POLITECNICO
MILANO 1863

Design Document - myTaxiService

Ciceri, Filippo
filippo.ciceri@mail.polimi.it

Cesaro, Federico
federico.cesaro@mail.polimi.it

Capecchi, Luca
luca.capecchi@mail.polimi.it

December 3, 2015

Contents

1 Introduction

1.1 Purpose

This DD is a specification document intended to outline the design choices being made to implement "myTaxiService". This includes a detailed component view, information about patterns used and stylistic choices, a description of the algorithms to be used and an overview of the UI. These specifications are for the developers that will be writing the code for the service.

1.2 Scope

This service will require the development of the following:

Client-Side

- A mobile application for customers
- A web interface for customers to access the service
- A mobile application for taxi drivers to interact with calls.

Server-Side

- A server application that handles live queues
- A database that stores user data, including credentials
- Another database that handles Registered user rides, memorizing both booked rides and past itineraries

1.3 Definition, Acronyms and Abbreviations

Guest A guest to the service, namely someone who may be a driver or a registered user but has not been authenticated.

Registered User/User A user of the service who has gone through the registration process, whose information is stored within the service's database.

Driver A user of the service that provides their taxi for the service. They are registered in a manner different from that of other users.

Customer Any person trying to use a taxi through the application, both a registered users and guests.

Ride Generic term to talk about a taxi trip

Instant Ride A ride ordered directly through the application which instantly assigns a taxi to a customer.

Reservation/Booked Ride A ride booked in advance through the application.

Ride Locking/Locked Ride Term used to indicate reservations that have been "locked" from cancellation. A reservation is considered locked when it can no longer be cancelled or modified (2 hours before the start of the trip).

1.4 Document Structure

The remainder of the document is divided into the following sections:

1. The introduction, which contains basic information on how the service works
2. Architectural description, where the components of the service are described, along with the different views, the stylistic choices and the design choices taken.
3. Algorithm design, the section where a basic description of the algorithms being used is given.
4. UX, the section concerning the UI design choices accompanied by some mockups.
5. Requirements traceability, where requirement from the RASD are mapped to the specifications within this document.

2 Architectural Description

2.1 Overview

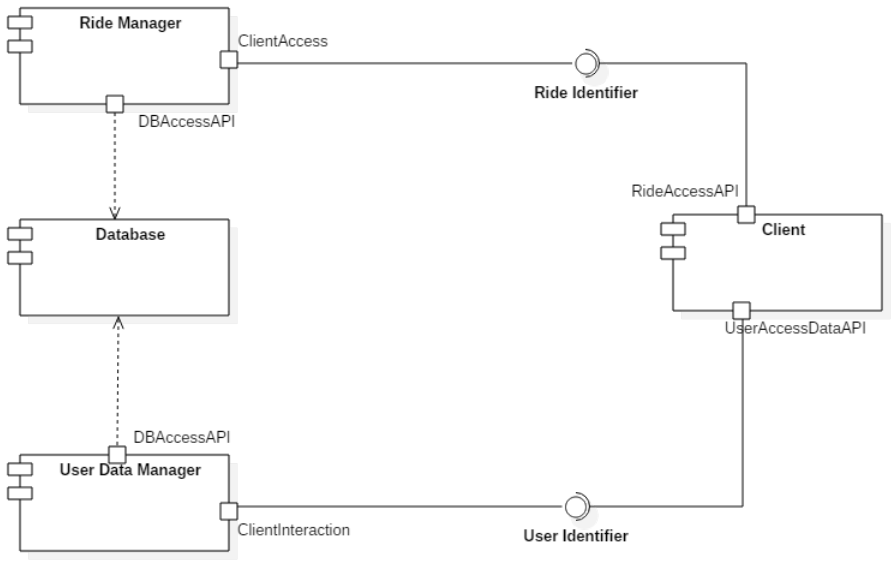
The logical structure of the service is divided in the following manner:

Client Interface Layer: Its purpose is to interact with all types of users. This includes both a web interface, reserved for customers, and a mobile application, used by both customers and drivers.

Business and Web Layer: This layer is responsible for the connection of the first layer with the Database Management Layer. It connects users to the information management system on the server. This layer operates server-side along with the Data Management Layer. It is also responsible for the management of driver queues and the creation of shared ride orders.

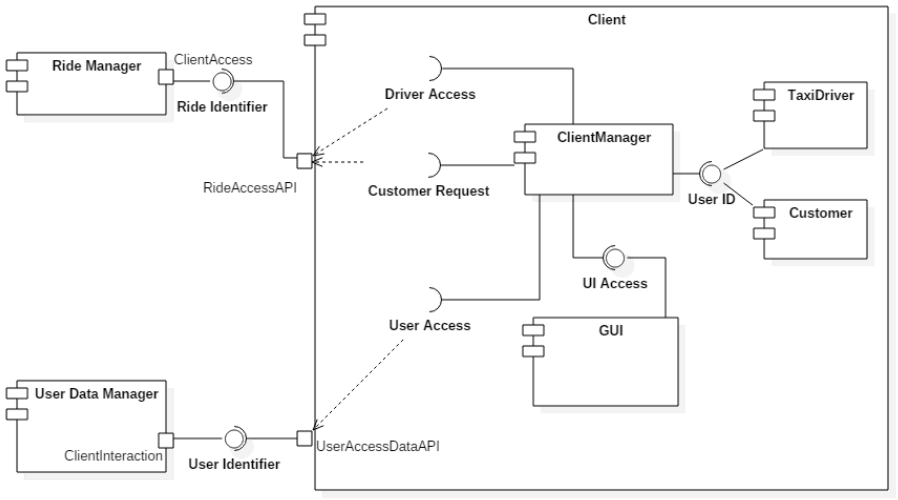
Database Management Layer: This layer is responsible for both user information management as well as reservation storage. Other than saving important and sensitive information about the user, it also stores information about the reservations made by users. It communicates with the Business Layer on a event basis when shared rides are to be created or locked.

