2019년12월18일

모바일앱프로그래밍 팀7 보고서

# 실행환경

바로 시작하려면 이와 같은 개체 틀 텍스트를 탭하고 입력을 시작하세요.

## 안드로이드

### SDK Version 26+

### 인터넷 연결 필요

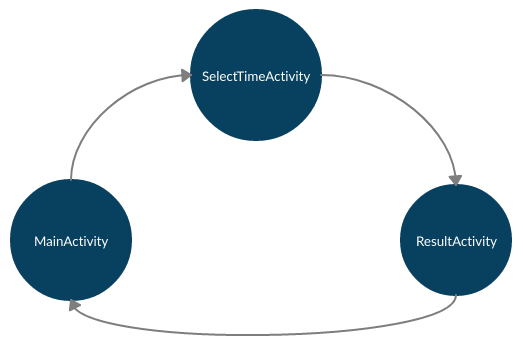
## 서버 (파이썬 Flask)

## We are currently hosting the server on Heroku which the app connects to. Thus it is not needed to locally run the server for the app to work. If you want to try to run the server locally, the required environment is as follows:

### Python 3.6

### Packages: korail2, flask (their dependencies are automatically installed by pip)

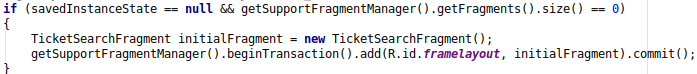
# 앱 구조 및 기능 구현



*Activity Diagram*

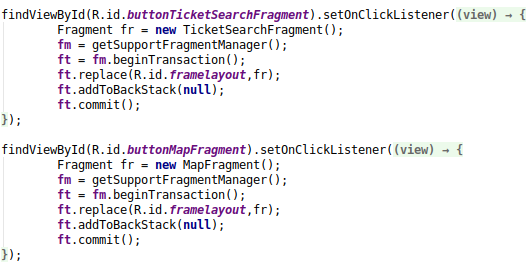
### MainActivity

Our app consists of 3 activities and 2 fragments. MainActivity contains a frameLayout which contains one of the fragments at a time. One fragment shows a map of the 경부선 train line and the other lets the user select a 출발역 and 도착역 to start searching for tickets. Two buttons switch between the fragments.

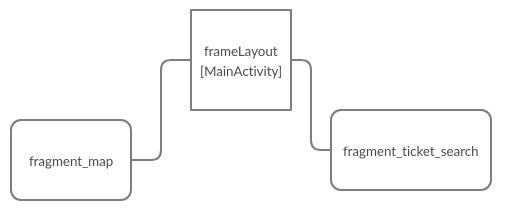


Here, if there is no fragment is loaded (when the app is first launched), the ticket search fragment is added to the frameLayout.



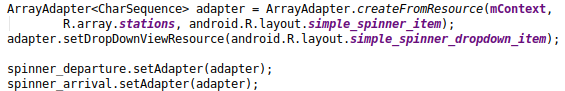


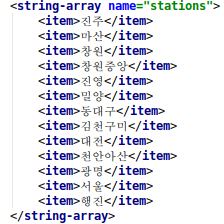
Here, the onClickListener of each button replaces the fragment in the frameLayout with the correct new fragment.



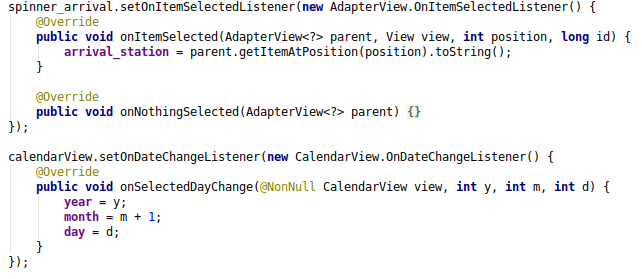
*Fragment Diagram*

The ticket search fragment contains a CalendarView to select the date and two spinners to select the 출발역 and 도착역.

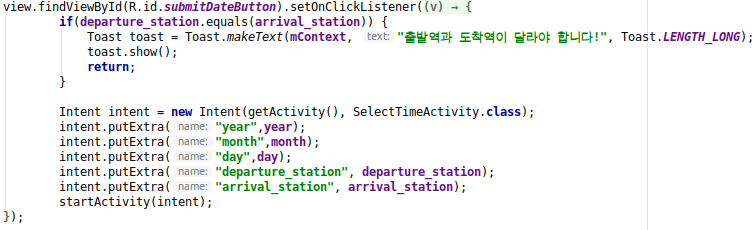




The stations for the spinner dropdowns are set using an adapter. Since both spinners should have the same stations, we reuse the adapter. The names of the stations are stored in res/values/strings.xml to improve maintainability.



