University Bordeaux 1

Internship report - Master Informatics in Software Engineering

Internship at Rainmaker Labs

By
Quang Anh NGUYEN

Department of Informatics
University Bordeaux 1

March 2016

Abstract

Mobile development is one of the most popular industries nowaday. Each year there are several hundreds millions of smartphones which were sold, in which Android devices and iOS devices hold the most part, around 98% of the market. I decided to learn about Android development, since it used Java language, the most familiar programming language to me. For that reason I applied into a branch of Rainmaker Labs in Vietnam, a mobile development outsourcing company.

Contents

List of Figures

1 Acknowledgement

I would like to express my gratitude to Mr. Tran Tan Phong, for allowing me join in the company, although I didn't have any experience in the field I applied for. I also would like to express my gratitude to Mr. Tran Thanh Tri, Mr. Pham Tien Vinh, tech leader and project leader, for training me and helping me understand many technical problems. To Mr. Le Vuong Thien, project manager, who helped me getting familiar with SCRUM method.

2 Introduction

2.1 Rainmaler Labs

Rainmaker Labs is the singaporean outsourcing company, specialized in mobile development, currently placed first among the competitors in Singapore and Asia Pacific.

Rainmaker Labs in Vietnam is the new branch which was established in Feb 2015. Though it's new, but thanks to the politics, cultures and remuneration policy, the offshore branch was able to recruit many talented and experienced individuals, some of them even hold high position in their previous jobs.

2.1.1 Company's culture

Rainmaker Labs is a young campany with young people, so the way it run is refresh to me. There isn't any dress code, not too shady clothes are good enough. Working time starts from 8h30 am, and ends at 5h30 pm, lunch break starts from 12h pm and ends at 1h pm. Everyone has their own seat, but you can move everywhere, even the isolated kitchen corner, if you want to emerge in your thoughts.

If you have an emergency matter to attend, or you just want to lay down and work on your bed, just give your team's leader and project's leader a call, tell them you want to work remote from home. Just make sure don't make it your habit, since interaction directly between team member is better than calling or chatting through internet.

2.1.2 Teams

Rainmaker Labs in Vietnam has many teams, such as : product team, web team, android team, iOS team, quantity controller team (QC team), marketing team and project manager team. Each team have their own jobs, like android & iOS team work on mobile development, QC team make sure that every functionality work right, and followed scope of work. Product team work on company's project (beside outsourcing project, company has its own project), web team work on front-end and back-end. Project manager is in charge of project, and make sure that project is sticked to its time line.

2.2 Projects

Although I was new to Android at that time, infact, I had never touch Android before, but I still managed to convince my seniors to let me participate in company's projects. The first project was the small project, named **Pedro**, which served at the **Charle & Keith Fashionable Awards 2015**. In this project I was able to study about the basic of Android development and some advanced technic.

The second one is a big project. This project is the digital version of Singapore government project, **Electronic Tourist Refund Scheme** aka **eTRS**. In this report I will concentrate on the technics, and knownledge which I was able to acquired.

3 Software Development Method

During the devlopment of an application, there are many tools and steps. Below are some important items :

3.1 SCRUM

In Rainmaker Labs, we use **SCRUM** as our software deployment method. With this method, we could deploy software to the customer as fast as possible, and be closest to customer's expectation. There are some important points in SCRUM such as :

- break down the project into small tasks, as small as possible.
- estimate the difficulties of each task and give points to them, then each member of team will select their task, base on their capability, such that all the team members have the same amount of points.
- during the development process, whenever a member encounter a problem which he couldn't solve, they could always ask for help. Team leader, or someone else who did solve the problem before could give hints to help him resolve his issue.
- everyday before starting working, team will have a stand-up meeting for about 10 minutes. Scrum master would hear team member reporting about their processes the previous day, and their planning for today's works.
- after each sprint (about a week), team will demonstrate what they have done to the client, allowing them to follow the development process, and correct the features to their ideal.

- after some sprints, when the product is about to finish, the tester team will join and dug in for bug. The development team now will start to fix the reported bug, along with working on their tasks.

Sadly in our company, SCRUM is not deployed as 100% as it means to be. The estimation of tasks is not executed by the whole team. In fact, it was the 2 most experience in our team that broken down and estimated every tasks, turn out that the estimation is not done objectively. Some tasks were over-estimated, some were under-estimated. And the most critical point is that both Android and iOS team have the same point for the same task, which is not true. Since there're tasks that could be done easily in Android while in iOS they have to struggle to make it right, and reverse.