2.2 High Level Components and Their Interaction

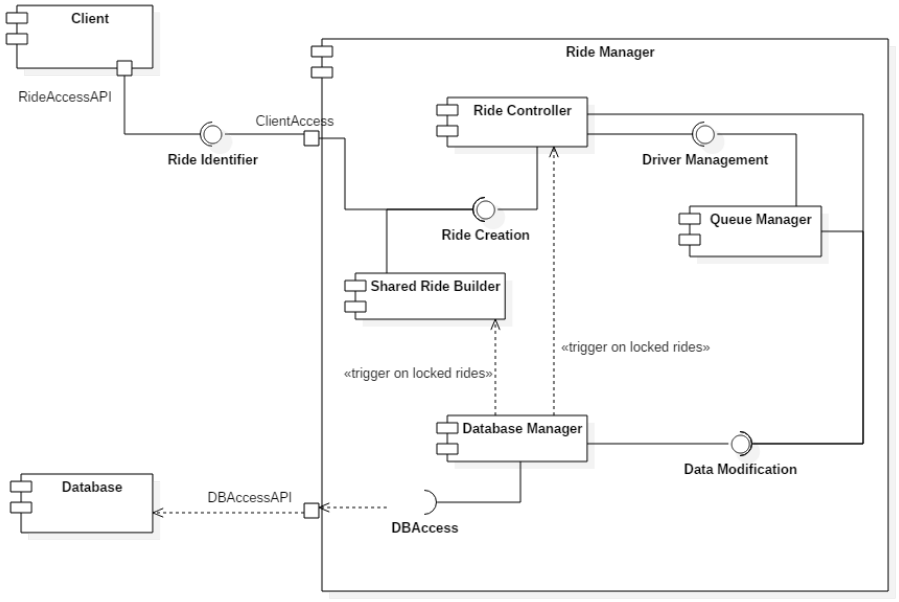


2.3 Component View

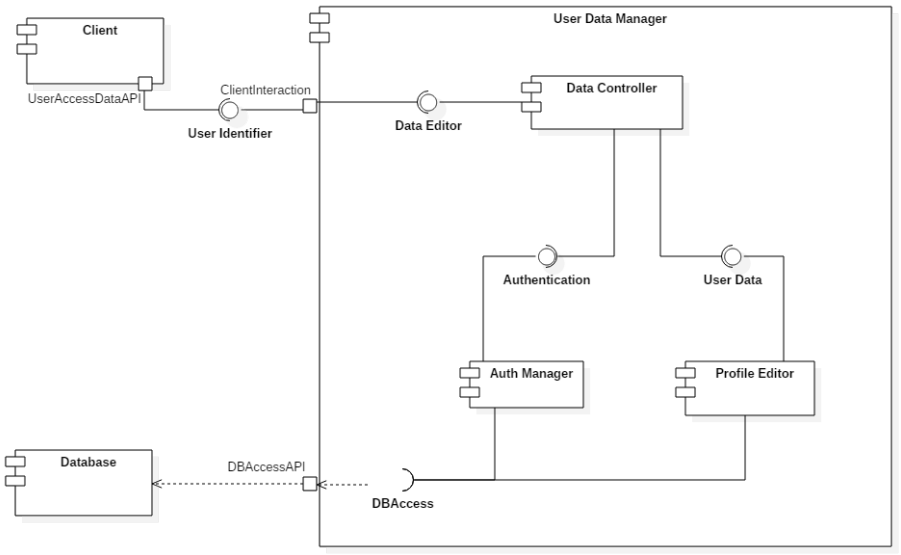
2.3.1 Client



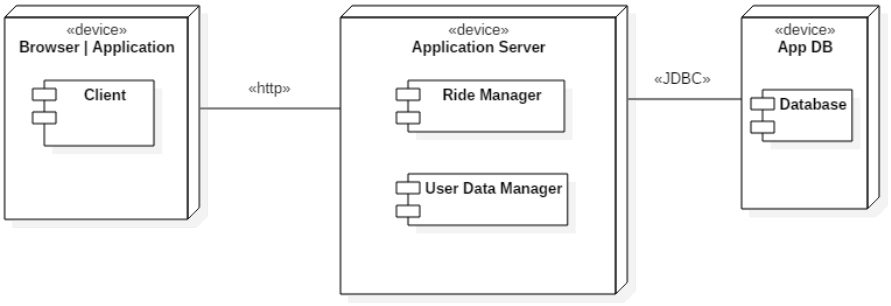
2.3.2 Ride Manager



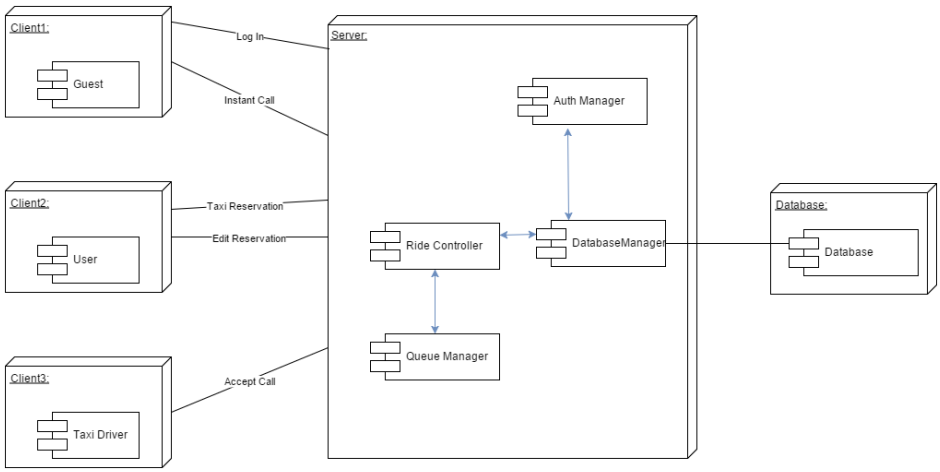
2.3.3 Data Manager



2.4 Deployment View



2.5 Runtime View



2.5.1 Sequence Diagram

In this section we show some service provided by our system to better understand how they work.

