Hao Zhang

# Design and Implementation of Social Event Application Based on Android

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor’s Thesis

14 April 2019

| | |
|---|---|
| Author<br>Title<br><br>Number of Pages<br>Date | Hao Zhang<br>Design and Implementation of Social Event Application Based on Android<br><br>34 pages<br>14 April 2019 |
| Degree | Bachelor of Engineering |
| Degree Programme | Information Technology |
| Professional Major | Mobile Solutions |
| Instructors | Kari Salo, Head of Degree Programme |

Over the years, more and more tourists come to Helsinki. In order to let more people know about the events happening in Helsinki area, with the help of the "City of Helsinki" organization, a social events viewing application based on Android platform was born. The main goal of this thesis is to produce a public events information platform based on Android to make it easier for people to find Activities happening in the moment or in the future and make people participate in Activities, integrate into local life and learn about Helsinki culture.

During the development process, back-end data is provided from the organization's open data which covers public data in the Helsinki region. The major application case used in this thesis is a completed social event application written in Kotlin and the specific location of the event will be marked on Google Maps. Besides, RecyclerView is widely used in this application to display specific event information, such as date, price, event publisher and so on.

Design pattern, as an essential part of computer science, is beneficial for keeping project's architecture scalable and testable. This thesis introduces Model-View-View-Model, a design pattern encouraged for Android development. In addition, MVVM design pattern will be demonstrated along with the extracted code from the application case.

In summary, this thesis implements a social event application based on MVVM design pattern and the UI of application conforms to the "Material Design" specification.

| | |
|---|---|
| Keywords | material design, MVVM, Android development, social event |

## Contents

List of Abbreviations

1 Introduction 1

2 Theoretical Background 3

2.1 Android Fragment and Views 3

2.1.1 Fragment 3

2.1.2 ViewPager 4

2.1.3 RecyclerView 7

2.2 Android Four Main Components 8

2.2.1 Activity 9

2.2.2 Service 10

2.2.3 Broadcast Receiver 11

2.2.4 Content Provider 12

2.3 Model View View-Model (MVVM) 12

3 Application Case 16

3.1 Application Description 16

3.2 Material Deisgn in Application 17

3.3 Paper Wireframe 20

4 Implementation 23

4.1 MVVM Implementation in Home Page 23

4.1.1 Model 23

4.1.2 Home Fragment View 24

4.1.3 View-Model 27

4.1.4 Repository 28

4.2 Project Result 29

4.3 GPU Rendering Testing 30

5 Conclusion 32

6 References 34

## List of Abbreviations

| | |
|---|---|
| MVC | Model-View-Controller |
| MVVM | Model-View-ViewModel |
| XML | EXtensible Markup Language |
| JSON | JavaScript Object Notation |
| HTML | HyperText Markup Language |
| API | Application Programming Interface |
| UI | User Interface |
| FPS | Frame Per Second |

# 1 Introduction

With the advent of the mobile Internet era, tourism applications are the product of a new era that has emerged in the context of the rapid development of the mobile Internet and tourism market industries. Some traditional travel issues can be solved by the client on the mobile smart terminal. Nowadays, visitors can get the latest information through various mobile applications, and visitors can make or change the travel plan and itinerary at any time. The customs and culture of the tourist destination can be easily obtained through the corresponding application. It can be said that people's dependence on the mobile Internet has been fully reflected in the tourism market. More and more tourists hope to get desired information and participate in the travel through this type of APP, this is also the core concept of this study’s application case.

Helsinki's tourism industry set a record in 2017, and the overall performance of Helsinki's tourism industry is good. The number of registered overnight stays in Helsinki exceeded 4 million, an increase of 13% over the same period last year. Overnight stays in Helsinki is 5.3 million. The number of foreign tourists staying overnight in Helsinki increased by 15%, while the number of domestic tourists staying overnight increased by 10%. [1] In addition, from the Helsinki Airport flight statistics, in 2017, international passenger traffic to and from Helsinki Airport increased by 11%, and the total number of passenger flights reached an all-time high. Most of the 19 million passengers flying to Helsinki Airport in 2017 were passengers on international flights. The largest increase was 21%, recorded in international transit travel. [2]

Through the above statistics, it can be foreseen that the future of the online travel market in Helsinki is broad, and the tourism client will also play an important role in the development of the online travel market.

The research content of this paper develops a public event platform for tourism based on the Android platform, which can provide destination-based events information service for travel enthusiasts. The application case used in this thesis is developed using the Kotlin language. The paper also introduces some of the components used in the application, such as Fragment, ViewPager, Services, and Broadcast Receiver.

As for the design framework, the traditional MVC is also known as the Massive View Controller, because when the MVC design pattern is used for development, the controller layer becomes bloated and difficult to manage because it carries too much business logic, data logic, and View-related services. This makes it difficult and powerless to modify programs, add features, and review code. As an enhanced version of MVC, the MVVM framework provides a separate View-Model layer to isolate data from UI. With the low coupling characteristics of MVVM, the maintainability and testability of the application are greatly improved.

The tourism industry in Helsinki is booming,  under the circumstance, mobile applications targeting tourists are getting more and more attention. This paper aims to implement a social event application for tourists based on the Android platform. The technologies used in the application, such as the four components of Android, will be introduced one by one. As for UI design, card design that meets the Material Design will also be incorporated into the UI design of the application case. In addition, the overall construction of the project will choose to follow the MVVM framework design. Furthermore, to test the performance of the final application case, GPU rendering test will be evaluated.

# 2 Theoretical Background

## 2.1 Android Fragment and Views

Android contains multiple views for user interface components, such as text-view, image-view, view pager, and recycler-view. Since ViewPager and recycler-view are widely used in the application case of the thesis, these two views will be discussed in this chapter along with android Fragment.

### 2.1.1 Fragment

One of the powerful features added to Android 3.0 (API 11) is Fragment, which is designed to provide more dynamic and flexible UI design support for large screens such as tablets. Fragments must always be managed by the Activity , and their lifecycle is directly affected by the lifecycle of the host Activity. [3]

For example, when an Activity  is paused, all of its Fragments will be synchronized to the paused state. If an Activity is destroyed, all Fragments inside it  will disappear. However, when the Activity  is running, each Fragment can manipulate independently, such as adding or removing them. to Activity , Fragment takes up less memory space. It is possible to design each Fragment as a modular. The diagram below shows the process when a Fragment is added to an Activity :

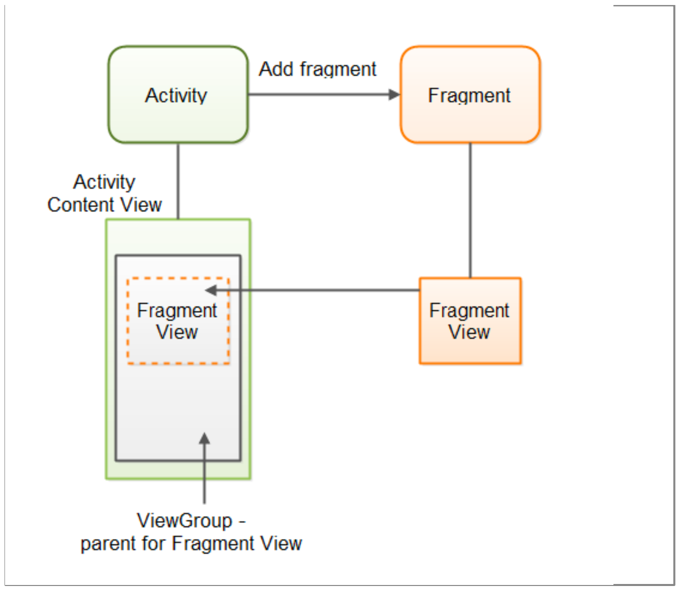


Figure 1. Workflow of adding a Fragment to Activity. Copied from [4]

As illustrated in Figure 1, in the first step, the Activity gets a reference to the Fragment. After that, the Activity gets a reference to the View-Group where the Fragment will be rendered inside of it. Secondly, the Activity adds the Fragment and the Fragment starts to generate its view based on its independent XML file. Finally, returning the created Fragment view to the Activity and the Fragment view means they will be inserted into the View-Group parent for displaying.

### 2.1.2 ViewPager

ViewPager is a component of SupportV4 and it extends from the View-Group class. Hence, ViewPager can be regarded as a view container. In practical working, Viewpager is usually used with Fragment. Also, each Fragment is placed in Viewpager, which is convenient for providing and managing the lifecycle of each page. [5]

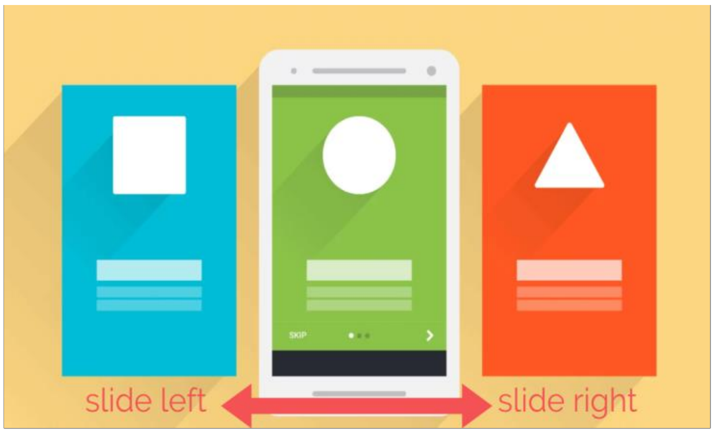
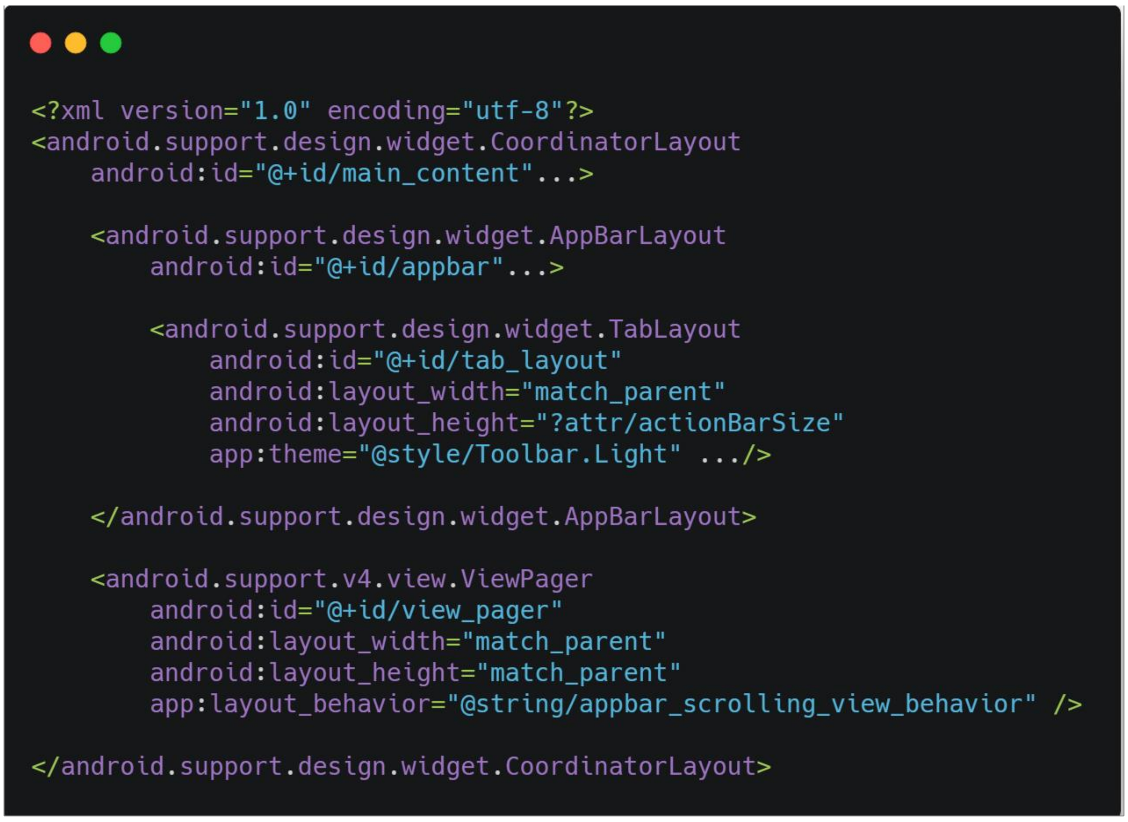


