Politecnico di Milano
A.A. 2015-2016
Software Engineering 2

Design Document

Alessandro Macchi
Caterina Finetti
Simone Manzoli

Sommario

1. Introduction

1.1 Description of the document

This document is focused on the Data design and on how the interaction between the system and the users should be.
The scope of this document is to provide the design of the application, describing and justifying the reason of our choices, for example the Data Base schema, the User Experience that our application should provide to our users and the main classes that we will make on the Implementation phase.
The Data Base design is essential to build a consistent Data Base that supports in an efficient way all the queries and the manipulations on the data.

1.2 Definitions, acronyms and abbreviations

User: is a person who is registered in the database of the application. He has access to all the functions of the program that involves the requiring of a taxi, shared or not. He also has the possibility to save a list of preferred locations, that he can automatically choose when the System require from him an address as starting position or destination.

Guest: is a person who is using the application but is not registered in the database. He has access only to the registration functions.

User Information: all the information that concern a user; most of them have to be inserted during the registration (Name, Surname, tel. Number, e-mail, password), some of them can be inserted at any time after the registration (such as the personals locations) and others are assigned by the system (for example the number of blank-calls or the feedback).

Feedback: the feedback measures the reliability of a user. Is a simple relation between the total number of calls and the number of blank calls that a user have made (so a feedback equals to 1 means that he never missed a call).

Basic User Information: The information that a taxi driver visualizes when he receive a call. They are: Name, Surname, Feedback, Telephone Number of the user.

Blank Call: we define Blank-call a call for a taxi where the client is not at the starting location when the taxi arrives with a maximum late of X minutes, or a call that the user cancel before X minutes.

Missed Call: we define Missed-call a call for a taxi where the client is not at the starting location when the taxi arrives (X+1) or more minutes late.

Partner: someone who share a run with a user

Pick-up place: the Address where a user asks a taxi to come

Taxi Driver: a registered Taxi Driver

BCE diagram: boundary-controller-entity diagram

UX diagram: User experience diagram

2. Architectual Design

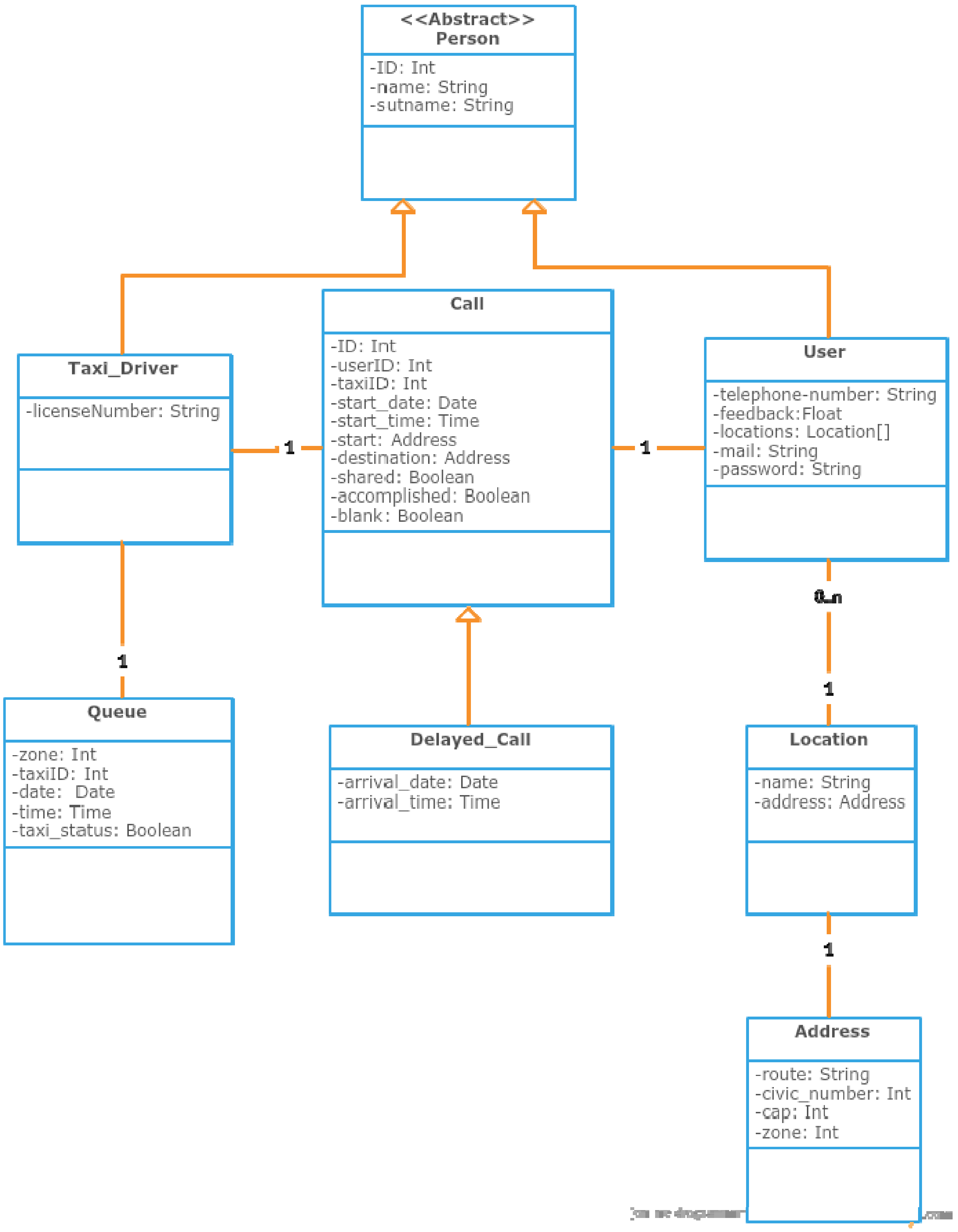
2.1 Overview

We decomposed the system in several sub-systems for visualize better every single part.
The sub-systems in which our system is divided are:
Call sub-system
Shared-call cost sub-system
User sub-system
Manage location sub-system
Manage account sub-system
Manage settings sub-system
Log in sub-system
Sign up sub-system
In this document we analyze every sub-system and show how the users will see and interact with our system.

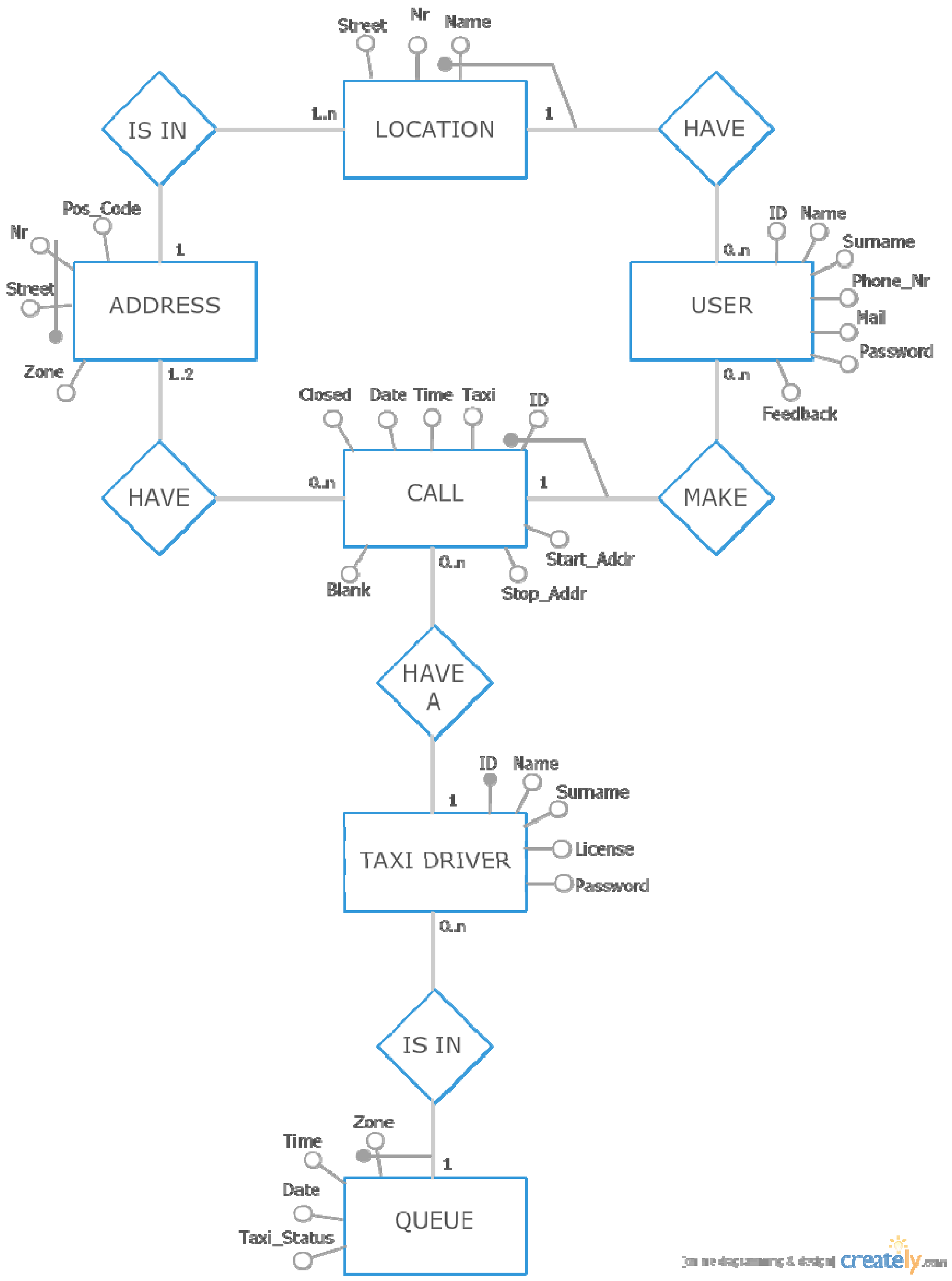
2.2 High level components and their interaction

In this section we provides to show the Class diagram and the concept diagram.
In the class diagram we defined and described the classes of our system and how they interact with each other.
The concept diagram has the scope of documenting in a clear and synthetic way the database structure that are part of our informative system.

