**6.User Interface Design**

**6.1 Design Choices**

The idea, always present within the project, is to create a tool that is easy and immediate for the user, and the user interface is designed to meet these requirements.

From the mockups already presented in the RASD, it is in fact clear that the aim is to present the users a neat and minimal interface, in order to make the procedures intuitive and as quick as possible.

6.2 **User Interface and Page Flow**

Here it is presented a diagram that, with the proper stereotypes ("page" and "form"), shows how pages are related, what important components are present, every input form, and how the navigation through the website is structured.

The graph is presented as a class diagram, and the symbols have the same meaning as if they were used in that kind of diagrams; it is important to notice that *<<page>>* means that the class represents a web page, and *<<form>>* identifies an input form contained in a specific page.

The home page structure is the same for all the users, focused on the fast request for a taxi; every user home page is then developed from this point, adding links and features associated to the relative user.

Following these links every customer can navigate through the pages, but only through those for which he has permission.

Here are presented only direct flows, associations representing cancellations or links to previous or home pages are omitted in order to simplify the reading of the graph.

6.3 User Interfaces

In order to make the style of the application more clear, here are presented some of the pages not already contained in the RASD, such as the taxi driver's home page, and the pages used by the administrator to manage users and taxi applications.

TAXI AND ADMIN PAGES

6.4 User Experience

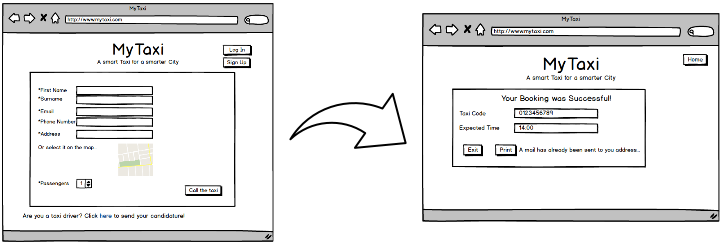
In order to make the service as immediate as possible, the home page of the application is composed of the module to send a rapid request or a taxi, inserting the indispensable data like name and phone number. On the side there are buttons to sign up and login. After the login every kind of user has a personal home page: the registered user has the options to book a taxi in advance and modify his data; taxi drivers can give/remove availability and accept requests; administrators have access to users data and to taxi driver applications.

Every option is fully described in the RASD section related to Use Cases, and the navigation uses only buttons to navigate through the pages, and forms to insert data: in this way the user can understand how to use the application at the first use.

Here are shown some page flows based on the mockups already presented, in order to fully describe how pages are related, and in which way the navigation is structured

6.4.1 Simple call

A guest already finds on the home page the form to call a taxi, when he has filled in the fields, he waits for the taxi, and when the call is accepted, the guest is brought on the confirmation page, with the call details.



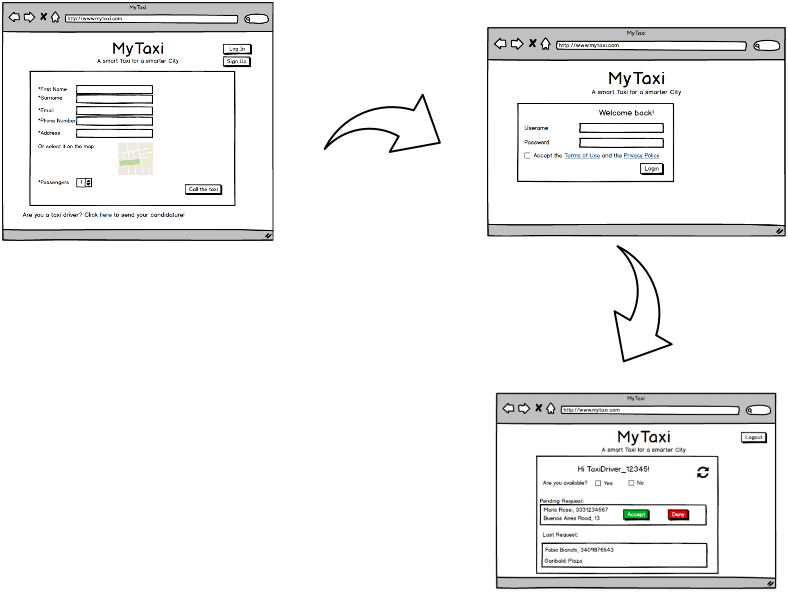
6.4.2 Sign Up and Reservation

If a guest needs to make a reservation, from the home page he has to click on the Sign In button: in the following page he will find the registration form; once he submit the data, he will be redirected on the home page, but this time as a user, and thus allowed to make a reservation. He finds a form very similar to the simple call one, that allows him to book a taxi. After the process a confirmation page will be shown, where the user can read a resume and save the page, or go back to the home page.



