School of Science and Engineering

CARPOOLING APPLICATION: KwiGo

Capstone Design

April 2015

Reda Aissaoui

Supervised by: Omar Iraqui Houssaini

ACKNOWLEDGMENTS

I would like to express my gratitude to my supervisor Mr. Iraqui Houssaini, whose expertise, patience and understanding, added considerably to my capstone project experience. His knowledge of internet technologies helped me develop a professional application that respects the enterprise-class applications principles. Knowledge about this kind of applications is really useful in job place and it represents a great asset for my career.

This is also the occasion to thank my family for all the support and love they provided me to make the best of this experience. It wouldn't have been possible without their encouragements and motivation. I would like also to thank all my friends that contributed to this work through their support and by being always available when I needed them the most. A special thanks goes to Shihab, Zakaria and Zakaria.

ABSTRACT

Carpooling has become a practical, cheap and stress-free way to move around. This project presents the requirement, design and implementation of an enterprise-class application for carpooling following a Model-View-Control model. The added features, compared to available applications, are different kinds of trips, a check-in system and social media integration. The two kinds of trips are single trips, which are trip between two cities, and frequent trips which are the ones that commuters do every day. The check-in system enables users to check in meetings points and notify all users about that. Users can also share their activities on the application thanks to social media integration. The application is designed to be scalable, extensible, highly available and with a good performance. The server is implemented using the powerful JavaScript server LoopBack. The server exposes a REST API, for the clients to consume, and makes the application compatible with multiple platforms. For the scope of this project, one client application is developed using Android. An iOS and Web client may be an improvement to this project given that the server is compatible with these technologies as previously stated.

Table of Contents

I- Introduction

With the increase of environmental concerns and the congestion of roads, carpooling has gained a lot of popularity when it comes to environment-friendly and cheap ways of travelling. Carpooling is when two or more persons share a ride in one of their personal cars. Carpooling reduces pollution since we have less cars on the road. It's also economic since the travel expenses are shared among the riders. Travelling alone may be stressful, so having other persons with you on a trip reduces the stress and is also the occasion to socialize and make the trip funnier.

Finding people to share a ride with is the challenge of carpooling as it is difficult to find a person going to the same place as you at a given time. Many websites and applications has been developed to help people meet to share rides. Those applications enable users to create and share their trip and find passengers. The downside of those applications is that they are usually location limited: they are available on few languages and for a limited number of countries only. Also, most of them are not socially enabled: they do not let users to share their trips on social media like Facebook.

The purpose of this project is to develop an application that tries to overcome the disadvantages of the other available applications. The application is to be generic, which means that it may work for any carpooler in any country in the world. Also, it is socially enabled by its integration to Facebook and possibly to other social media. KwiGo, which is the name chosen for this application, is also a real-time application: any person taking part of a trip can check-in the meeting point to let the other persons now he/she has arrived to the meeting point.

The main objective of the work presented throughout this report is to develop an enterprise-class server that represents the backbone of the application and ensure its compatibility with multiple platforms including web, Android and iOS. Moreover, an example of a client Android application is developed for the users to access the services of the application from handheld devices and serve as a companion during travelling.

II- Feasibility study

The first step of my project consisted of assessing the different available carpooling applications in order to come up with requirements along with improvements. Given that Play Store is the official source of applications of Android, I used their search engine to find carpooling application by typing the keyword “carpooling”. A set of similar applications that hold the same icon showed up in the results. All those applications are from the same publisher but the difference between them is that each one is for a different country (carpooling.fr, carpooling.co.uk, …). After installing one of this set of applications and exploring the different features, I found out that it was offering trips between two cities along with frequent trips. The only disadvantage is that the application works only in France and in order to have access to other countries you have to download a separate application. The other apps were similar to the one previously stated. Another app called Carma offered to make the payments between the passengers and drier goes through the application.

Another set of applications offer different kind of trip: the one that make parents or tutors that do a trip frequently to take kids to school, clubs or sports meet so that they take turn. These applications are only available through web and not as a mobile application. An example of this kind of applications is HopWays.

After exploring the different applications, I came up with essential features that are feasible and also some improvements that should be considered. Single trips and frequent should be implemented in order to have an application that answers the need of the market.

For the scope of this project, the plausible features to implement in order to improve what is available on the market are:

- Location independent application: the same application (no need to download a country specific app) should work everywhere in the world.
- Socially enabled: Login using Facebook, Google Plus … and share content to social media.
- Pay through the app: The payments for trips goes through the application.
- Map picker: for picking the meeting points.
- Payment system: The payment can go through the application, this is difficult to implement given the complexity of this system and its legal implication

III- User and system requirement document

1) Project description

The following section contains the user and system requirements for the carpooling application. The application is a meeting point for carpoolers, both drivers and passengers. Users can share and find rides. The application will be divided into two main parts. The first one is for intercity trips where users can post their trips and register for trips created by other users. In addition to that, a check in system is available to notify the users when the driver or the passenger reaches the meeting point. The other part is for frequent trips. Frequent trips are trips that occur on a weekly basis. A person who commutes to work, for example, may be interested in creating a frequent trip to find other passengers to ride with. Given the fact that the application should be socially enabled, the user should be prompted to share his trips on social media. The access of the application is only granted to authorized users.

a) Users

The users of the application are travelers and commuters who want to go from one place to another or users that are driving a trip and want to find passengers. Users can act as both passengers and drivers while using an application. The users use their social media accounts in order to log in the application. Any user of the application can act as:

- **A driver** is any person that owns a car and wants to go from one place to another and publishes his trip on the application in order to find passengers to share the ride with.
- **A passenger** is any person that doesn't own a car and wants to join a driver in a trip he posted and agrees to all the conditions specified (price and general behavior).

b) Dictionary

In order to avoid ambiguities and to facilitate a good comprehension of this report, following is the frequent used terms.

TERM	DEFINITION
DRIVER	Any person that owns a car and wants to go from one place to another and publishes his trip on the application.
PASSENGER	Any person that doesn't own a car and wants to join a driver in a trip he posted and agrees to all the condition specified (price and general behavior).
REGULAR TRIP	A onetime long distance planned travel between two point (usually cities) with a defined departure time and price.
FREQUENT TRIP	A short to medium distance frequent (daily/weekly) between a neighborhood and a workplace, school or other point of interest.

c) Dependencies

The application will highly depend on the geo-localization and mapping system of Google. This will be used for showing itineraries and maps. Also, GPS data will be processed using Google Maps. Third-party authentication systems are also being used for logging users. In this case, Facebook Authentication is used to verify the identity of users.

1) Functional requirements

a) General application requirements

a.1) Login

Since all the operations that can be done using the application requires both the driver and passenger to be logged in, they can use the login forms of either Google Plus or Facebook. For this matter, the user is prompted to connect the app to his account and then proceed for sign in/up.

After the user authorizes the application to access his social media account, the server retrieves his info. If he has never logged to the application before, a new account is created for him.

a.2) Modify profile information

All users can modify their profile information. The profile information contain: name, phone number, email, type/color of car if any. The user can easily edit these information in order to be contacted and recognized.

a.3) Social media sharing

In order to attract more users to the application and help users find passengers, users should be able to share their activity on the application on social media. A suggestion for sharing trips' creation, trips' registration or check in should pop-up whenever those previous actions are performed. The sharing should be authorized by the users and not done automatically by the application in order not to spam the users' account and gain the users' confidence.

a.3) Rate driver/passenger

Both the driver and passenger can rate each other in other to gain reputation. The importance of the rating is to encourage users to be helpful and nice during the trip so that they gain popularity in the application. It is also a way to ensure users of who can be trusted or not. The ratings represent a relative guarantee for the users to trust each other.

b) Regular trips

b.1) Create new regular trip

The driver can create a new trip to be displayed when passengers search for trips. The application will prompt the driver or information of the regular trip which consists of destination, origin, meeting point (which can be pointed in a map), departure time/date , estimated arrival time and traveling preferences (number of free spots, price, size of bags, smoking/non-smoking, pets, stops …). After providing this information, the user publishes it in order to find passengers. Upon the creation of the trip, a user can share the trip he just created in social media to find passengers to drive with.