2.2.1 Class Diagramm (UML)



2.2.2 Concept Diagram



2.3 Component view

The final result of our database will have this structure:

USER (<u>ID</u>, Name, Surname, Phone_Nr, Mail, Password, Feedback)
CALL (<u>ID, User</u>, Taxi, Date, Time, Start_Addr, Stop_Addr, Closed, Blank)
TAXI_DRIVER (<u>ID</u>, Name, Surname, Licence, Password)
QUEUE (<u>Zone, Taxi</u>, Date, Time, Taxi_Status)
ADDRESS (<u>Street, Nr</u>, Zone, Pos_Code)
LOCATION (<u>User, Name</u>, Street, Nr)

create
table User (ID integer(12) primary key, Name varchar(25), Surname varchar(25), Phone_Nr varchar(18), Mail varchar(40), Password varchar(20), Feedback float between 1 and 100, foreign key (ID) references Call (User) reference Location (User), on delete no action on update cascade)

create
table Call (ID integer(12), User varchar(25), Taxi integer(n), Date date, Time time, Start_Addr integer(n), Stop_Addr integer(n), Closed boolean, Blank boolean, primary key (ID, User), on delete no action on update cascade)

create
table TaxiDriver (ID integer(6) primary key, Name varchar(25), Surname varchar(25), Licence varchar(18), Password varchar(20), foreign key (ID) references Call (Taxi) reference Queue (Taxi), on delete no action on update cascade)

create
table Queue (Zone integer(6), Taxi integer(n), Date date, Time time, Taxi_Status boolean, primary key (Zone, Taxi), on delete no action on update cascade)
create
table Address (Street varchar(25), Nr integer(4), Zone integer(6), Pos_Code integer(5),primary key (Street, Nr), foreign key (Street, Nr) references Location (Street, Nr), on delete no action on update cascade)

create
table Location (User integer(12), Name varchar(25), Street varchar(25), Nr integer(4), primary key (User, Name), on delete no action on update cascade)

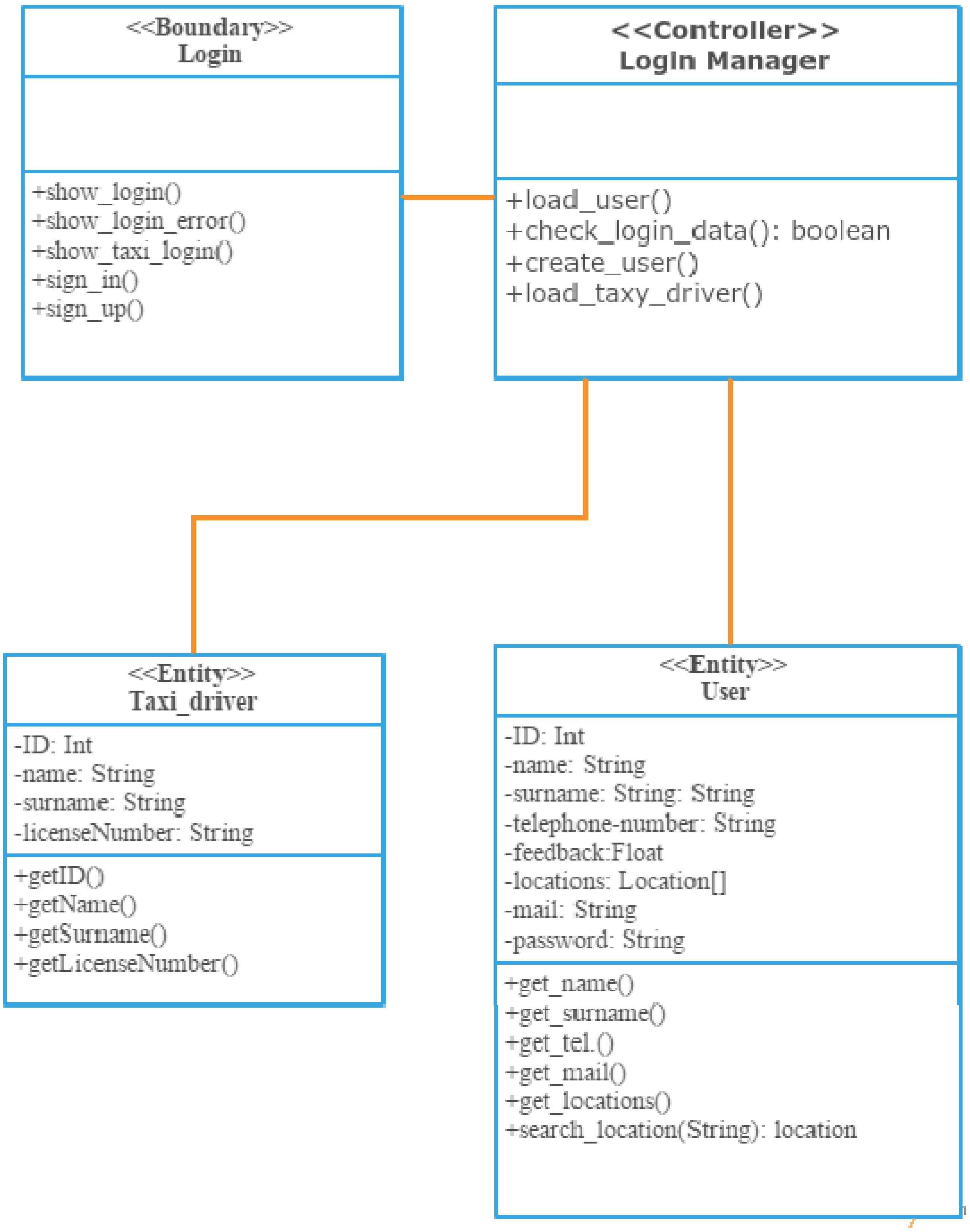
2.4 Deployment view

In this section we will show our boundary-controller-entity diagrams that are defined as:
Entities *(model)*
Objects representing system data, often from the domain model.
Boundaries *(view)*
Objects that interface with system actors (e.g. a user or external service). Windows, screens and menus are examples of boundaries that interface with users.
Controls *(controller)*
Objects that mediate between boundaries and entities. These serve as the glue between boundary elements and entity elements, implementing the logic required to manage the various elements and their interactions. It is important to understand that you may decide to implement controllers within your design as something other than objects – many controllers are simple enough to be implemented as a method of an entity or boundary class for example.

In the diagrams we will separate our system in several sub-systems as said before and we will show one BCE diagram for each sub-system.

2.4.1 Sign up and sign in

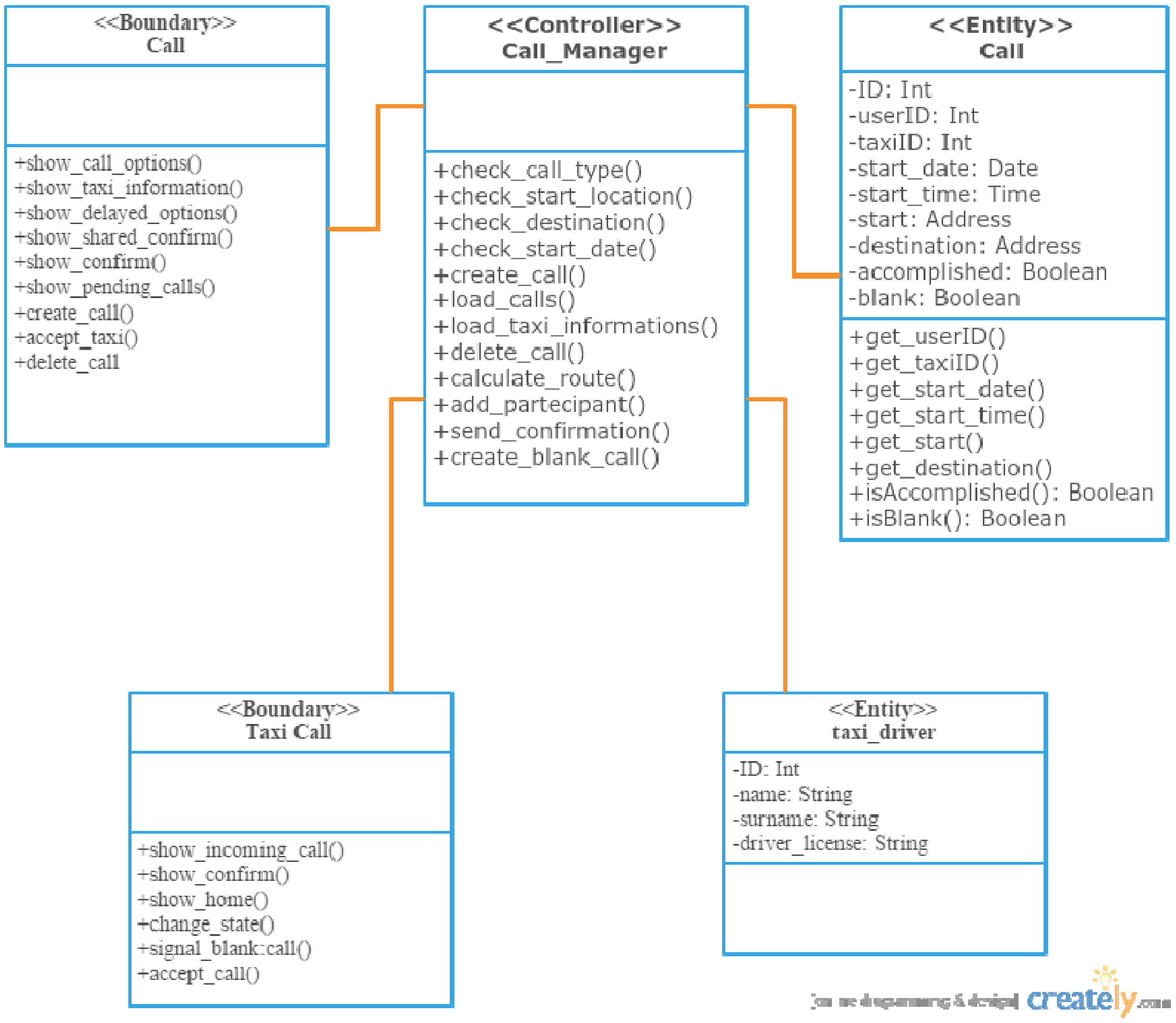
The subsystem that controls the log-in and sign-up operation is extremely simple, because his only function is to verify the access data in the first case, and to create a new user in the database in the second case.
The boundary is the same for both operations: we decided to include the button fort 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 information: 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.