One more drawback, is that actually in our company, point is equal to hour. So a 4 points task is given 4 hours to complete. This is also not true, since point should be given based on the complexity of the task itself.

Anyway, it was interested to experience method SCRUM myself, though it was not 100% it supposed to be.

3.2 JIRA

JIRA is used for managing task. Project manager could monitor everybody's process by following JIRA. Tester could report every bugs they've met on JIRA, and developer could fix the problems thanks to the details of how to produce such bug.

3.3 GitHub

To manage code, we use github system (paid version, not free one). Developer will create his own branch for each task in JIRA, and after making sure that everything work right, merge that branch into the main branch.

3.4 Charles

In mobile development, when interacting with internet, for example when calling API, catching network package is very important, since you can then prepare model associate with each kind of reponse, request. For that reason we used Charles. Charles is a software allowed catching package which go through its port, and helped user analyze package's content detail, from header to body.

3.5 Testing

One of the important phase in developping an application is testing. While working on this project, it became clearer to me how important testing is. One of the strangest cases that I've ever met, is that the application worked fine on my phone (Sony), but crashed on another one (Samsung). Without a tester team, and various kinds of devices, it could be impossible to spot this out.

4 My Project

4.1 Project's purpose

Electronic Tourist Refund Scheme (ETRS) is a government's project, which allowed the tourists who visit Singapore could reclaim Good and Service Tax if he bought a product in Singapore.

To be able to reclaim the refund money, a condition have to be satisfied:

- receipt's value must exceed 100 SGD
- merchant or shop who generated the receipt must registry for the ETRS project
- if receipt's value don't exceed 100 SGD, it could be grouped with maximum another 2 receipts, such that the total value of this group must exceed 100 SGD and all the receipts within this group must belong to the same merchant or shop

Our job is to realize the idea of the ETRS project into mobile device.

4.2 Important technics implemented

4.2.1 BlinkID

BlinkID is a library allow scanning the passport of tourist. From the information we got from the scan, we can create account for the tourist, thus each person is associated

with his/her passport. So even if a tourist lost his account, the refund process still could going on if he still keep his passport.

The mechanic of the scanning is that it will read the **Machine Readable Zone** (MRZ).

Figure 4.1: Passport with MRZ zone

Scanning passport included 2 tasks : get information from the MRZ zone, and take the cropped picture of the passport. Getting information from the MRZ zone is actually easy, since it's the main purpose of BlinkID, but to get the cropped picture of the passport, I have to work more. Luckily BlinkID have supported this feature too, but it's not as easy as the other one. I have to include a *ImageListener* into the *RecognizerView* (the camera container), once the images were captured, it would fire an event inside *ImageListener*. But to use it, we had to link it with our main process.

Because the scanning function is not only in 1 screen, so to make the process of implementation more easier, I wrapped up the *RecognizerView* and *ImageListener* into one class, and every communication, including getting cropped image, must go through this new class. That basically solved the problem. But then 1 more problem appear: the *ImageListener* not always return the cropped image. Sometimes the scanning process happened too fast that it could only get the image of the MRZ zone of the passport. To resolved this, I make a *Handler* that examine all taken images, if no desired image was taken when the scanning had done, repeat again. Once successfully capture the cropped image, I fired an event in my custom *Listener* to notify the main thread. Then it would go to the next screen.

4.2.2 QR code

Each tourist, after registry their account, is given 1 QR code. In case that he don't want to show his passport, he could use this QR code to identify himself. Each merchant or shop would have their own QR code, and the tourist can scan their QR code for the ID of the shop.

In this project, I've implemented the scanning and generating QR code using a library, created by JourneyApps.

- website: http://journeyapps.com/
- github: https://github.com/journeyapps/zxing-android-embedded

4.2.3 Android location service

Using this service, the application will notice the tourist if he's near one of the shop registry for the ETRS project. This service locates the user's position either by wifi or by GPS.

4.2.4 Geofencing

Given coordinates of a location, and a radius, we could define a zone. Geofencing then notify if a target is getting in or out of this zone. We used this to detect if the user is going in or out of certain zones, thus allow us to take correspondant behavior.

4.2.5 Push notification

This allows the server to notify users when there're changes in their data, such as if their request to refund is realized, or refused.