b.2) Search for regular trips and reservation

When a passenger needs to find a driver for a destination, he can use a search form which asks for destination, origin, departure date/time. He can also specify the travelling preferences. When he finds a suitable trip, he can reserve a spot easily in by taping a button which will send a notification to the driver telling him that a passenger has reserved.

b.3) Check-in trip

Whenever the driver or passenger arrive to the meeting point at the time agreed upon, he can check-in the meeting point in order to notify the other user and to show his punctuality. The application will use the devices GPS in order to make sure that the users are in the meeting point. When somebody checks in, a notification is sent to all the carpoolers saying that somebody is in the meeting point.

c) Frequent trips

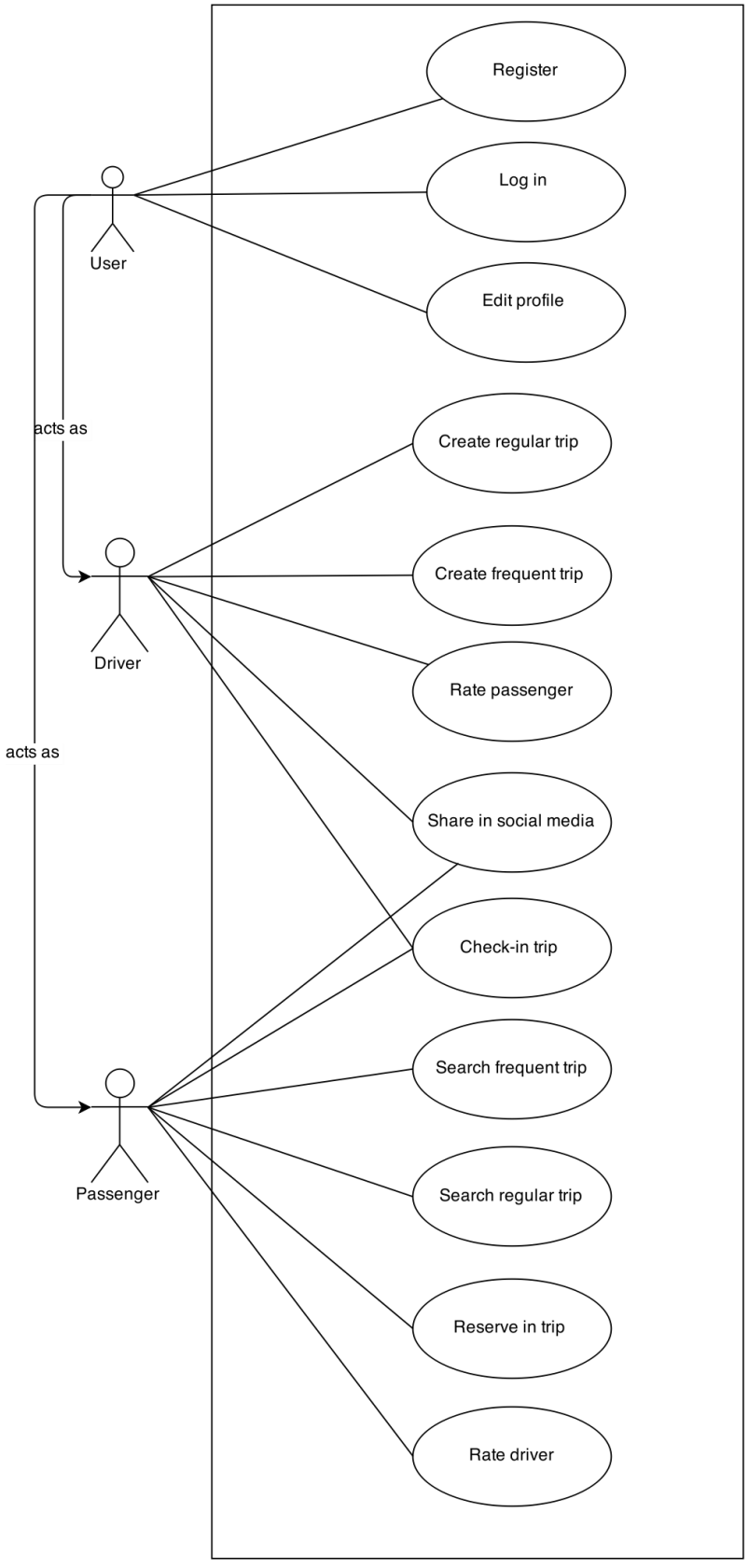
c.1) Add frequent trip

The driver can create a frequent trip where they show the origin and destination, departure and return times in addition to the frequency (daily and weekly).

c.2) Search frequent trips

A passenger can search for a frequent that he can join. The passenger should specify the departing neighborhood, destination, departure times and frequency. The application will try to match it with the best trip. If the passenger is satisfied, he can register to the frequent and will be given the contact of the other members.

d) Use case diagram



2) Non-functional requirements

a) Performance

The application has to offer a very quick response time as the meeting between the driver and passengers is done through notifications. In other words, the server should be able to treat notifications and propagate them instantly. The application should handle 1000 users sending queries at the same time.

b) Scalability

The application should respond properly to a high increase of users. It should be able to handle from 10 000 users to 100 000 users. And also from 100 000 to one millions users.

c) Extensibility

The application should by extensible in order to support multiple platforms including iOS, Windows Phone and Web.

d) Availability

Since a lot of information about the trips and check in are available in the application, it has to be highly available and guarantees a good server up-time. The server should allow only 1 hour down time per year which is 99.99% up-time.

e) Privacy and Security

The application should ensure the privacy of the users including the trips they take part in, their social media accounts and their accounts. The login system should also be robust where only authorized users can post and edit their own information.

f) Maintainability

Since the application may be developed in the future by adding other features, it should be easily maintainable.

II) Technology enablers

The choice of technology enablers that will be used for the development of the application is essential for its success. The technology enablers should provide a suitable way to fulfill the requirements stated before. Principles of enterprise class applications should be kept in mind during the choice of these technologies. The two main ones are that there is no best technology but instead suitable ones and that we shouldn't reinvent the wheel which means that we should take advantage on what was already implemented and offered to the community. Throughout this document, I will present a comparison between different technologies and the chosen one along with the different reason of the choice for the application server, the client-side and other components.

1) Server technologies

a) Application server

	**Java** is the most used language for application servers. The applications are installed through WAR and EAR files that are deployed on the servers. The application runs on the Java Virtual Machine which makes the applications runnable on different operating system. GlassFish and JBoss are examples of application servers that uses Java
	**.NET** is an application server technology that was developed by Microsoft that runs in Windows.
	**JavaScript** is a dynamic programming language that is mainly used for client-side scripts and web development. Lately, programmers have been able to implement standalone application servers using this technology. A famous JavaScript solution is Node.JS

Technologies vs. Requirements

In order to choose the best technology for the previously stated requirements, I will see what each technology has to offer regarding some important requirements. Given that the application should follow the enterprise class applications principles, for this comparison I'll consider Java EE platform instead if normal Java.

Scalability

Java (JEE)	Java EE achieve scalability through the distribution of components across multiple servers. The web server and application server may be running on different machines. It scales to large number of clients by having a thread for each user which may create concurrency problems.
Microsoft .Net	Like Java, .Net supports physical scalability by replicating servers.
Node.js	Node.js, which is event driven, can delegate processes (especially I/O operations) to other components which gives it time to process other requests. It is also single threaded so it doesn't take advantage of high performance CPUs that's why scaling-out is preferred when thinking about scaling.

Performance

Performance is also an important criteria when choosing a suitable web application for the application. Many studies have been conducted in order to have a performance comparison between JEE, Microsoft .Net and Node.js. The studies consist of implementing similar applications on different technologies and running them under the same computer configuration. The first study (Hamed and Kafri, 2009) concluded that Java EE performs better than .NET. The second study (Adhao and Gaikwad, 2013) states that Node.js performs 20% faster and handles more requests/second. From both studies, we can deduce that Node.js has the best performance.