<<Boundary>>
Login
+show_login()
+show_login_error()
+show_taxi_login()
+sign_in()
+sign_up()
<<Controller>>
Login Manager
+load_user()
+check_login_data(): boolean
+create_user()
+load_taxy_driver()
<<Entity>>
Taxi_driver
-ID: Int
-name: String
-surname: String
-licenseNumber: String
+getID()
+getName()
+getSurname()
+getLicenseNumber()
<<Entity>>
User
-ID: Int
-name: String
-surname: String: String
-telephone-number: String
-feedback:Float
-locations: Location[]
-mail: String
-password: String
+get_name()
+get_surname()
+get_tel.()
+get_mail()
+get_locations()
+search_location(String): location

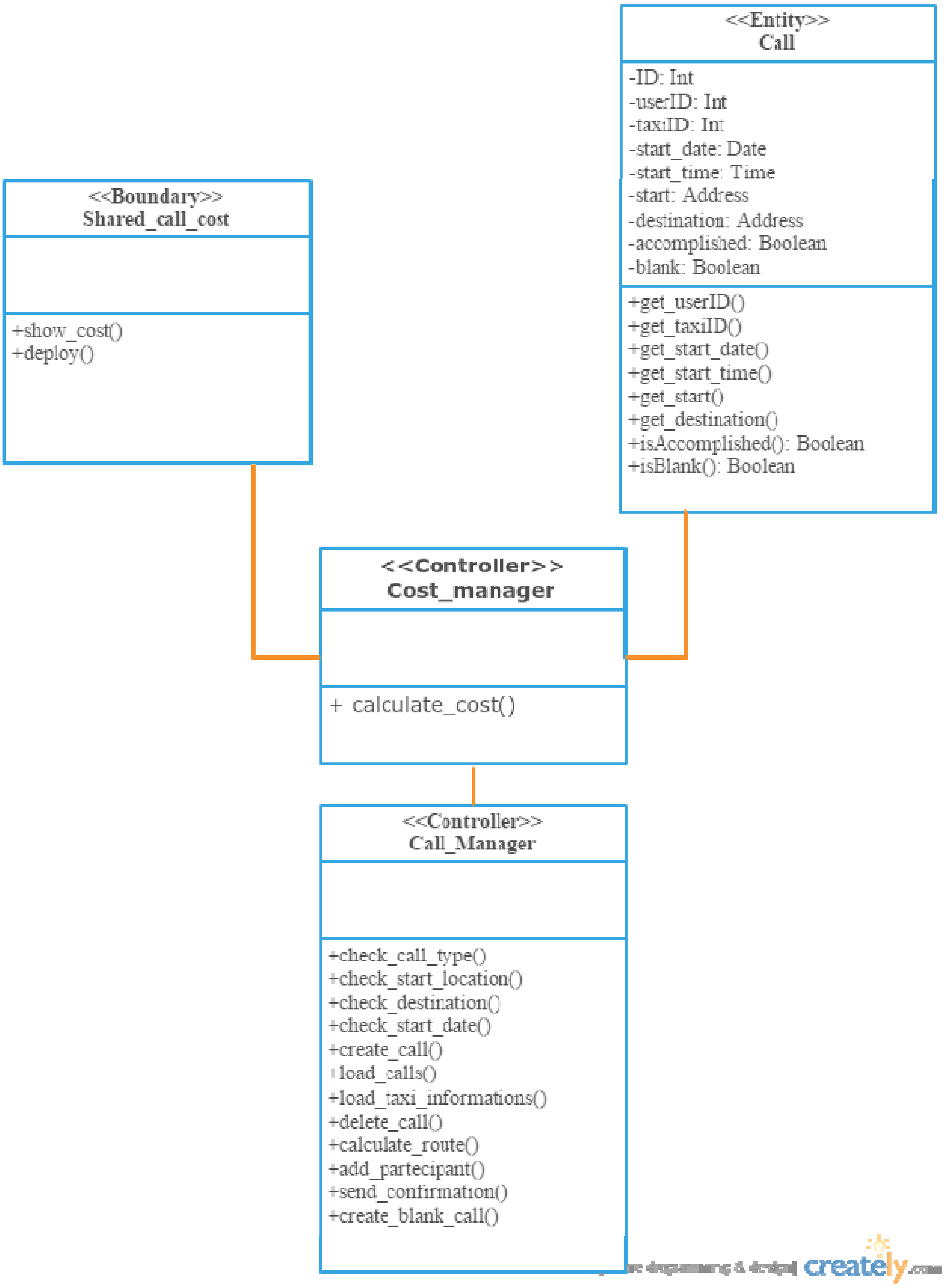
2.4.2 Make a call

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.


<<Boundary>>
Call
+show_call_options()
+show_taxi_information()
+show_delayed_options()
+show_shared_confirm()
+show_confirm()
+show_pending_calls()
+create_call()
+accept_taxi()
+delete_call
<<Controller>>
Call_Manager
+check_call_type()
+check_start_location()
+check_destination()
+check_start_date()
+create_call()
+load_calls()
+load_taxi_informations()
+delete_call()
+calculate_route()
+add_partecipant()
+send_confirmation()
+create_blank_call()
<<Entity>>
Call
-ID: Int
-userID: Int
-taxiID: Int
-start_date: Date
-start_time: Time
-start: Address
-destination: Address
-accomplished: Boolean
-blank: Boolean
+get_userID()
+get_taxiID()
+get_start_date()
+get_start_time()
+get_start()
+get_destination()
+isAccomplished(): Boolean
+isBlank(): Boolean
<<Boundary>>
Taxi Call
+show_incoming_call()
+show_confirm()
+show_home()
+change_state()
+signal_blank:call()
+accept_call()
<<Entity>>
taxi_driver
-ID: Int
-name: String
-surname: String
-driver_license: String
creately.com

2.4.3 Calculate cost

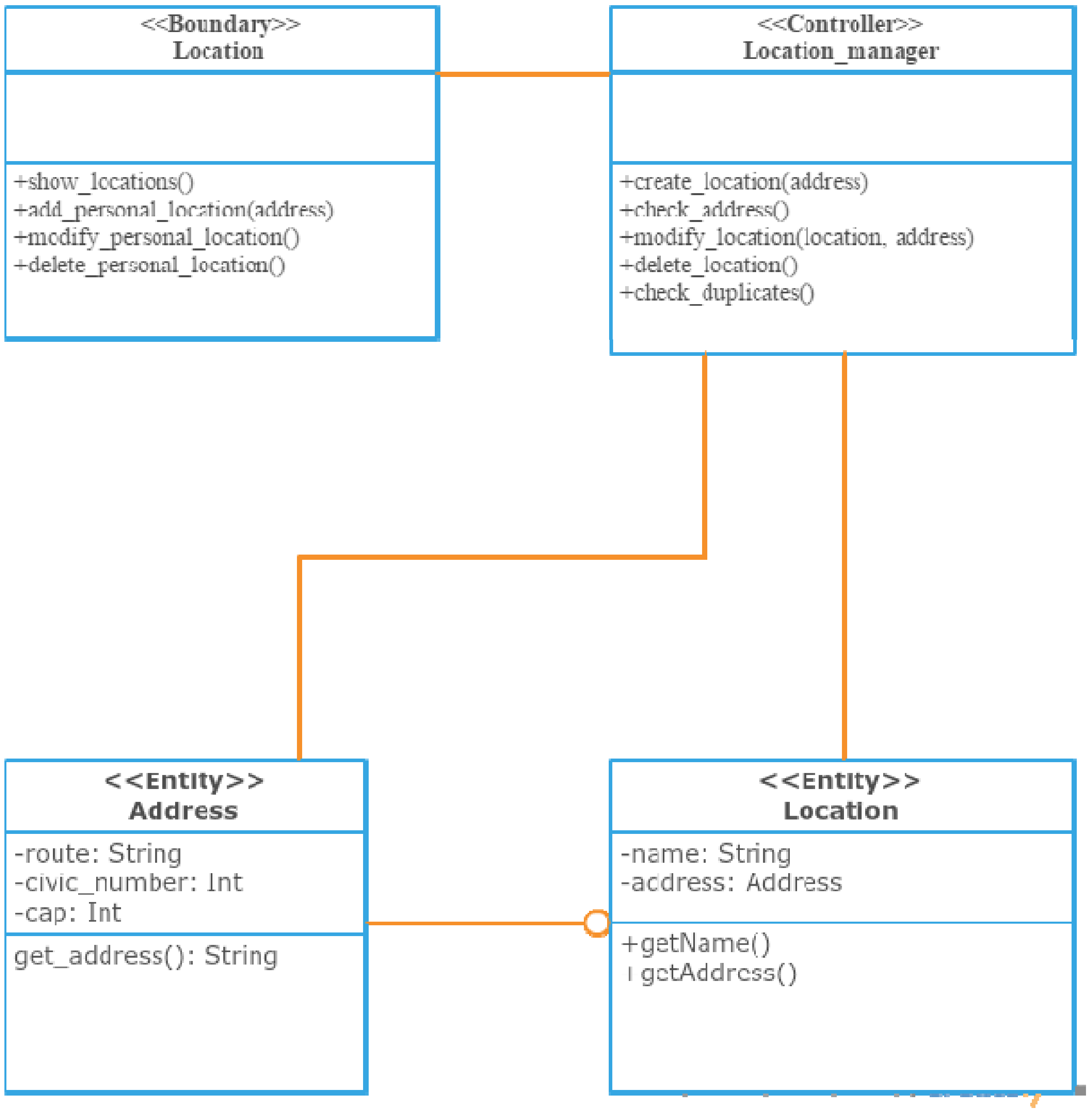
This particular subsystem has no interactions with the users: his task is just to calculate the cost of every single passenger of a shared call, when he get down from the taxi.
The boundary is visualized by the taxi driver while he is performing the ride, and he visualizes the number of passengers still 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).



