Midterm

Building a simple task management app in Android using Kotlin

Prepared by Azimkhanov Bauyrzhan

Almaty, 27.10.2024

Table of contents

Executive summary

The goal of this assignment is to gain hands-on experience with Android development and Kotlin programming language. I used Kotlin language, gradle-kts (Kotlin style) and Android Studio IDE. From Android development SDK I used layout system, fragments, different methods for fragment/activity lifecycle, Navigation Component and other tools and utilities. As a result, I received an application with functions for creating tasks, filtering posts and other features from the technical specifications.

Introduction

Mobile apps have become an integral part of everyday life. They are everywhere, and in the near future, I think their role will only increase. They usually allow users to perform tasks in a very convenient way, since users can work without even "getting off the couch." For businesses, mobile apps are crucial for reaching customers, improving service delivery, and collecting valuable user data.

Kotlin has become the preferred language for Android development because it simplifies coding by reducing lines of code and increasing readability. Java was no sweeter because Kotlin gave us so much "syntactic sugar" to make our lives as software developers easier. Additionally, Kotlin supports functional programming and coroutines, which allow us to write smoother asynchronous programs. Google's support for Kotlin as an official language for Android has also added confidence to the future of this technology.

My main motivation for making a task manager using abovementioned stack of technology was the desire to master a new stack of relevant technologies and the desire to get a good grade at the end of the semester.

Project objectives

Among the goals of this project, I can highlight the creation of a prototype of the application with the necessary requirements. All this is necessary in order to better understand this subject and master the material.

List of goals:

1. Developing a functional mobile application
2. Understanding the Android lifecycle
3. Utilizing Kotlin features
4. Styding navigation component, fragments, different layout system and so on

Overview of Android development and Kotlin

How to setup Android Studio and configuration for Kotlin development? You should follow these general steps:

1. Find and install Android Studio from the official website
2. Create a new project in the desired working directory with the type "empty activity"
3. Import all required and/or desired libraries, modules, registries, etc.

There are some features of Kotlin comparing with their “father” Java:

1. **Concise syntax** - Kotlin has shorter code compared to Java. This means developers can write less code to do the same tasks. It makes the code easier to read and develop.
2. **Null safety** - Kotlin helps prevent crashes caused by null values. It forces developers to handle nulls, making apps more stable.
3. **Extension functions** - Developers can add new functions to existing classes without modifying them. This makes it easier to enhance code.
4. **Smart casts** - Kotlin automatically checks the type of a variable. This means developers don’t need to cast types manually, reducing errors.
5. **Coroutines** - Kotlin makes it simple to manage tasks that run in the background. This helps keep apps responsive and fast.

Improvements over Java:

1. **Less long code structure**: Kotlin reduces repetitive code. This saves time and makes projects easier to manage.
2. **Better readability**: The clean and simple syntax of Kotlin makes it easier for new developers to understand the code.
3. **Interoperability**: Kotlin works well with Java. Developers can use both languages in the same project, which makes the transition easier.
4. **Modern features**: Kotlin has many modern programming features that Java lacks, making it more enjoyable to use.

Functions and lambdas in Kotlin

To develop this app I created a lot of methods. Let me descripe couple of them.

Let us start with method performFiltering from RecyclerViewAdapter.kt file. This method get ArrayList of taks and filter it by username that was also provided by the application user.

You can see the basic filtering process on line 8, where each record in the original dataset is checked for a specific username. If there is a match between the current username and the desired username (they are equal), we add the current element to a new result array, which will be shown as the filtered output.

```
override fun performFiltering(input: CharSequence?): FilterResults {
    val filteredArrayList: ArrayList<RecyclerViewItem> = ArrayList()
    if (input.isNullOrEmpty()) {
        filteredArrayList.addAll(initialDataSet)
    } else {
        initialDataSet.forEach {
            if (it.username.lowercase().contains(input)) {
                filteredArrayList.add(it)
            }
        }
    }
    val results = FilterResults()
    results.values = filteredArrayList
    return results
}
```

Figure 1. performFiltering() method

The onCreateView() method is an important part of an Android Fragment. It helps create the user interface for the fragment.