Notifications

The application should able to send notification instantly to both the driver and passengers when somebody checks in. Android, for example, manages its notifications through a web service called Google Cloud Messaging. The fact that It's a web service makes it multi-platform and then compatible with all technologies presented. There is also a web service called Pushwoosh that enable to centralize the notification system for more than 19 platforms (Android, iOS, Web …).

Results: Regarding scalability JEE and Node.js both offer good solutions: they both scale to an increasing number of users. Notification system are offered as a web service which makes it multi-

platform. Node.js offers more performance, while JEE and .Net come in the second and third place respectively. All in all, the three technologies are quiet similar when it comes to scalability and notification system. Node.js has a better performance than JEE which is an advantage sum up, I will choose Node.js as an application server because it offers more performance and also because it's a new trending technology that I want to discover.

b) Data storage and persistency

SQL	**SQL** uses a relational model to store data using tables that consist of rows and columns. The different columns are created when a row is created even though the fields are to be kept empty.
NoSQL	**NoSQL** uses the documents to store data. Unlike SQL, the different attributes of an object may be created on the fly if there is a need, so space is not taken if an attribute is empty. NoSQL is used for applications that may have the need to grow rapidly through receiving a high throughput of data.

Node.js (Javascript) and NoSQL format data the same way using JSON, which improves performance because there is no need to reformat data. Reformatting will be the case if we use an SQL database since the result need to be processed to meet the JSON format.

As stated before, an application that is expected to be scalable should use a NoSQL database. NoSQL provides a flexible way to store data where the rules are stated by the programmer and not by a relational model. Also, the choice of Node.js as an application server pushes the choice of a NoSQL database as it is more supported than a traditional SQL one.

In the context of this application, I'll be using Mongo DB which is a NoSQL database that uses JSON like documents in order to store data.

2) Client side technologies

	Android is an open-source operating system developed by Google for smartphones, tablets, cars, TVs and smart watches. Today, more than 84% of smartphones in the world are using this operating system which makes it hold the biggest market share.
	iOS is an operating system developed by Apple for it smartphone.
Web	Enables the user to access the services using any web browser on any device using the HTTP protocol, HTML, Ajax, JavaScript...
Desktop	A program that runs directly on the target desktop operating system.

The application should be implemented using Android, iOS and web. Smartphone implementation is mandatory as the application is to be used during travels for both the passenger and driver. Also, it should be also available on the web as it provides a larger display and a comfortable way to browse trips. Desktop clients programs are to be omitted since the web provides a multiple operating system solution.

The application is firstly intended for handheld devices, so for the scope of this project the client side part of the application will be developed using Android as it is a smartphone operating system and it is the most used one. The application may be expanded to iOS and Web.

3) Other technologies

a) Google Maps

Google Maps is mapping application developed by Google. For this application, I'll be using its services for:

- Maps: Google Maps API for Android enables to show a map given GPS coordinates or choose a point on a map and get the GPS coordinates back. This will be useful for determining the meeting point for the trips and store them in the database. Also, Google uses a textual identifier that uniquely identifies a place. The latter is useful to unify the name of places and avoid users' confusion. For example, Ifrane will ever be identified by ChIJCxQt83LXoQ0Ro3xMQB3PU3I.
- Directions: Google Maps API provides a web service to determine the distance between two GPS coordinates, that will be used at the time of the check-in to determine if the user is close to the meeting point or not.

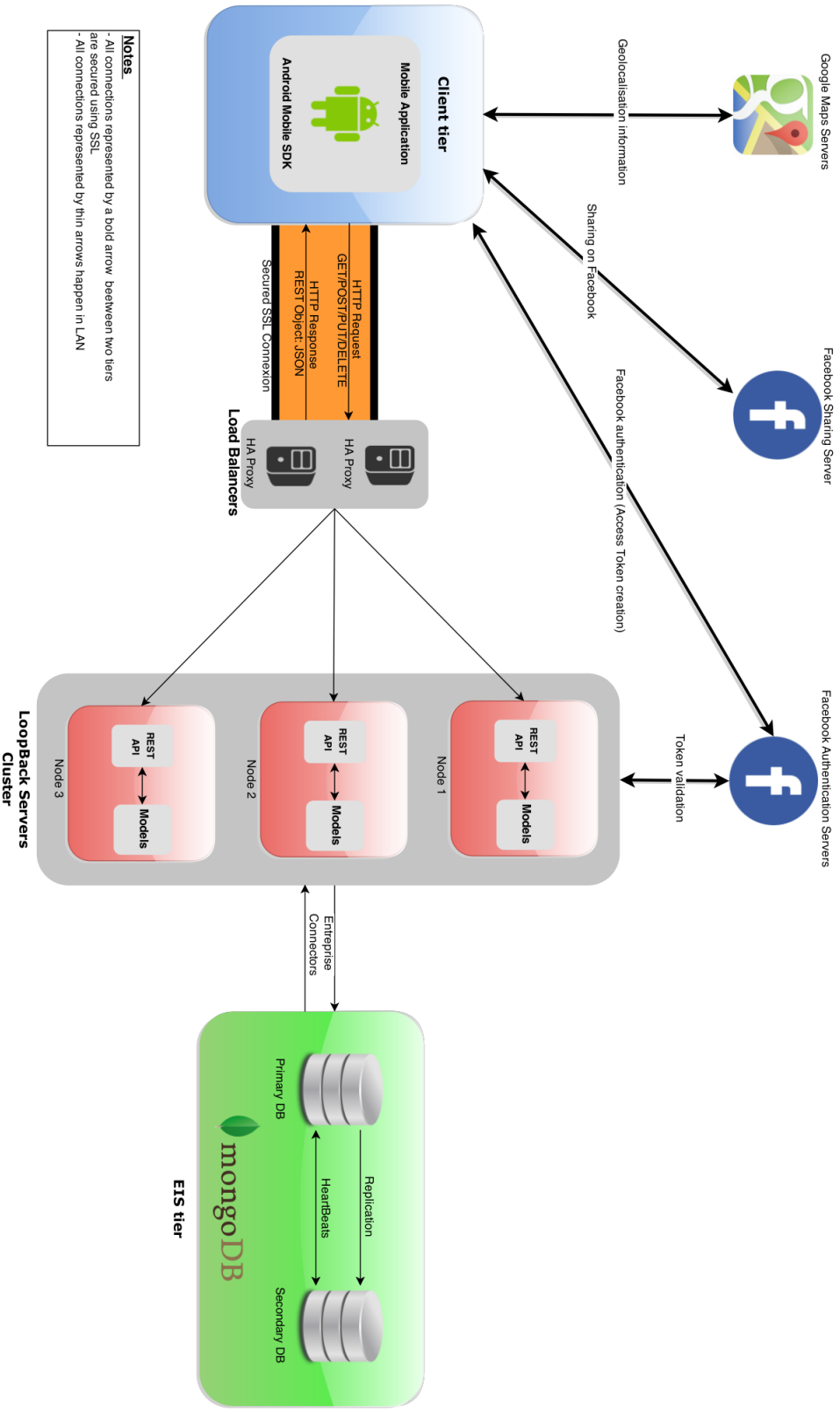
b) Facebook and Google Plus

In order for users to be able to sign in using Facebook and Google logins, I'll be using their APIs for this matter. They both enable to retrieve the user's information as soon as he authorizes the application. Content may be shared to these social networks which will ensure a presence of the application on them.

III) System design and architecture

1) System architecture

The following diagram shows the general system architecture



The application architecture has been chosen according to the previously specified requirement.

In order to ensure a high availability and increased performance, the backbone of the application is composed of a cluster of LoopBack servers. HA proxy has the responsibility of distributing the work load over the LoopBack servers' cluster. The cluster also facilitates scalability as we can add as many servers (scaling out) as we want if we have an increase of user. Actually, a second load balancer is available in case one of them encounters a failure. At the level of the Enterprise Information System tier, a primary database receive the different queries for data persistence and access. It also replicates all the data on a secondary back-up database. Both databases exchange heartbeats in order for each one to know if the other is up and working. The second database will take over the operations, in case the primary one fails. In this way we are sure that data is redundant to avoid data loss and that a database is always available and working.