<<Entity>>
Call
-ID: Int
-userID: Int
-taxiID: Int
-start_date: Date
-start_time: Time
-start: Address
-destination: Address
-accomplished: Boolean
-blank: Boolean
+get_userID()
+get_taxiID()
+get_start_date()
+get_start_time()
+get_start()
+get_destination()
+isAccomplished(): Boolean
+isBlank(): Boolean
<<Boundary>>
Shared_call_cost
+show_cost()
+deploy()
<<Controller>>
Cost_manager
+ calculate_cost()
<<Controller>>
Call_Manager
+check_call_type()
+check_start_location()
+check_destination()
+check_start_date()
+create_call()
+load_calls()
+load_taxi_informations()
+delete_call()
+calculate_route()
+add_partecipant()
+send_confirmation()
+create_blank_call()
creately

2.4.4 Location

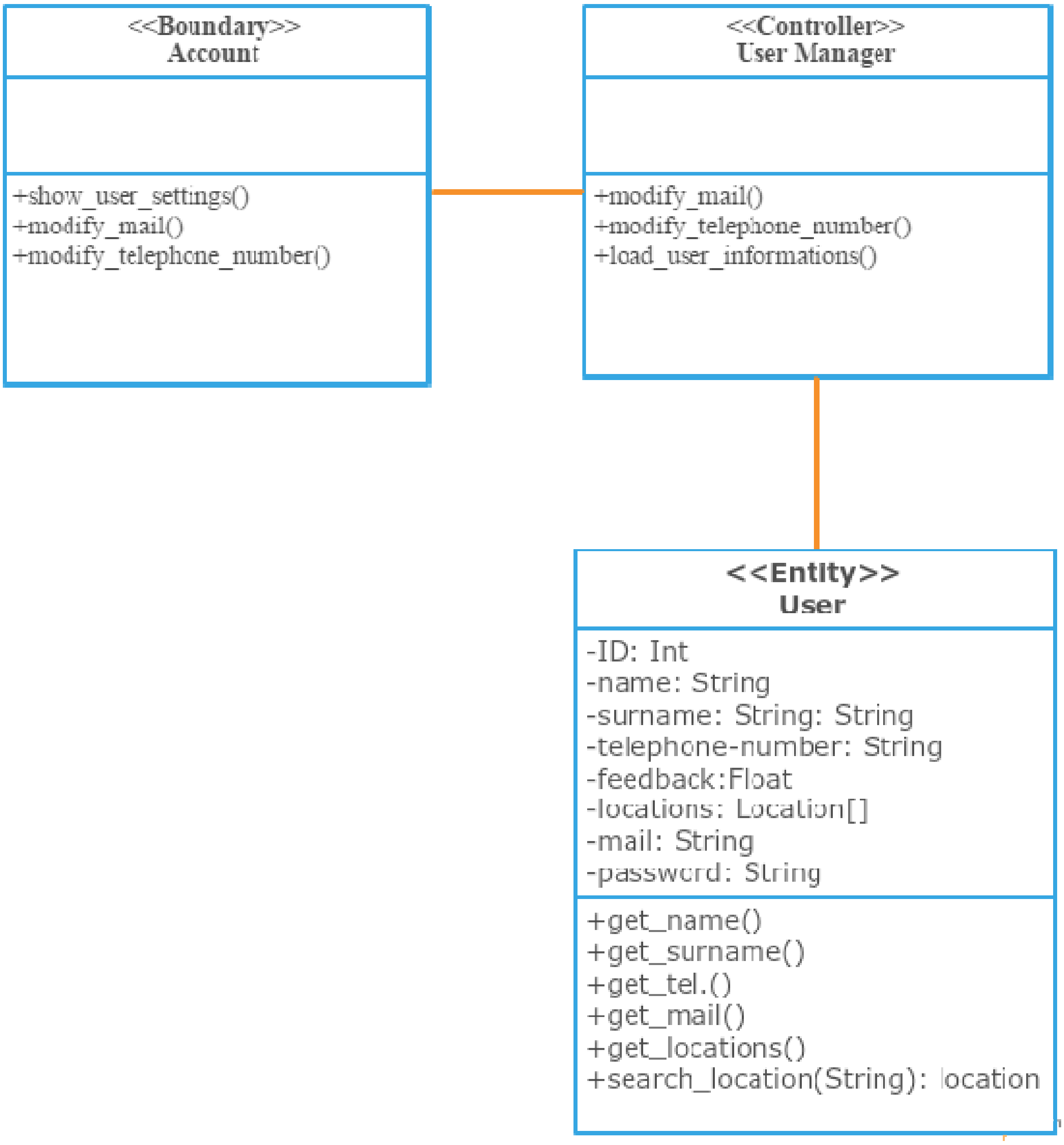
This subsystem is used for the personal locations of a user.
The Boundary is a list of all the locations already in database for the user; from this screen, he can add a new one, or modify or delete an existing one.
The controller has all the methods associated with this functions, including “check_address” that ensures that a location is valid, and “check_duplicates”, that ensures that the user has not another existing location with the same name.



<<Boundary>>
Location
+show_locations()
+add_personal_location(address)
+modify_personal_location()
+delete_personal_location()
<<Controller>>
Location_manager
+create_location(address)
+check_address()
+modify_location(location, address)
+delete_location()
+check_duplicates()
<<Entity>>
Address
-route: String
-civic_number: Int
-cap: Int
get_address(): String
<<Entity>>
Location
-name: String
-address: Address
+getName()
+getAddress()

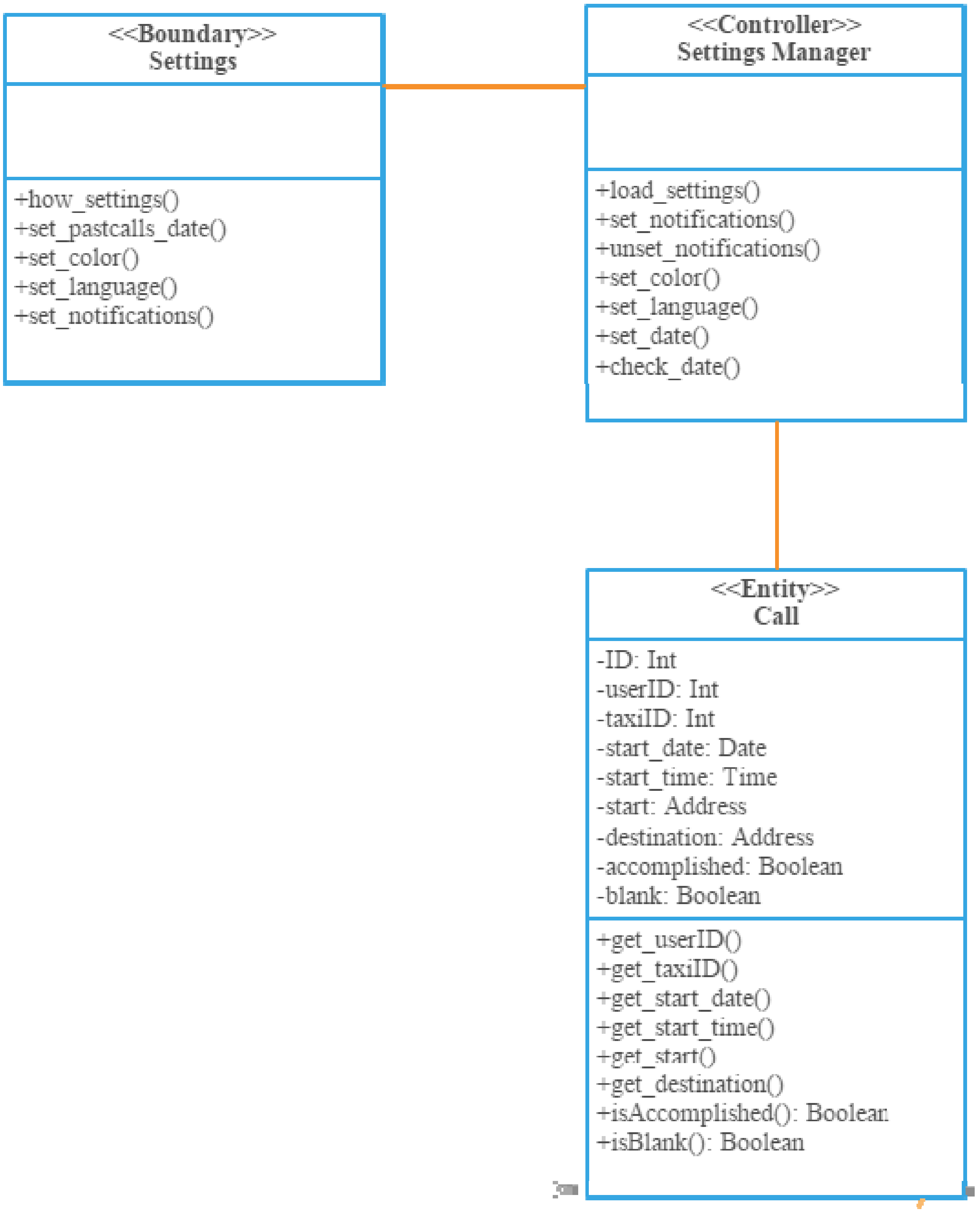
2.4.5 Account options

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.


<<Boundary>>
Account
+show_user_settings()
+modify_mail()
+modify_telephone_number()
<<Controller>>
User Manager
+modify_mail()
+modify_telephone_number()
+load_user_informations()
<<Entity>>
User
-ID: Int
-name: String
-surname: String: String
-telephone-number: String
-feedback:Float
-locations: Location[]
-mail: String
-password: String
+get_name()
+get_surname()
+get_tel.()
+get_mail()
+get_locations()
+search_location(String): location

2.4.6 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.