We then set appropriate listeners for each widget.

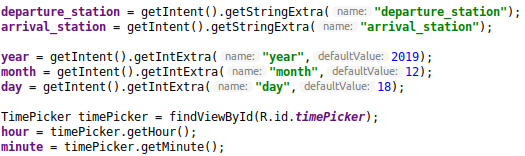


The submit button checks whether the 출발역 and 도착역 are different, else it throws an error. If they're different it creates an intent to go to SelectTimeActivity, adds the selected 검색조건, and starts the activity.

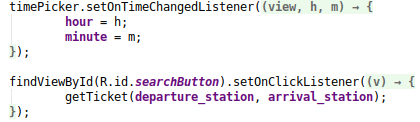
The map fragment only contains an ImageView.

### SelectTimeActivity

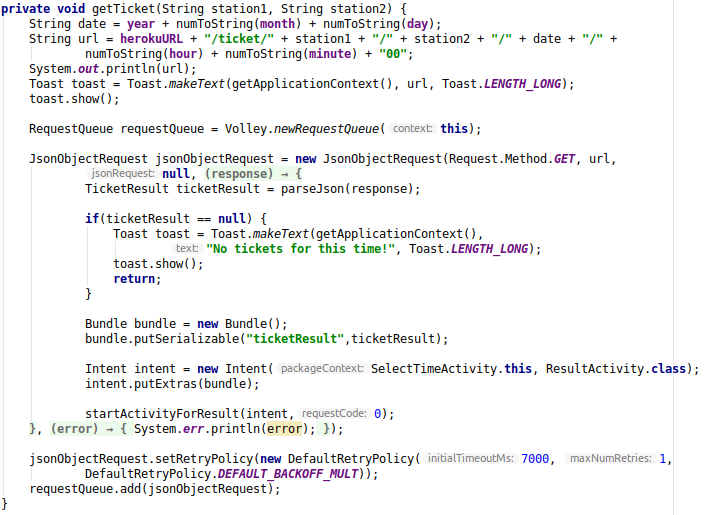
The SelectTimeActivity layout only contains a TimePicker and a submit button.



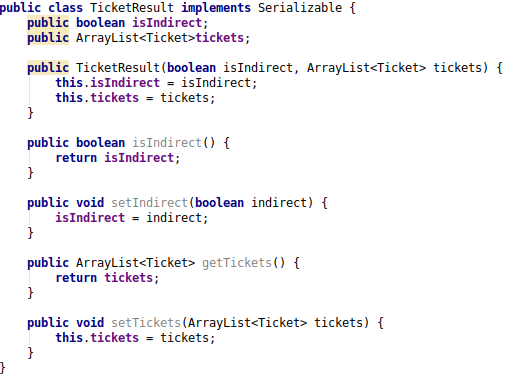
We first get the extras that were added to the intent and then set some default values for the time.

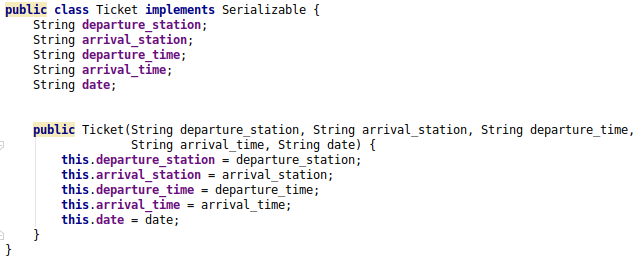


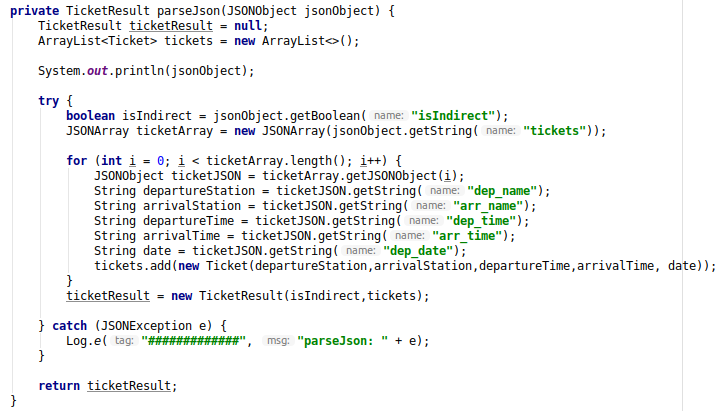
Here we set listeners for each widget. The submit button first needs to make a request to the server to find tickets for the given 검색조건 and then go to the next activity. For maintainability and modularity we have moved this to a separate function.



