.2 Taxi driver mobile app

Since taxi drivers will be familiar to the app after a short time using it every day, the same ease of use enforced for customer interfaces is not needed. It's still mandatory that a taxi driver should be able to use the basic features of the app, i.e. 3.2.3, as well as have understood system assumptions and rules at the utmost after attending a 15 minutes long training.

3.6 Software system attributes

3.6.1 Reliability and availability

3.6.1.1 Mobile apps

Mobile Apps should encounter unhandled runtime exceptions with a MTTF (Mean Time to Failure) of at least 5 days of use (assuming 15 mins of use a day). All exceptions should be reported to the system and stored for further analysis and statistics.

The MTTR (Mean Time to Repair) is considered negligible as it's the time spent to reopen the app on the mobile device.

Runtime errors that cause permanent damage to an app installation (and thus compelling the user to reinstall it) are unacceptable; if ever discovered they must be addressed with the maximum priority (i.e. a fixed app release should be issued in no more than 7 working days).

3.6.1.2 Customer Web service

The Web Service will have a required uptime of 99.5%. Required MTTF is the same of mobile apps (with the same assumptions of mean use). MTTR is considered even here negligible as it's the taken to reload the web page.

As for mobile apps, runtime errors that bring about Web server failure with consequent restart needed are unacceptable and should be reported and fixed within 7 working days.

3.6.2 Security

MyTaxiService platform security will be guaranteed both at user interface and backend level.

3.6.2.1 Mobile Apps and Web Service

User interfaces will enforce a Username/Password login. The password will be stored in the database as a hash (minimum 512 bits of length) to prevent unauthorized accesses even to MyTaxiService technicians and system administrators who can read the database. It will be available to the user a procedure to set up a two-level authentication login, involving for example entering a code received on the mobile phone or via email.

3.6.2.2 System Backend

The machine (virtual or physical) running MyTaxiService backend will not be exposed directly to Internet. There will be a firewall appliance which will check incoming traffic at application level (thus being able to prevent even light DDoS attacks) and then forward them to the backend.

The backend-running machine will be accessible only to authorized administrators, with strict password complexity requirements. Passwords should be at least of 10 characters, of which at least one from the following types: Capital letters, special characters and numbers. All logins (success/failed) will be logged.

Periodic automatic backups will be scheduled. Backups will be stored in another physical machine and on removable media such as backup tapes or removable HDDs. The minimum backup history to keep is 4 backups.

3.6.3 Maintainability

MyTaxiService has to be an extendible platform and in, addition to expose a set of API for external applications, needs to me easily maintainable and modifiable itself.

The System Backend code should be as most as possible modular and keep in different modules at least the following features: Communication with user interfaces, notifications management, Taxi location handling and search, taxi selection policy, booking management.

3.6.4 Portability

3.6.4.1 Mobile Apps

The mobile apps should be available and compatible with the following mobile platforms: Android, iOS, Windows Phone.

3.6.4.2 Web Service

MyTaxiService website should be designed responsive. It should behave and look good both on PCs and Mobile devices (Tablets and smartphones).

3.6.4.3 System Backend

No portability is needed for the backend. It will be developed using Microsoft ASP and .Net technologies and use only them.

3.7 Scenarios

3.7.1 Scenario 1 (Customer) – Registration and taxi ride booking

Giulio is a night owl: he loves to spend the night out and partying with friends. The problem is, he finds himself finishing late all the time, always looking for someone who may take him home.

His friend Alessia tells him about MyTaxiService: it seems pretty simple and easy to use.

Giulio downloads the MyTaxiService app on his brand new smartphone and register to the service by inserting his data. After finishing the registration process, he logs in and try to book a taxi ride for the coming Saturday night's party: he just needs to specify the location and the pick-up time.

Luckily, the club hosting the party is reached by the taxi service: with a tap, the booking process ends.

Now, Giulio can go to the club without having to worry about transport issues: a few minutes before the end of the party, the system dispatches a request for the pick up to occur. A taxi accepts the task and reaches the location, allowing Giulio to go home.

3.7.2 Scenario 2 (Customer) – Log in and "on the fly" taxi request

Rosy is an investment and risk analyst working for a big financial company. Being a consultant requires her to move in the city quite often and keeps her always busy: since meetings may be scheduled anytime, Rosy finds herself using the city's taxi service very often.