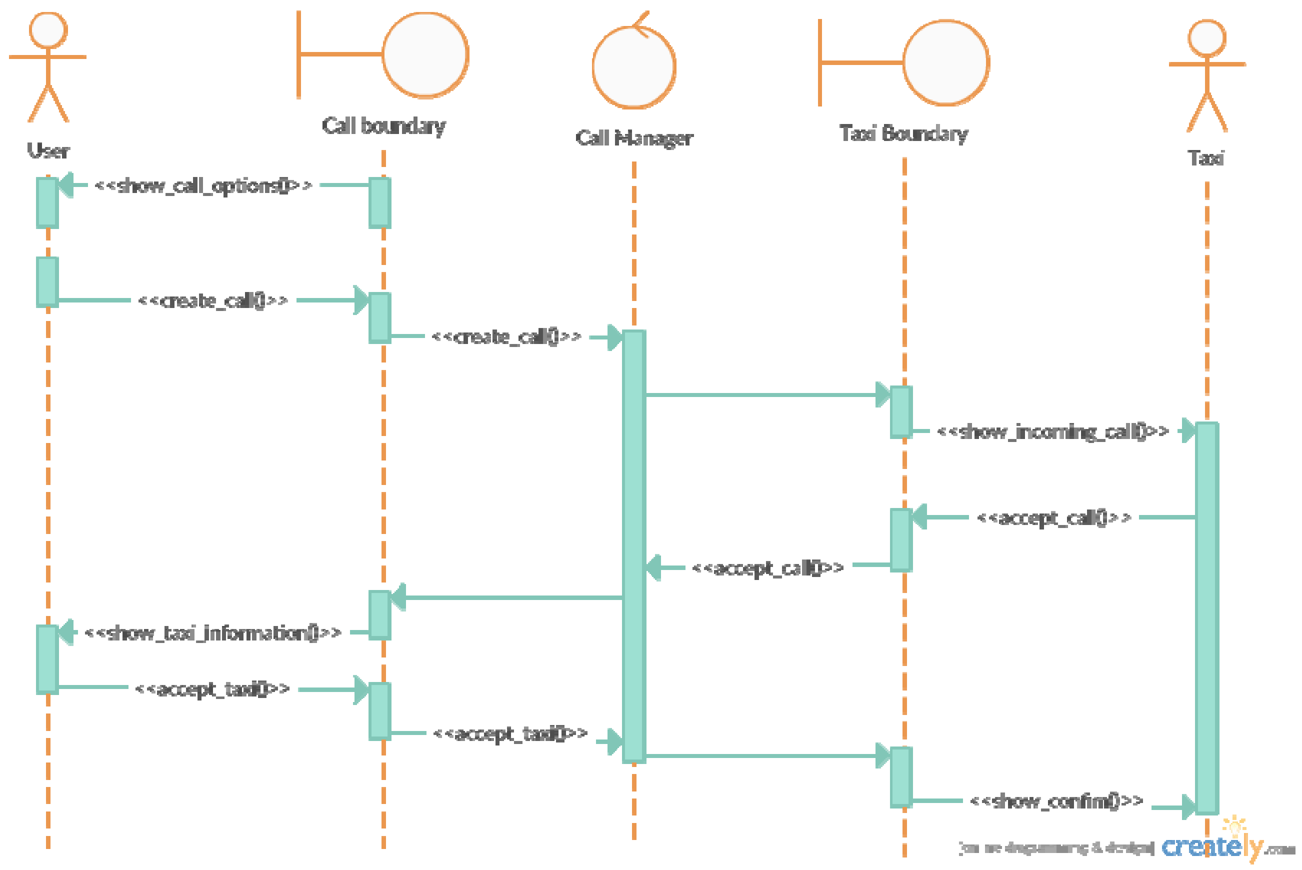
<<Boundary>>
Settings
+how_settings()
+set_pastcalls_date()
+set_color()
+set_language()
+set_notifications()
<<Controller>>
Settings Manager
+load_settings()
+set_notifications()
+unset_notifications()
+set_color()
+set_language()
+set_date()
+check_date()
<<Entity>>
Call
-ID: Int
-userID: Int
-taxiID: Int
-start_date: Date
-start_time: Time
-start: Address
-destination: Address
-accomplished: Boolean
-blank: Boolean
+get_userID()
+get_taxiID()
+get_start_date()
+get_start_time()
+get_start()
+get_destination()
+isAccomplished(): Boolean
+isBlank(): Boolean

2.5 Runtime view

In this section we will show some examples of how the users, boundary, controller, and entity will interact with each other. We tried to pick up the most explanatory examples for every BCE diagram.
We used some sequence diagram to make the simplest view of what is happening.

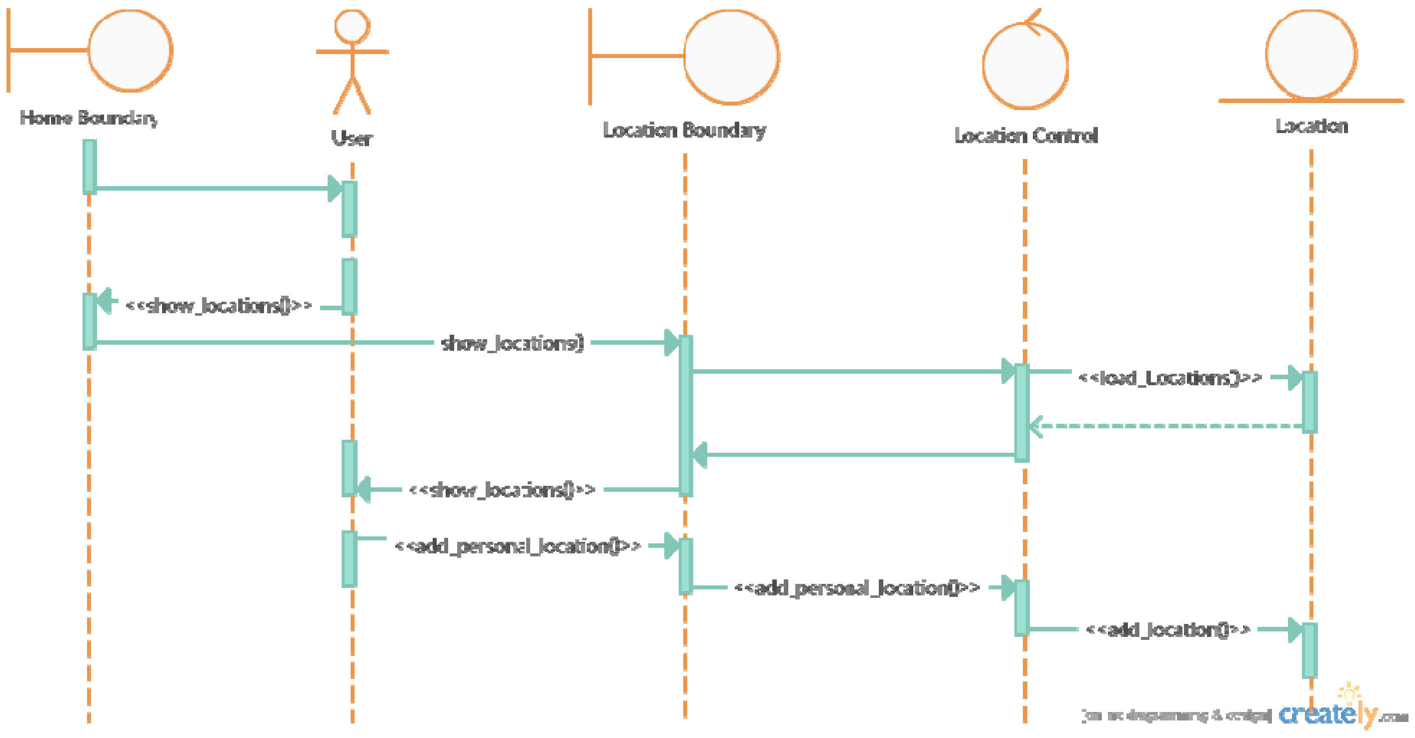
2.5.1 Create a new taxi call

The following sequence diagram represents the creation of a new call by the user in the optimal case where both user and taxi driver accept the call.



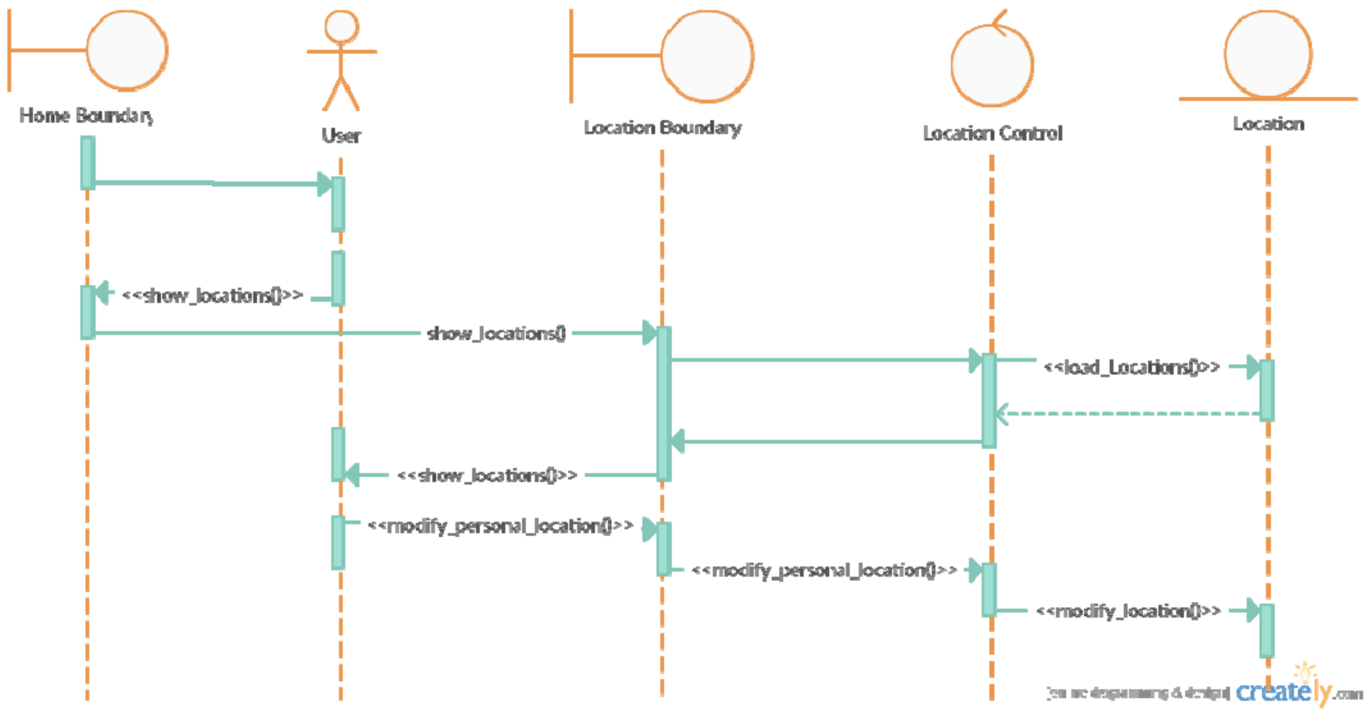
2.5.2 Create a new personal location

The following sequence diagram represents the creation of a new personal location and the system adds it to user's favorites. The user starts from the homepage of the app and the load his locations page.