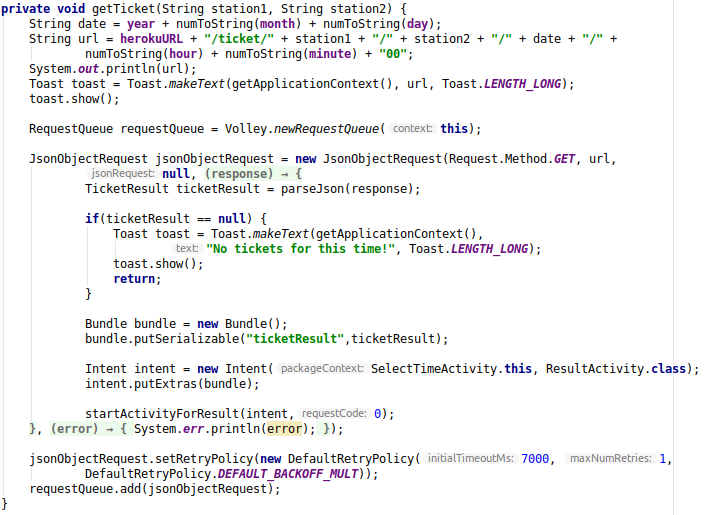
First, we create the correct URL that will give us the ticket. Since the focus of this project lies on the app, the server simply receives the 검색조건 as part of the URL. We created a function numToString to format numbers to fit the Korail API. We then use the **Volley** library to make the request. Since the server will respond with a JSON object, we create a JsonObjectRequest. In the response listener, we first parse the json and transform it into an object of the “TicketResult” class that we’ve created ourselves.





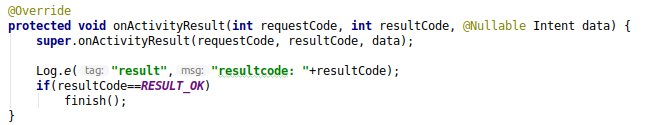


The classes are simple data classes that store the info about the tickets. They implement Serializable to easily store them into the bundle that will be given to the next activity. In the parseJson() function, we first get the JSON array of the tickets, then loop over it, adding each ticket to the list of tickets. This means our app would still work fine if more than 1 or 2 tickets were returned by the server. “isIndrect” is a boolean that indicates whether the traveler needs to change seats or trains during the journey.



After the tickets have been fetched, we check whether they are null and pop up an error if they are. If tickets were correctly returned, we put them into a bundle, add it to a new intent which then starts a new activity.

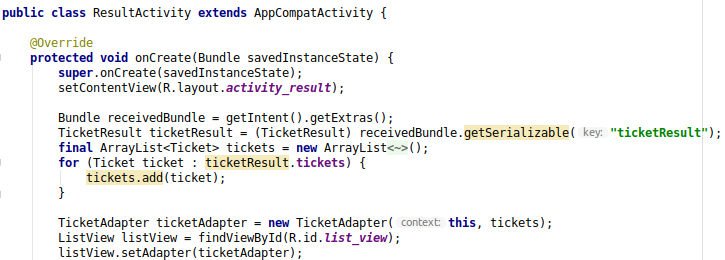
Because the Heroku server is located in the US, response times can sometimes be slow, hence we have set the timeout of the JSON Object request to 7 seconds before adding it to the request queue.