Figure 2. Guide pages on Android application

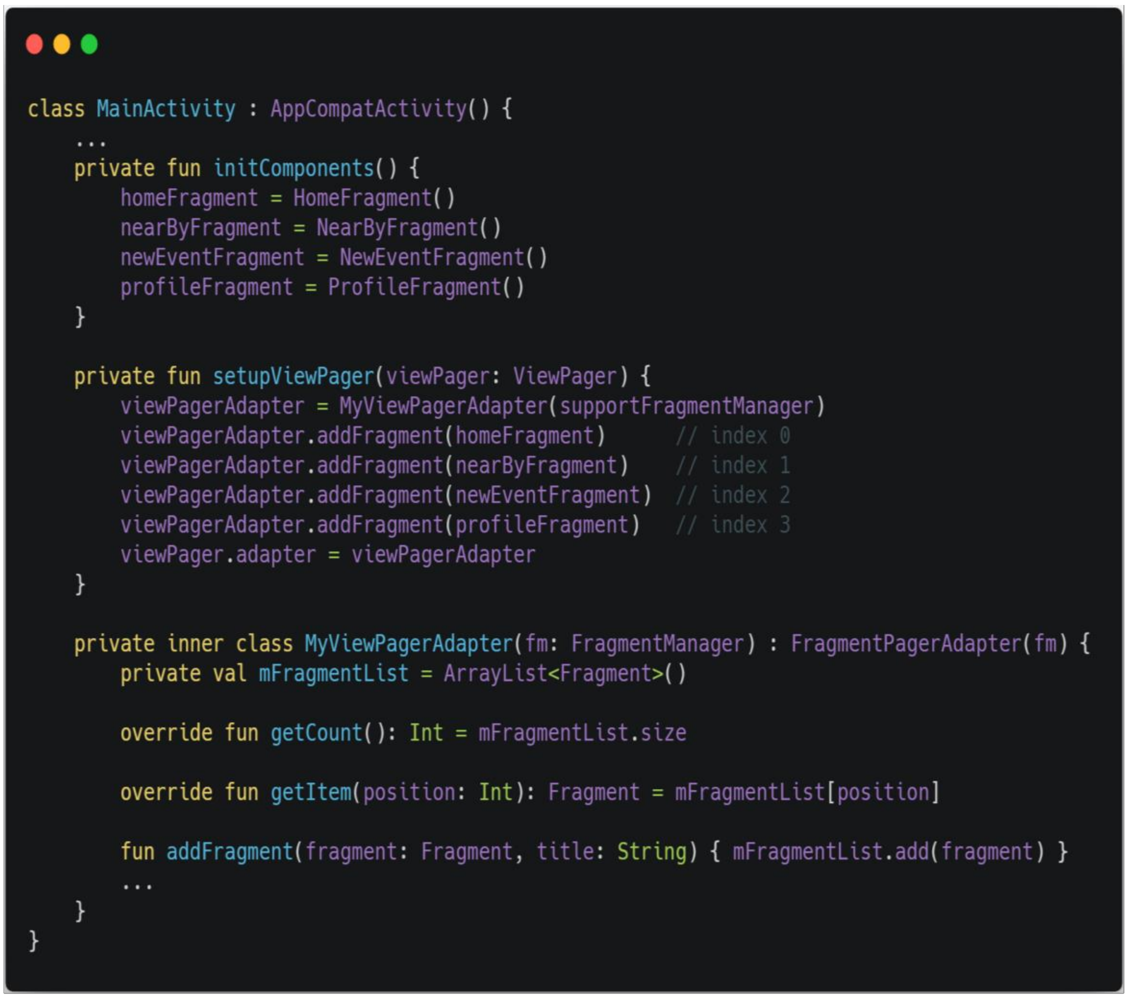
ViewPager, as a view container, has multiple views inside of it which allows the user to switch pages by flipping left or right as seen in Figure 2. Other than making guide pages, the most commonly used of ViewPager is to make an automatic slide show to display the hottest events or advertising exhibitions.



```xml
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    android:id="@+id/main_content"...>

    <android.support.design.widget.AppBarLayout
        android:id="@+id/appbar"...>

        <android.support.design.widget.TabLayout
            android:id="@+id/tab_layout"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            app:theme="@style/Toolbar.Light" .../>

    </android.support.design.widget.AppBarLayout>

    <android.support.v4.view.ViewPager
        android:id="@+id/view_pager"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_behavior="@string/appbar_scrolling_view_behavior" />

</android.support.design.widget.CoordinatorLayout>
```

Figure 3. Layout of MainActivity

In the application case of this thesis, ViewPager is used to combine with Fragments as the main interface framework. As observed in Figure 3, the layout of the main screen is not very complicated. There is a tab bar at the top, and the rest of the page is filled with a ViewPager which will be filled with four Fragments.



```
class MainActivity : AppCompatActivity() {
    ...
    private fun initComponents() {
        homeFragment = HomeFragment()
        nearByFragment = NearByFragment()
        newEventFragment = NewEventFragment()
        profileFragment = ProfileFragment()
    }

    private fun setupViewPager(viewPager: ViewPager) {
        viewPagerAdapter = MyViewPagerAdapter(supportFragmentManager)
        viewPagerAdapter.addFragment(homeFragment)      // index 0
        viewPagerAdapter.addFragment(nearByFragment)    // index 1
        viewPagerAdapter.addFragment(newEventFragment)  // index 2
        viewPagerAdapter.addFragment(profileFragment)   // index 3
        viewPager.adapter = viewPagerAdapter
    }

    private inner class MyViewPagerAdapter(fm: FragmentManager) : FragmentPagerAdapter(fm) {
        private val mFragmentList = ArrayList<Fragment>()

        override fun getCount(): Int = mFragmentList.size

        override fun getItem(position: Int): Fragment = mFragmentList[position]

        fun addFragment(fragment: Fragment, title: String) { mFragmentList.add(fragment) }
        ...
    }
}
```

Figure 4. Main interface framework of the application case

The ViewPager in the main page just serves as a container, mainly responsible for dynamically loading the four Fragments and each Fragment will display distinct page. As shown in Figure 4, inside the setupViewPager() function there are functions that do not do the following things: instantiate a PagerAdapter.object and by invoking addFragment() function to insert 4 Fragments to the ArrayList that inside the adapter. By assigning the customized ViewPager adapter object to the adapter property of the view pager, finally, hook up the PagerAdapter to the ViewPager, The ViewPager class requires an adapter class to provide data to it, as observed in Figure 4, the data source used in PagerAdapter is an ArrayList to carry data items. There is an inner-class that inherits from FragmentPagerAdapter abstract class and getItem() method is needed to be implemented to support instances of Fragment as new pages. Besides the Adapter of ViewPager is compulsory to override the getCount() function, whose return value is the amount of pages the adapter will generate.

### 2.1.3 RecyclerView

RecyclerView is one of the components in Android Lollipop 5.0, after RecyclerView came out. Soon it became a substitute for ListView. Compared with the classic ListView, the RecyclerView is a more progressive, powerful and flexible version of ListView especially when considering the of displaying a list of items with a bunch of data that needs to be modified frequently. [6]

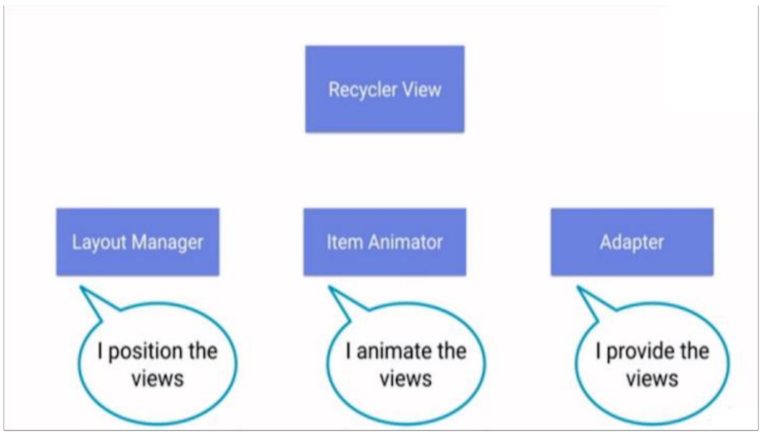


Figure 5. Three extensions for RecyclerView

RecyclerView provides a less coupled way to implement what ListView can do, Therefore, it is widely used in the application case of this thesis. There are three main advantages in RecyclerView:

1. RecyclerView itself does not care about where and how the item is displayed, there is an object called layoutManager, as a massive enhancement brought to the RecyclerView, which provides a list with different structures. As for ListView, it only has the view structure in the vertical direction and the official view does not provide a ListView with a horizontal structure view. Things have changed in RecyclerView, it offers many customization options in RecyclerView, such as

LinearLayoutManager, StaggeredLayoutManager and GridLayoutManager. If none of these above layouts suits the project’s need, developers are unable to create their own layout structure by extending the RecyclerView.LayoutManager abstract class.

2. RecyclerView itself does not care about the effect of item addition and deletion animation. The tranditional ListView has limited support for animations and cannot provide multiple animation effects, but the RecyclerView uses an animator to alter its appearence. Developers can define their own animator object by extending RecyclerView.ItemAnimator. With the RecyclerView.ItemAnimator class, animating the views becomes easier and intuitive. On the other hand, animation makes the display of list elements more in line with Material Design specifications.

3. RecyclerView aims at knowing how to take the view, the views in the list are represented by view holder objects. These objects are instances of a class you define by extending RecyclerView.ViewHolder. Each view holder is in charge of displaying a single item with a view. ViewHolder mode is recommended in ListView, but it is by no means mandatory. As for RecyclerView, it is necessary to use the RecyclerView. ViewHolder class. When implementing a RecyclerView, this class is used to define a ViewHolder object which is used by the adapter to bind ViewHolder with a position. In this way, RecyclerView avoids a heavy operation of finding views by ids every time.

## 2.2 Android Four Main Components

The Android application is mainly composed of four components: Activity, Service, Broadcast Receiver and Content Provider. The development of Android application is inseparable from these 4 to 5 major components.