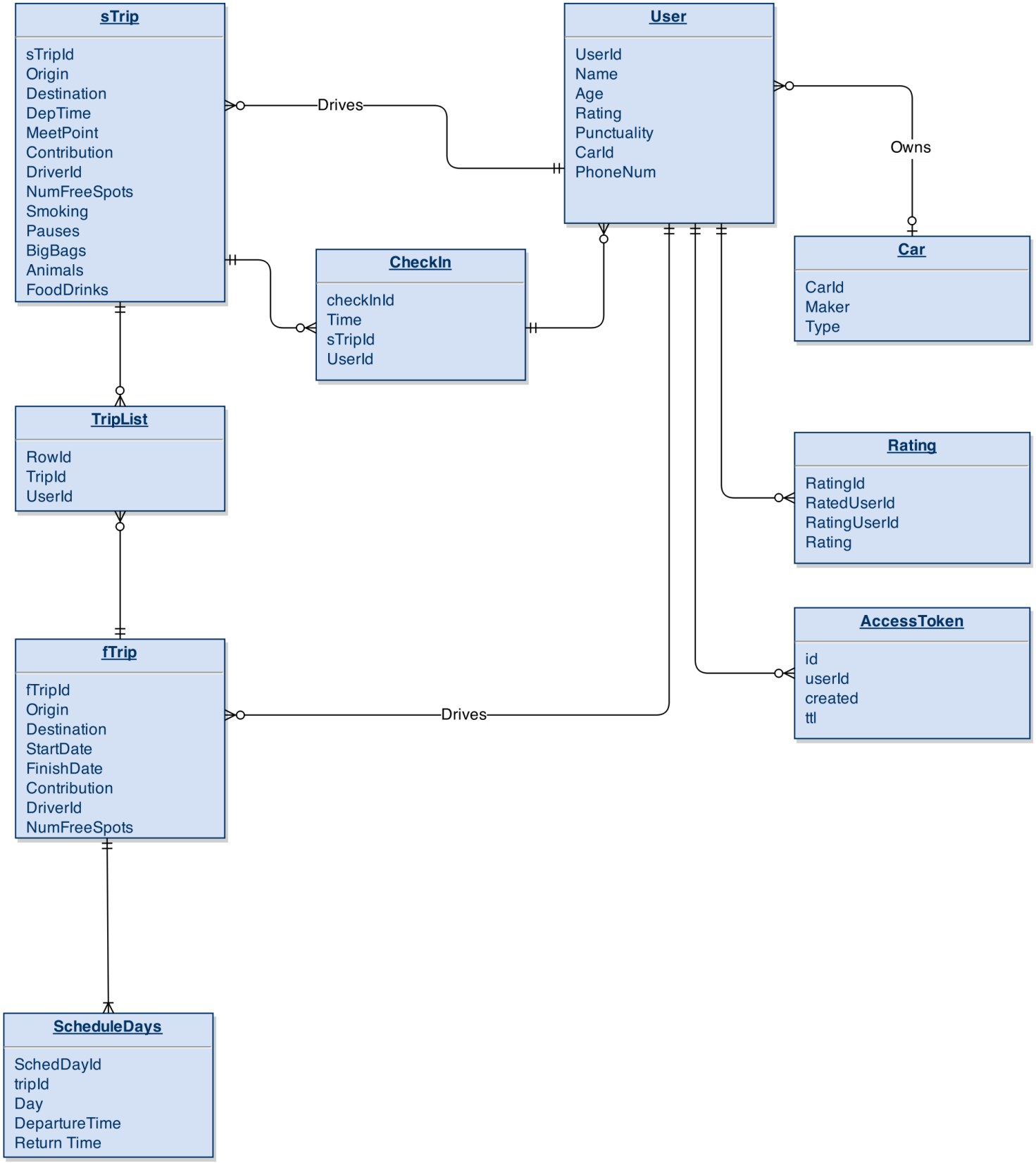
The server side exposes a RESTful API that will be consumed by any REST client. This answer the extensibility requirement as any device can consume those services. The data exchanged is formatted using JSON.

The client tier is implemented using Android. The Android application communicates with the server using SSL to ensure the privacy of the user.

There are two web services used in the application. Facebook Graph API is the first one, it is used to query the essential data from Facebook for a given user and share his trip on the social media. Personal information like first and last name, age, hometown, profile picture, and email address for example can be retrieved using this API. This information will be used to populate the application's database. Also, upon the creation of a trip or registration in one, this API may be used to share the trips on Facebook in order to attract more users. This API is also used to

The second web service used is Google Maps which is useful for all what concerns geolocalisation. In order to organize and standardize the destination and the origin of the trips, this web service offers a place identification using a unique ID. This will unify how the places are referred to and avoid confusion.

2) Entity relationship diagram



In order to persist data on the database, the entities should be defined. The following entities will be represented at the level of the server as models and persisted on the database as collections. The following table gives the detail about each entity including its name, its attributes and a description on what it is used for.

Entity name	Attributes *type*	Attribute's Description	General Entity Description
sTrip	sTripID *String*	The ID of the trip which is unique for each trip	This entity represents a single trip. It stores different information about the single trips. The different passengers that will take part in this single trip will be stored in a common entity called sTripList described below.
	Origin *String*	The origin city represented by a place ID from Google Maps	
	Destination *String*	The destination represented by a place ID from Google Maps	
	MeetPoint *String*	A GPS coordinates of the meeting point for the trip	
	Contribution *double*	The amount each passenger should pay to take part in the trip	
	DriverId *String*	The ID of the user that created the trip and will be driving for the trip	
	NumFreeSpots *int*	The number of passengers that the driver wants in his trip	
	Smoking *boolean*	A boolean stating if smoking is allowed during the trip	
	Pauses *boolean*	A boolean stating if stopping for rest is allowed during the trip	
	BigBags *boolean*	A boolean stating if there is enough room for big bags	
	Animals *boolean*	A boolean stating if the passenger can bring a pet for the trip	
	FoodDrinks *boolean*	A Boolean stating if eating or drinking in the car is allowed during the trip	
TripList	RowId *String*	The ID to the row	This entity is used to keep track of the users that are taking part of a
	TripId *String*	The trip own which this row's user is taking	

	UserId *String*	The user that will take part in the trip referenced by this row	trip part (can be frequent or single trip)
CheckIn	CheckInId *String*	The ID of the checkin	This entity stores the check ins of the single trips and uses the check in time to calculate the punctuality of the user.
	Time *Date*	The time the user checked in that is used to calculate the punctuality of the server	
	sTripId *String*	The single trip assigned to the check in	
	UserId *String*	The user that proceeded to the check in	
AccessToken	id *String*	The id of the access token	This entity stores the application access token for users.
	userId *String*	The owner of the access token	
	created *Date*	The date and time when the access token has been created	
	ttl *Integer*	The time to live for the token. It represents the duration for which to token is valid	
User	UserId *String*	The ID of the user	This is the entity that represents every user of the application. This user can act as a driver or a passenger.
	fbToken *String*	The token received from the Android application for validation	
	Name *String*	The name of the user retrieved from Facebook	
	Age *int*	The age of the user	
	Rating *double*	A double ranging from 0.0 to 5.0 that represents the average rating of this user by other ones	
	Punctuality *double*	A double ranging from 0.0 to 5.0 that represents the punctuality of the user calculated from the time of check ins	

	PhotoURL *String*	The photo URL got from Facebook to be showed in the profil of the user	
	CarId *String*	The id of the car of the user	
	PhoneNum *String*	The phone number of the user to be contacted by passengers or drivers	
Car	CarId *String*	Car ID	This entity stores car information for the user to choose from
	Maker *String*	The maker of the car	
	Type *String*	The type of the car	
Rating	RatingId *String*	The rating's id	This entity stores all the ratings of users. The pair RatingUserId and RatedUserId can only exist if the Rating user has taken a trip with the Rated user and vice versa
	RatingUserId *String*	The ID of the user that rated	
	RatedUserId *String*	The ID of the user that is being rated	
	Rating *int*	The rating is an integer that ranges from 0 to 5	
fTrip	fTripId *String*		This entity represents a frequent trip. It stores different information about the frequent trip. The different passengers that will take part in this single trip will be stored in a common entity called TripList described above.
	Origin *String*	The origin place represented by a place ID from Google Maps	
	Destination *String*	The destination place represented by a place ID from Google Maps	
	StartDate *Date*	The starting date of the frequent trip	
	FinishDate *Date*	The end date of the frequent trip	
	Contribution *Double*	The amount to pay for every trip in the frequent trip (One way)	
	DriverId *String*	The ID of the user that created the trip and will be driving for the trip	
	NumFreeSpots *int*	The number of passengers that the driver wants in his trip	