Luckily, she has learned about MyTaxiService from her colleagues: she has already registered and is ready to test the product capabilities.

Rosy needs to go to the other side of the city in order to discuss about an important investment: that sure seems a perfect occasion to use MyTaxiService.

After logging in with her smartphone, Rosy is able to ask for a taxi with a few taps; since her device's localization features are enabled, she doesn't even need to insert her current location!

A few instants after completing the request, Rosy is notified that a taxi will come to pick her up in 4 minutes. After accepting the request, a taxi comes picking Rosy up at the scheduled time.

3.7.3 Scenario 3 (Customer) – Scheduled taxi ride cancellation

It is a wonderful day for Luca: his dear friend Louis, who lives in another country, is coming visiting him.

Luca is a customer of MyTaxiService and, since he needs to take care of his little sister and cannot go to the airport himself, he has already booked a taxi that would pick Louis up.

Luca receives a call: it is Louis, still waiting at his country's airport, telling him that the flight has been delayed by one hour due to luggage loading issues. That's not too much of a problem: Luca connects to the MyTaxiService website, logs in by inserting his credentials and displays the status of the scheduled ride. By clicking the appropriate button, he can now edit the information about the taxi's arrival time.

Since the old request has not been processed yet, Luca can confirm the change and tell his friend not to worry anymore: the system asks for a taxi to reach the airport at the new specified time. The first taxi driver in the airport taxi zone queue refuse the task, which is forwarded to the next one: the second taxi driver accepts and goes to the airport, allowing Louis to reach his friend.

3.7.4 Scenario 4 (Taxi driver) – Log in and request acceptance

Steve is a new employee at the most popular taxi agency in the city. After being hired, he has been told about the MyTaxiService app: he needs to download the app on his mobile phone in order to be consistent with the adoption of the new service.

Luckily, he doesn't need to register his own account: the agency provides him the login credentials.

After opening the app, Steve logs in and navigates through the main app's features: everything seems simple and pretty straightforward.

After setting up the taxi which has been assigned to him, Steve receives a new ride request through the MyTaxiService app: the device's screen shows the customer's location and asks him to accept or refuse it.

Steve decides to accept the task and starts his first experience with MyTaxiService: he reaches the target location at the scheduled time, taking the passenger to the destination.

After the task has been accomplished, the ride details are recorded into the ride history and Steve is ready to receive new assignment.

3.7.5 Scenario 5 (Customer) – Request not fulfilled (worst-case scenario)

Paul is going back to his hometown today: he needs to go to the city station and take a train.

Right after leaving his place, Paul starts considering taking a taxi in order to avoid carrying his heavy luggage all the way to the station: he takes his smartphone and, after logging in the MyTaxiService app, asks for a taxi to come picking him up.