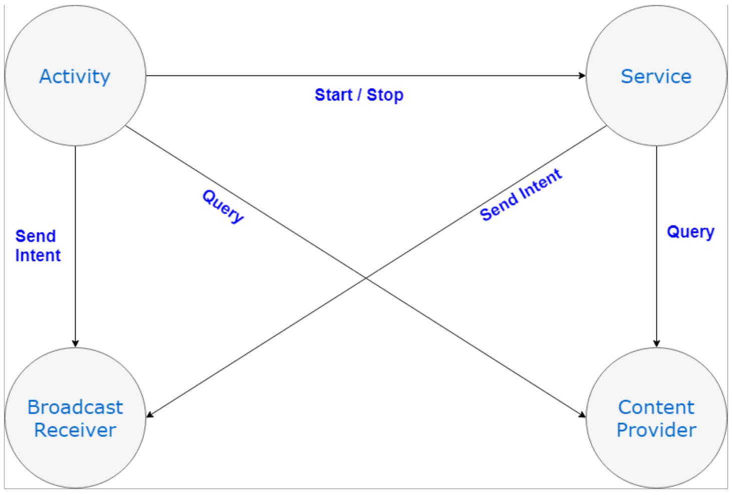


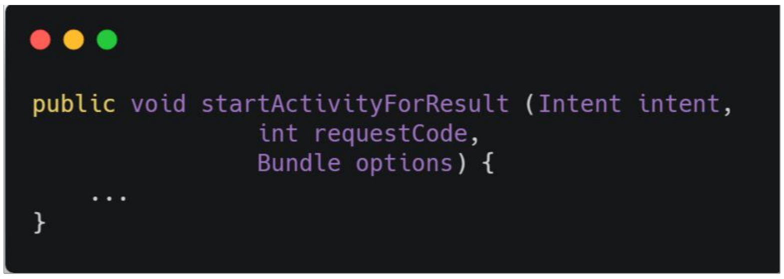
Figure 6. Relationship between Android four major components

In the process of implementing the application case, with the help of the four components. Functionality such as map display, user location monitoring, components communication and so on are realized. The relationships between components are shown in Figure 6. Each component will be specifically introduced in the following paragraphs.

### 2.2.1 Activity

The most frequently used in Android is the Activity  component, which is one of the most basic modules in Android. Activity  takes care of generating a window for placing UI with setContentView  ---- (View) method. Activities are usually presented to the user as a full screen window. These interfaces basically belong to or depend on the Activity . In one Android application, one Activity  usually refers to a screen of the mobile device. If compare a mobile device to a browser on the web side, the Activity  is similar to a web page. Similar to the Input, H1, Span and other elements contained in a HTML page, Button, Checkbox and other controls can be added in the Activity. From another perspective, Activity and web page have a lot in common in terms of concepts.

An Android application has at least one Activity and switching between different Activities is allowed. For example, pressing a Button may jump to other Activities. When new Activity is loaded, the previous Activity will turn into paused state and pushed into the top of the Activity stack. The previously opened Activity can be shown when the user presses the back button on the phone. There is a choice of Activities that are not necessary to keep, thus, those Activities can be selectively removed from the Activity stack. A little different from web page jumping, jump between Activities may return values. For example, from Activity  A to Activity  B, then when Activity  B finishes running, it may give Activity  A a return value. As shown in Figure 7, Launch an Activity for which the previous Acticity would like a result when it finished.



```
public void startActivityForResult (Intent intent,
                                    int requestCode,
                                    Bundle options) {
    ...
}
```

Figure 7.  Start an Activity and wait for a result

In addition, the Activity  component is designed to have a system control in the form of a lifecycle. The Activity mainly has: onCreate, onStart, onResume, onPause, onStop, onDestroy, and onRestart lifecycle methods. When developers design the functionality of an application, they only need to match the lifecycle according to the business to determine what needs to be done at different lifecycle methods.

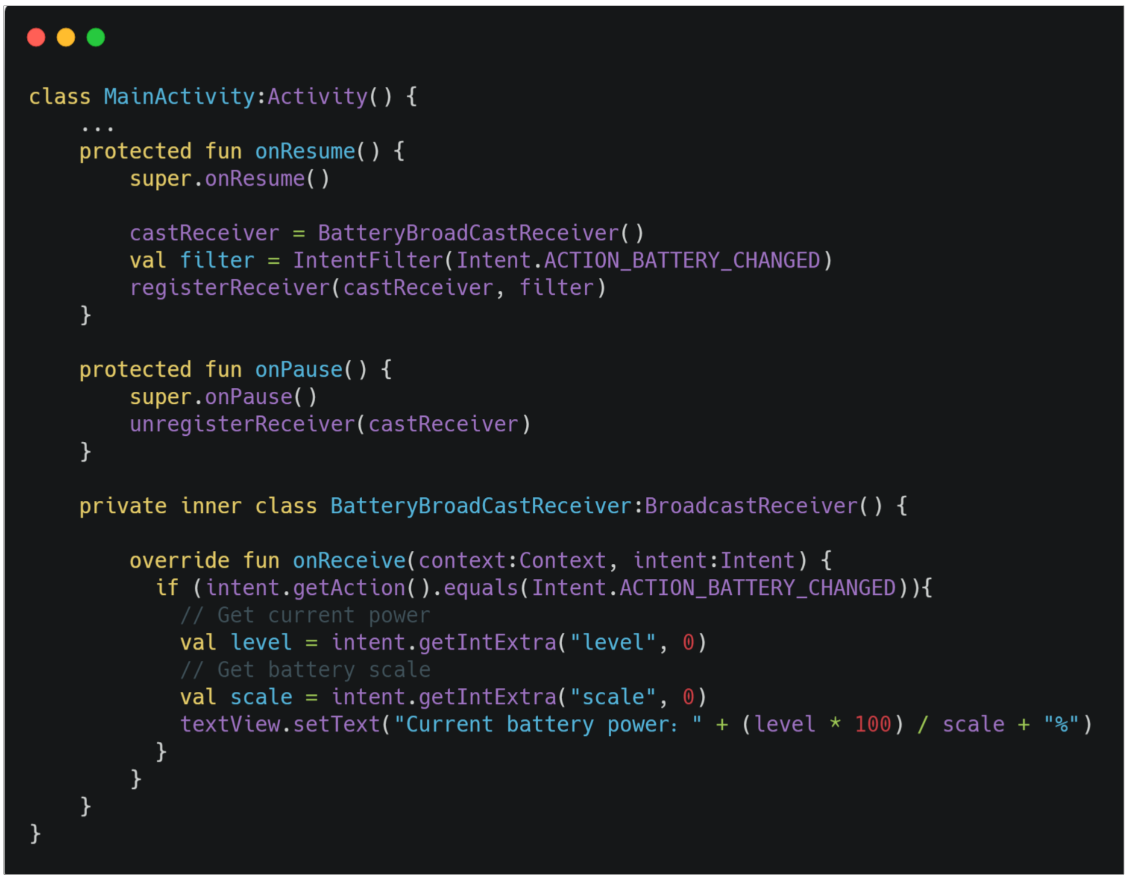
### 2.2.2 Service

Service is a solution for running programs in the background process. It is very suitable for tasks that do not need to interact with users but also run for a long time. The operation of the service does not depend on any user interface, but can be used to interact with other components, even if the current application is switched to the background, or the user opens another application, The service is still up and running because the service is not running in a separate process, but rather on the application process in which the service was created. When an application process is killed, all services that depend on it will stop running.

The service can be used in applications with multiple occasions, such as detecting changes in the file on the SD card, or recording changes in the location of your geographic information in the background. The commonality of the above cases is that these operations do not require an interface display through the service. A typical case is music application in which an Activity starts a service running on user interaction, with the service probably downloading music from the web server. The user can continue to interact with the Activity while the service runs since it executes in the background.

### 2.2.3 Broadcast Receiver

Broadcast Receiver is a component that can be regarded as a messaging system across applications and outside of the normal user flow. Similar to the Service component, it has no user interface. The principle behind of Broadcast Receiver is kind of like publish-subscribe pattern in Youtube. In youtube, one user can subscribe to multiple videos to get an update reminder. Similarly, the user can also post videos and be subscribed to by others.



```kotlin
class MainActivity:Activity() {
    ...
    protected fun onResume() {
        super.onResume()

        castReceiver = BatteryBroadCastReceiver()
        val filter = IntentFilter(Intent.ACTION_BATTERY_CHANGED)
        registerReceiver(castReceiver, filter)
    }

    protected fun onPause() {
        super.onPause()
        unregisterReceiver(castReceiver)
    }

    private inner class BatteryBroadCastReceiver:BroadcastReceiver() {

        override fun onReceive(context:Context, intent:Intent) {
          if (intent.getAction().equals(Intent.ACTION_BATTERY_CHANGED)){
            // Get current power
            val level = intent.getIntExtra("level", 0)
            // Get battery scale
            val scale = intent.getIntExtra("scale", 0)
            textView.setText("Current battery power: " + (level * 100) / scale + "%")
          }
        }
    }
}
```

Figure 8. Example of detecting battery change using Broadcast Receiver

Back to adnroid, after the broadcast receiver is defined by one application, other applications call it according to its defined rules and send a broadcast to it. After receiving the broadcast, the received data can start an Activity or a Service to start the follow-up function. In additon, an application can define multiple broadcast recipients, either dynamically by inheriting from the BroadcastReceiver class as illustrated in Figure 8 or statically registered in the AndroidManifest file.

### 2.2.4 Content Provider

In Android, the protection of data is very strict. Except for the data placed in the SD card, the database, files and other contents held by an application are not allowed to be directly accessed. Android certainly does not really make every application an "island", it has a ”window” for all applications to interact with data from other application, this ”window” is the Content Provider.

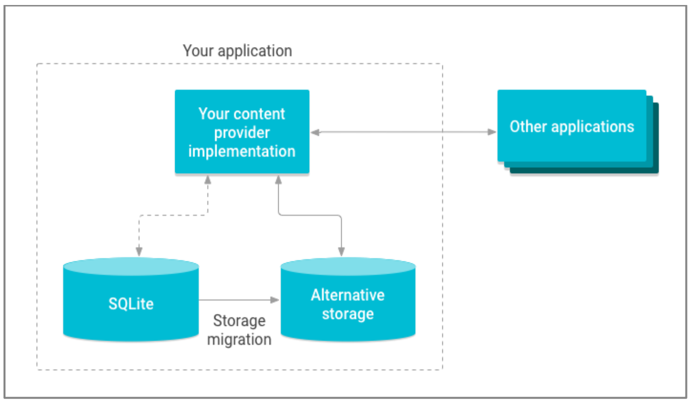


Figure 9. illustration of migrating content provider storage. Copied from [7]

The Content Provider is an access plan for third-party application data provided by Android. Content providers are used to share a given data set of an application to other applications, and other applications can obtain data from a content provider. The relational database SQLite provided in Android system will create its own data set for each application level, and only the content provider can share data between each application. Content Provider is able to modify application data store implementation without affecting other existing applications that rely on accessing data. In this case, only Content Provider is affected, not the application that accesses it. For instance, the SQLite database can be replaced with an alternate storage, as shown in Figure 9. [7]

## 2.3 Model View View-Model (MVVM)

In the process of large-scale software system development, if developers do not pay attention to the architecture of the program, the modular design of the code and the decoupling of the function modules, this may lead to some undetectable and difficult to locate errors. Especially these undetectable problems occur when the program reaches a certain scale. Due to the high degree of coupling between functions and services, it can lead to difficulties in reconstruction, finally, it can only be achieved by redesigning the software. In order to improve development efficiency and lay the foundation for future software maintenance. it is crucial to define a reasonable program architecture in the early development process.

