Glovo App

Notes about the development of the Glovo Challenge

Michele Franco

Abstract

The Glovo challenge has been really interesting in terms of designing and developing of a product. The following review explains firstly the process and choices behind Glovo App. Then, it will be given details about the implementation of the application.

A product must be conceived starting from the bottom.

As general rule of thumb, it's not possible to erect buildings without foundations. This rule affects the development process of a digital product too. The skeleton of a mobile application are data models. Therefore, the starting point has been the detection of data on which has been built the Glovo App.

This task has been fairly simple, mainly because, as for the most part of application, information comes from servers. Hence, responses from them has been translated into data models properly.

By Scanning responses, it has been reached to a conclusion:

API provided has been designed to work in a web app environment.

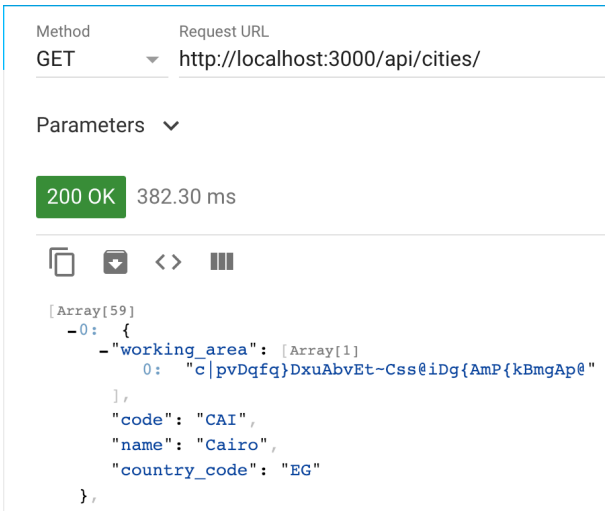
Glovo App has two different data to deal with: Country and City.

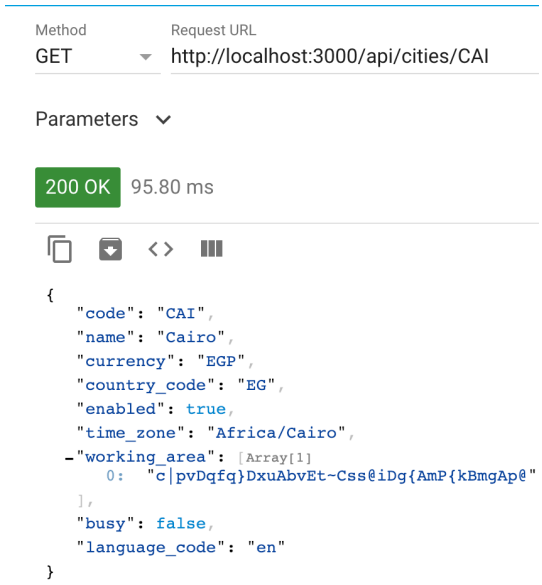
Focusing on City model, there are two different API for them, as it's possible to see below:

Data returned are literally the same, the left request gives back a subset of the response of the right one. This has been done because a web application relies only on Internet connection, so this kind of design should be acceptable in that area.

Conversely, moving to mobile devices that can work offline, it should be a good habit to minimise requests to the server,

especially if they return same objects. Since of non-homogenous data, cities can have different states, that could bring developers to adapt the implementation to a bad design of API.





Created models, it's the turn to make features.

It's time to speak about interfaces.

User Experience as vehicle to design interfaces.

"Two of the most important characteristics of good design are discoverability and understanding.

Discoverability: Is it possible to even figure out what actions are possible and where and how to perform them?

Understanding: What does it all mean? How is the product supposed to be used? What do all the different controls and settings mean?"

(The Design of Everyday Things - Donald Norman).

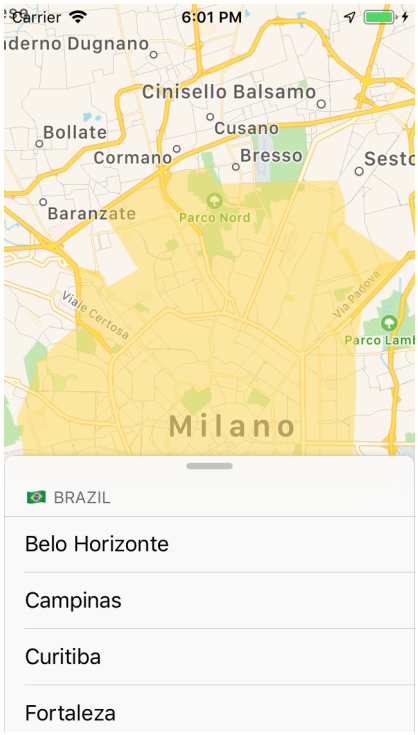
Everyone who ends up to create a project should satisfy users needs.