ScheduleDays	SchedDayId *String*	The ID of the row	In order to persist the schedule of a frequent trip and given the fact that the schedule may differ from a week day to another, each day will have its dedicated schedule.
	ftripId *String*	The ID of the frequent trip that this schedule is assigned to	
	Day *String*	The day this schedule refers to. Monday, Tuesday, Wednesday…	
	DepTime *Date*	The departure time a that day	
	RetTime *Date*	The return tim from destination to origin in that day	

3) Business model

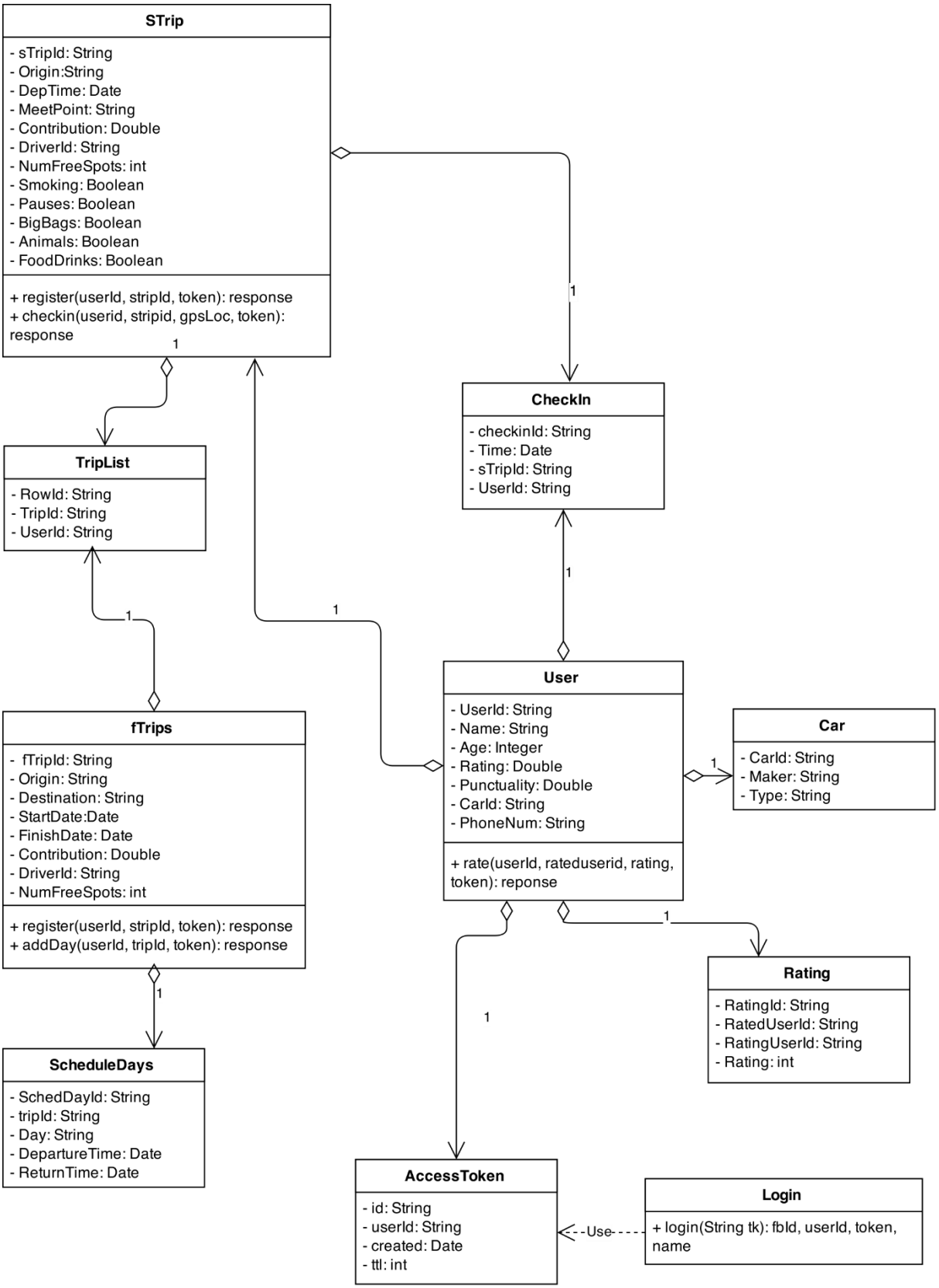
The following functions need to be implemented in the server in order to be exposed to the mobile application through REST. The basic CRUD (create, read, update and delete) are automatically created by the Loopback framework, however they are hidden from the user that can only access the Read function of some models. The following methods are, on the other side, custom remote methods that implement the business intelligence. The methods are static methods of a model and are exposed using a REST endpoint. The name of the model followed by the name of the method is used to reach it using REST. For example, in order to use the method login, the end point /Login/login is used.

Method	Description	Input	Output	Model
login(String tk)	This method takes as an input the Facebook token received from the Android device and validates it against Facebook's servers. When validated, the method identifies the user, creates an application token for him and sends it back.	***tk*** – Facebook token	**fbId** – User's Facebook ID ***userId*** – User's app ID ***token***– application token to be used for further requests. ***name*** – User's first name to be displayed ***email*** – User's email address	Login

			picture – URL of the user's picture	
register(String userId, String tripId, String token)	This method should be implemented for both regular trips and frequent trips. At first, it checks the number of users already registered on the given trip and if the user is not already registered. Then it registers the user by adding a row in the trip list	***userId*** – User's app ID ***tripID*** – Regular or frequent trip ID ***token*** – User's app token	***response*** – A Boolean stating if the registration has been done ***message*** – A string containing a message for the user	Both sTrip and fTrip
checkin(String userId, String stripId, String gpsLoc, String token)	This method is used to check the user in a trip. The server should verify if the user is close to the meeting point by using the received coordinates and the meeting point's coordinates. The based on the time	***userId*** – User's app ID ***stripID*** – Regular trip ID ***gpsLoc*** – the GPS coordinates of the user ***token*** – User's app token	***response*** –A Boolean stating if the checkin has been done ***message*** – A string containing a message for the user	sTrip
rate(String userId, String ratedUserID, Double rating, String token)	This method is used to add a rating for a user after he took part in a driver's trip	***userId*** – User's app ID ***ratedUserId*** – rated user's app ID	***response*** – A Boolean stating if the rating has been done ***message*** – A string containing a message for the user	User
addDay(String userId, String tripId, String token)	This method is used to add a day to a frequent trip	***userId*** – User's app ID ***tripID*** –Frequent trip ID ***token*** – User's app token	***response*** – A Boolean stating if the day has been added ***message*** – A string containing a message for the user	fTrips