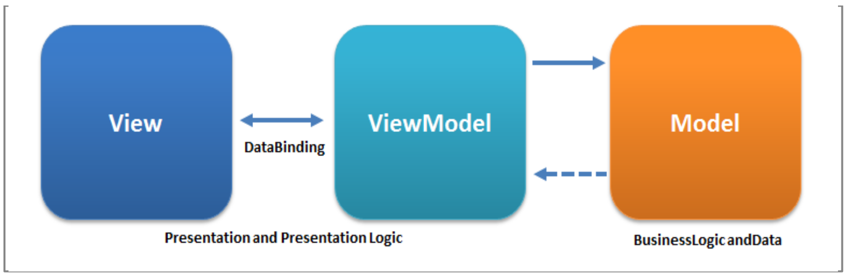


Figure 10. The relationships between View, View-Model and Model (Copied from David Britch (2017) [8])

The MVVM framework model was first proposed by Microsoft and applied in software development. With the rise of mobile development, MVVM is also encouraged by Google, as shown in Figure 10, MVVM is similar to the well-known MVC pattern to some extent because Model layer and View layer are relatively identical. The only difference is C (controller) and VM (view model).

1. View

   The View layer is only responsible for UI-related work, UI and data are strictly separated. It does not perform logical processing and update the UI in the Activity or Fragment. UI renew is implemented by changing the data source in the View-Model layer. Simply put: In View layer, the main role of View is informing the View-Model about the user’s actions

2. Veiw-ModelDid yowiåt

   The View-Model layer does exactly the opposite of the View layer. View-Model only does things related to business logic and data source. It does not do anything related to UI. The View-Model layer does not hold any reference to the UI element and it is unable to update the UI by reference to the UI element. Simply put: View-Model is not tied to the view, the main role of the View-Model layer is to wrap the model and arrange observable data needed by the view.

3. Model

The biggest feature of the Model layer is that it is assigned the responsibility of data acquisition. View-Model is used with Model to capture and save data. The Model provides a data acquisition interface for the ViewModel to invoke, and through data transformation and manipulation and finally mapping to the properties of a UI element of the View layer.

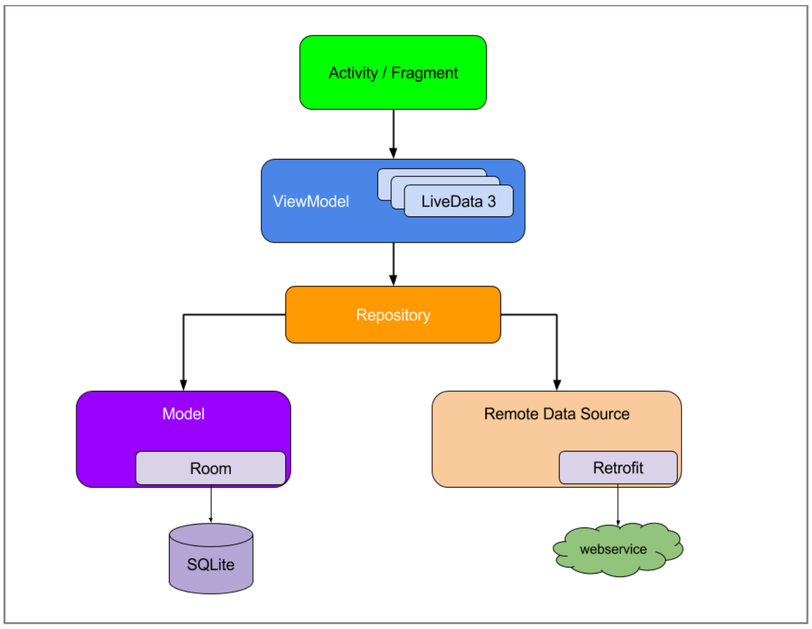


Figure 11. Google recommended application architecture (Copied from Google Developers guides [12])

However, In the actual development process, a Repository layer is usually added between the Model layer and the View-Model layer as shown in Figure 11, this Repository layer is equivalent to a mediator between different data sources, such as persistence models, web services, and caches. The main role of the Repository module is to handle data operations, they offer a clean API so that the rest of the application can easily retrieve this data and they aslo know where to get the data from and what API calls to make when data is updated. Moreover, the structure diagram shown in Figure 11 is also the official application structure recommended by google [12].

in summary, each component depends only on the component that one level below it. For instance, Activities and Fragments only depend on the View-Model layer and the Repository layer is the only module that depends on many data sources; in this example, the Repository relies on a persistent data model and a remote back-end data source [12].

Both the unit test of the UI and the unit test of the business logic are low-coupling. This greatly improves the testability of the application. In addition, due to the low coupling of the MVVM framework, team development is more convenient, such as one developer handling business and data, and another developer responsible for specialized UI processing.

# 3 Application Case

## 3.1 Application Description

The City of Helsinki organization supplies backend database called “Open Data” and API interface to provide data for this project and our project team is mainly responsible for front-end development based on the Android platform.

The core idea of the application is to provide a platform for people living in Helsinki or traveling in Helsinki to find real-time public events. The project adopts the MVVM framework design, and the client interface design follows the specification of material design, MVVM architecture has been described in previous chapter and Material Design is further discussed in the following sections. Moreover, the client is programmed with Kotlin and uses some third-party libraries, such as Retrofit, Google-Map and so on.  The main functions of the project are as follows:

1. List upcoming events and display specific event information such as time, location, price, publisher, etc.

2. Show the specific location of the event as a marker on the Google Map

3. Filter result by category, age, price, postal code, date and so on

4. Create peronal event and post it to Open Data

5. Display multiple itineraries to the event location

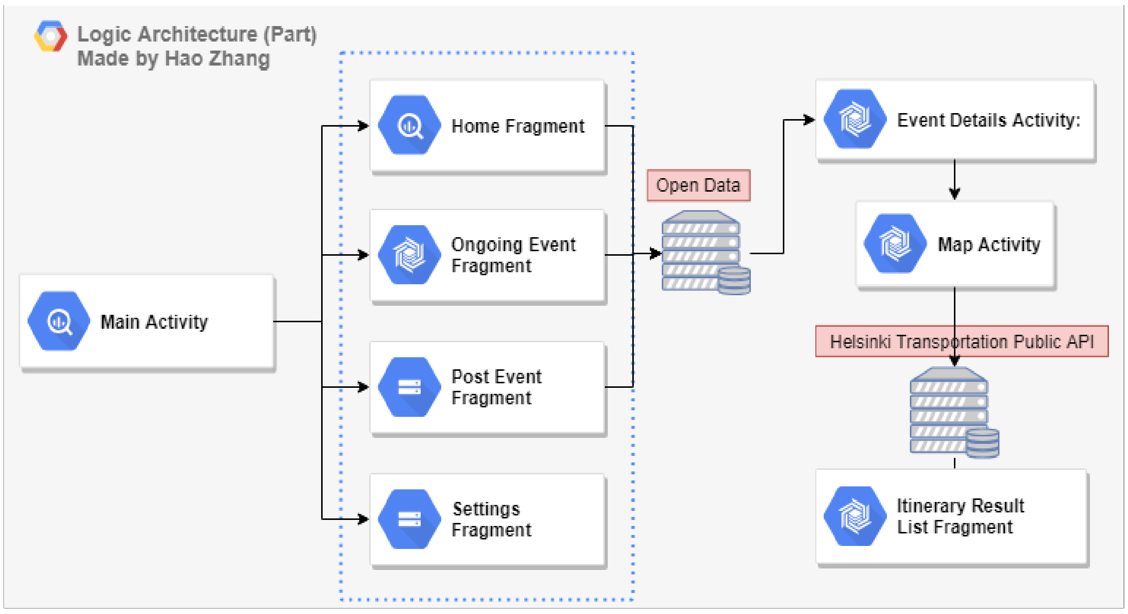


Figure 12. Part of the logic diagram of the project

As shown in Figure 12, the main interface MainActivity uses a ViewPager as a container to load different Fragments. The HomeFragment is mainly used to display activities of a specific theme. The ongoing event fragment is used to display events that will take place in the Helsinki area in the coming week. Moreover, by clicking on each activity card will get data by requesting open data and jump to the event details page. In the event details page, users can click the map card shown in Figure 15 to get the specified route plan through the API of Helsinki public transportation. Similarly, users are able to post an event to the public. SettingsFragment is the place where users can change the theme, language and other options. It should be noted that the above Figure 12 is only a part of the overall logic diagram. It mainly lists the usage of ViewPager, RecyclerView, Activity and other components are listed in Chaper 2. S  ince the thesis focuses on the implementation of the MVVM framework in Android applications, thus, Manager classes Implementation , such as networking class, data persistence class and so on, will not be covered in this thesis.

## 3.2 Material Deisgn in Application

Application's interface design is kind of like the industrial design in industrial products, which is an important selling point of the product. The criteria for verifying an interface are neither the opinion of a project development team leader nor the result of a project member's vote, but the user's feelings. An application with a reasonable and pretty interface will not only bring a comfortable visual enjoyment, but also bring people closer to the product. The UI design of this thesis's application case uses Material Design as guideline to create a consistent interface and user experience.

In the period of disorder, Android is full of the style of freedom, since Google did not impose any restrictions about design guidelines. Developers can arbitrarily upload applications designed with their own ideas to the store without review. At that disorder period, Android is like a wasteland opened by Google that everyone can use at will. The products of that period had no user experience, and users were forced to adapt to different interaction styles.

However, over the years, In the rapid development of mobile development, interface design is getting more and more attention, especially after Google released Material Design on Google I/O conference in 2014. Google rethinks the user experience on the Android platform, trying to bring the experience and  physical feedback of the real world to the screen. At the same time, remove the impurities and randomness in reality, retain its most primitive and pure form, spatial relationship, change and transition. Finaly, restore the most realistic experience and achieve a simple and intuitive effect [9].

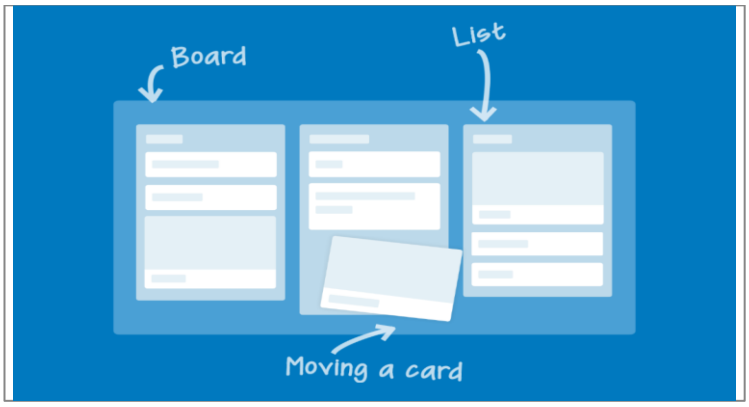


Figure 13. Cards design used in Trello. Copied from [11]

Card, as one of the component in Material Design, borrows the features and concepts of cards in the real world, it has been widely used in mobile development. For example, Facebook's feed uses cards with infinite scroll loading to carry a quick preview of events. In addition, Trello's task list also uses a card design, which is very helpful for users to manage different tasks as shown in Figure 13.