4.2.6 Google Cloud Message (GCM)

The application allow user to define their events that need to be notified when the time come, for example the flight's time. Application will use GCM to notify the user when such time come.

4.2.7 Social network sharing

The application allow user to share on their social network, such as Facebook, Wechat, Sina Weibo. And this is abig challenge, because China's social network (Wechat, Sina Weibo) do not have many tutorials in English, and the procedure to register application ID on these network is complicated too.

4.2.8 Map

Google Map service is intergrated into the application, thanks to the need of path finder. It's not very complicated, but then the problem rise. As Google Map can't be used inside China, we have to adopt another map service from China, Baidu map, and this process is extremely difficult, from the registration process to implementation process, since this service only work if you are in China, even fake GPS couldn't help.

4.2.9 Communication method between client and server

We mainly use RestAPI, since it's lightweight, and easy to implement. For security purpose, we generate for each user each times they login an **user-token**. Everytime a client want to make a request, the user-token will be included into header of request. Beside, base on this user-token, we could identify user, and prevent the client from spamming request to server.

The response is under form of **JSON** format. And since Google have greatly supported JSON, parsing a JSON response into Java model is plain easy.

4.3 Algorithm utilized

I was given the task to write the algorithm that filter a list of receipts and return receipts which satisfy the condition in 4.1.

4.3.1 Brief description

Given the list of receipts, could be generated by many shops, extract a list of receipts which satisfy the conditions in 4.1.

4.3.2 Solution

We devide the list of receipts into many lists such that the receipts in each list belong to one shop only. This lead to a new problem: Given the list of receipts, all receipts belong to 1 shop, extract a list of receipts which satisfy the condition in 4.1

Step 1

Extract all the receipts which value greater or equal 100. Then sort the rest of list in increased order.

Step 2

Divide the ordered list into many smaller lists :

Algorithm 1 Split list of receipts into many sublists

```
1: function SplitList(receipts)
2:     j = 0, i = 0
3:     while i < receipts.length do
4:         if ⌈receipts[i].value / 10⌉ ≤ j then
5:             list_j.add(receipts[i])
6:             i++.
7:         else
8:             j++
```

Step 3

From the sublists, we will extract groups of receipts which satisfy the condition. There's some preconditions we could make for simplifier the process.

- for a group of 2 receipts, if they belong to $list_i$ and $list_j$ then $i + j \geq 9$
- for a group of 3 receipts, if the belong to $list_i$, $list_j$ and $list_k$ then $i + j + k \geq 8$

First of all we need a function that check the condition of a set of receipt. If satisfy, remove the receipts from their correspondant sublist, and add a new group of receipt to our list of group and return true. Otherwise return false.

Algorithm 2 Check if the correspondant receipts could form a group

```
1: function CHECK(i, j, k, a, b, c, groups)
2:     r_a = list_i[a]
3:     r_b = list_j[b]
4:     r_c = list_k[c]
5:     if r_a.value + r_b.value ≥ 100 ∧ (i ≠ j ∨ a ≠ b) then
6:         groups.add(new Group(r_a, r_b))
7:         list_i.remove(r_a), list_j.remove(r_b)
8:         return true
9:     if r_b.value + r_c.value ≥ 100 ∧ (j ≠ k ∨ b ≠ c then
10:        groups.add(new Group(r_b, r_c))
11:        list_j.remove(r_b), list_k.remove(r_c)
12:        return true
13:    if r_a.value + r_c.value ≥ 100 ∧ (i ≠ k ∨ a ≠ c) then
14:        groups.add(new Group(r_a, r_c))
15:        list_i.remove(r_a), list_k.remove(r_c)
16:        return true
17:    if r_a.value + r_b.value + r_c.value ≥ 100 ∧ (i ≠ j ∨ a ≠ b) ∧ (i ≠ k ∨ a ≠ c) ∧ (j ≠
   k ∨ b ≠ c) then
18:        groups.add(new Group(r_a, r_b, r_c))
19:        list_i.remove(r_a), list_j.remove(r_b), list_k.remove(r_c)
20:        return true
       return false
```

The next function is the main one that extract a list of groups of receipts from the sublists.