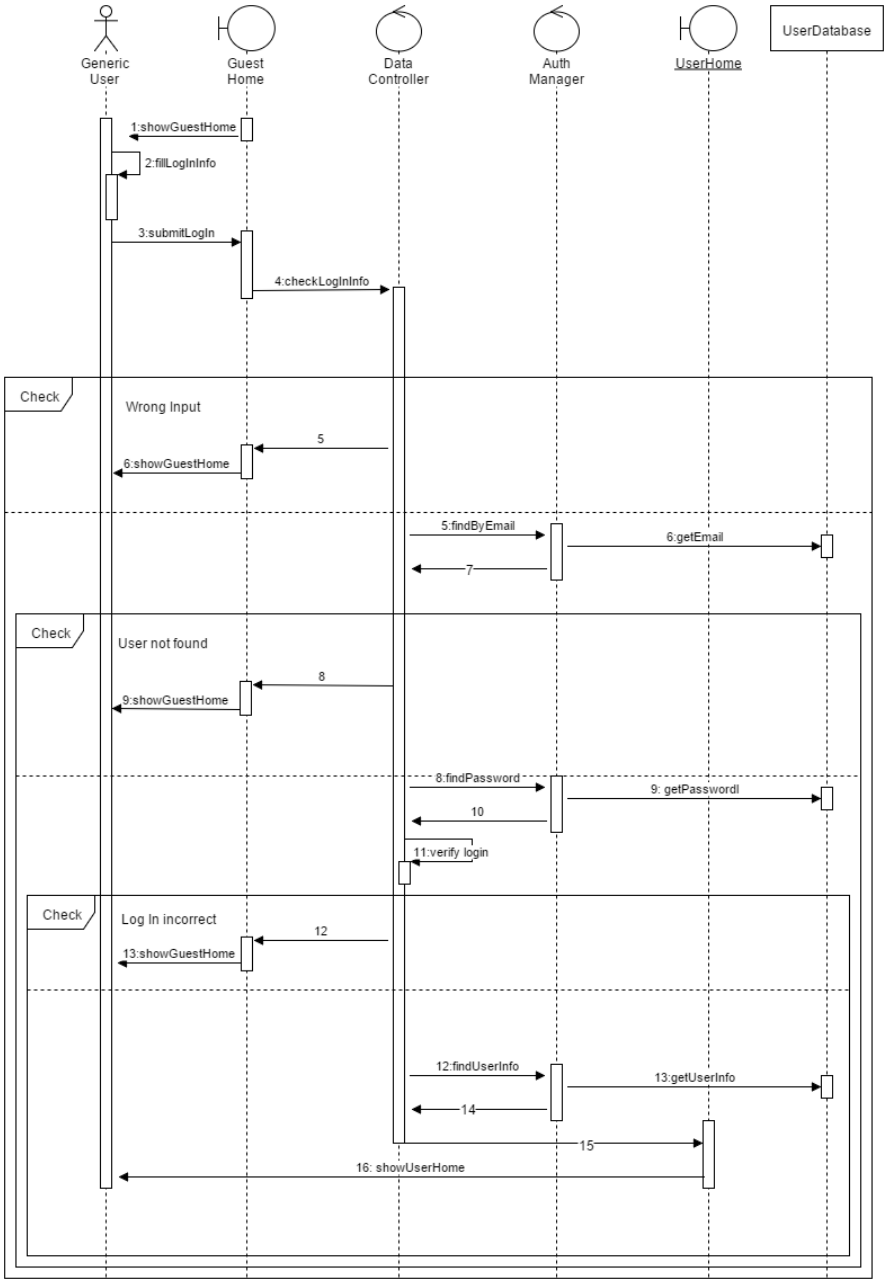
2.5.1.1 Log In

In order to Log In the guest must:

- Open the service.
- Fill the fields with is account data.

During the Log In the system must check that:

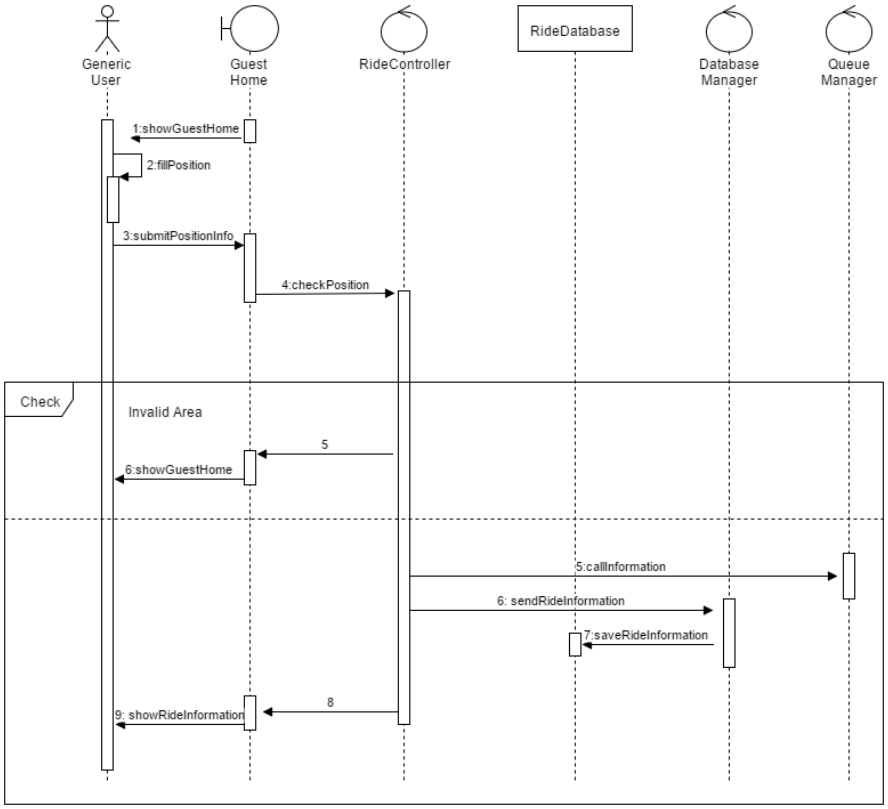
- All the field are filled with valid data (The first field must be an email and the second must follow the password rules).
- The email is stored in the database and connected to an account.
- The emails-passwords must coincide.


Generic User
Guest Home
Data Controller
Auth Manager
UserHome
UserDatabase
1:showGuestHome
2:fillLogInInfo
3:submitLogIn
4:checkLogInInfo
Check
Wrong Input
5
6:showGuestHome
5:findByEmail
6:getEmail
7
Check
User not found
8
9:showGuestHome
8:findPassword
9: getPasswordI
10
11:verify login
Check
Log In incorrect
12
13:showGuestHome
12:findUserInfo
13:getUserInfo
14
15
16: showUserHome

2.5.1.2 Instant Call

Both the guest and the user must tell their location to call a taxi.
The system must:

- Check the location given by the guest/user.
- Extract from the queue the closest taxi to the given location.
- Send the request to the taxi driver.
- Confirm the request to the guest/user.