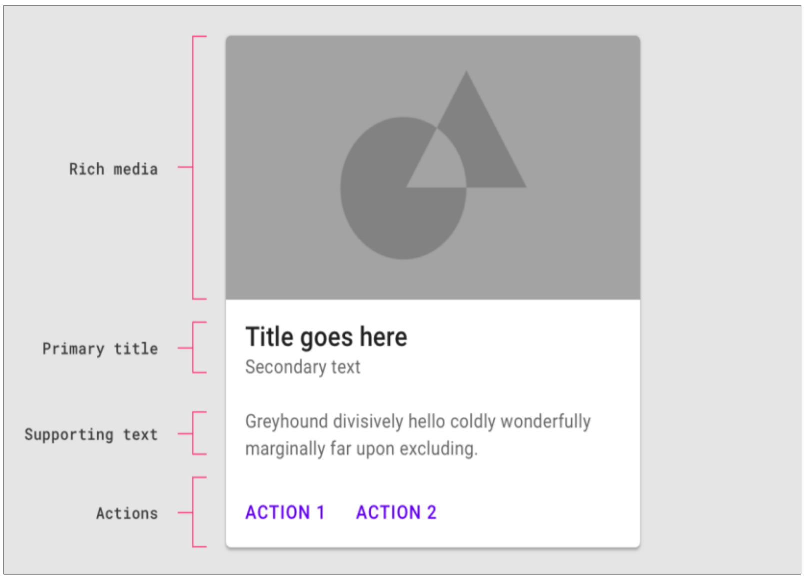
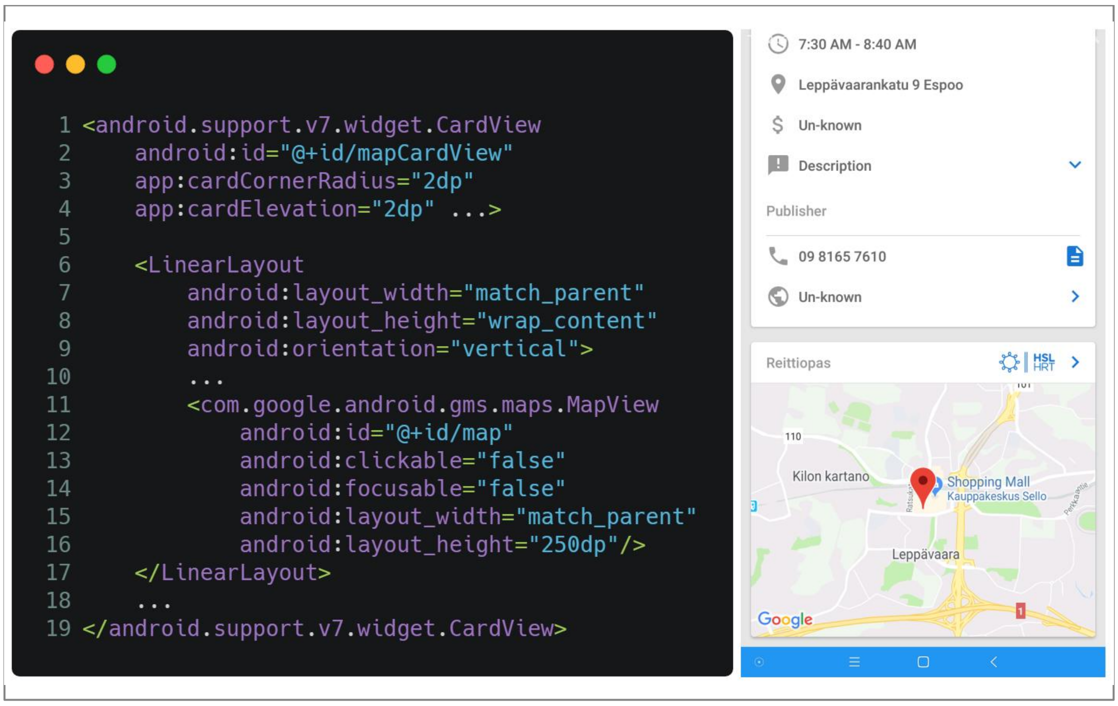


Figure 14. Structure of a single card component in Material Design. Copied from [10]

Usually the cards in the UI are rectangular, which carries different elements such as images, text, links, buttons, and so on. Different elements perform their duties in different cards as observed in Figure 14. Due to the independence of each card, this makes the information more portable and easier to share. Moreover, gestures are the main interaction in the mobile design, interesting gestures and interactions can create a fun and enjoyable experience with cards. For example, choose whether you  like the content in the card by swiping the card left or right, and organize the card with a long press click.



```xml
<android.support.v7.widget.CardView
    android:id="@+id/mapCardView"
    app:cardCornerRadius="2dp"
    app:cardElevation="2dp" ...>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">
        ...
        <com.google.android.gms.maps.MapView
            android:id="@+id/map"
            android:clickable="false"
            android:focusable="false"
            android:layout_width="match_parent"
            android:layout_height="250dp"/>
    </LinearLayout>
    ...
</android.support.v7.widget.CardView>
```

7:30 AM - 8:40 AM

Leppävaarankatu 9 Espoo

Un-known

Description

Publisher

09 8165 7610

Un-known

Figure 15. Sample of map card in the application case

In the application case of this paper, the card layout design is used in the project. Meanwhile the card is used together with RecyclerView, each card is used as an item in the RecyclerView to form a Facebook-like event stream and timeline. Different content is loaded into different cards, for example, some cards display specific information about the activity, while some cards are used to load a map to display the location information of the event as shown in Figure 15.

## 3.3 Paper Wireframe

During the development process, because there are no professional designer in our group, therefore, I not only bear the responsibility of development, but also shoulder the responsibility of product design. Wireframe, Prototype, Mockup are three common prototyping tools for visualizing product concepts, I chose wireframe to show page design to team members.

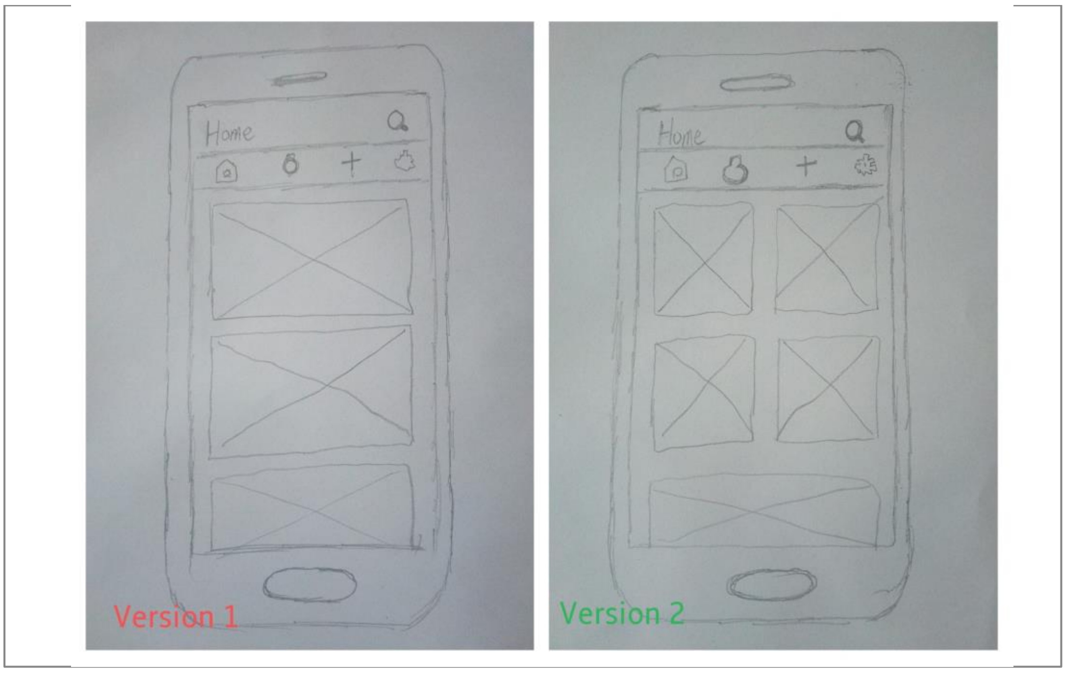


Figure 16. Design of different versions of the homepage

Wireframe is a communication tool used to express product concepts, product architecture, content prioritization, page layout, and operational logic. It focuses on the main functions of the product and how it should behave in different scenarios, so visual elements should be as simple as possible, even they can be ignored. In the early stages of the application case, our team used wireframe to quickly visualize product concept and let team members quickly understand it for further discussing and collecting feedback from each team member. The feedback is collected to improve the product, so some of the wireframes are probably not the final version of the product, and the feedback gathered in the discussion may make the product change a lot, which is why visual details do not need to consider too much.

About the design of homepage, in the first version, the content cards occupy almost the width of the screen as shown on the left side of Figure 16. Although this can maximize the display of the cover image of the event, but it also causes a reduction in the displayable content card within a certain screen space. Especially when there is a lot of content, users need to keep sliding down to load more content, which is not conducive to user experience. After a group discussion, card size was changed to a small card to load more content on the same screen. As observed on the right side of Figure 16, the actual card size is half of the original version.

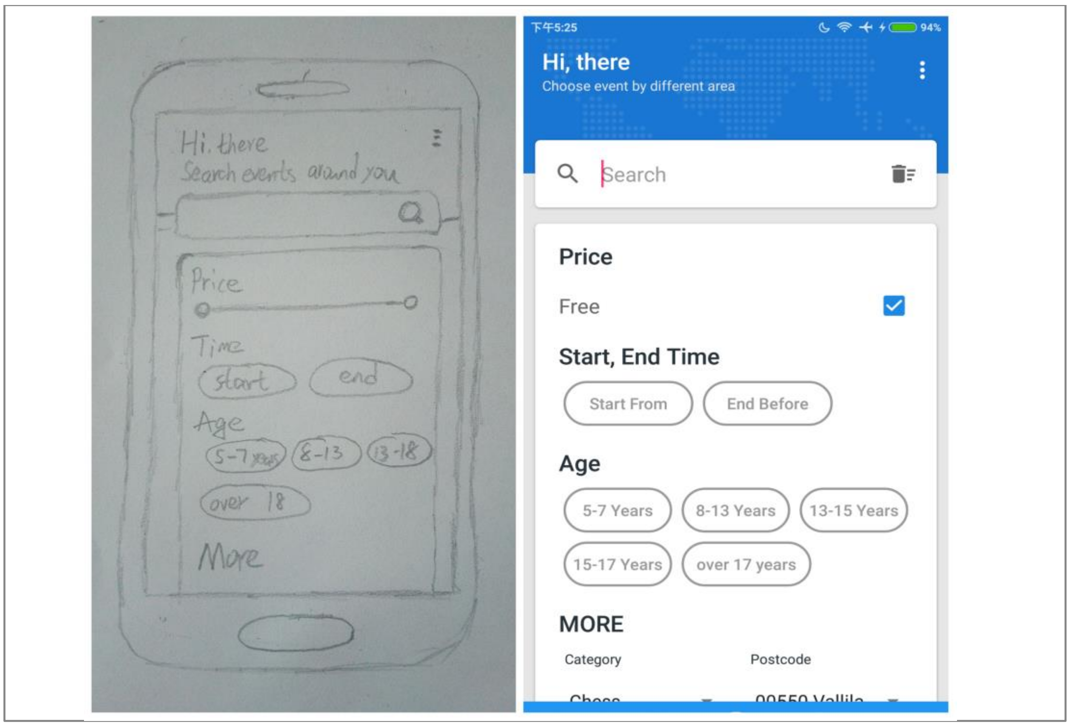


Figure 17. Paper prototype and actual display effect of the search page

The wireframe of the search interface is similar to the actual product in the end, as shown in Figure 17, there has been no major change. Below the search box, there is a card that contains multiple filtering conditions. There is a slight change in the price filter. The original version is to display a slider to let the user select the price range. If the user wants to search for free activities, user needs to slide the points at both ends of the slider to the position of 0 at the same time. In the final version, the price filter displays a checkbox by default. The user only needs to check this box to indicate that the search for free activities, and only when the checkbox is unchecked, the slider for displaying the price range will be displayed.