Algorithm 3

```
1: function FILTER
2:     i = 0, j = 0, k = 0
3:     groups
4:     loop_i:
5:     for i = 0 to 9 do
6:         a = b = c = 0
7:         while a < list_i.size do
8:             loop_j:
9:             for j = i to 9 do
10:                b = c = 0
11:                while b < list_j.size do
12:                    loop_k:
13:                    for k = j to 9 do
14:                        c = 0
15:                        while a < list_i.size ∧ b < list_j.size ∧ c < list_k.size do
16:                            if i = j then
17:                                if a = b then
18:                                    if i + k ≥ 9 then
19:                                        if j ≠ k ∨ a ≠ c then
20:                                            if ¬CHECK(i, j, k, a, b, c, groups) then
21:                                                c + +
22:                                        else
23:                                            c + +
24:                                    else
25:                                        goto loop_k
26:                                else
27:                                    if k = j ∧ (c = a ∨ c = b) then
28:                                        if i + j ≥ 9 then
29:                                            if ¬CHECK(i, j, k, a, b, c, groups) then
30:                                                c + +
31:                                        else
32:                                            goto loop_k
33:                                    else
34:                                        if ¬CHECK(i, j, k, a, b, c, groups) then
35:                                            c + +
36:                            else
```

```
37:                                 if j = k then
38:                                     if b = c then
39:                                         if ¬CHECK(i, j, k, a, b, c, groups) then
40:                                             c++
41:                                     else
42:                                         if i + j + k ≥ 8 then
43:                                             if ¬CHECK(i, j, k, a, b, c, groups) then
44:                                                 c++
45:                                         else
46:                                             goto loop_k
47:                                 else
48:                                     if i + j + k ≥ 8 then
49:                                         if ¬CHECK(i, j, k, a, b, c, groups) then
50:                                             c++
51:                                     else
52:                                         goto loop_k
53:                         b++
54:                 a++
55:     return groups
```

At this stage we get a list which consists of :

- receipts which are greater than 100
- groups of receipts, maximum 3 receipts, which are greater than 100

Repeat for all lists of receipts of different merchant or shop, and we got the solution.

4.4 Database's synchronization

This application support offline usage, so that if there're no internet, user still could use some of its features. So I had to design a local database that help the application work when internet is out. But then, when internet is back, there would be conflicts between local database and server's database. That's why we have to synchronize between 2 databases.

To do that, client size will send a request, in which it attach the most recent update for each table. Server then search for records that have *last_update* greater than or equal the time sent by client, and send them back to client. Client will update it database local based on the reponse it received.

I also made a function that check for network available, and send the generated data during offline time to server. This process is done in the background, to prevent affectation on user's experience.These data would be marked with special status, to distinguish from other data.

5 Knowledge acquired

During the time of this project, I have learnt many things, from non-technical to technical, such as teamwork, communication, code re-used, code convention.

5.1 Design Pattern

Code re-used is an important factor in software development industry. You can work faster or not depends on how good your code re-using skill is. And design pattern is a part of code re-used. In the project, there are some views that were used in many screens. At first I repeat the same implementing process everytime, and the code for each screen grow quite big. That's when I started to use design pattern, and suprisingly, it work well.

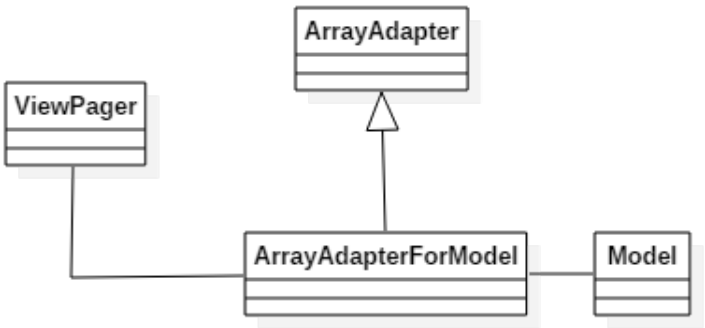

Figure 5.1: Class diagram for normal use of ViewPager

ViewPager is a view provided by Android, it's used to perform slide action when there'are many images, or sections. Above is a class diagram when normally implementing ViewPager. An instance of *Model* is a data object that we want to present it within *ViewPager*. So if we have many class *Models*, we have to rewrite *ArrayAdapter* corresponding to each *Model*. So my first approach is below :