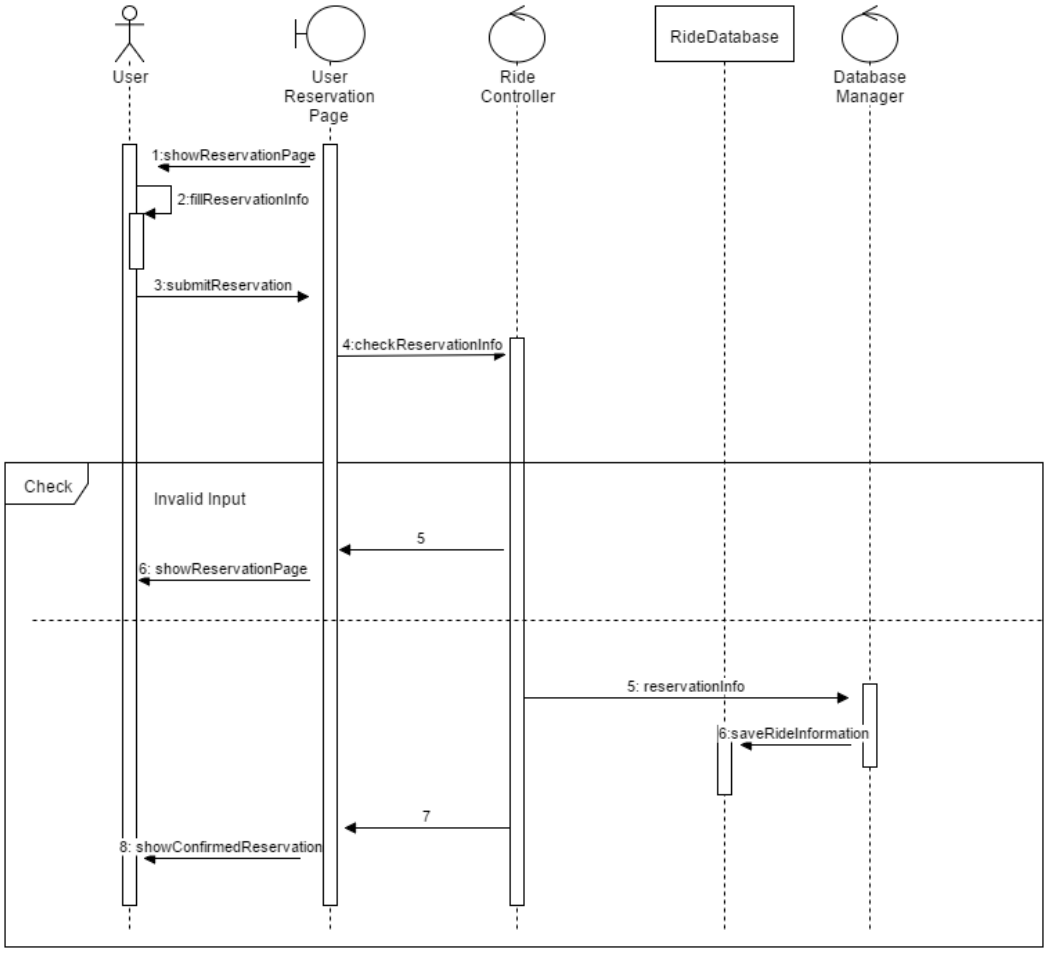
2.5.1.3 Taxi Reservation

A user can book a taxi with or without the sharing option by:

- Acces the booking section.
- Fill the requested fields.

The system must:

- Check the input given by the user.
- Update the Database.



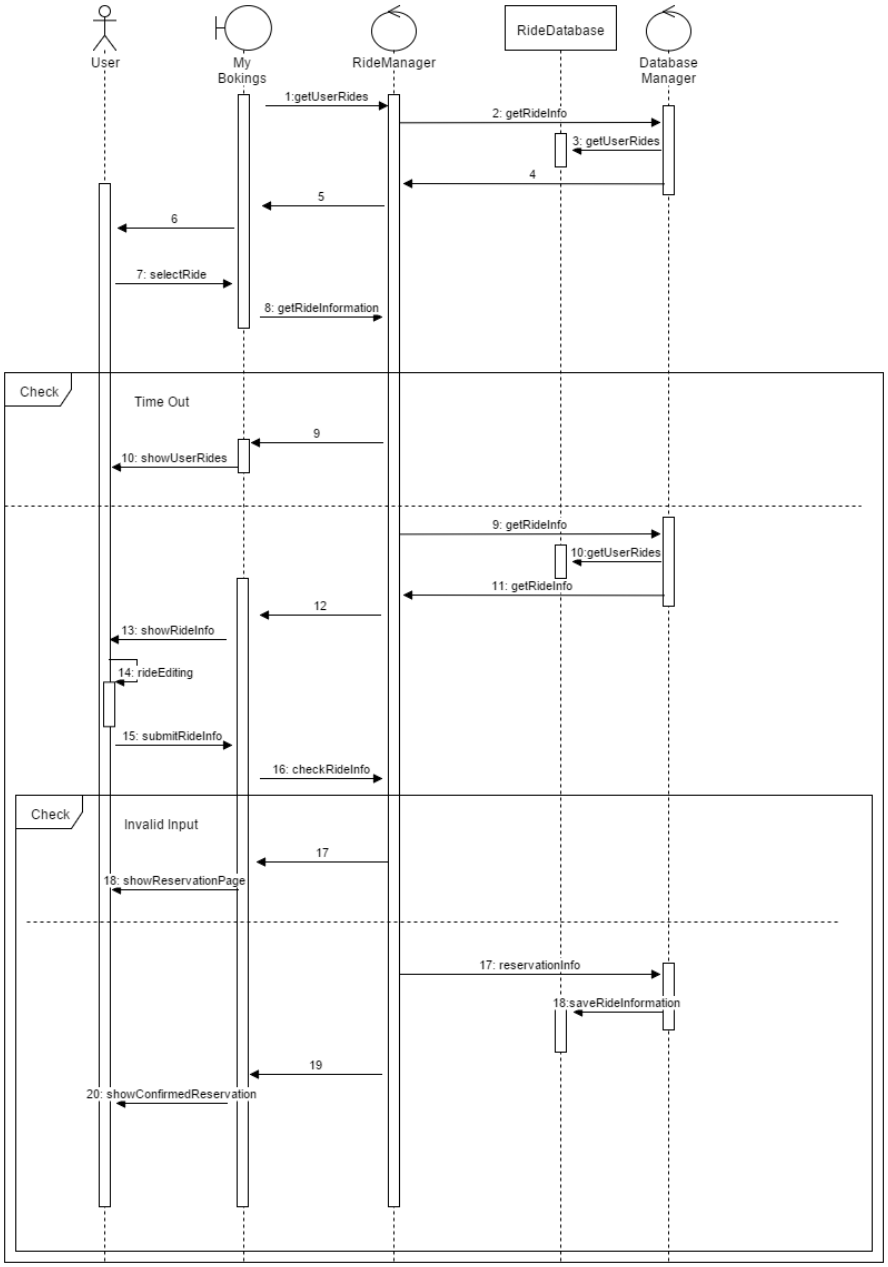
2.5.1.4 Edit Boking

A user can edit a reservation by:

- Selecting the ride to edit from a list.
- Edit/cancels the ride.

The system must:

- Show the user's Booked Ride.
- Check if the ride can be edited.
- Check the new input given.
- Update the Database.