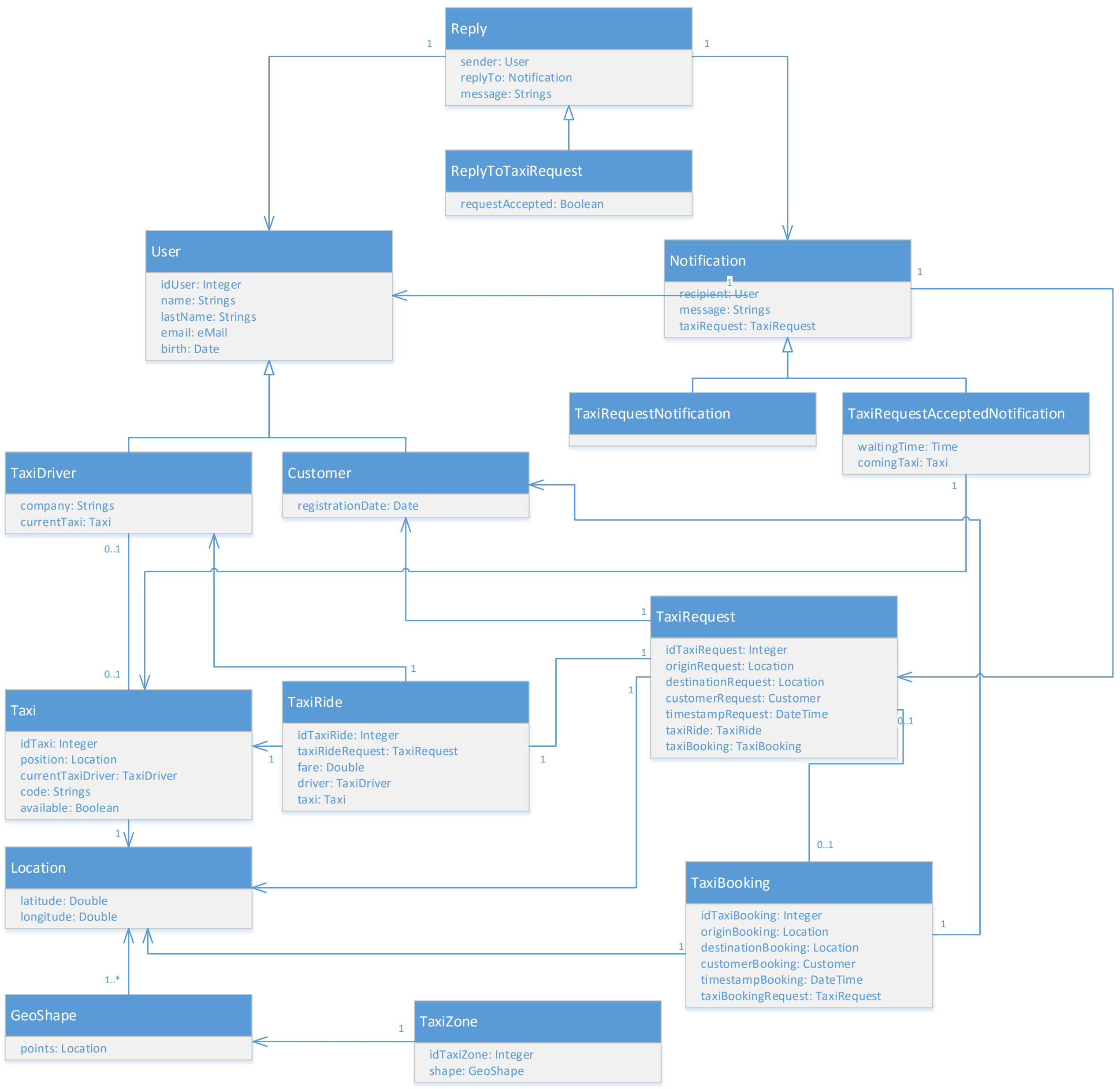
The request is processed by the system and forwarded to the first taxi driver in the taxi zone queue: since the driver is about to have his lunch break, the task is refused and moves through the queue.

Unluckily for Paul, the second taxi driver is about to end his work day and refuses the assignment, too.

Since there are just two taxi in the taxi area, the request goes through the queue once more, being denied again: after running through the queue twice, the system notifies Paul that the service will not be delivered. Paul needs to manually call a taxi or to use the public transport.

4. Supporting Information

4.1 UML Class Diagram



4.2 Alloy

4.2.1 Model

```
sig Integer {}

sig Double {}

sig Strings {}

abstract sig Boolean {}

sig True extends Boolean {}

sig False extends Boolean {}

sig eMail {
      account: one Strings,
      domain: one Strings,
}

sig Date {
      day: one Integer,
      month: one Integer,
      year: one Integer
}

sig Time {
      hour: one Integer,
      minutes: one Integer
}

sig DateTime {
      date: one Date,
      time: one Time
}

sig Location {
      latitude: one Double,
      longitude: one Double
}
```

```
sig GeoShape {
      points: some Location
}

abstract sig User {
      idUser: one Integer,
      name: one Strings,
      lastName: one Strings,
      email: one eMail,
      birth: one Date
}

sig TaxiDriver extends User {
      company: one Strings,
      currentTaxi: lone Taxi
}

sig Customer extends User {
      registrationDate: one Date
}

sig Taxi {
      idTaxi: one Integer,
      position: one Location,
      currentTaxiDriver: lone TaxiDriver,
      available: one Boolean,
      code: one Strings
}

sig TaxiZone {
      idTaxiZone: one Integer,
      shape: one GeoShape
}

sig TaxiRequest {
      idTaxiRequest: one Integer,
      originRequest: one Location,
      destinationRequest: one Location,
      customerRequest: one Customer,
      timestampRequest: one DateTime,
      taxiRide: one TaxiRide,
      taxiBooking: lone TaxiBooking
}
```

```
sig TaxiRide {
      idTaxiRide: one Integer,
      taxiRideRequest: one TaxiRequest,
      fare: one Double,
      driver: one TaxiDriver,
      taxi: one Taxi
}

sig TaxiBooking {
      idTaxiBooking: one Integer,
      originBooking: one Location,
      destinationBooking: one Location,
      customerBooking: one Customer,
      timestampBooking: one DateTime,
      taxiBookingRequest: lone TaxiRequest
}

sig Notification {
      recipient: one User,
      message: one Strings,
      taxiRequest: one TaxiRequest
}

sig TaxiRequestNotification extends Notification {}

sig TaxiRequestConfirmedNotification extends Notification {
      waitingTime: one Time,
      comingTaxi: one Taxi
}

sig Reply {
      sender: one User,
      replyTo: one Notification,
      message: one Strings
}

sig ReplyToTaxiRequest extends Reply {
      requestAccepted: one Boolean
}
```