The method starts by inflating the layout for the fragment using *inflater.inflate(R.layout.fragment_home_feed, container, false)*. This means it takes the XML layout file (fragment_home_feed.xml) and turns it into a view that can be shown on the screen. Next, it finds the RecyclerView in the inflated view with *view.findViewById(R.id.recycler_view)*. This is where the list of tasks will be shown. It sets the layout manager for the RecyclerView using *recyclerView.layoutManager = LinearLayoutManager(context)*. The LinearLayoutManager arranges the items in a vertical list, which is common for displaying tasks. The method also initializes a navigation controller with *navigationController = findNavController()*. This allows the fragment to navigate between different screens in the app. Finally, it returns the created view with *return view*. This makes the layout visible to users.

In summary, this method sets up the layout, finds the RecyclerView, arranges it, and prepares for navigation. It is essential for displaying the list of tasks in the fragment.

```
override fun onCreateView(
    inflater: LayoutInflater, container: ViewGroup?,
    savedInstanceState: Bundle?
): View? {
    // Inflate the layout for this fragment
    val view = inflater.inflate(R.layout.fragment_home_feed, container,
false)
    recyclerView = view.findViewById(R.id.recycler_view)
    recyclerView.layoutManager = LinearLayoutManager(context)
    navigationController = findNavController()
    return view
}
```

Figure 2. onCreateView() method

I used lambda function in NotificationsRecyclerViewAdapter.kt file to bind value of textView in short and laconic way. See figure 3.

```
override fun onBindViewHolder(holder: ViewHolder, position: Int) {
    holder.notificationItemView.text = with(dataSet[position]) {
        "$username: $action at $timestamp"
    }
}
```

Figure 3. OnBindViewHolder() method

Object-oriented programming in Kotlin

In Kotlin, classes and data classes are used to represent tasks:

1. A class can define a instance with fileds like title, description, and etc. It can also include methods to perform actions, such as markAsCompleted() to change the task's status.
   Example you can see on the figure 4.

```
Class RecyclerViewAdapter(private val 6ataset: ArrayList<RecyclerViewItem>,
private val navigationController: NavController) :
    RecyclerView.Adapter<RecyclerViewAdapter.ViewHolder>(), Filterable {
    private var initialDataSet = ArrayList<RecyclerViewItem>().apply {
        addAll(6ataset)
    }

    private val searchFilter: Filter = object : Filter() {
        override fun performFiltering(input: CharSequence?): FilterResults {
            val filteredArrayList: ArrayList<RecyclerViewItem> = ArrayList()
            if (input.isNullOrEmpty()) {
                filteredArrayList.addAll(initialDataSet)
            } else {
                initialDataSet.forEach {
                    if (it.username.lowercase().contains(input)) {
                        filteredArrayList.add(it)
                    }
                }
            }
            val results = FilterResults()
            results.values = filteredArrayList
            return results
        }

        override fun publishResults(input: CharSequence?, results:
FilterResults?) {
            if (results?.values is ArrayList<*>) {
                6ataset.clear()
                6ataset.addAll(results.values as ArrayList<RecyclerViewItem>)
                notifyDataSetChanged()
            }
        }
    }

    class ViewHolder(view: View) : RecyclerView.ViewHolder(view) {
        val usernameItemView: TextView =
view.findViewById(R.id.username_item_text_view)
        val imageItemView: ImageView =
view.findViewById(R.id.image_item_image_view)
        val commentItemTextView: TextView =
view.findViewById(R.id.comment_item_text_view)
        val likesItemTextView: TextView =
view.findViewById(R.id.likes_item_text_view)
```

```
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
ViewHolder {
        val view = LayoutInflater.from(parent.context)
            .inflate(R.layout.feed_recycler_view_item, parent, false)
        return ViewHolder(view)
    }

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        holder.usernameItemView.text = initialDataSet[position].username
        holder.imageItemView.setImageResource(initialDataSet[position].image)
        holder.commentItemTextView.text = initialDataSet[position].comment
        var likesItemTextViewText: String =
initialDataSet[position].likes.toString() + "likes"
        holder.likesItemTextView.text = likesItemTextViewText
        val username = initialDataSet[position].username
        holder.usernameItemView.setOnClickListener {
            try {
                val transitionToProfileMenuAction =
HomeFeedFragmentDirections.actionHomeFeedMenuToProfileMenu(username)
                navigationController.navigate(transitionToProfileMenuAction)
            }
            catch (exception: Exception) {
                println("Transited from wrong origin - HomeFeedFragment:
$exception")
            }
            try {
                val transitionToProfileMenuAction =
SearchFragmentDirections.actionSearchMenuToProfileMenu(initialDataSet[positio
n].username)
                navigationController.navigate(transitionToProfileMenuAction)
            }
            catch (exception: Exception) {
                println("Transited from wrong origin - SearchFragment:
$exception")
            }
        }
        holder.likesItemTextView.setOnClickListener {
            initialDataSet[position].likes++
            likesItemTextViewText = initialDataSet[position].likes.toString()
+ "likes"
            holder.likesItemTextView.text = likesItemTextViewText
        }
    }

    override fun getItemCount() = 7ataset.size

    override fun getFilter(): Filter {
        return searchFilter
    }

}
```