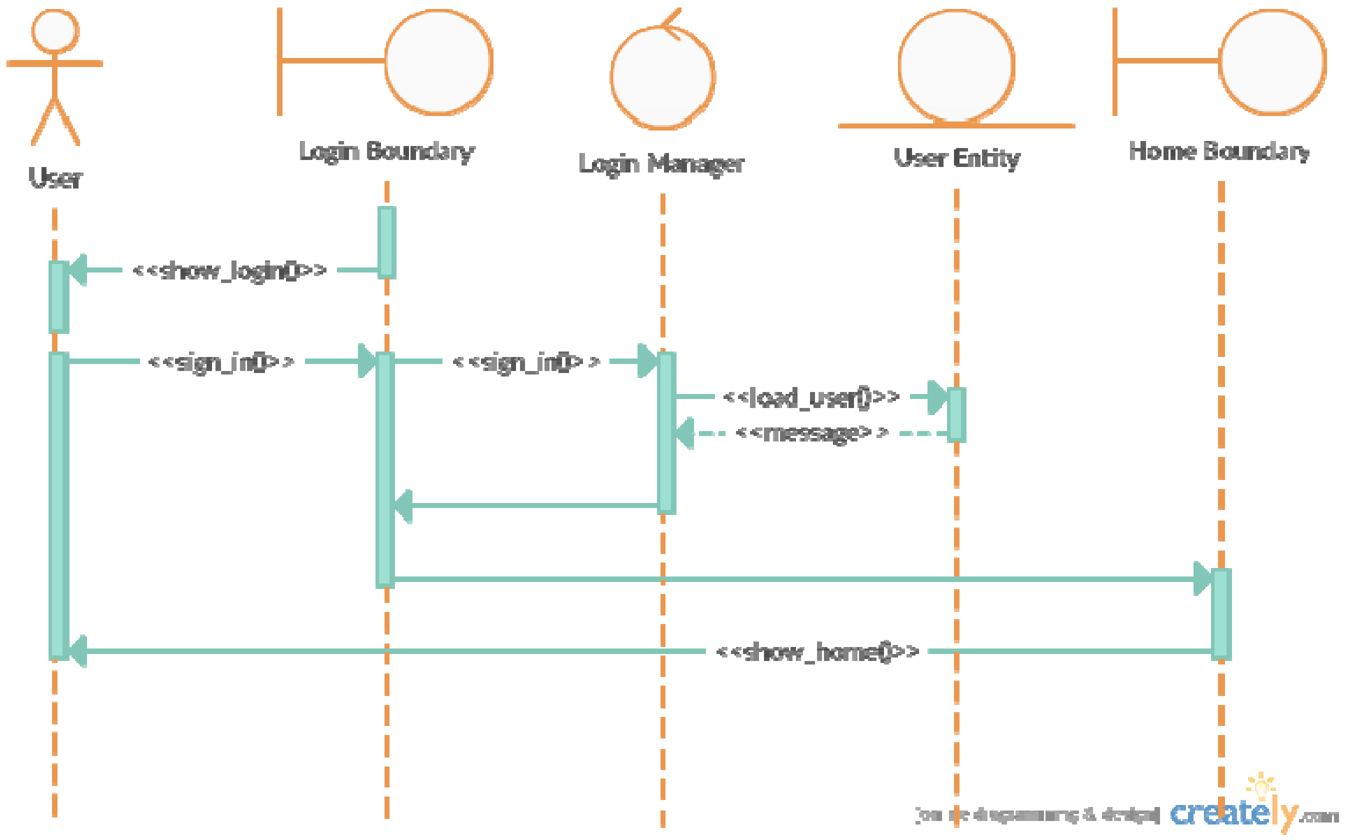
2.5.3 Modify personal location

The following sequence diagram represents the modification of a personal location. The user wants to change one (or every) field in the location (e.g. Name) and the database will be updated.



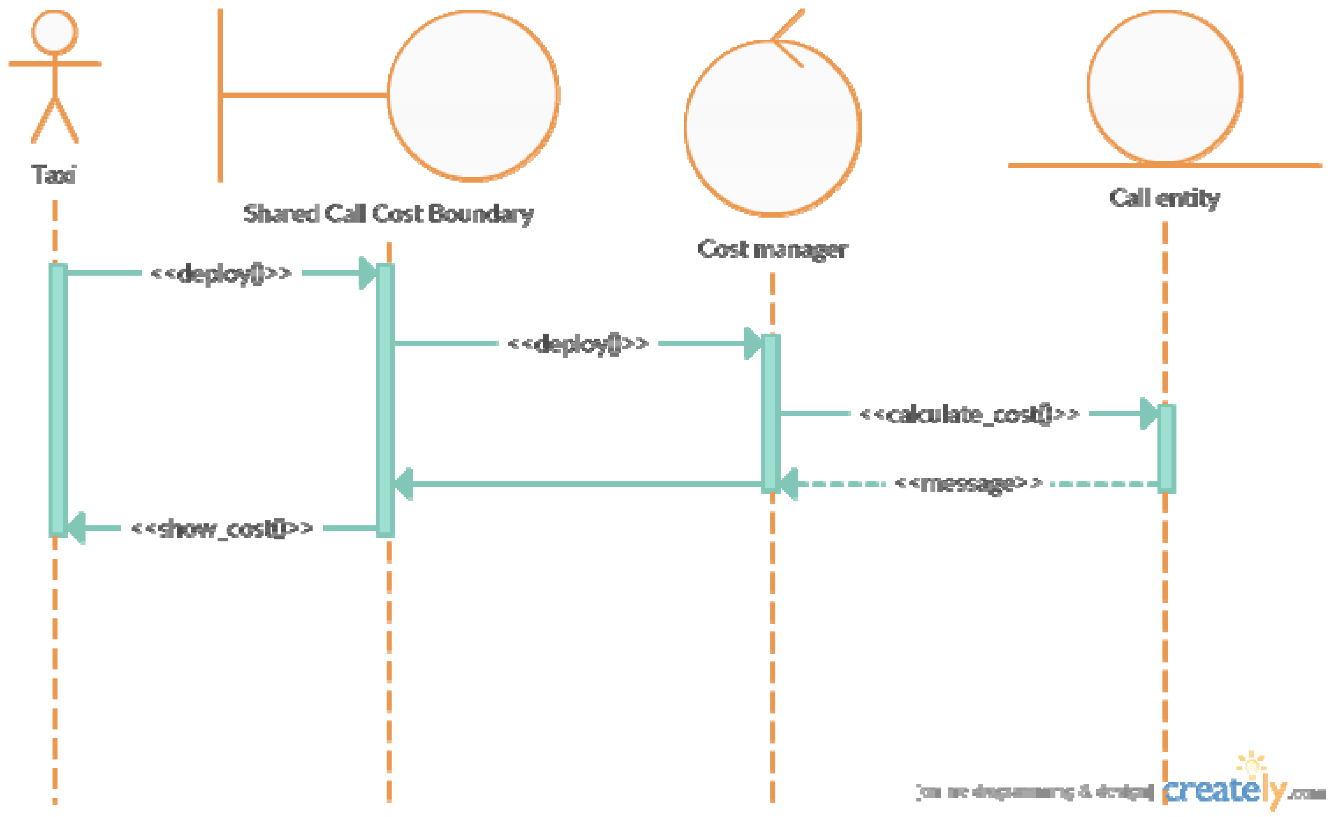
2.5.4 Log in

The following sequence diagram represents the log in of a user. The user begins from a guest page and after a successful log in it will see his homepage.



2.4.5 Shared Cost

The last sequence diagram represents the situation where, during a shared call, one if the users arrives at his destination and the app must calculate the cost of his travel. The taxi driver (here call simply taxi) signals to the system that one passenger arrived and the system will show the amount to the taxi driver.