Donald Norman, a pioneer in the UX field, tries to track the lane for successful results. How can Glovo App have discoverable and understandable components?

Have a look to the home screen of the application on the left.

The map fills the screen in order to have the right importance for users, even though the presence of a big floating button on the upper right corner.

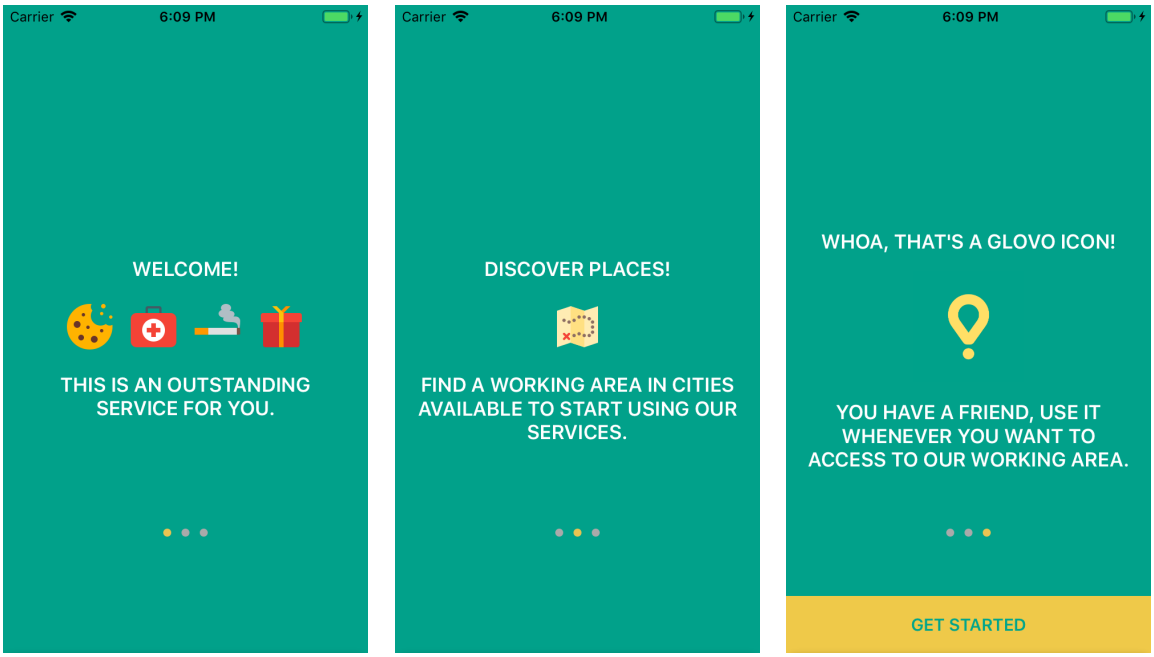
Users can decide to know working areas available. This kind of information has been displayed through panels visible “on demand”. As the user needs details, a panel is presented from the bottom.



Components are designed to be discoverable and understandable. The table in the left image has a clear sign for dragging up the list. On the other hand, panel on the right has been designed to be fixed.

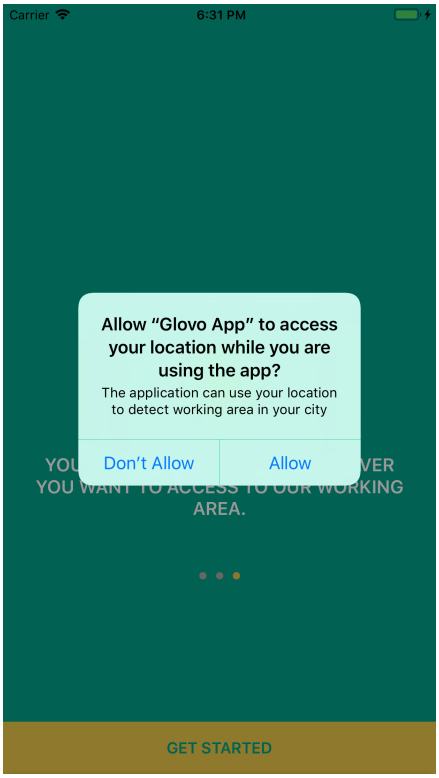
But concretely, what does the application? How can users know what they are able to do at the beginning?

To be sure of a complete clarification of the purpose of the project, it has been added an “on boarding” flow.



On Boarding flow explains goals briefly

Through the on boarding process, the user has a global understanding of features of the product. For this reason, the request of permission of user location has been put to as final action of the flow, mainly because users has been more open to allow permissions if it has a goal.



Geek mode Enabled, Speaking about the implementation 🤓

Ok, I have the entire drawing in my mind. How should it implement components? It has been said the map is the main element for the application and it deserves to be the first to talk about.

The question is clear: Is it better Mapkit or Maps SDK by Google?