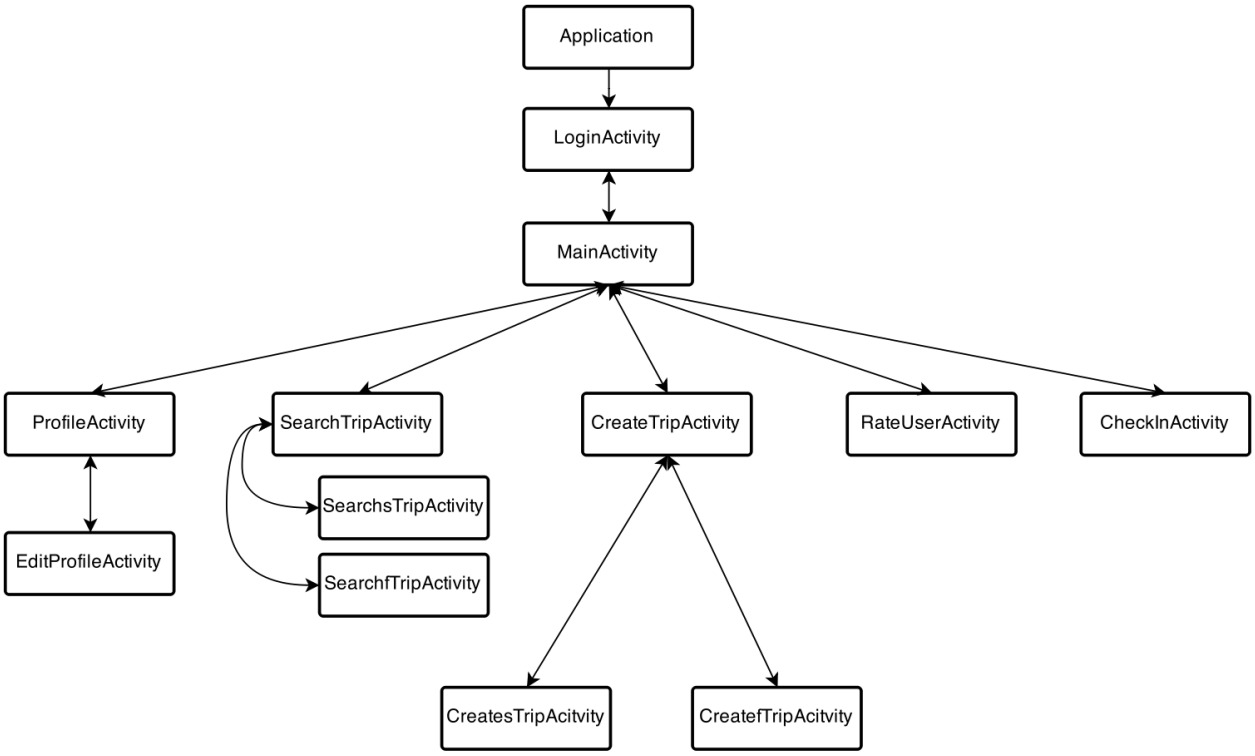
The input validation is taken care of at the level of the level when defining a model and when persisting the model in the database. This is to add a level of security by validating data received directly through REST. For the one received from the Android application, it will be already validated by the application itself before being sent to the server.

Following is the class diagram for the server, with all the methods described above.



4) Android application design

The Android's user interface is built around Activities which are single focused thing that a user can do. They are directly related to the functional requirement defined for the application. For example, the functional requirement stating that a use should be able to login in the application, implies that there should be an activity that enables the user to login. Following are the derived, from the user requirements, activities that should be implemented to fulfill the functional requirements.



Activity Name	**Activity description**
LoginActivity	This activity is the first activity that is opened in when the application is launched. It contains the buttons to login uses any social media chosen. When the login is successful, it opens the *MainActivity*.
MainActivity	The *MainActivity* is opened only after the login of the user. It has icons for all the tasks that can be done by the application. At the top it includes a little bar with the logged user's information and photo.
ProfileActivity	This activity is used by the user to see all his information including ratings. If the user wishes to edit some of his information he can use a dedicated button on this activity that opens *EditProfileActivity*.
EditProfileActivity	This activity is used to change the user's information. However, not all information can be changed.
SearchTripActivity	This activity gives the choice to either search a single or frequent trip.
SearchsTripActivity	This activity is used to search for single trips using by giving origin, destination and date.
SearchfTripActivity	This activity is used to search for single trips using by giving origin, destination, date and other information.
CreateTripActivity	This activity gives the choice to either create a single or frequent trip.
CreatesTripActivity	This activity is used to create a single trip
CreatefTripActivity	This activity is used to create a frequent trip
RateUserActivity	This activity shows all the recent users that the user has taken a trip with. Then it gives him the ability to rate those users
CheckInActivity	If the user is registred for a trip that is in the upcoming 10 minutes and he is close to the meeting point, the user can check in the trip to improve his punctuality score

IV) Implementation and Testing

1) Implementation

The implementation process followed an iterative process in both the server and client side. Each function(requirement) is implemented and then tested.

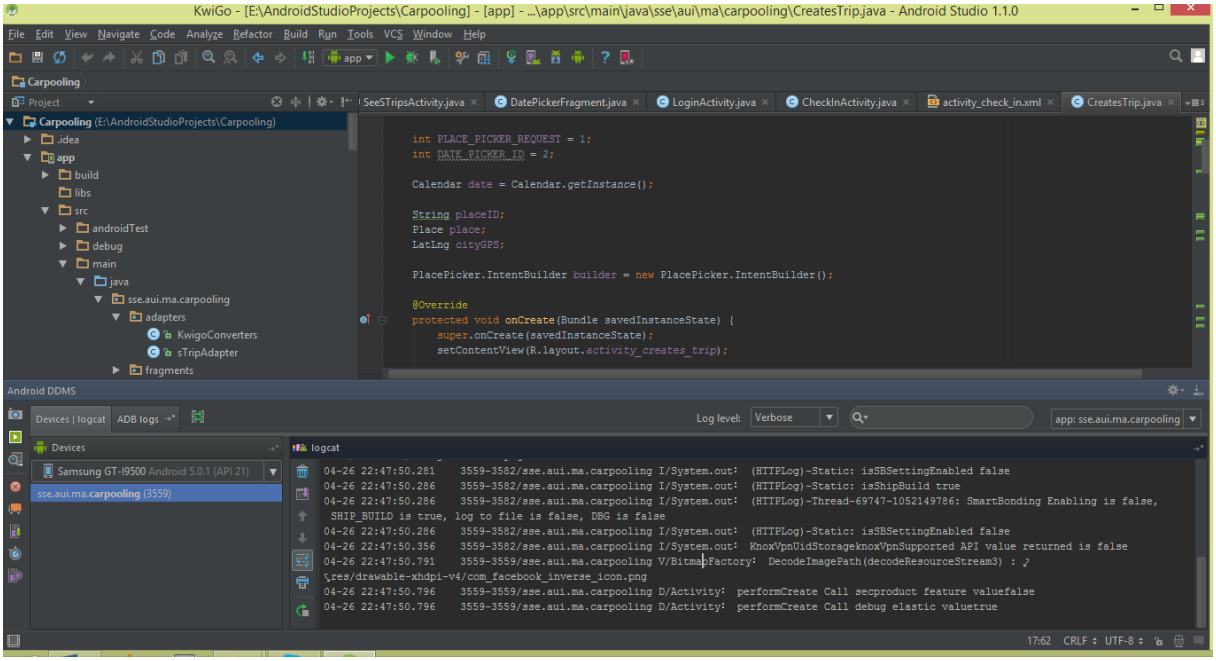
a) Server side

The process of implementation started by the developing the server first because it is the backbone of the application and it is required for the application to be tested. There is no IDE to develop Loopback applications. However, loopback has an in line tool to create the different models.