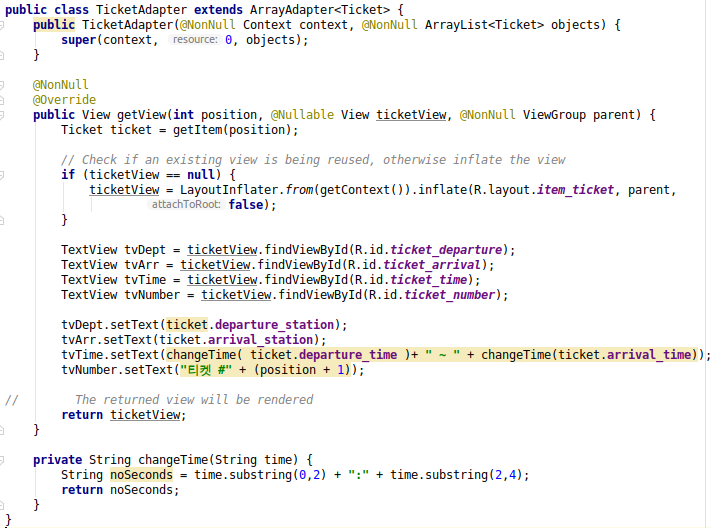
This code ensures that if the next activity is finished by clicking “홈으로”, this activity finishes too, hence returning to the MainActivity.

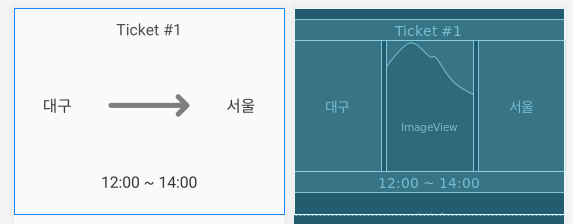
### ResultActivity

The main widgets of the layout of ResultActivity are a TextView to show the date of the ticket, a **ListView** that shows each of the tickets, a **button to open the 코레일의 공식적인 기차예매 앱 "코레일톡",** and a button to directly go back to the MainActivity.



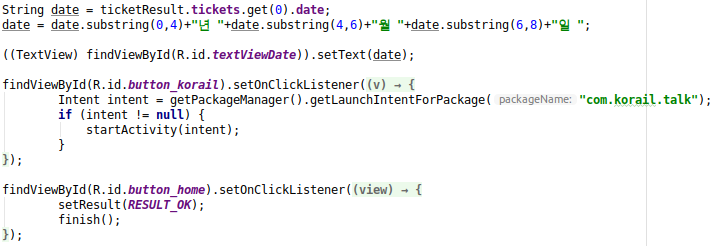
First, we unpack the bundle from the intent. Then, we create a new adapter for the ListView, using the tickets from the bundle. The listview is then set to use this adapter. Since we want to show more than just one string on each item of the listview, we have created our own listview item layout and a custom adapter class that can be instantiated directly from an array of tickets, to improve maintainability and modularity.





The item\_ticket layout uses a few TextViews to display the ticket info and and ImageView for the arrow.

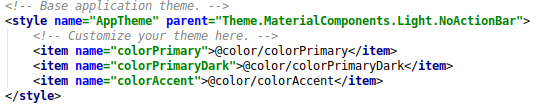
The TicketAdapter has to extend the ArrayAdapter class. It first needs to inflate the view with our item\_ticket layout. It then gets the attributes of the ticket corresponding to the view and displays them in the various widgets.

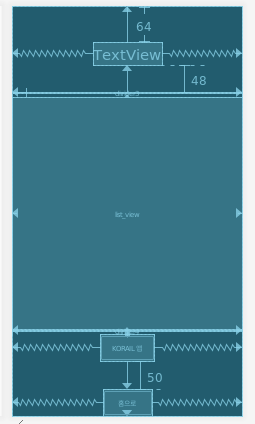
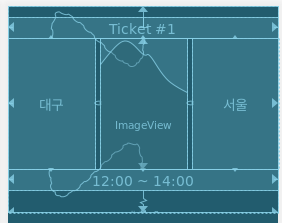


Going back to the remainder of the ResultActivity, the main points here are the button listeners. The first button opens the Korail app. It first creates an intent to the Korail app. The intent needs to be null-checked in case the user does not have the Korail app installed. If it does have the app, the activity is launched, starting the Korail app.

The 홈으로 button simply sets the activity’s result and finishes it.