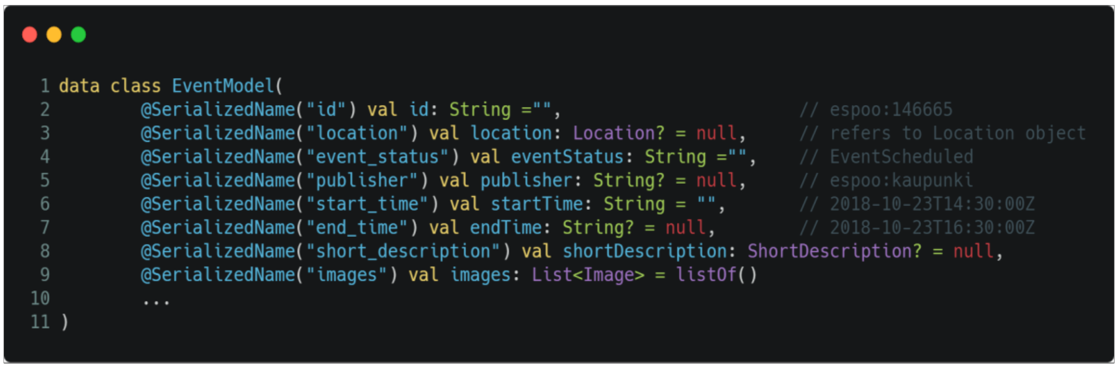
# 4 Implementation

## 4.1 MVVM Implementation in Home Page

Since this project is developed using the MVVM framework, there are multiple MVVM blocks in the application, for study purpose, only one MVVM block that related event details will get a concrete explanation as the study case. this block includes the following classes: EventModel, EventRepository, HomeFragment and HomeFragmentViewModel which represent Model, Repository, View, ViewModel respectively. For those classes that are not related to MVVM, such as management classes, tool classes, etc., will not be mentioned in this paper.

### 4.1.1 Model

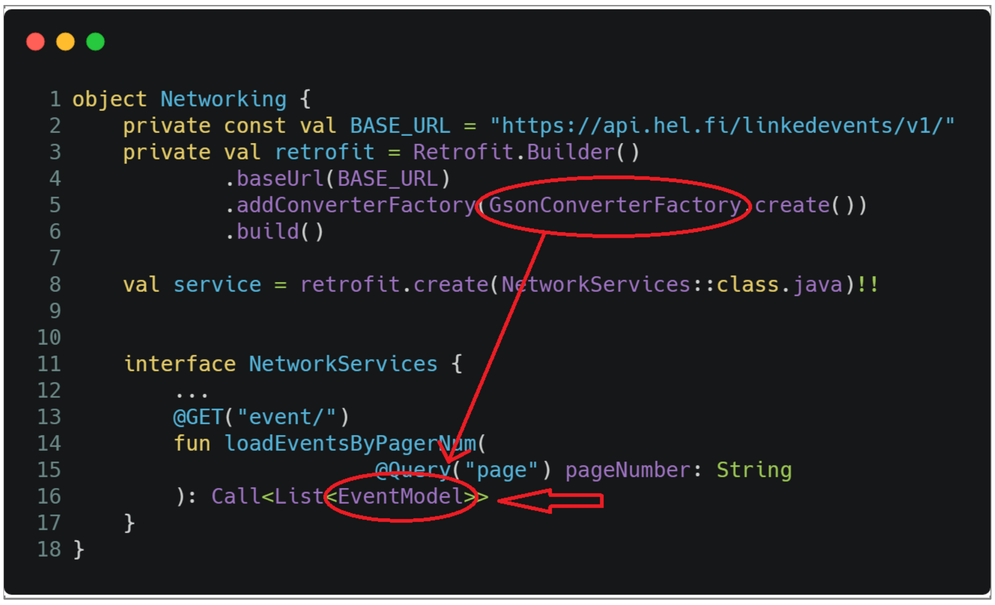
The data displayed on the home page are encapsulated into multiple data models. In the event model, as shown in the Figure 18. since there are more than 30 fields in event data model, only a few important parameters are displayed in the Figure.



```
data class EventModel(
        @SerializedName("id") val id: String ="",                              // espoo:146665
        @SerializedName("location") val location: Location? = null,            // refers to Location object
        @SerializedName("event_status") val eventStatus: String ="",           // EventScheduled
        @SerializedName("publisher") val publisher: String? = null,            // espoo:kaupunki
        @SerializedName("start_time") val startTime: String = "",              // 2018-10-23T14:30:00Z
        @SerializedName("end_time") val endTime: String? = null,               // 2018-10-23T16:30:00Z
        @SerializedName("short_description") val shortDescription: ShortDescription? = null,
        @SerializedName("images") val images: List<Image> = listOf()
        ...
)
```

Figure 18. Sample of event model used in Home Page

The meaning of each field in the event model is self-explanatory. Notice that the location field represents an object containing information about the location. There is a location id in the location object. The location id can be used to obtain the location details, similar to the joint query in the database.



```kotlin
object Networking {
    private const val BASE_URL = "https://api.hel.fi/linkedevents/v1/"
    private val retrofit = Retrofit.Builder()
            .baseUrl(BASE_URL)
            .addConverterFactory(GsonConverterFactory.create())
            .build()

    val service = retrofit.create(NetworkServices::class.java)!!


    interface NetworkServices {
        ...
        @GET("event/")
        fun loadEventsByPagerNum(
                    @Query("page") pageNumber: String
        ): Call<List<EventModel>>
    }
}
```

Figure 19. The usage of Model class in Retrofit

The data retrieved from the backend server is returned to the client as JSON string format. As for Model classes, the main role in the entire application is to map values of the JSON. In the project, a third-party library called Retrofit is used to process the network request. As shown in Figure 19, when building a Retrofit object, a GsonConvertor object needs to be passed in to initialize the Retrofit object. Retrofit will use the converter library chosen to handle the deserialization of data from an object. In the NetworkingServices interface, there is a loadEventsByPageNum fuction which return value is a parameterized Call<T> object to perform GET HTTP request, in this example, the return value is Call<List<EventModel>>. EventModel class is  used to map the response JSON KEY parameters to their respective variables, for example, the value of key endTime will be mapped into endTime property in the Event Model class

### 4.1.2 Home Fragment View

The recommended way to communicate between View and View-Model is the observer pattern, which is available in an observable way provided by LiveData or other libraries. In the project, LiveData was chosen to build a bridge between these two layers, LiveData is an observer wrapper over a model data that is lifecycle aware, for example: LiveData<MutableList<EventModel>>, a list of the Event Model classes is wrapped as observable data as illustrated in Figure 20.

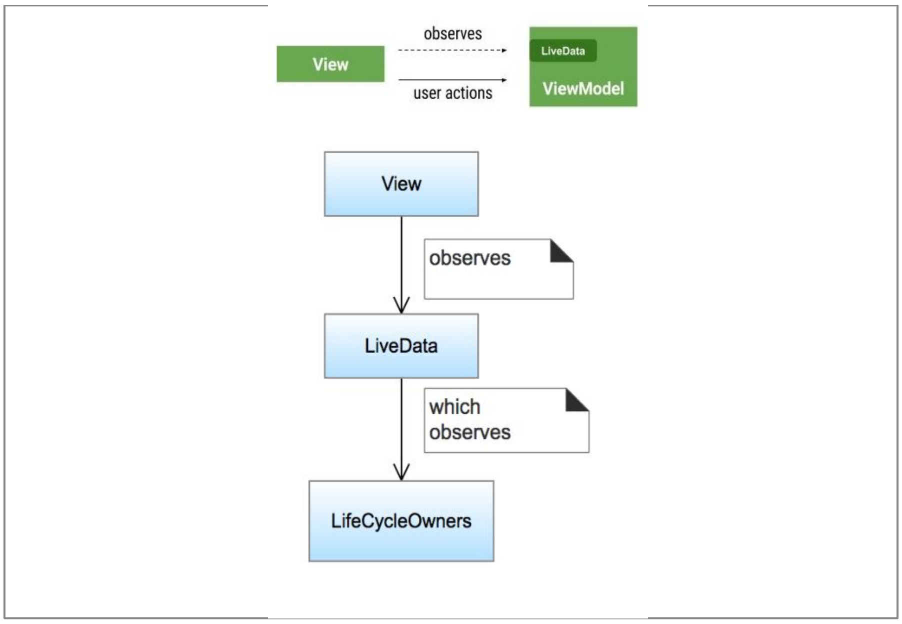
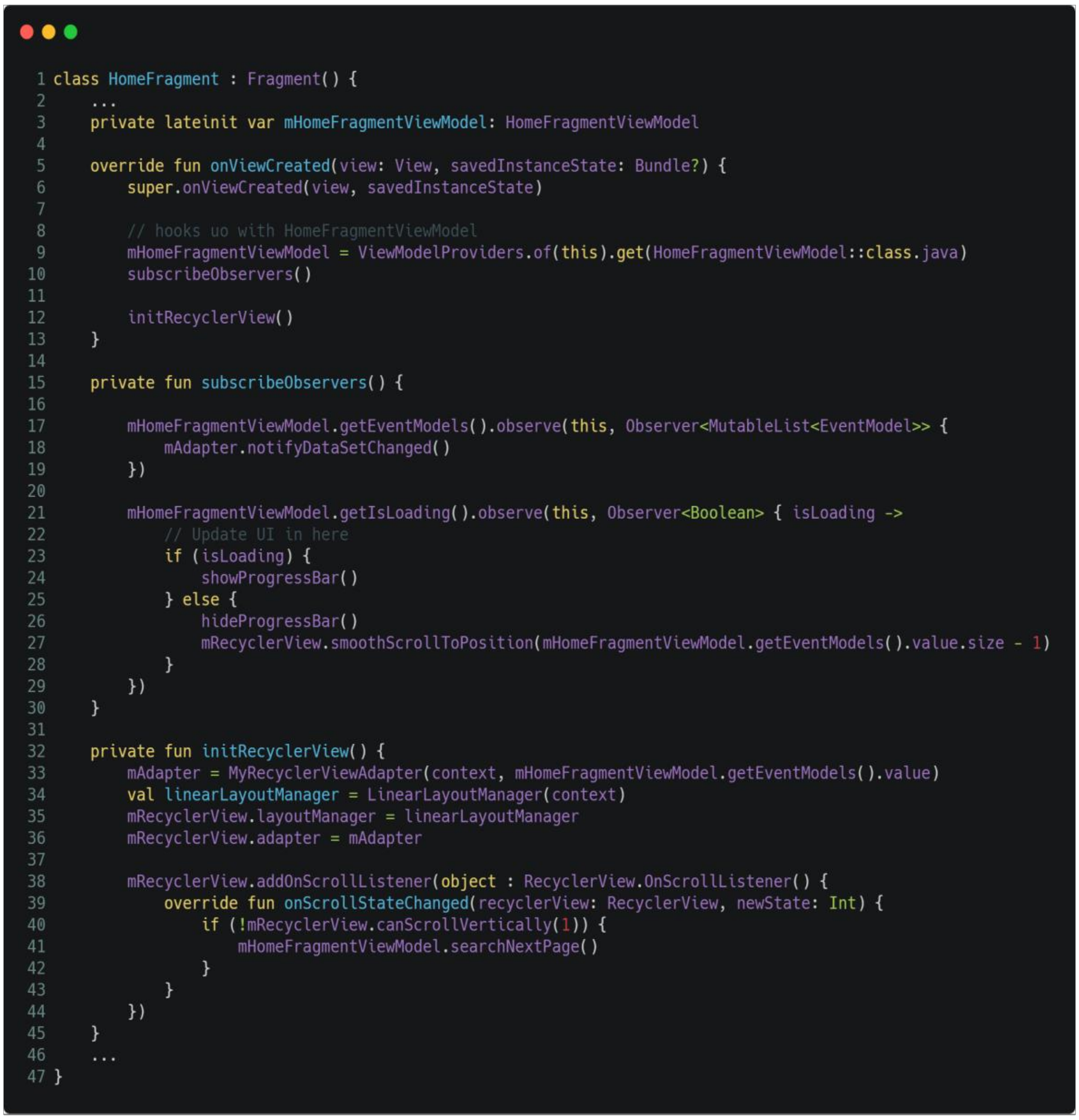