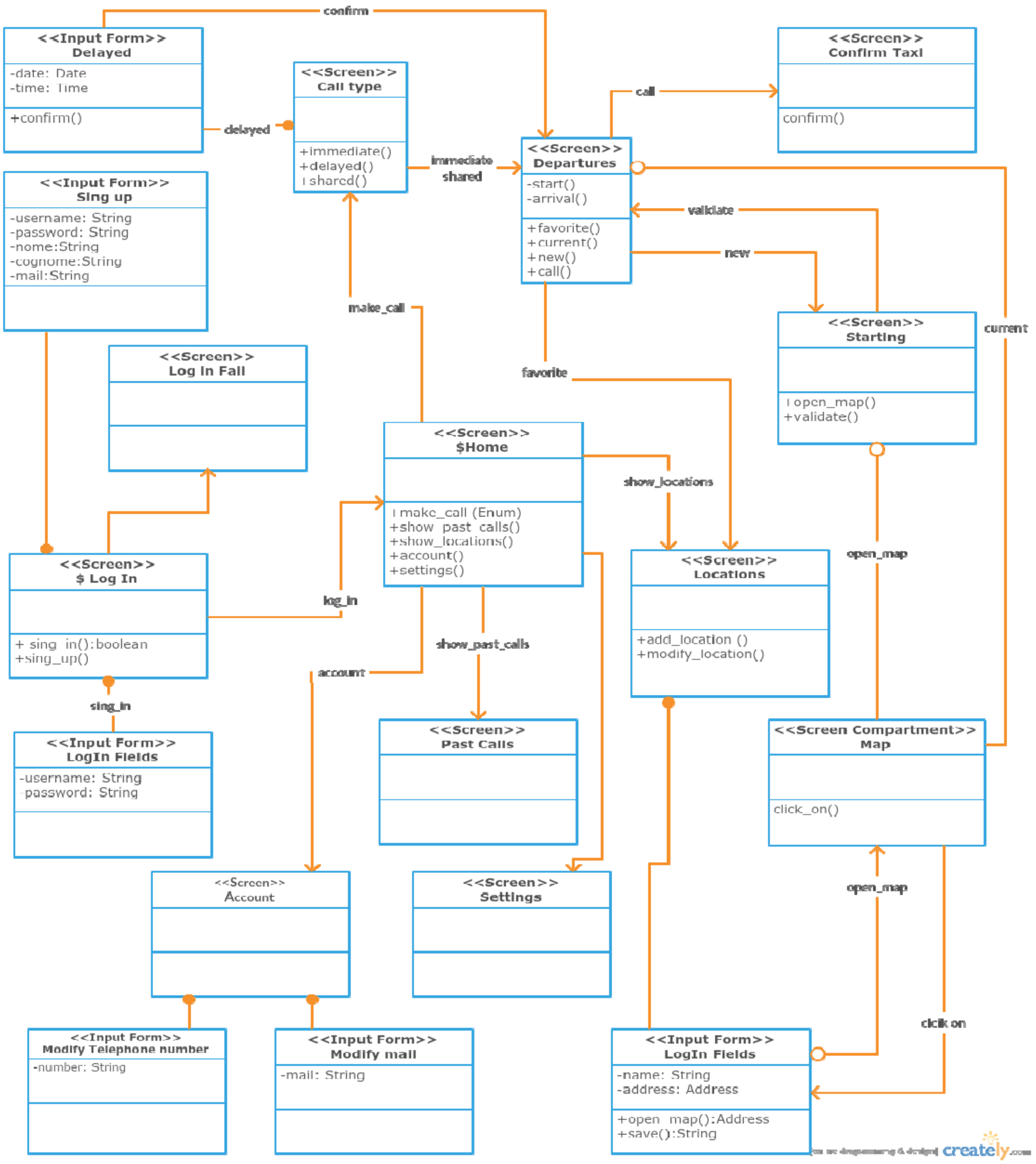


3. User interface Design

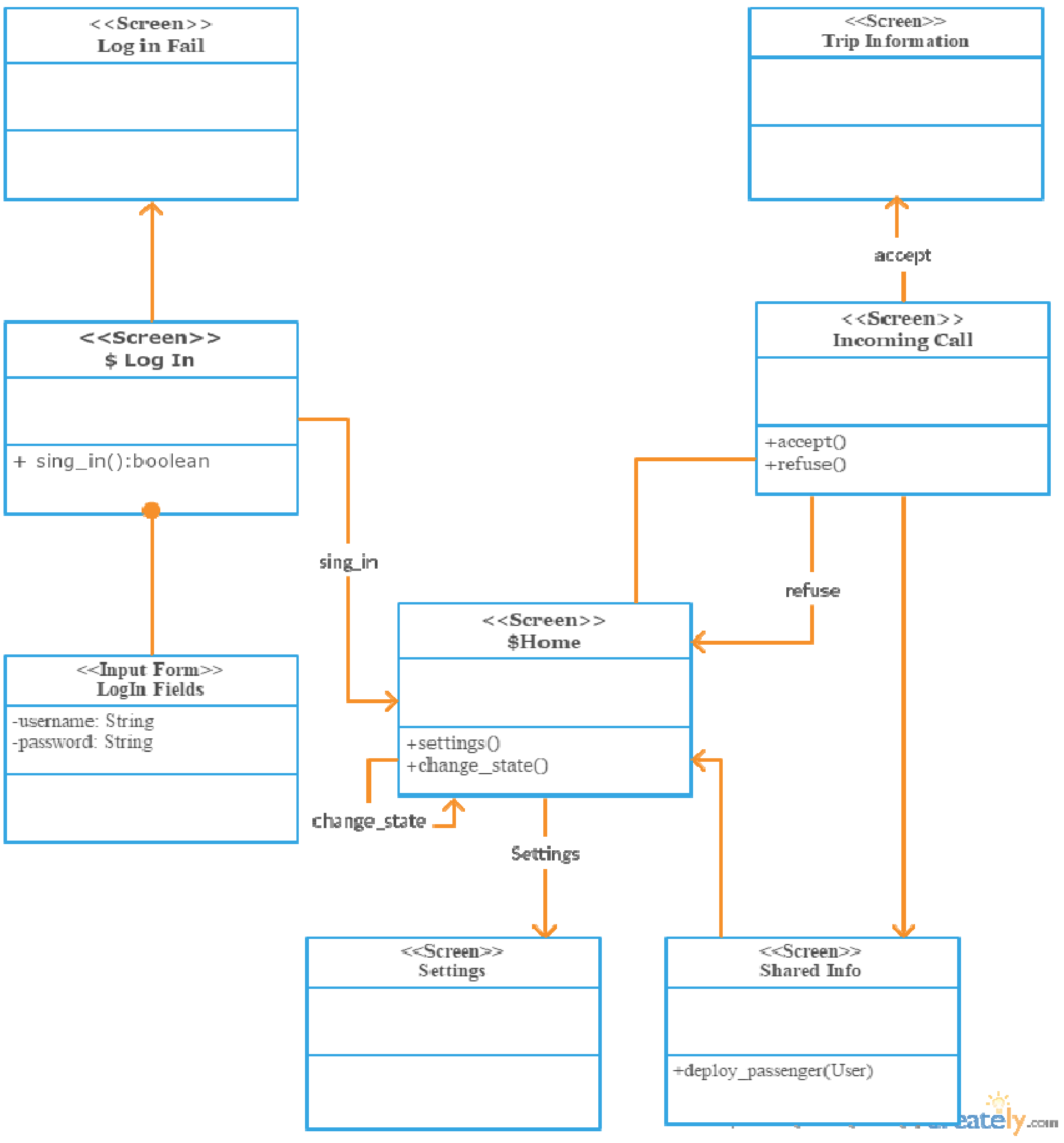
In this section we will show some models of the major pages of the app. They will simple and show only what they must have, not how they must be.
We will show also the Ux diagrams that will show how every screen will be connected and how is possible to pass from one screen to another.

3.1 UX Diagrams

3.1.1 UX Diagram User view



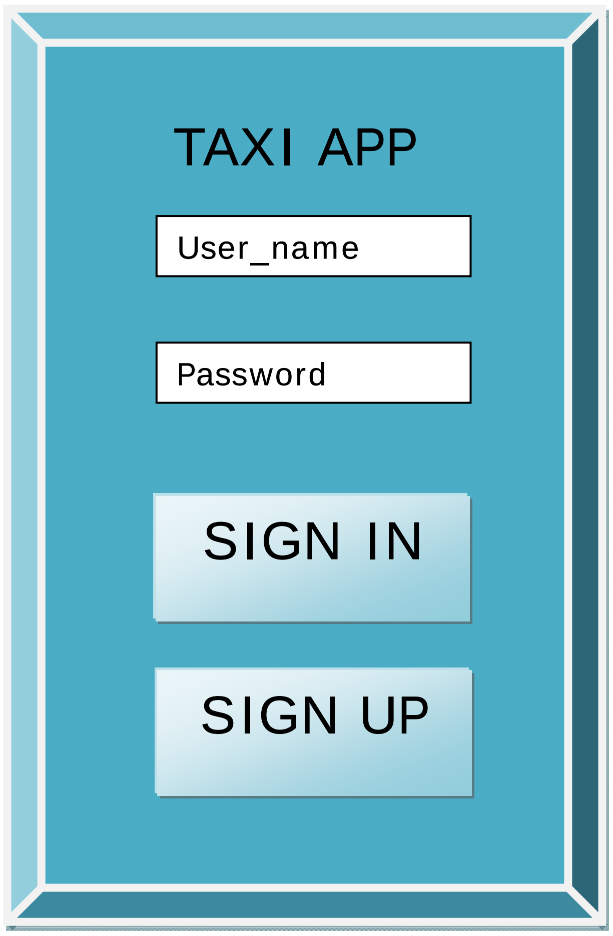
3.2.1 UX Diagram Taxi driver view



3.2 Interface

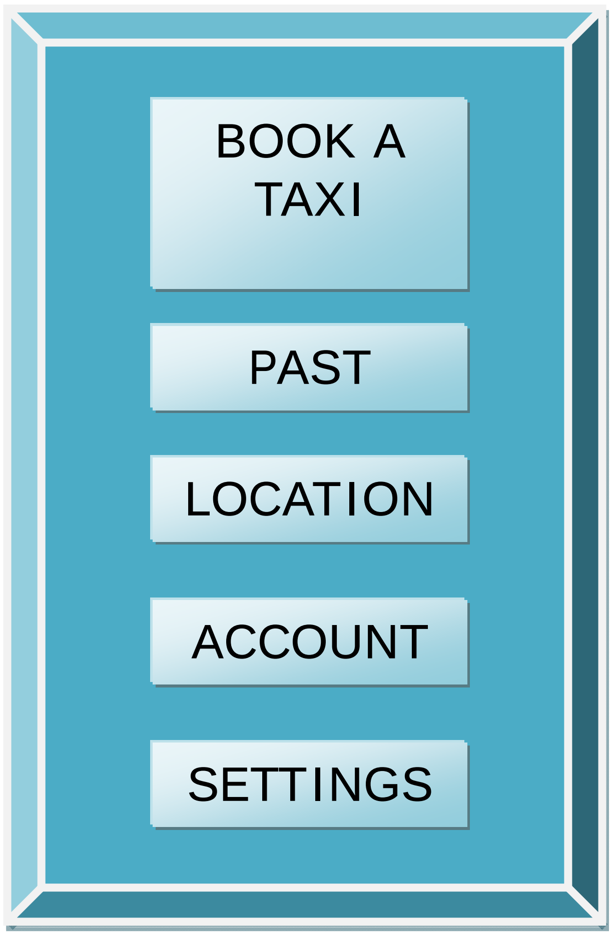
3.2.1 Login Page

This is the page visualized when the app is opened. A user can do the login if he is already signed up; otherwise he can decide to sign up and register himself.



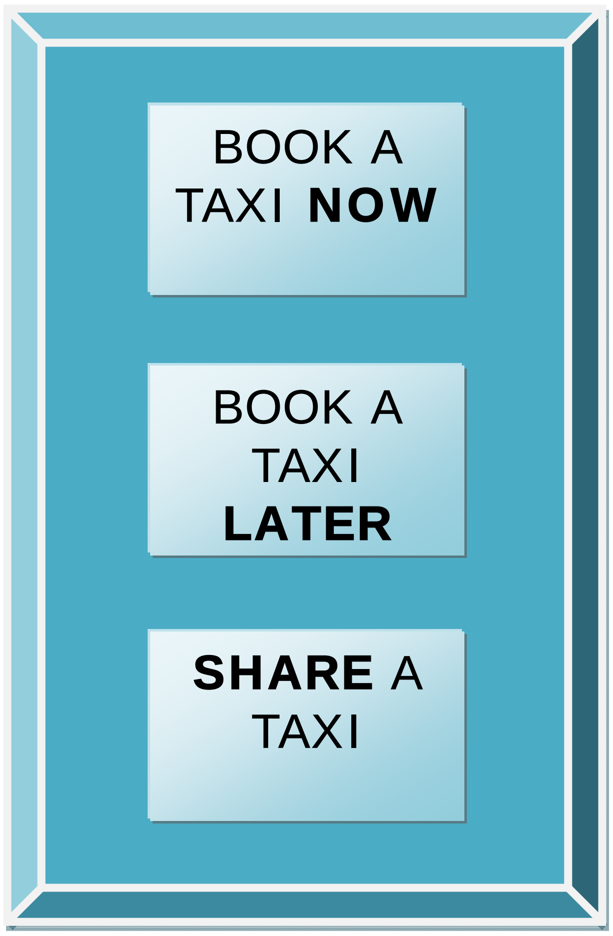
3.2.2 Homepage

This is the homepage, from where is possible to navigate into the app.