4.2.2 Facts

```
fact TaxiRideNeedsTaxiRequestAss256 {
      all tr: TaxiRide | one req: TaxiRequest | tr.taxiRideRequest = req
}
```

```
fact RequestEqualsBooking {
	all y: TaxiBooking, r: TaxiRequest | r in y.taxiBookingRequest =>
	(r.originRequest = y.originBooking && r.destinationRequest =
	y.destinationBooking)
}

fact InverseProperties {
	taxiRide = ~taxiRideRequest
	taxiBooking = ~taxiBookingRequest
	currentTaxi = ~currentTaxiDriver
}

fact RequestNotificationOnlyToDrivingTaxiDrivers {
	all y: TaxiRequestNotification | some t: TaxiDriver | y.recipient = t
	all y: TaxiRequestNotification | some t: Taxi | one b: True |
	y.recipient in t.currentTaxiDriver && t.available = b
}

fact Replies       {
	all r: ReplyToTaxiRequest | one t: TaxiDriver | r.sender = t
	all r: ReplyToTaxiRequest | one c: Customer | r.sender = c
	all n: TaxiRequestNotification | one t: TaxiDriver | n.recipient = t
	all n: TaxiRequestNotification | one r: ReplyToTaxiRequest | r.replyTo
	= n && r.sender = n.recipient
}

fact NoDuplicates {
	no disj u1, u2: User | u1.idUser = u2.idUser
	no disj t1, t2: Taxi | t1.idTaxi = t2.idTaxi
	no disj q1, q2: TaxiRequest | q1.idTaxiRequest = q2.idTaxiRequest
	no disj r1, r2: TaxiRide | r1.idTaxiRide = r2.idTaxiRide
	no disj b1, b2: TaxiBooking | b1.idTaxiBooking = b2.idTaxiBooking
}
```