The answer is: It depends on several factors.

Mapkit is the native framework for maps by Apple and the main reason it has been chosen it over the Google library is purely for avoiding too many external dependencies. Since the requirements were to develop just the iOS version, Mapkit has been sufficient to reach the goal.

A pros for Google Maps is the common interface between platforms, so that users cannot lost themselves even switching

from iOS to Android and viceversa. Moreover, Google Maps has a native tool to decode polylines.

Actually, the algorithm for decoding polylines is vey simple, thus Mapkit allows to not overload project with external frameworks. So the main effort has been the writing the codes for retrieving coordinates from a string encoded.

Snapkit hasn't been deliberately used in the project since it's a library which stands on the top of UIKit simply, providing just a more convenience syntax than the native framework by Apple. So again, it would have been another external dependency with the project.

The same has been applied for networking. Apple provides solid API for managing network data transfer tasks.

Obviously, there are tons of outstanding frameworks like Alamofire that fits great for doing this job, but sometimes adding libraries on a project could be dangerous(iOS developers know how it has been frustrated the migration to the different new versions of Swift about frameworks.)

That being said, it would be wrong erasing the possibility of using them, because the open source communities are really a treasure for developers; it is enough to benefit of libraries parsimoniously.

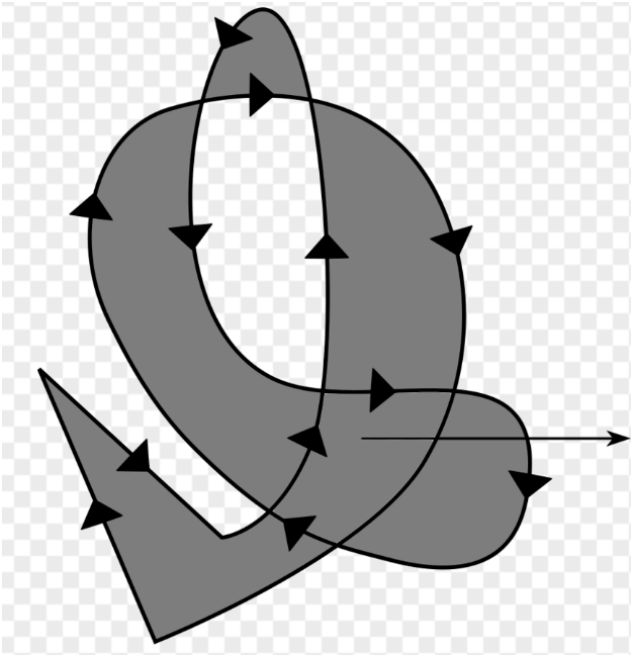
The only Third-party framework utilised has been necessary for implementing the menu for the choice of working area, exclusively to speed up the process.

Carthage has been the dependency Manager for Glovo App. The most part of companies tends to select CocoaPods over Carthage.

Each one has inner pros and cons. The cool thing of Carthage is that it requires more effort that CocoaPods for importing

frameworks into the application, but you will never lost the control of the organisation of the project.

At the time of this writing, it hasn't been able to finalise the combination of all polygons for each city into one polygon to avoid a patchwork look of working areas quickly. The algorithm used has been the "non-zero winding rule".



This rule plots a ray from any region to be evaluated toward the bounds of the drawing, then counts the closed path elements that the ray crosses: +1 for counterclockwise paths, -1 for clockwise. The rule defines interior regions as those where the sum of crossings is nonzero, and exterior regions as those where the sum of crossings is zero.
So in the example above, an arrow shows a ray from a point P heading out of the curve. The ray is intersected in a clockwise direction twice, each contributing -1 to the winding score: because the total, -2, is not zero, P is concluded to be 'inside' the curve.

Using this technique, paths has been recalculated correctly.

The bad news is that it hasn't been able to drawing the new path on the map in a proper way. I preferred to continue focusing on the

rest of the project but, even if this request hasn't been satisfied yet, in the next days I'll try to solve this issue.

The Developer Life cycle: Branching, Merging, Committing, Pushing

A very self explained title, developers are literally addicted to repositories. They can save the developer's Life in some circumstances.

Therefore, mastering the Git Flow could be a free pass to the Heaven. Generally speaking, it should be a one-to-one relationship between features and branches. There are many cases in which developers could modify same things concurrently, and to use branches is the best tool for having a personal state of application. Once job has been terminated, it's time for merging and then committing to produce a current snapshot of the product.

The following image is a piece of history from Glovo App repository.

Merge branch 'feature/api_management' into develop
api integrated & minimum unit tests written
Models of the App integrated
Merge branch 'feature/map_integration' into develop

To sum up...

It's never an easy job to make a mobile application, especially with small devices like phones but combining several user experience techniques with a solid technical implementation can be the key to the success of a product.

Let's rock Glovo Challenge.