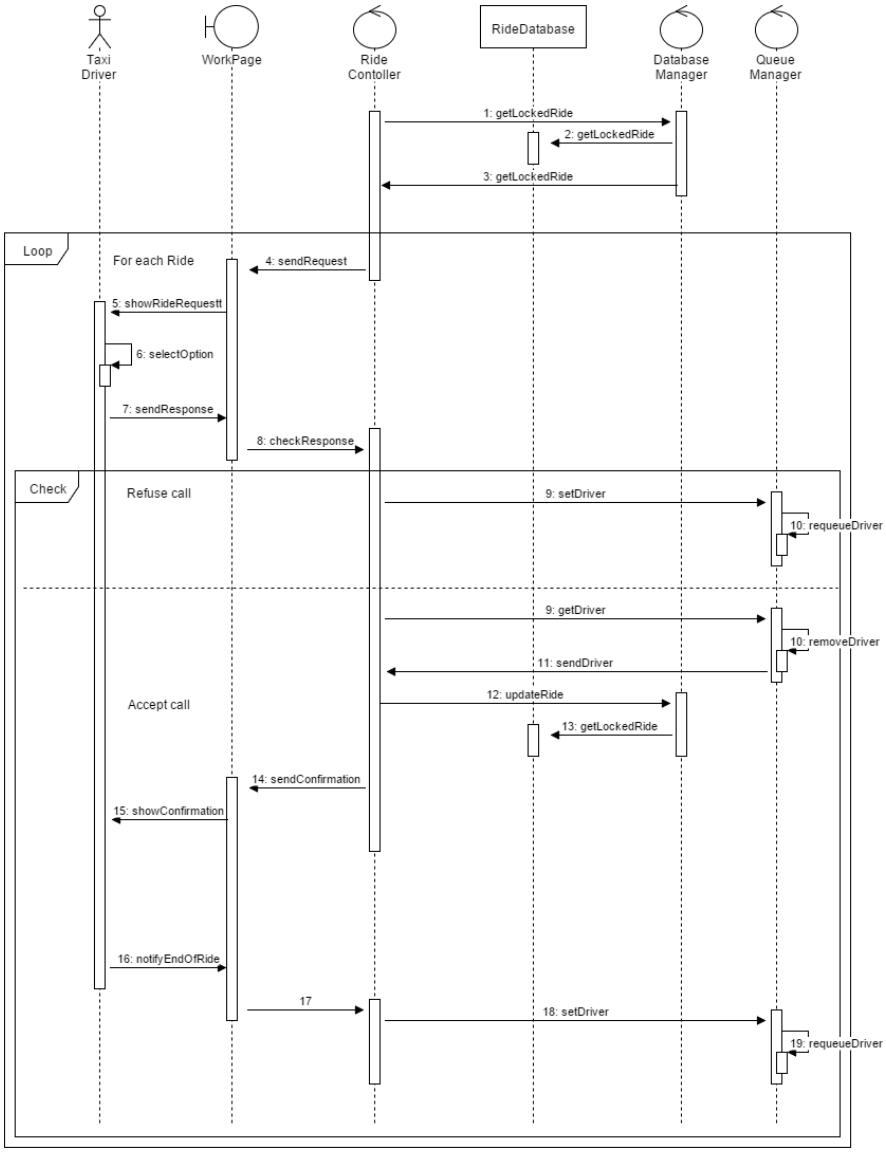

User
My Bokings
RideManager
RideDatabase
Database Manager
1:getUserRides
2: getRideInfo
3: getUserRides
4
5
6
7: selectRide
8: getRideInformation
Check
Time Out
9
10: showUserRides
9: getRideInfo
10:getUserRides
11: getRideInfo
12
13: showRideInfo
14: rideEditing
15: submitRideInfo
16: checkRideInfo
Check
Invalid Input
17
18: showReservationPage
17: reservationInfo
18:saveRideInformation
19
20: showConfirmedReservation

2.5.1.5 Accept/Refuse Call

The taxi driver must respond to a request sent by the system, and must notify the end of the ride.
The system must:

- Push a notification to the first driver in the queue near the location of the ride.
- Check the answer.
- Handle the queue due to the answer given.
- Update the Database.

The Ride Manager gets a notification from a Database Trigger when a Ride is Locked . A notification is pushed instantly if it is an Instant Call or only a small time before the departure time of a Booked Ride.


Taxi Driver
WorkPage
Ride Contoller
RideDatabase
Database Manager
Queue Manager
1: getLockedRide
2: getLockedRide
3: getLockedRide
Loop
For each Ride
4: sendRequest
5: showRideRequestt
6: selectOption
7: sendResponse
8: checkResponse
Check
Refuse call
9: setDriver
10: requeueDriver
9: getDriver
10: removeDriver
11: sendDriver
Accept call
12: updateRide
13: getLockedRide
14: sendConfirmation
15: showConfirmation
16: notifyEndOfRide
17
18: setDriver
19: requeueDriver

2.6 Component Interfaces

2.6.1 Intra-component

Ride Identifier Methods contained within this interface are used to send data form the Client to the Ride Manager, to enable ride creation, modification and deletion. It is used for both reservations and instant calls. Methods here also allow taxi drivers to interact with rides, by "effectively" modifying the status of the rides.

User Identifier This interface handles User access to the system. It has methods for both authentication and insertion of user data. All methods in this interface use end-to-end encryption to protect sensitive user data.

2.6.2 Client

Driver Access This interface is used to access driver-only functions which regard the status of rides. It is used to retrieve ride information for the driver and allows a driver to set their working status and accept/refuse jobs.

User Access This interface is used for authentication and access to the user information database.

Customer Request This interface gives users the methods to do all things related to rides.

User ID This interface gives separate methods to taxi driver users and regular customers to login to the service.

UI Access Methods in this interface are used to render the graphical part of the mobile application.

2.6.3 Ride Manager

Ride Creation This interface handles all incoming requests to edit and create rides. It also gives the the shared ride builder an object that allows it to access the database in order to query for shared rides.

Driver Management This interface allow the controller to insert/remove a taxi driver into the queue and allows the controller to retrieve the first available driver in the queue to be assigned to a ride.

Data Modification This interface is an intermediary between the controller and db accessor.

DBAccess This interface handles database access.

2.6.4 User Data Manager

Data Editor This interface takes all incoming requests to the data controller.

Authentication The methods in this interface let users and drivers log in to the system.

User Data This interface handles profile editing.

DBAccess This interface deals with database access.

2.7 Selected Architectural Styles and Patterns

Multi-Tier Architecture: We separated the system into multiple tiers, since we cannot rely on a solo machine, due both to the client-server nature of the application

Component Based Architecture: To simplify the implementation of future functionalities, we resort to dividing the system into components, rendering the development process easier to break-down and making extension of the service simpler

Event-Driven System: The most important part of the service, namely the assignment of taxis to drivers, is entirely event driven. Assignment, queueing and creation of shared rides are all driven by database triggers.

Singleton - Queue: To simplify the access to the queue from the various components we use the singleton pattern to render the queue element unique.

3 Algorithm Design

3.1 Taxi queueing system

The taxi queue algorithm is responsible for the assignment of taxi jobs to drivers.

3.1.1 Insertion and Removal

Taxi drivers can be inserted into the queue in 2 ways, either by starting their work period or by finishing their current job. In both cases, insertion is made at the end of the queue. Due to the nature of the queue, memorizing the time of entry and exit into the queue is not necessary, since the order is enough to determine the assignment priority.
Drivers may be removed from the queue by either selecting to stop working or by accepting a job. After having accepted a job, the drivers doesn't have to worry about queueing up again until the end of the trip. At that point, the driver can select the end of ride function within the application to be reinserted in the queue.

3.1.2 Insertion of Jobs into the Queue

There are different types of jobs that can be inserted into the queue. Currently we divide them in 3 types:

- Regular "instant calls", the most basic type of ride. These are more or less equivalent to calling the taxi through a phone. These offers are immediately assigned to the first taxi in the queue to accept them. More information about assigning the offers can be found in the next paragraph.
- Single order reservations, which are locked two hours before they are supposed to take place. Ten minutes before the reservation is scheduled, the offer is inserted into the queue and is assigned to a driver. When this offer is sent to the drivers, all details specified during the booking process are revealed to the driver, information like the number of passengers, the destination if specified, etc.....
- Shared ride reservations, which, much like single order reservations, are locked two hours before taking place. Since itineraries for these types of jobs are very important regarding the nature of the ride, all details

are sent to the driver. This includes all stops the driver has to make, the number of people being picked up at each stop, and an estimate of the amount of money the driver is going to receive.

3.1.3 Assignment of Passengers to Taxis

To allow a fair management of the queue, two things have to be taken into account when assigning passengers to taxis:

- Time - the time a taxi driver has been in the queue, which should give priority to a driver over another.
- Proximity - the distance a taxi driver is from the passenger they are supposed to be picking up.