A simple text editor has been used to edit the different files required by the server. The server is the started using command lines

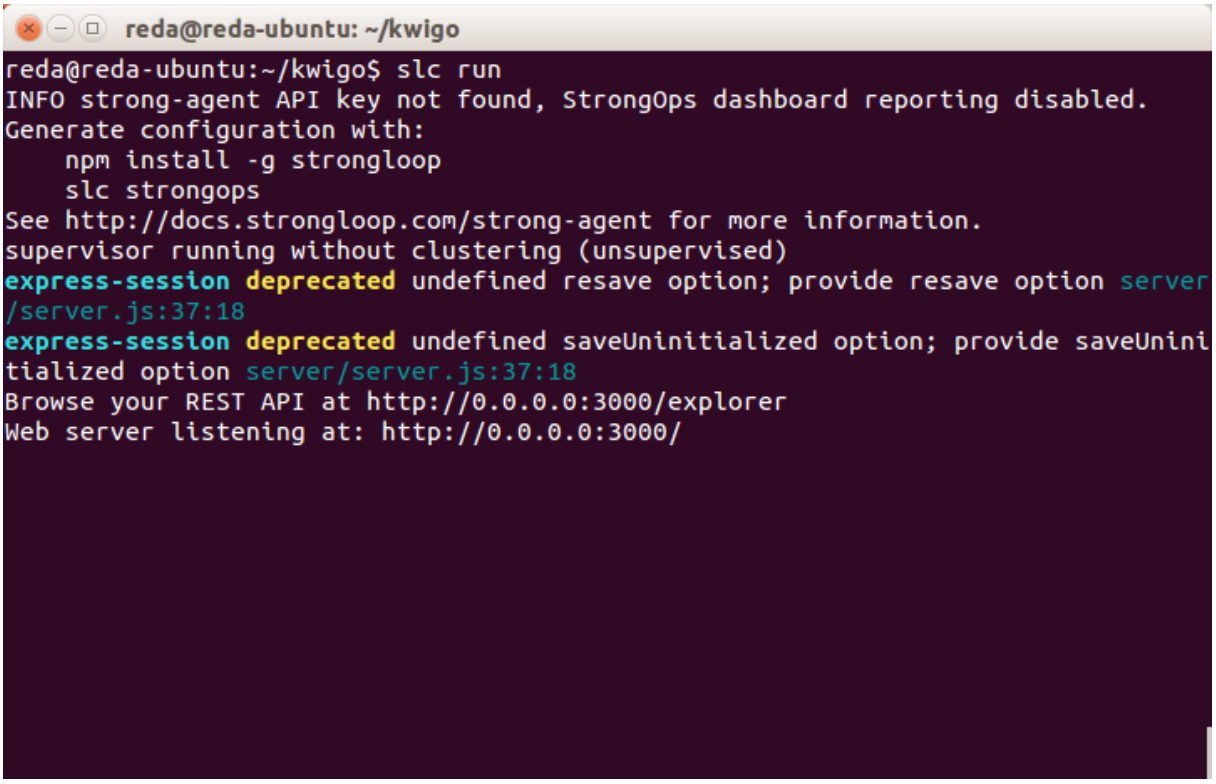
b) Client side

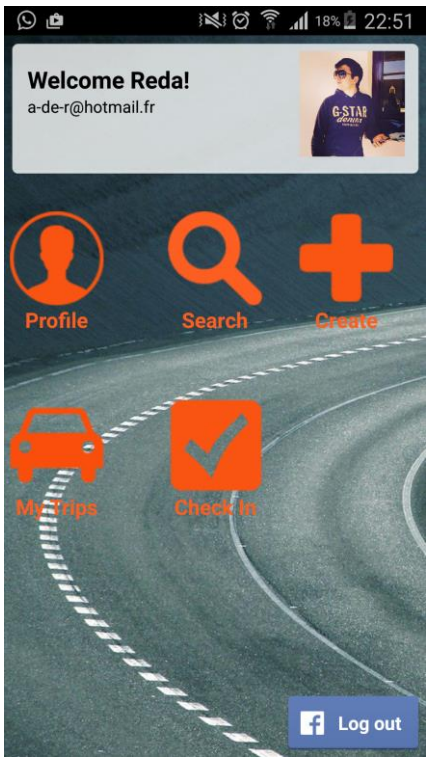
The Android application has been developed using the Android Studio IDE. It is targeted to the newest Android OS, Android 5.0 Lollipop and higher. The implementation respect the design previously presented.

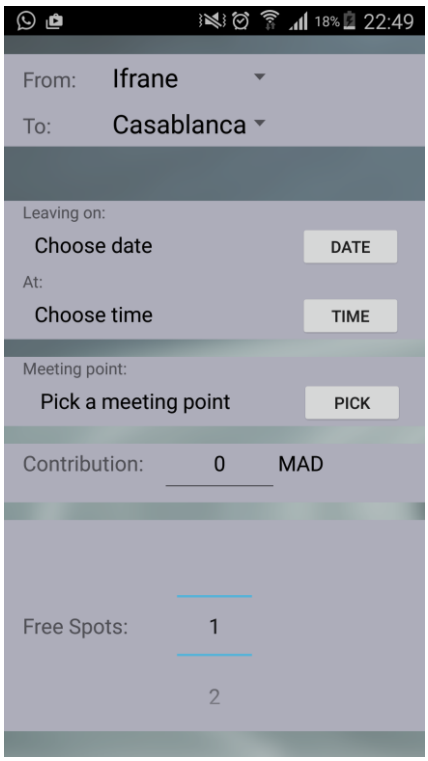


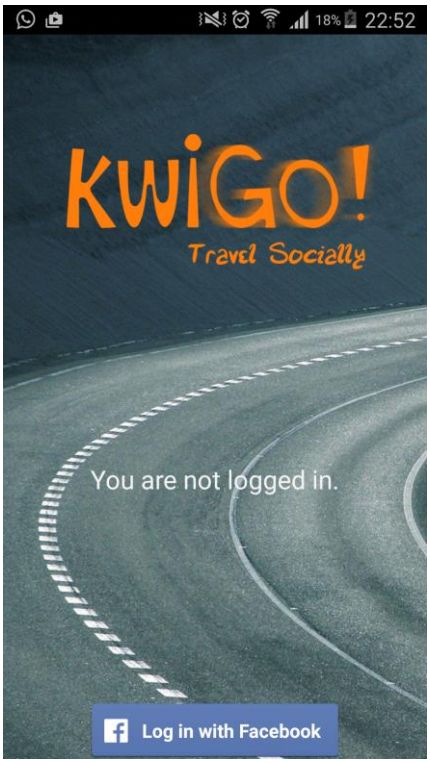
c) Results

Following are screen shots of the application server and application.




22:51
Welcome Reda!
a-de-r@hotmail.fr
Profile
Search
Create
My Trips
Check In
Log out


22:49
From: Ifrane
To: Casablanca
Leaving on:
Choose date
DATE
At:
Choose time
TIME
Meeting point:
Pick a meeting point
PICK
Contribution: 0 MAD
Free Spots: 1
2


22:52
kwiGo!
Travel Socially
You are not logged in.
Log in with Facebook


22:51
Results
Ifrane to Casablanca
100.0MAD
29-04-2015 13:15

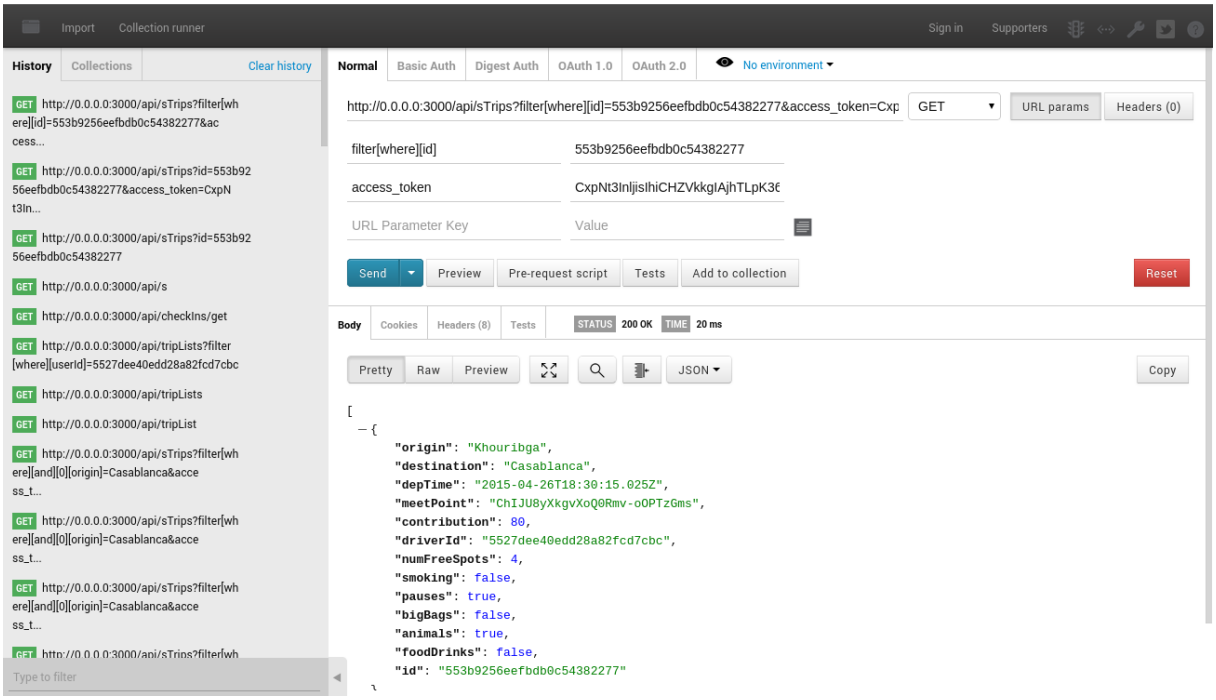
2) Testing

For testing purposes, I set up a Linux machine running Ubuntu 14.0 that runs the servers. On that machine, a created a Wi-Fi hotspot to enable the Android phone to reach the server. The Android phone is a Samsung Galaxy S4 running Android 5.0.1 Lollipop. The phone was connected through USB to a Windows 8 machine where Android Studio was installed.