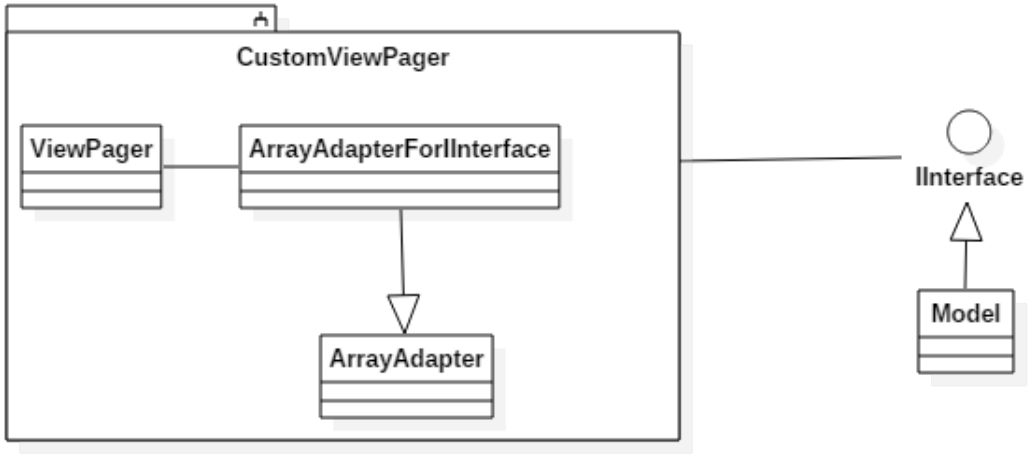

Figure 5.2: First approach

With this approach, my code reduce significantly. I just have to write some more functions defined in *IInterface* for *Model*. But then there's another problem. I have to rewrite some classes *Model* that I had created long before, and everyone were working on it, so modifying existed file need to be prevented. Then I came up with second approach, based on the first one :

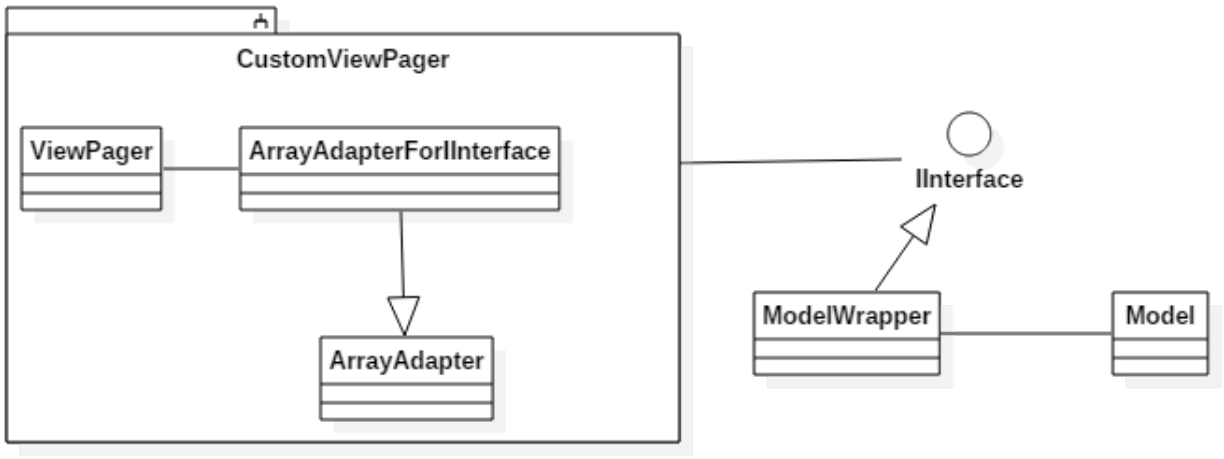

Figure 5.3: Second approach

This time, instead of extending class *Model*, I used a medium *ModelWrapper* which envelop *Model* and implement interface *IInterface*. Then each time I want to apply *CustomViewPager* for a new *ModelA*, I just need to create a class *ModelAWrapper* (normally this wrapper had around 20 lines of code).

And after reviewing this scheme, I just realized that I had just apllied *Adapter Pattern* in *Design Pattern*.

5.2 Custom View

Apart from *Design Pattern*, making custom view and re-used it is a great way of re-using code too. In this project, I had made a textview in which the text is align with the diagonal of the textview's bound, a circle view which has shadow, and a layout with shadow.

Custom component could make your work easier, depend on how convenient it became. In Android, a *Spinner* is a dropdown combo box, which let user choose an item from its dropdown list. The normal way of using such component, is to enter a list of *String*, and when user choose one item, this *Spinner* will return the position of the selected item, and then we can retrieve the corresponding object.

