**Sign up and sign in**

The subsystem that controls the log-in and sign-up operation is extremely simple, because his only function is to verify the access data in the first case, and to create a new user in the database in the second case.

The boundary is the same for both operations: we decided to include the button fort the sign up in the really first page that the user opens; obviously we suppose that he will use this function only once.

If the controller receives a log-in request, he will check the username, and if it is valid he will load the user information from the database, through the method “load\_user”. Then it will check if the information matches, and if they are correct it will allow the access to the Home page.

If the controller receives a sign-up request, he will just open an input form the user will compile with his personal informations: Name, Surname, mail, Tel. Number; then, through the method “create\_user” the controller will put it in into the database, and the user will visualize again the log-in boundary.

**Make a call**

The subsystem that controls the calls has to manage two different aspects of it, because there is an interaction between the user that makes the call and the taxi driver who receives it.

We have two different boundaries, one for the user and one for the taxi driver.

The boundary “call” is the first one, and it starts with a screen where the user has to choose which type of call he wants to create. Then he will choose the departure address from another screen. At this point, the system assigns to him a taxi driver, and the user will display the estimated arrival time, and he will have to confirm the call.

The boundary “taxi\_call” is the second one, and starts when the system assigns a call to the taxi driver. He visualizes the user basic information, and he can accept or decline the call. If he accepts, he has to wait until the user confirm the call (after visualizing the driver information).

The controller in this case is just one, and it interacts with both the actors. First, when he receives a request for a call, it checks the call type through the method “check\_call\_type”, just to decide how to manage it. Then he will receive the address information (could be just the start for an immediate call, or start and arrival otherwise) and it check if they are valid. For a delayed call, it also use the method “check\_start\_date” to ensure that is a valid one. For the shared call, it has to calculate the route in order to pick up (or not) other participants; to do that, it uses the method “calculate\_route”, that will be supported by an external service (like google maps). If the user does not confirm the call, the manager will delete it from the database using the method “delete\_call”. The method “load\_taxi\_information” is necessary to load the information that the user will visualize on the screen.

**Calculate cost**

This particular subsystem has no interactions with the users: his task is just to calculate the cost of every single passenger of a shared call, when he get down from the taxi.

The boundary is visualized by the taxi driver while he is performing the ride, and he visualizes the number of passengers still on the taxi and he have the button “deploy”, that he has to press when someone arrives at destination. After the system has calculated it, he will visualize on the screen the cost of the ride.

The controller has just one method, associated with the only function that is request to him. Is connected to the call manager, that is the one who can load the call information from the database (the departure address is needed).

**Location**

This subsystem is used for the personal locations of a user.

The Boundary is a list of all the locations already in database for the user; from this screen, he can add a new one, or modify or delete an existing one.

The controller has all the methods associated with this functions, including “check\_address” that ensures that a location is valid, and “check\_duplicates”, that ensures that the user has not another existing location with the same name.

**Account settings**

This subsystem manages the changes to the account information of a user.

The user interacts through the boundary, that visualizes the screen containing the information; the user is allowed to change only the email and the tel. number, he can’t modify his name and surname, and obviously can only read but not modify his ID and his feedback.

The manager has a method to load all the user information, and the two methods to modify the mail and the tel. number: this methods will also check if the new data are valid.

**Settings**

This subsystem is used to manage the app settings.

The boundary is the one that permits to visualize the actual settings and modify some of them.

The manager has the methods that permits to change the background color of the app, to select a different language, to activate or deactivate the notifications when the app is not open. The method “set\_date” is used to create a filter for the past calls: the app loads from the server just the calls from that date beyond. The method “check date” is used to ensure that the selected date is a valid one.