To respect these factors, the system has to mix them into the assignment algorithm.

1. The system finds the position of the passenger, and plots it into a coordinate plane, setting the passenger's position as the origin.
2. The system looks for taxis within a certain range of the passenger. Ideally this number is between 500m and 1km.
3. If the system finds at least 5 taxis within range, it moves to the next step. Otherwise, the system increments the maximum distance until at least the minimum number of taxis is found in range.
4. At this point, the system sorts the drivers found by their position in the queue. It starts by sending the offer to the first driver. At this point the driver may either accept or refuse. In case of refusal, the system sends the request to the next taxi. This process continues until someone accepts the offer or until all "eligible" drivers have refused. If the offer has still not been taken. The system goes back to the second step, looking for new drivers by incrementing the distance again. Note that at this point drivers that have already refused the ride will not be sent the offer again, since they have already been on the assignment list. The process continues until one driver accepts the offer.

3.1.4 Movement in the queue

The queue has to guarantee a fair assignment of passengers to drivers. Therefore, the queue has to change as little as possible. There are only 3 methods

by which the queue changes Two of these have been described above(insertion and removal). The third method involves refusing a job offer. When a ride is assigned to a driver, the driver has a short time frame to accept or refuse the job. Refusing the ride results

3.2 Shared ride creation system

3.2.1 Structure of Shared rides

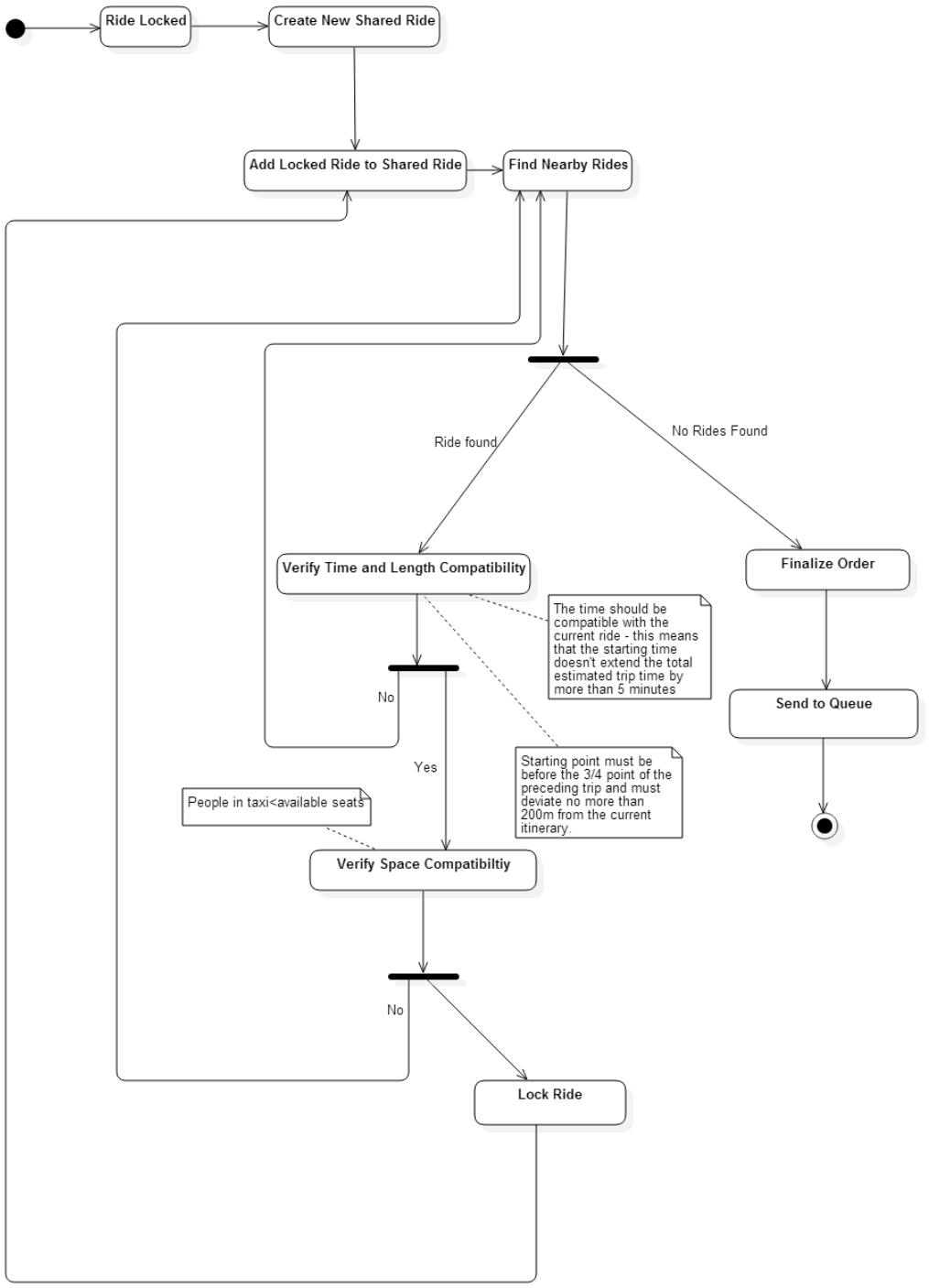
Shared rides are composed by a single itinerary, which includes all stops a taxi makes on a route, and by a set of individual reservations, each referring to a separate order. The itinerary has to include the starting point and destination of each sub-ride, and must be structured in a way that at no point in time the number of passengers on the taxi is greater than the number of seats in the taxi.

3.3 Composition of Rides

A ride is created on a need basis. Before being assigned to a ride, all shared-ride orders are only seen as requests in the system, but do not correspond to an actual job order. The order is only created when a ride has to be locked. When a ride is supposed to be logged, the system computes the best possible shared ride for the selected order. Shared rides need to consider three things mainly:

- The gain of the passenger - Is the shared order netting high enough savings to a passenger to justify sharing the ride. This refers not only to the money being saved, but also the time being "lost" vs a regular ride.
- The length of the ride. Since the shared ride can theoretically exist as a combination of multiple rides, it is important that the ride is not too long, so there must be an upper limit to the total estimated sum of the duration of a ride.
- The number of rides included into the shared ride - since a shared ride is the sum of multiple sub-rides, it is crucial that the system maps the trip accordingly. For example, the algorithm should never produce a ride composed of 10 consequent short rides, to the point where it would almost feel like a bus ride.