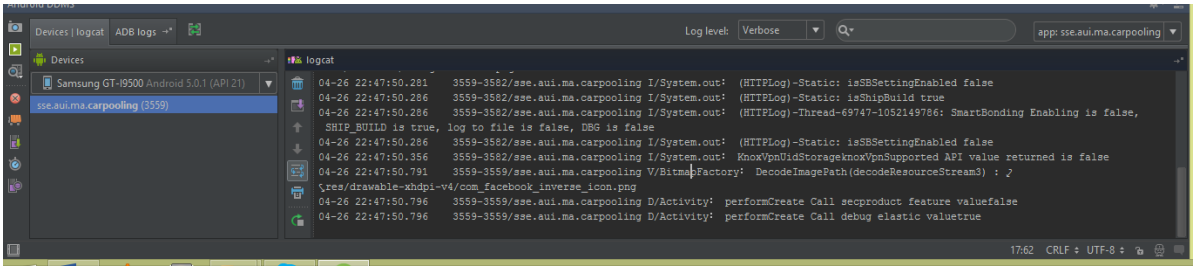
a) Server side

Following each iteration, I tested the implemented function of that iteration. Since the server is exposed using REST, I used a REST client called Postman. The tool helps building an HTTP request, send it to the server and read the response.

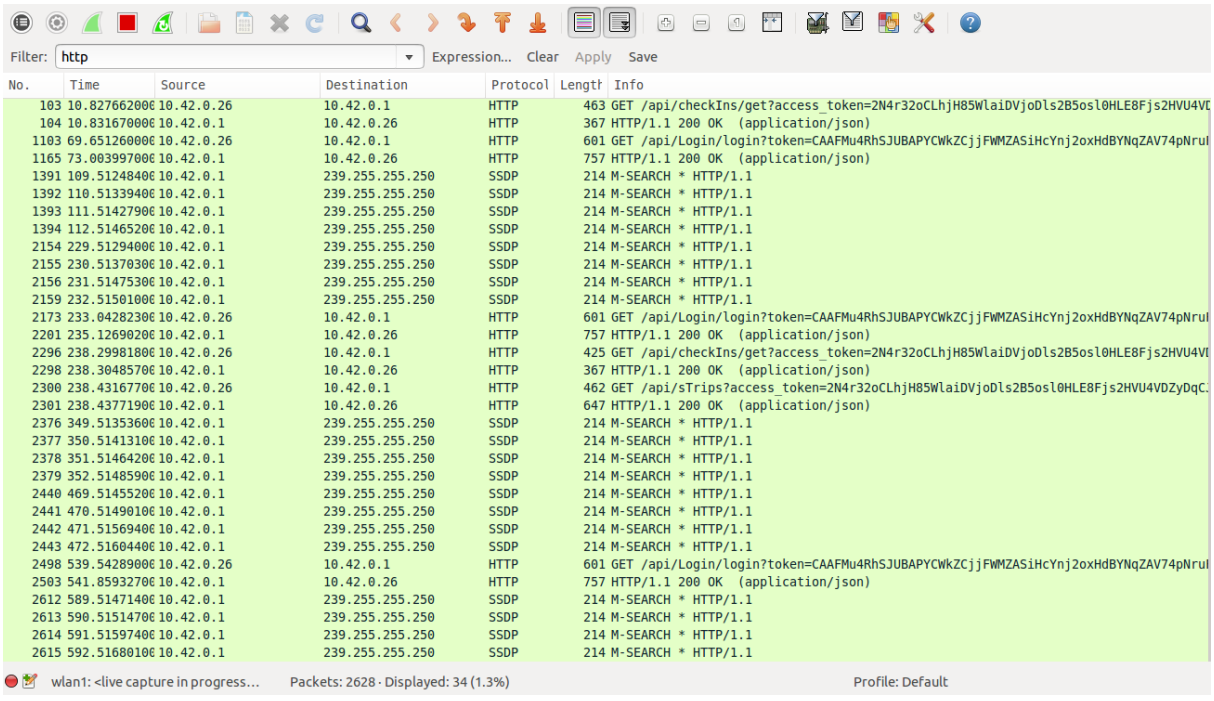
b) Client side



Android Studio enable developers to run their application directly in an Android phone a displays the log of the application. I used this feature to run the application and watch the different log entries.



In order to see what HTTP requests the server and the application are exchanging, I used a packet sniffer to explore the requests and diagnose the source of a bug. The packet sniffer used is WireShark.



The application was given to users to test and I noted their behaviors to improve my application in upcoming iterations.

V) STEEPLE analysis

Social

Increase social interaction and solidarity

Meet new people during rides and make new friends

Driving with people is better than driving alone since it involves less stress

Technology

Smartphone penetration is increasing day after day

Use of technology to create matches between drivers and passengers

The application is accessible from anywhere using a smartphone

Real time communication between actors

Environmental

Increase of high occupancy vehicles which will lead to a decrease of CO2 emissions.

Less cars on the roads that leads to safer roads and fluent traffic.

Economical

Savings as the price of gas and highways is shared among the travelers in a context of an increasing gas price.

Political

Increase of support for initiatives that decreases greenhouse gas emission

Support from government thank to the benefits of carpooling.

Legal

Insurances of drivers and passengers: In case of accidents, the owner of the car should have an insurance to cover any medical expenses.

Ethical

Client confidentiality should be kept: all information related to trips' history should only be communicated to their respective user

VI) Conclusion

This carpooling application is an application that complies to the enterprise class application principles. It is designed to be performing, scalable, extensible, and highly available. It also ensures the privacy of the users' data and secures its access. Given that it may be improved in many ways, the application is also easily maintainable.

The result achieved in this project is a working Android application and server that perform the requirements stated in this document. It is still not ready to be deployed on the Play Store for the public. The main reason is that the server should be deployed on stronger hardware with a good Internet connection.

The constraint that should have been considered is that developing a server and an Android application demand a lot of work. This should be considered in the time allowed for each one of these activities. Due to this lack of time, many things can be improved in the present application. This includes a better user interface with more attractive styles. Also, adding more support for authentication systems can be an improvement.

References

Journal Articles

Hamed, O., & Kafri, N. (2009). Performance Prediction of Web Based Application Architectures Case Study: .NET vs. Java EE. *International Journal of Web Applications, 1*(3), 146-156. Retrieved March 4, 2015, from http://www.dirf.org/ijwa/v1n30609.pdf

Adhao, A., & Gaikwad, A. (2013). ASYNCHRONOUS AND NON-BLOCKING INPUT/OUTPUTUSING BACKBONE.JS AND NODE.JS. International Journal of Application or Innovation in Engineering & Management, 2(11), 201-207. Retrieved March 3, 2015, from http://www.ijaiem.org/volume2issue11/IJAIEM-2013-11-20-058.pdf

Websites

http://www.idc.com/prodserv/smartphone-os-market-share.jsp

http://nodejs.org/industry/

http://www.mongodb.com/what-is-mongodb

http://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js

http://loopback.io/