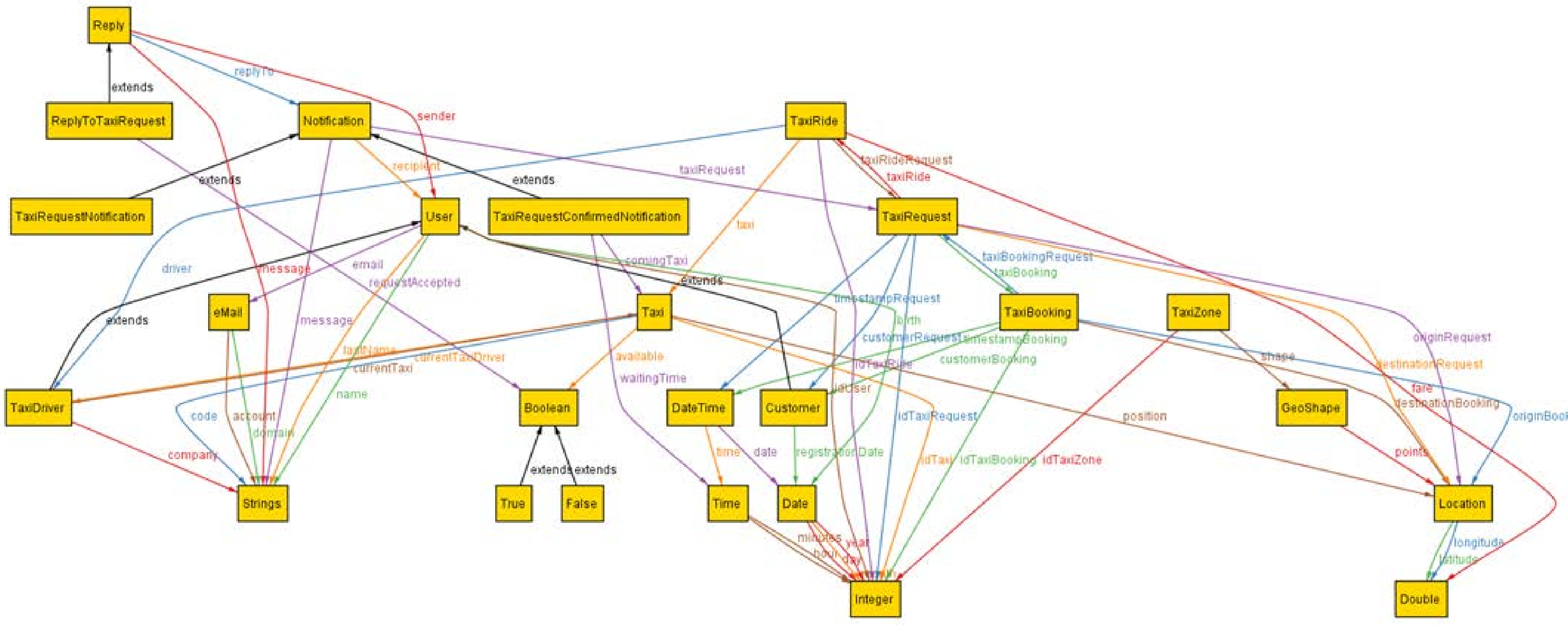
4.2.3 Assertions

```
assert Assumption251 {
	no u: User | one c: Customer, t : TaxiDriver | u = c && u = t
}
check Assumption251

assert Assumption253 {
	no r: TaxiRide | ! one tr:TaxiRequest | r.taxiRideRequest = tr
}
check Assumption253
```

```
assert BookingSameLocations {
      no r: TaxiRequest, y: TaxiBooking | r in y.taxiBookingRequest && (
      r.originRequest != y.originBooking || r.destinationRequest !=
      y.destinationBooking )
}
check BookingSameLocations

assert NoRequestNotificationsToCustomersOrUnavailableTaxiDrivers {
      no n: TaxiRequestNotification | #n.recipient.currentTaxi < 1
}
check NoRequestNotificationsToCustomersOrUnavailableTaxiDrivers

assert RequestNotificationOnlyToTaxiDrivers {
      all y: TaxiRequestNotification | one t: TaxiDriver | y.recipient = t
      all y: TaxiRequestNotification | ! one c: Customer | y.recipient = c
}
check RequestNotificationOnlyToTaxiDrivers

assert TaxiDriversAlwaysReply {
      all n: TaxiRequestNotification | some r: ReplyToTaxiRequest, t:
      TaxiDriver | (n.recipient = t) => r.replyTo = n
}
check TaxiDriversAlwaysReply
```

4.2.4 Generated World

The diagram of the generated world is in the following page.


Reply
extends
replyTo
sender
ReplyToTaxiRequest
Notification
TaxiRide
extends
recipient
taxiRequest
taxiRideRequest
taxiRide
TaxiRequestNotification
extends
User
TaxiRequestConfirmedNotification
TaxiRequest
taxi
driver
message
email
comingTaxi
requestAccepted
taxiBookingRequest
taxiBooking
extends
eMail
extends
Taxi
timestampRequest
birth
TaxiBooking
TaxiZone
message
customerRequest
timestampBooking
originRequest
taxiName
currentTaxiDriver
available
idTaxiRide
customerBooking
shape
destinationRequest
currentTaxi
waitingTime
idUser
fare
destinationBooking
TaxiDriver
code
account
name
Boolean
DateTime
Customer
idTaxiRequest
position
GeoShape
originBooking
domain
company
time
date
registrationDate
idTaxi
idTaxiBooking
idTaxiZone
points
extends
extends
Strings
True
False
Time
Date
Location
minutes
year
hour
day
longitude
latitude
Integer
Double