Figure 20. Observer logic diagram with LiveData

UI of Home Fragment is driven by the data in the Event Model class. According to MVVM principle, View layer relies on the ViewModel layer, thus, HomeFragment holds a reference of HomeFragmentViewModel as shown in Figure 21 and this reference is used to subscribe. In the subscribeObservers funtion, LiveData was exposed by ViewModel and observed by View, if LiveData changes, View will be notified and update itself with new data. This avoids the developer taking the time to re-update the data by findViewById method for the View layer, which simplifies operations and improves development and testing efficiency.



```kotlin
class HomeFragment : Fragment() {
    ...
    private lateinit var mHomeFragmentViewModel: HomeFragmentViewModel

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        // hooks uo with HomeFragmentViewModel
        mHomeFragmentViewModel = ViewModelProviders.of(this).get(HomeFragmentViewModel::class.java)
        subscribeObservers()

        initRecyclerView()
    }

    private fun subscribeObservers() {

        mHomeFragmentViewModel.getEventModels().observe(this, Observer<MutableList<EventModel>> {
            mAdapter.notifyDataSetChanged()
        })

        mHomeFragmentViewModel.getIsLoading().observe(this, Observer<Boolean> { isLoading ->
            // Update UI in here
            if (isLoading) {
                showProgressBar()
            } else {
                hideProgressBar()
                mRecyclerView.smoothScrollToPosition(mHomeFragmentViewModel.getEventModels().value.size - 1)
            }
        })
    }

    private fun initRecyclerView() {
        mAdapter = MyRecyclerViewAdapter(context, mHomeFragmentViewModel.getEventModels().value)
        val linearLayoutManager = LinearLayoutManager(context)
        mRecyclerView.layoutManager = linearLayoutManager
        mRecyclerView.adapter = mAdapter

        mRecyclerView.addOnScrollListener(object : RecyclerView.OnScrollListener() {
            override fun onScrollStateChanged(recyclerView: RecyclerView, newState: Int) {
                if (!mRecyclerView.canScrollVertically(1)) {
                    mHomeFragmentViewModel.searchNextPage()
                }
            }
        })
    }
    ...
}
```

Figure 21. Home Fragment Controller

In addition, HomeFragmentViewModel exposes two ViewModels, one is related to Event Model, another one is in the way like LiveData<Boolean> which is used for detecting data state and determine whether the data is loaded successfully. For example, in Figure 21, A scroll listener is added to the RecyclerView, which triggers the searchNextPage method when RecyclerView reaches the bottom. Then, a ProgressBar will appear when pulling up to load the next page and it will disappear when next page is loaded.

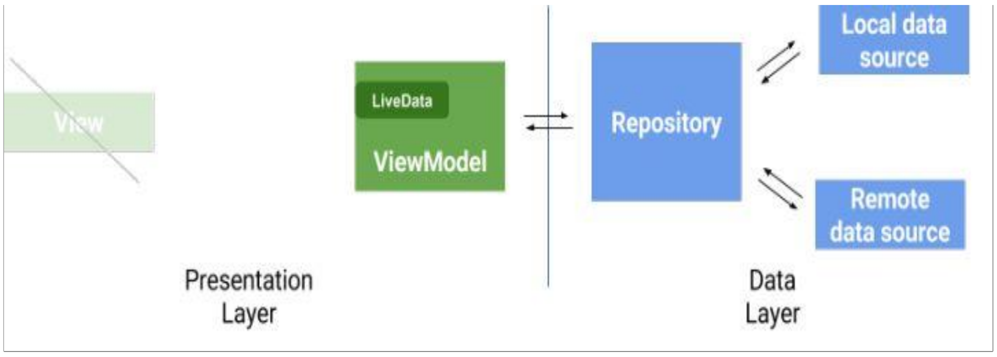
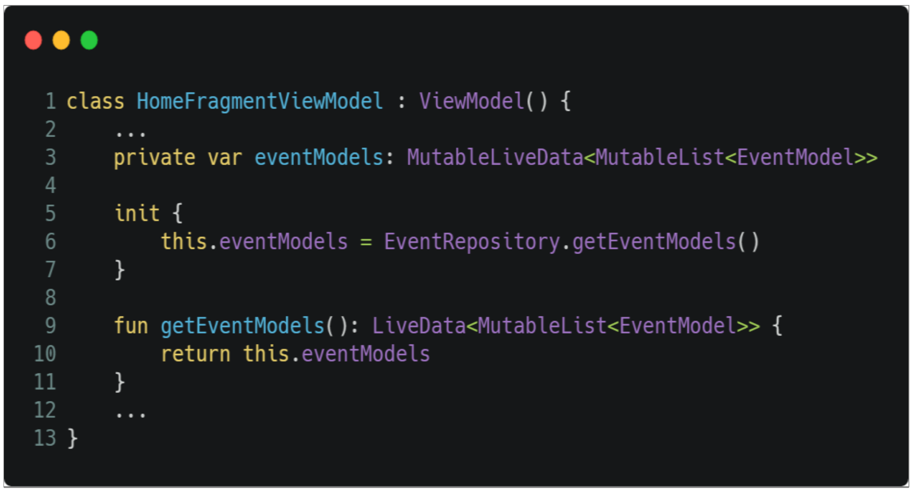


Figure 22. Observer logic diagram with LiveData

Another point to note is that LiveData knows about the View layer's state, since "this" as a parameter in the observer function is set to current LiveData's lifecycle owner, in other words, HomeFragment is in charge of both LiveData<MutableList<EventModel>> and LiveData<Boolean>, therefore, LiveData will not be triggered if its lifecycle owner is destroyed as illustrated in Figure 22.

### 4.1.3 View-Model

ViewModel is used to store and manage UI-related data in a life-cycle-aware manner. It allows data to survive without loss when the configuration changes. For example, the most common one is when the Activity is switched between horizontal and vertical, this Activity will be destroyed and recall the onCreate() function, but LiveData is not be trigeted, thus, data survives.



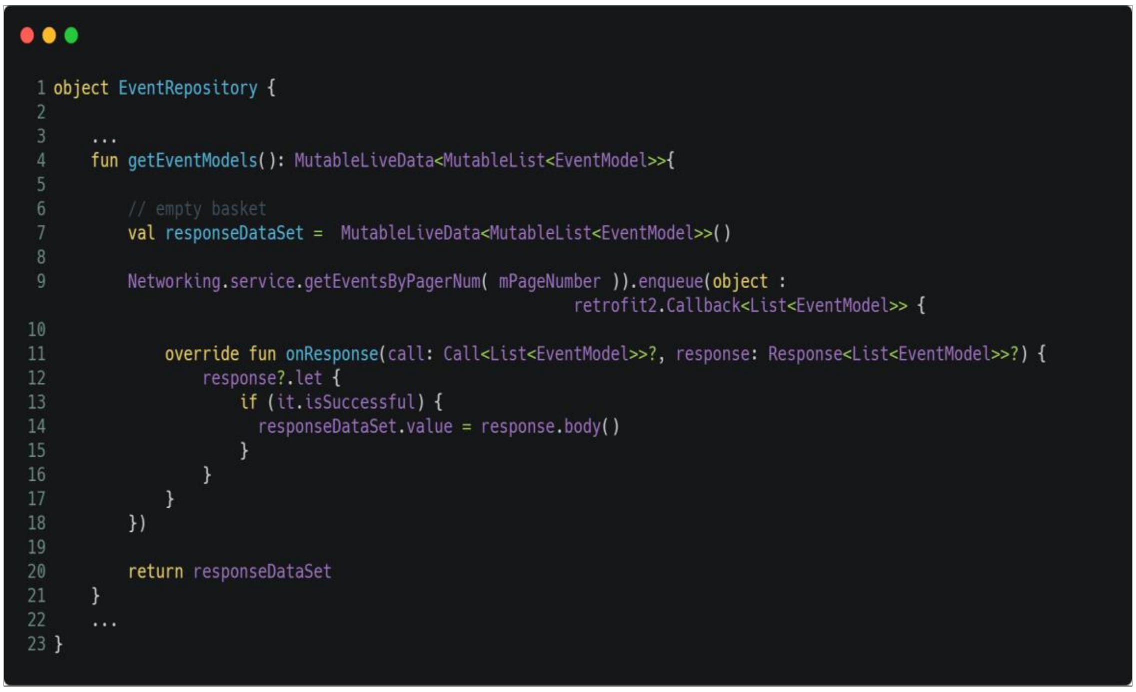
```kotlin
class HomeFragmentViewModel : ViewModel() {
    ...
    private var eventModels: MutableLiveData<MutableList<EventModel>>

    init {
        this.eventModels = EventRepository.getEventModels()
    }

    fun getEventModels(): LiveData<MutableList<EventModel>> {
        return this.eventModels
    }
    ...
}
```

Figure 23. Sample of View-Model for HomeFragment.

As observed in Figure 23, the HomeFragmentViewModel class extends from the View-Model class, and there is a LiveData inside it and ViewModel has no view's references, so there is no memory leak and no need to handle lifecycle events, such as, unsubscribe observers in onDestory() lifecycle methods.

### 4.1.4 Repository

Repository is responsible for obtaining data from different data sources. for example, the data can come from the network (Retrofit + OKHttp), the local Database (Room), or cache, etc. When ViewModel gets data from the Repository, ViewModel is no need to pay attention to which data source the data comes from.