6.4.3 Taxi Driver Login and Request Notification

A taxi driver is ready to work, so on the home page he clicks on the Login button. In the following page he inserts his username and password, and he will be redirected to the home page dedicated to taxi. This page is comprehensive of everything he needs: in fact here he can see his state and his rides. In a dedicated box he will find new requests, ready to be accepted or rejected just by clicking on the specific button.



2.4 COMPONENT INTERFACES

Every component needs to interact with other elements of the system. The communication between components is managed using interfaces. Every interface takes a fixed number of arguments, and returning the desired result, so that components can avoid knowing how every operation is implemented, and so reducing coupling.

Here is presented a brief description of every interface, in order to perfectly explain the interaction between components.

**Localizer API**

**+getPosition (Bin signal : String coordinates)**

It is the interface used to calculate the client position using an external GPS service. It receives the signal and provides the coordinates for the client.

**Request**

**+createRequest (String name, String surname, String email, String phone\_number, String position, String address, Int passengers : Int taxi\_code, Time expected\_time)**

It is used by a user to ask for a taxi ride, it takes essential data as an input, and after all the procedure to find the taxi, returns the assigned taxi code and an estimated waiting time

**+viewRequest()**

**Reservation**

**+createReservation (String username, String start\_point, String end\_point, Int passengers, Date reservation\_date, Time reservation\_time : Bool confirmation)**

This interface allows the user to book a taxi ride in advance; in order to do so the interface only needs the client username (to retrieve user details) and ride details such as start and end addresses, number of passengers and date and time of the ride. The request will be created a couple of minutes before the indicated time, so the only thing returned to the user is the outcome (positive or negative) of the reservation

**Available Taxi**

**+Accept (String request\_ID, String taxi\_code : Bool accepted)**

This interface represent the attempt to assign a taxi to a specific request: it takes the request id to identify univocally the request, and sends the request to the taxi; it then sends back to the system the taxi driver answer.

**Queue Service**

**+insertTaxi (String taxi\_code, String position : String zone)**

The target of this method is to insert an available taxi in a queue, so it receives the taxi code and its position, sending back the name of the zone in which the taxi is positioned

**+getTaxi(String request\_start\_address : String taxi\_code)**

The request server ask the queue server a possible taxi to forward a specific request, sending the start address indicated in the request. The queue server detects the associated zone queue and query the queue manager for a taxi; as soon as it receives the response, it forwards the taxi code to the request server

**Queue Manager**

**+getTaxiFromQueue(int queue\_index : String taxi\_code)**

The queue manager receives the queue index and so requires the first taxi in the indicated queue. When the taxi is found, its code is sent to the queue manager, ready to be sent to the waiting request

**Send Notification**

**+sendEmail(String email,** **String event\_type, String event\_id : void)**

**+sendSMS(String phone\_number, String event\_type, String event\_id : void)**

**+sendNotification(String username, String event\_type, String event\_id : void)**

This set of methods is created in order to delegate notifications at the specific component. The system sends the data related to the event that has to be notified, and the target media for the notification (email, mobile phone or mobile application). The notification server then provides to send the proper notification to the user.

**DB Service**

**+insertRequest(Request req : void)**

**+insertUser(User usr : void)**

**+insertTaxi(Taxi taxi\_driver : void)**

**+insertReservation(Reservation reserv : void)**

**+insertAdmin(Admin admin : void)**

**+modifyUser(User newusr : void)**

**+modifyTaxi(Taxi newtaxi : void)**

The set of methods defined by this interface is meant to define how entities are inserted, modified, or deleted from the databases. They take the entities classes, with every detail related, that will be decomposed in basic type pieces of information and inserted in the database. The removal of entities is not a physical removal, it is managed by a flag that shows if that user is active or not.