Keeping in mind these criteria, the algorithm shall behave as shown below:



4 UX - User Experience and Interface Design

This Section describes the User Experience, meaning all the webpages that a customer will be able to reach and all the functionalities he will have access to.
We used a Class Diagram with three stereotypes:

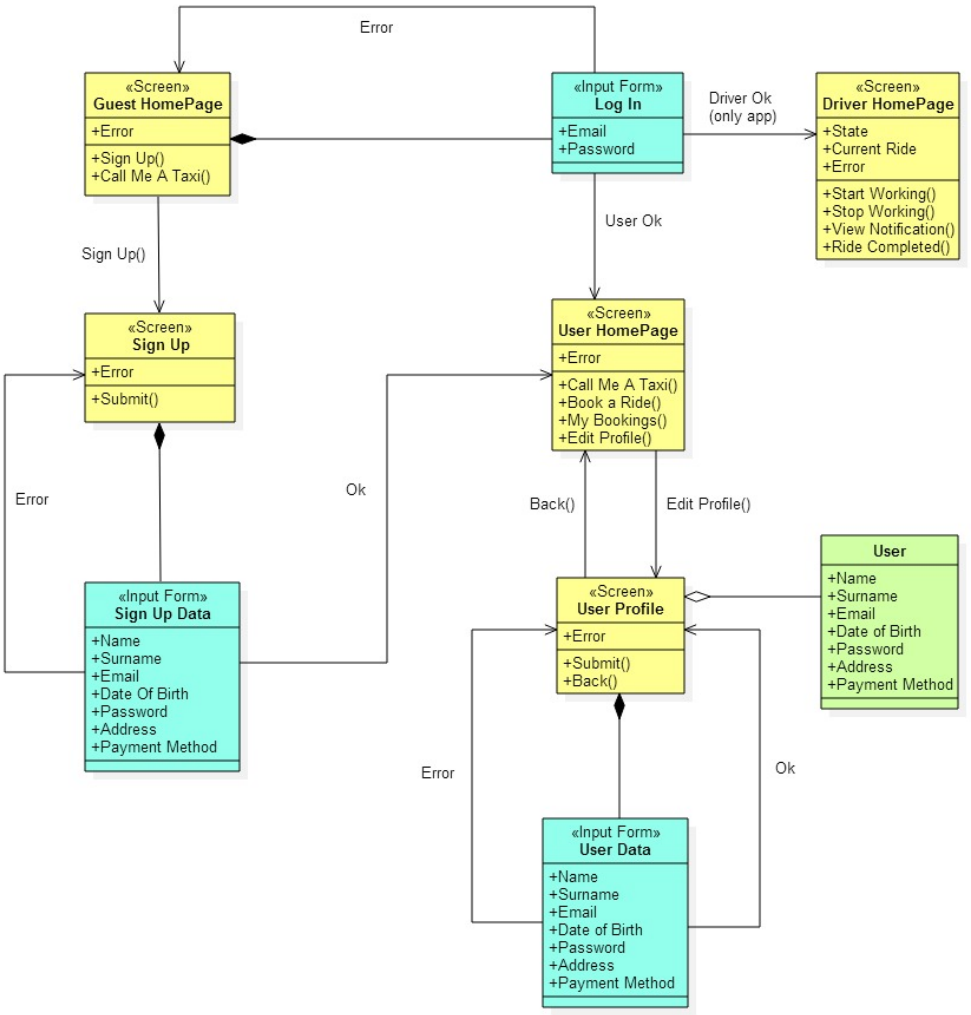
- *Screen* (*yellow*)*:* represents regular webpages, like the HomePages.
- *Input Form* (*blue*)*:* represents input fields, like the standard email-password form used to log in.
- *Regular Classes* (*green*)*:* represents the Classes that are present in the regular Class Diagram.

We divided our UX Diagram in three parts, in order to better highlight the connections between specified pages which have related functionalities:

1. **Log In, Sign Up and Data Editing**
   This UX Diagram shows the pages and the functions related to the management of the Customer data, like Log In, Sign Up and data modification
2. **Taxi Calls**
   Here all the connections between the pages used to call a taxi in all the possible ways are explained. In the Booked Ride case the possibility to modify the reservation data is also shown.
3. **Driver's Pages**
   The last Diagram shows the pages created for the Driver's use via the app. This includes functionalities like the status management and the job acceptance.

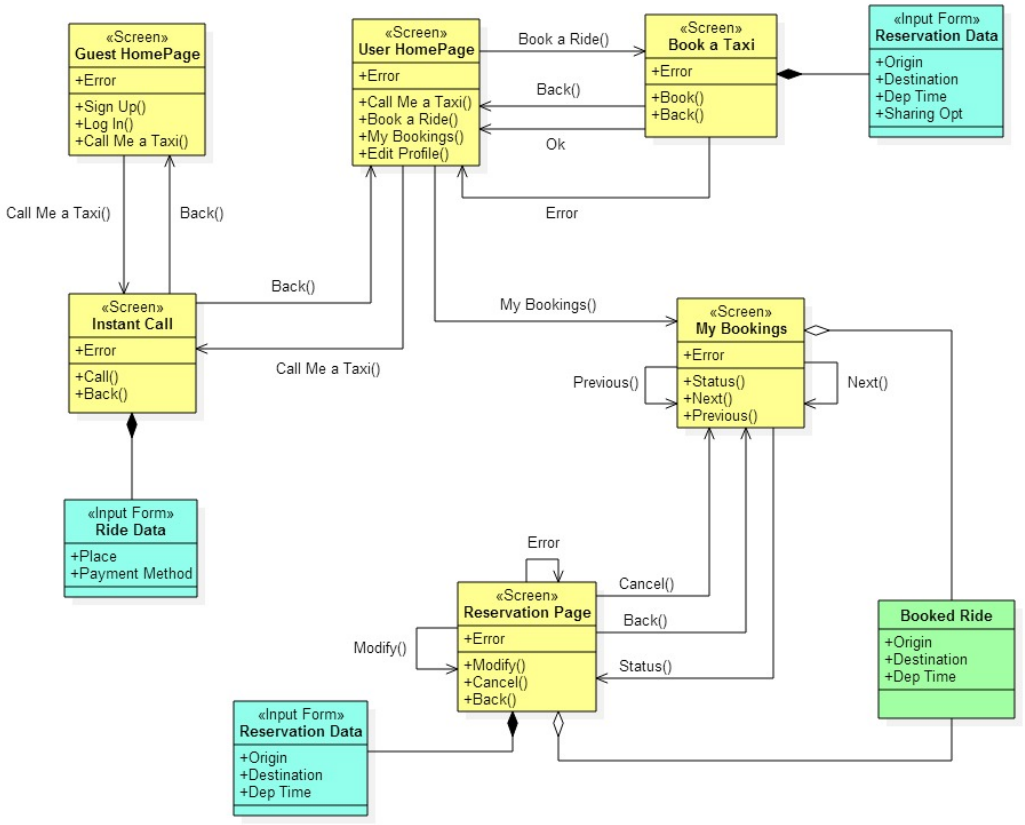
4.1 Log In, Sign Up and Data Editing

As shown in the diagram below, when someone opens either the website or the app he gets to the Guest's HomePage. Here he can call a taxi with the Instant Call function or fill the Input Form to Log In, in order to have access to the User's functions. Also a Guest that hasn't registered before can click on the *Sign Up* button that will take him to the dedicated webpage, where he can fill in the fields and register.
Users can also modify their data getting into the User Profile webpage and modifying an Input Form similar to the Sign Up one.
Lastly, Taxi Drivers (only via app) can perform their special Log In to get to the Driver's special HomePage.