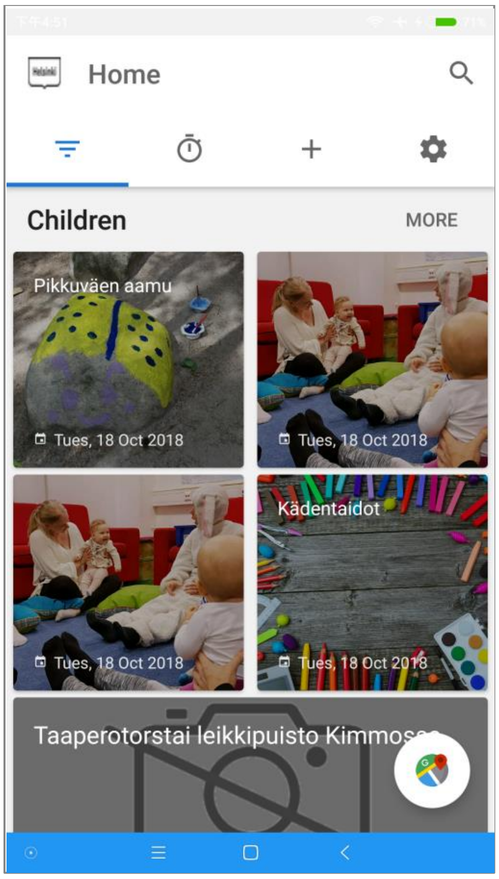
```kotlin
object EventRepository {

    ...
    fun getEventModels(): MutableLiveData<MutableList<EventModel>>{

        // empty basket
        val responseDataSet =  MutableLiveData<MutableList<EventModel>>()

        Networking.service.getEventsByPagerNum( mPageNumber )).enqueue(object :
                                                        retrofit2.Callback<List<EventModel>> {

            override fun onResponse(call: Call<List<EventModel>>?, response: Response<List<EventModel>>?) {
                response?.let {
                    if (it.isSuccessful) {
                      responseDataSet.value = response.body()
                    }
                }
            }
        })

        return responseDataSet
    }
    ...
}
```

Figure 24. Fetch data from server in Repository

As observed from Figure 24, This EventRepository class is defined using the "object" keyword in Kotlin, making it a singleton pattern, so that Kotlin will not create multiple Repository instance objects when fetching data. In this EventRepository, there is a method called getEventModels which is used to fetch data from server by calling the HTTP request.  After successfully acquiring the data, the value of responseDataSet variable will be reassigned and return it to the View-Model. At the same time, since View layer relies on the View-Model, therefore, View layer will get notification of data changes.

## 4.2 Project Result

The result of the application case can be observed in the following Figure 25. The entire main page can be swiped left and right to switch between different pages. The home page displays the activities with the theme of children and family by default. Each activity is displayed in the form of a card. One of the advantages of card type is modularity, which also means reusability. This not only ensures the aesthetics of the UI, but also reduces the amount of code. Click on the floating button on the bottom right corner, it will open an Activity to specifically load Google Maps and apply for permissions, then dynamically get the location of the user, and finally achieve displaying the relevant activities that near the user.



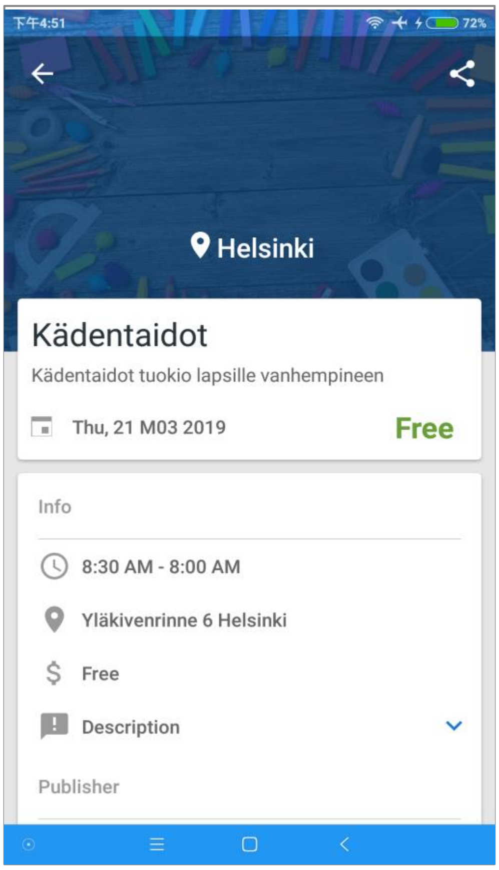


Figure 25. Home Page and Details Page

The activity details screen is displayed on the right side of the Figure 25. Users can share the event to their friends. In addition, in order to create a simple and user-friendly operating environment for the user, the icons used in the application case are relatively intuitive. Moreover, The use of the MVVM framework in the application also makes it easier to add new features and maintenance code.

## 4.3 GPU Rendering Testing

The fluency of application has always been regarded by users as an important criterion for measuring the visual experience of the application. FPS is often used as a measure of whether an application is fluent. FPS is frames per second, if the number of frames is 0, indicating that the page is at rest. As long as the page is moving, the number of frames will change. To give a simple example, the early cartoons were actually made by successively flipping pages with hand-drawn pictures. When a single image is switched fast enough, the user's eyes will think that this is a continuous action. So high frame rates give user smoother and more realistic animations. On the other hand, when the picture switching speed is not fast enough, the corresponding visual experience is clunky. In summary, the value of FPS reflects the display performance of the application

GPU rendering performance testing is based on the following environment:

1. Android Studio 3.2

2. Hardware environment: Huawei Honor 6 plus ( Android 6.0 )

GPU Monitor was removed from Android Studio since Android Studio 3.0 and this test used the GPU rendering tool on the phone by activating developer options. In the Android system, 60 FPS is specified as a full frame. [13] In other words, as long as the device is running at 60 FPS, the frame will not be dropped or delayed. Rendering 60 frames in 1 second, which means that each frame is rendered for a maximum of 16 milliseconds to ensure visual fluency. The horizontal green line in Figure 26 refers to 16 milliseconds and each frame below this green line means rendering time of the current frame is qualified.

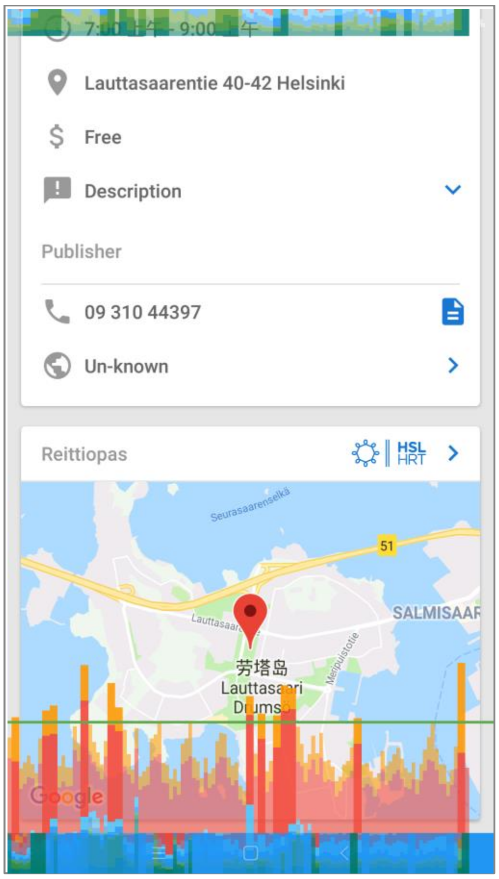


Figure 26. GPU Rendering on the Details Page

As observed from the Figure 26, most of the vertical bars are under the green line, only some of the frames are higher than the green line. Careful observation shows that these vertical bars that exceed the green line are having a red portion with a large proportion. According to the official documentation, the red part indicates the time it takes for Android's 2D renderer to issue commands to OpenGL to draw and redraw the list [14]. In other words, the more views, the longer the time it takes to render. In addition, this also warns developers, in order to ensure the smoothness of page scrolling, do not add too many views and complex custom views in the same Activity layout.

# 5 Conclusion

The goal of this paper was to achieve a social activity platform in which the main target group is tourists. In the process of research, the basic concepts and main roles of some technology stacks used in the application case are introduced. The understanding of these concepts and characteristics is the basis for implementation of the application case in this paper. In addition, the use of the MVVM framework design has greatly improved the scalability and reusability of the program. As for the GPU rendering evaluation, the overall result is qualified and it  does not affect the smoothness of page scrolling, but the evaluation result also alerts the developer to pay attention to reduce the complexity when designing the page.

On the other hand, this thesis also encourages further studying about the use of framework design in mobile development. From the MVC architecture model to the MVVM, after decades of development and evolution, the MVC architecture pattern has appeared in various variants and has its own implementation on different platforms. The role of the various architectural patterns is to separate concerns and distribute the functions belonging to different modules into the appropriate locations. Meanwhile the goal is minimizing the interdependence of individual modules and reducing the glue code that needs to be contacted. The choice of specific architectural modes depends on multiple factors, such as the complexity of the project, and functional design. In short, it is  best only when it fits you.

The research and development of tourism information service application software based on Android system has come to an end, but its subsequent development is still in the initial stage. As a mobile terminal, in order to satisfy the customer's individualized demand for tourism information service system and various complex operations, the follow-up needs to be improved to make many aspects more perfect. The following content can be added as extensions of the application:

1. Connect weather API to get destination weather information.
2. Intelligently recommending Event information through user’s browsing history.
3. Online payment module to facilitate users to purchase e-tickets for example Online.

In the end, tourism and information industries are gradually merging with each other, and the global informatization wave has promoted the informatization of the tourism industry. Meanwhile, along with the popularity of mobile phones, it not only improves the ability of tourists to obtain information but also makes the social application of tourism have a broad customer base and great potential.

# 6 References

1 Helsinki tourism enjoys record growth in 2017; [online]; URL: https://www.hel.fi/uutiset/en/kaupunginkanslia/record-growth-2017 Accessed 13 February 2019.

2 Pekka Mustonen. Helsinki tourism enjoys a record-breaking year – but what next ?; 2018. [online] URL: https://www.kvartti.fi/en/articles/helsinki-tourism-enjoys-record-breaking-year-what-next Accessed 13 February 2019.

3 Official documentation of Fragment [online] Google Corporation; URL: https://developer.android.com/guide/components/Fragments Accessed 23 February 2019.

4 Anupam Chugh. Android Fragment Lifecycle [online]; URL: https://www.journaldev.com/9266/android-Fragment-lifecycle Accessed 22 February 2019.

5 Official documentation of ViewPager [online] Google Corporation; URL: https://developer.android.com/reference/android/support/v4/view/ViewPager Accessed 27 February 2019.

6 Create a List with RecyclerView [online] Google Corporation; URL: https://developer.android.com/guide/topics/ui/layout/recyclerview Accessed 27 February 2019.

7 Official documentation of Content Providers [online] Google Corporation; URL: https://developer.android.com/guide/topics/providers/content-providers.html Accessed 1 March 2019.

8 Model View ViewModel MVVM Android Example. [online]; URL: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel Accessed 23 March 2019

9 Principles of Material Design [online] Google Corporation; URL: https://material.io/design/introduction/#principles Accessed 23 February 2019.

10 Cards in Material Design [online] Google Corporation; URL: https://material.io/design/components/cards.html#anatomy Accessed 18 March 2019.

11 Amit Agarwal. How to Get Things Done with Trello; 2015 [online] URL: https://www.labnol.org/internet/trello-basics-getting-started/29044/ Accessed 12 March 2019.

12 Guide to app architecture [online] Google Corporation; URL: https://developer.android.com/jetpack/docs/guide Accessed 23 February 2019.

13 Test UI Performance [online] Google Corporation; URL: https://developer.android.com/training/testing/performance Accessed 4 April 2019.

14 Inspect GPU rendering speed and overdraw [online] Google Corporation; URL: https://developer.android.com/studio/profile/inspect-gpu-rendering#profile_rendering Accessed 3 April 2019