### UI Design

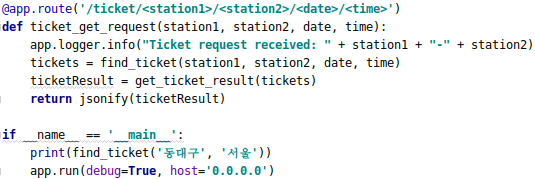


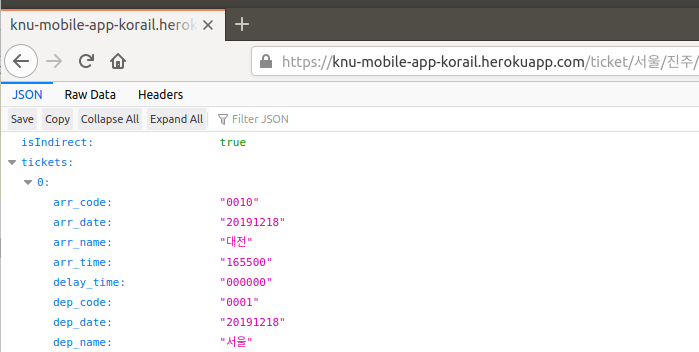


We have used the Material UI theme without an app bar. The layouts are based on ConstraintLayout, with one FrameLayout is used to hold the MainActivity’s fragments.

# 서버 구조 및 기능 구현

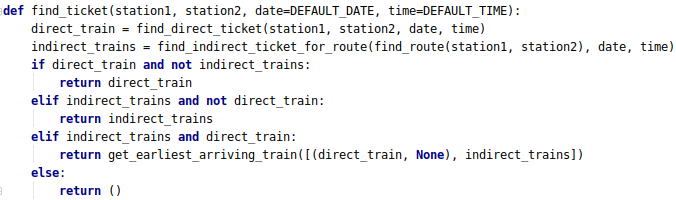
We have chosen to develop a **Flask** server, meaning it is coded in Python. It uses the “korail2” library to access the Korail API.





The only route that our app serves is shown here. It gets the 출발역, 도착역, 출발날짜 and 출발시간 from the URL and uses them to find appropriate tickets. If the fastest ticket available is a direct ticket, this one ticket is returned. If there is a faster indirect ticket, where e.g. the traveler has to change to a different seat during the trip, the two tickets that make up the indirect ticket are returned. This is returned as a JSON.

The rest of the code is entirely the logic to find the fastest indirect and direct tickets and compare them. Since this part of the code is not really related to mobile app programming we will just give an overview. The flow is as follows.



1. The “fastest direct ticket” is searched
2. The "fastest indirect ticket" is searched. This is done by looping over all intermediary stations inbetween the departure station and the arrival station. It then searches for two tickets, one that arrives at the intermediary station and one that departs from it after the first train has arrived. This includes both a. tickets of the same train but different seats. b. tickets of two different trains.
3. If only one of the two types of tickets was found, this one is returned. If both were found, arrival times are compared and the quickest one is returned. If neither was found, an empty tuple is returned.

# 공식 코레일 앱 "코레일톡", 코레일 웹사이트에 비해 우리 앱 유닉한

# 장점

1. 우리 앱은 "매진"인 기차도 탈 수 있게 만든다. 예를 들어, 서울에서 동대구 가는 기차가 "매진"으로 뜬다. 간혹 이런 경우에 그 기차는 서울에서 대전까지 빈 자리 있고 대전에서 동대구까지 다른 빈 자리도 있지만 쑥 빈 자리가 없어서 매진으로 뜬다. 대부분 승객들은 대전에서 자리 옮겨야 해도 더 빨리 도착하는 기차를 택하겠는데 이런 경우를 코레일 공식 앱이나 웹으로 발견할 수 없다는 점이 상당히 아쉽다. 우리 앱은 이 단점을 보충하여 이런 "같은 기차 환승" 티켓을 찾아준다.
2. 코레일 공식 웹/앱으로 "다른 기차 환승"을 찾을 수 있지만, 이것은 따로 선택해 해야 하며 "직통 티켓"과 동시에 볼 수 없다. 이러므로 둘 다 중에 뭐가 더 빠른지 비교하고 싶을 때 불편하다. 게다가 "다른 기차 환승"을 찾을 수 있는 기능이 분명하지 않고 모르는 사람이 많다. 우리 앱은 항상 가장 좋은 티켓을 찾아줘서 이런 문제점들을 해결한다.
3. 코레일 공식 웹/앱은 낡아서 느리고 인터페이스 깔끔하지 않다. 특히 웹이 그런데, 앱도 로딩시간이 상당하고 깔끔하지 않으며 날짜, 시간 검색조건 입력이 우리 앱보다 상당히 불편하다.