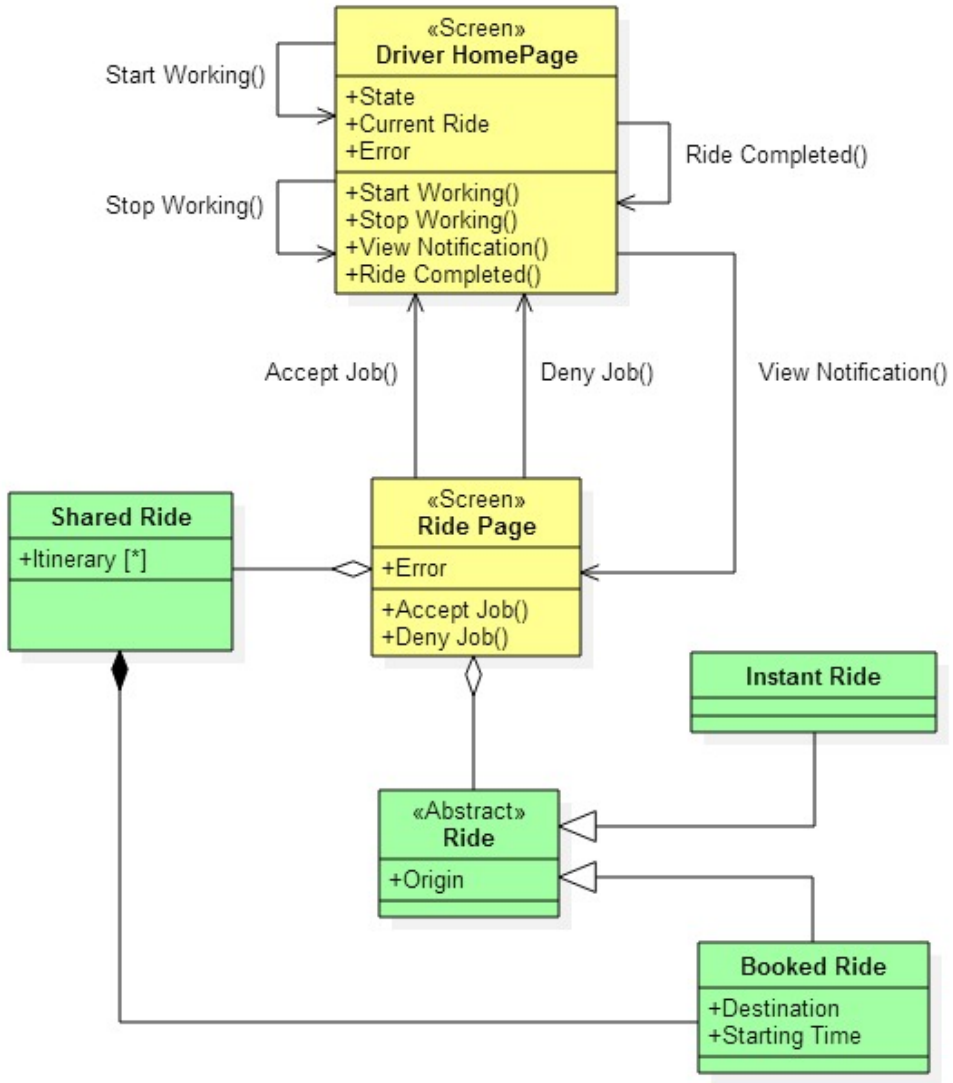
4.2 Taxi Calls

As described in the RASD, the Instant Call is a functionality that is granted to both Guests and Users; for this reason, the *Call Me a Taxi* button is present in both the HomePages.
In the Instant Call page the Customer can fill an Input Form to specify his location and the payment method.
On the other hand the Booking and sharing functions are User exclusive, so the dedicated button is only in his HomePage. This button opens a page with an Input Form used to specify all the data about the requested ride and lets a user eventually enable the Sharing option.
A User also has the possibility to visualize his past rides and see the data of the booked ones. By clicking the *Status* button a page with the current state of a ride and will be given the possibility of cancel or modify it (if it's not locked) will be shown.



4.3 Driver's Pages

The Driver's part of the UX Diagram is much simpler than the others because the Driver is supposed access the system only via mobile. Most of the functionalities are accessible from the main page, like the one concerning the management of the working status.
The only other page viewable is the Ride Page, shown when a notification of a new possibile job is recieved. In this page the Driver can see the Ride's data and decide if accept it or not.



5 Requirements Traceability

5.1 Log In, Sign Up and Data Editing

5.1.1 System Registration

The Sign Up function is implemented by the ClientManager. Using the User Access interface, it comunicates with the UserDataManager and saves the new User's data.

5.1.2 User LogIn

The LogIn is performed by the ClientManager that, as in the Sign Up, communicates with the UserDataManager through the User Access Interface. Here the AuthManager verifies the correspondance between the info inserted by the Customer and the ones stored in the DataBase using the DBAccess Interface and answers to the ClientManager.

5.1.3 User edit information

Similarly to the LogIn function the ClientManager communicates with the Data Manager. Here the Profile Editor, through the DBAccess Interface updates the User's data.

5.2 Registered User Requirements

5.2.1 Instant Calls

This function is implemented by the Ride Controller, that recieves a request from a Client via the Ride Creation Interface, creates a Ride, looks for Taxi Drivers that may accept the jobs and reports it to them. The Controller also communicates with the DataBase to store the Ride's information.

5.2.2 Booking and Sharing

In this case the Ride Controller behaves the same way as the Instant Calls, with a difference: when a Ride has to be locked, it's passed to the Shared Ride Builder that looks for other Rides that could match; then it creates a Shared Ride composing the Rides that it found and sends the result to the Ride Manager

5.2.3 Booking Editing

To modify a Ride's data the Client passes to the Ride Controller the new information. The Controller checks if the modification follows the rules and through the Data Modification Interface writes the new infos in the DataBase.

5.3 Taxi Driver Requirements

5.3.1 Driver LogIn

The Driver's LogIn works the same way as the Customer's LogIn does.

5.3.2 Driver Work Settings

The Driver, through the UI Access interface, comunicates to the ClientManager the new set of his status. The ClientManager provides the information to the RideManager.

5.3.3 Driver Ride Acceptance and Ending

As in the previous case, the Driver comunicates his decision (accept, refuse or end a ride) through the UI Interface.
This information arrives to the QueueManager from the ClientManager using the Ride Identifier Interface; then the QueueManager will perform the adequate changes to the Driver's position in the queue.

6 Appendix

6.1 Software and Tools Used

- MiKTeX (http://www.miktex.org/) to format and create this document.
- StarUML (http://www.staruml.io/) used to create UML diagrams: Use Case and Class Diagrams as well as State Charts.
- Draw.io (http://www.draw.io/) to create advanced sequence diagrams

6.2 Work Hours

Time spent on the creation of this DD:

- Filippo Ciceri: 24 hours
- Federico Cesaro: 23 hours
- Luca Capecchi: 21 hours