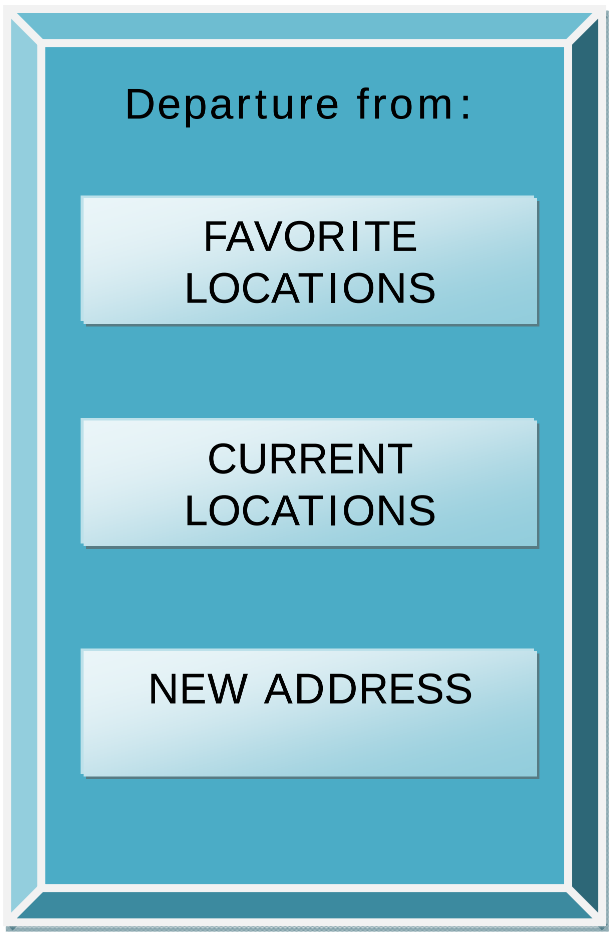
3.2.3 Call type

In this screen the user can select which type of call he wants to do



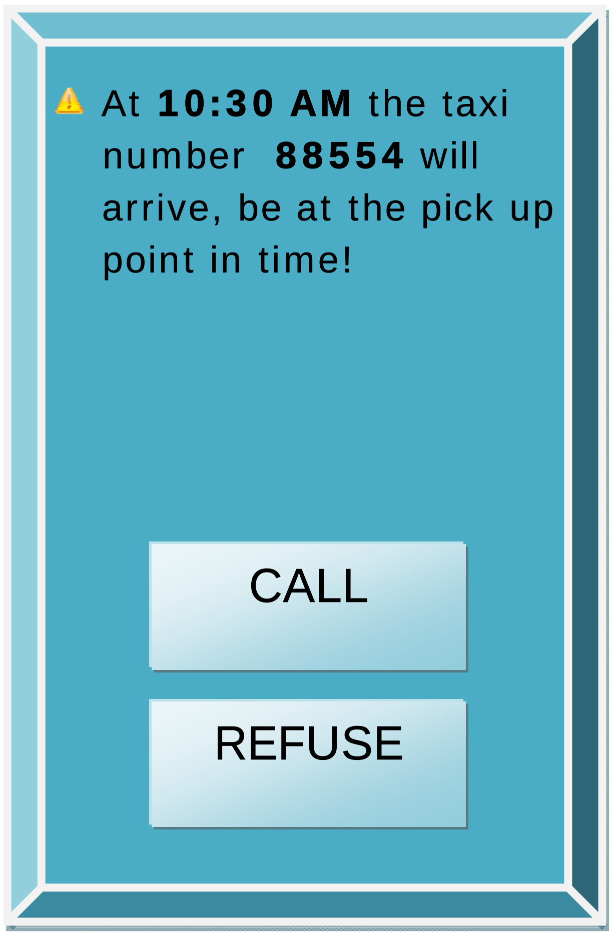
3.2.4 Departures

This is where the user can select the departure address.
Clicking on favorite locations will open the list of the users' personal locations.
Clicking on current location will open a support map (google maps). GPS connection required.
Clicking on new address will open an input form where the user can manually insert an address.



3.2.5 Confirm taxi

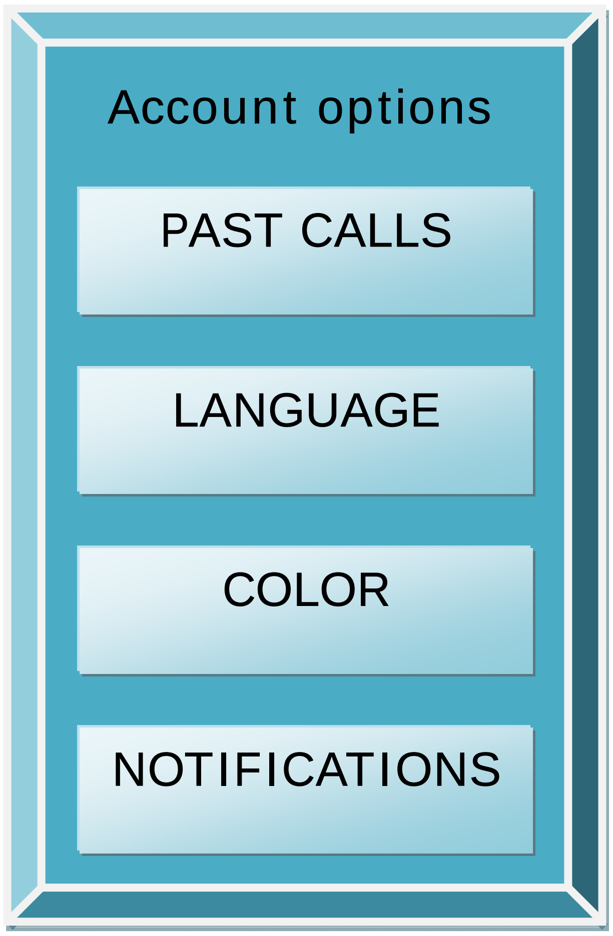
After the selection of the departure address, the system will calculate the estimated arrival time of the taxi who have accepted the call, and communicate it to the user. Now he can accept or refuse from this screen.



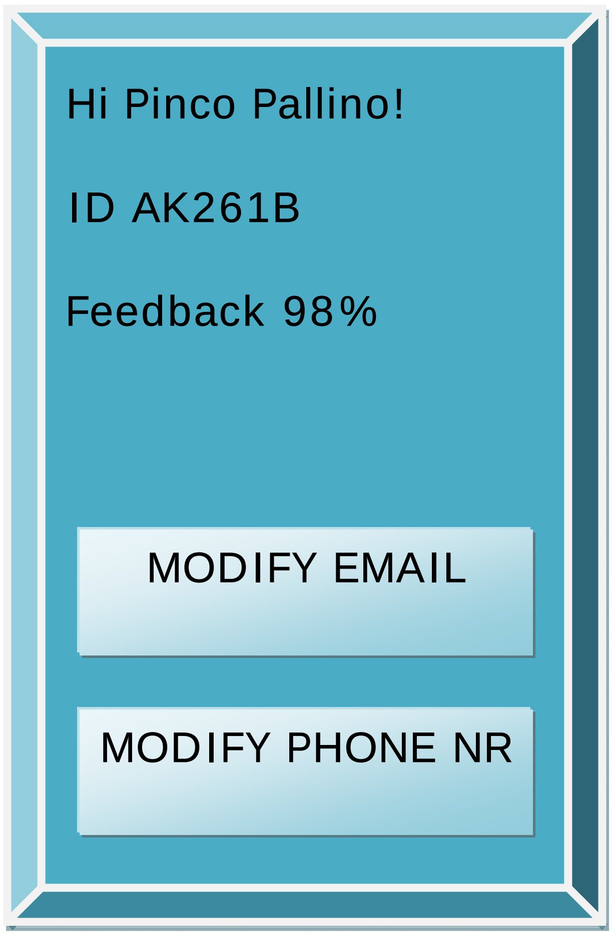
At **10:30 AM** the taxi number **88554** will arrive, be at the pick up point in time!

3.2.6 Settings

This is the homepage, from where is possible to navigate into the app.



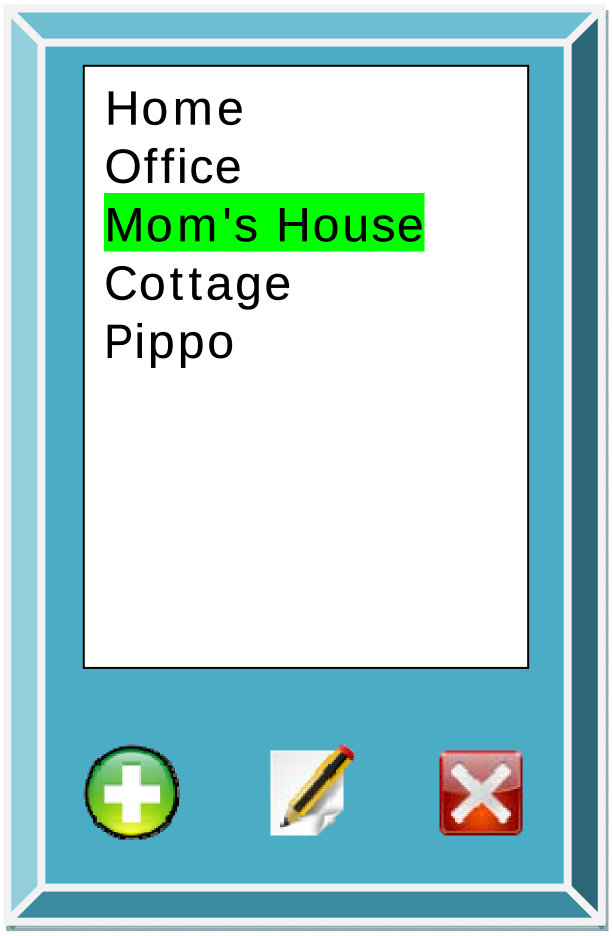
3.2.7 Account options


Hi Pinco Pallino!

ID AK261B

Feedback 98%

3.2.8 Locations



Home
Office
Mom's House
Cottage
Pippo