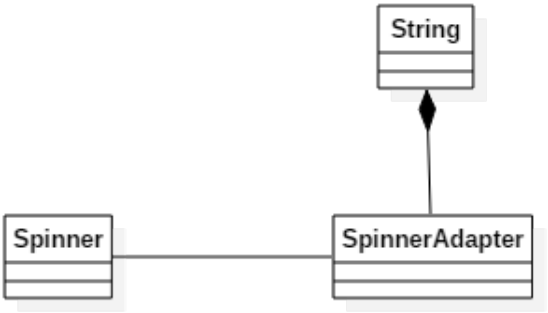

Figure 5.4: Tradition Spinner

Let analize the example below :

I have an array of *Good*, like Candy, Wine, Clothes, etc. If I want to make a *Spinner* using the above method to display good's name, and available quantity, it would be a little more complicated, since each item has 2 values: a name, and a quantity number, but it's still doable. Then if we want to add more functions, such as reduce the quantity, it would mean that we have to clear the data of the *Spinner*, and re-insert the new data each time we updated the data.

So I came up with another solution. Instead of using the tradition *String* as input for *Spinner*, I decide to use *Good*. And every time an instance of *Good* is updated, its automatically update the *Spinner* as well. And moreover, when I selected an item in the dropdownlist, unlike before, I obtain a *Good*'s instance, instead of a *String*. And that make my work faster and easier.

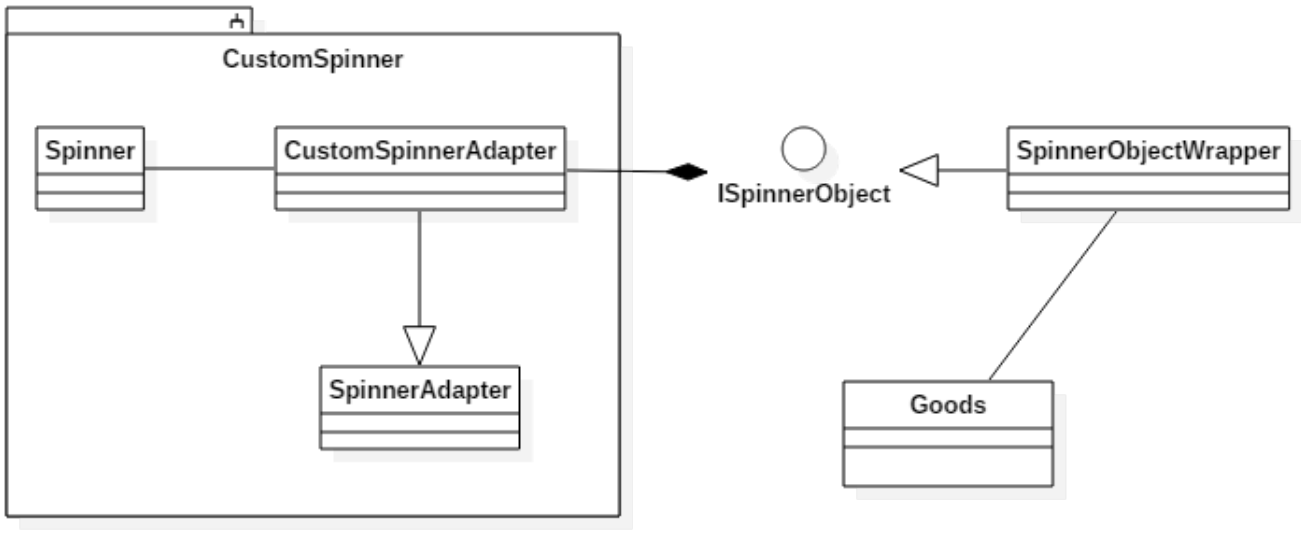

Figure 5.5: My Spinner

6 Conclusion

Thanks to my job in Rainmaker Labs, I came to understand the development procedure, from scratch to finish. I was able to make acquaintance with SCRUM method. I had chances to aplly my university's knowledge, such as algorithm, design pattern, and for that reason, I actually could understand them more than before. I also could analyze a problem by looking it from many perspectives. I was able to learn about Android development, and have a broader view on mobile development (both Android and iOS).

I had chance to experience teamwork in a big group doing a big project. I made acquaintance with many colleagues, and learned a lot from them, since I'm more theoretical than practical. These relationships and knowledge will help me alot in my future career.