Figure 4. Example of Kotlin class

2. Data classes are simpler and are mainly used to hold data. They automatically generate functions like equals(), hashCode(), and toString(), making them ideal for tasks.
   Example you can see on the figure 5.

```
data class Task(
    var name: String,
    var description: String,
    var completed: Boolean
)
```

Figure 5. Example of data class

Encapsulation is an important principle of Object-Oriented Programming (OOP). It helps organize code by hiding details and exposing only what is necessary.

Properties and methods can be marked as private or public. This means you can hide some data from outside access. For example, you might want to keep the task's internal state private and only expose methods to interact with it.

Using getter and setter methods allows controlled access to properties. This ensures that data is safe and changes happen only through defined methods.

Working with collections in Kotlin

As mentioned above in Figures 1 and 4, I used forEach, filter, and other methods to search, iterate, and perform various operations on Kotlin collections. Below in Figure 6, you can see an example of how I used an ArrayList to store task data. This variable "dataset" stores instances of the Task data class that are declared when the ArrayList is created.

```
val dataset = arrayListOf(
    RecyclerViewItem("Prepare assignment 2", "Requirements in Google
Documents", true),
    RecyclerViewItem("Prepare for midterm", Requirements in Google Documents,
false)
)
```

Figure 6. Kotlin collection example

Android layout design

In my app I have a main_activity and different fragments to navigate around the app. For all layouts (fragment and activity) I used ConstraintLayout. The only exception was the notification page where I used LinearLayout. You may ask why I do this? Well, I do this because of the requirements of this job which declared the responsibility to implement the view using different layouts. ConstraintLayout looks simpler to me because each UI element (TextView, ImageView, etc.) is positioned automatically. You only need to define rules for them.

Below in figure 7 you can see main_activity.xml. It has:

1. The root element is <androidx.constraintlayout.widget.ConstraintLayout>. This is a flexible layout that helps position child views relative to each other and to the parent.
2. The first child is <androidx.fragment.app.FragmentContainerView>. This is used to host fragments.
   Attributes:
   1. android:id: Sets a unique identifier for this view.
   2. android:name: Specifies that this view will be a navigation host for fragments.
   3. app:navGraph: Links to a navigation graph, defining navigation paths.
   4. app:defaultNavHost: Indicates that this is the main navigation host.
   5. Layout Constraints: It is constrained to the top, bottom, start, and end of the parent layout, making it fill the space above the bottom navigation view.

3. The second child is <com.google.android.material.bottomnavigation.BottomNavigationView>. This provides a bottom navigation bar.
   Attributes:
   1. android:id: Sets a unique identifier for this view.
   2. android:background: Sets the background color of the navigation bar.
   3. Layout Constraints: It is constrained to the left, right, and bottom of the parent layout, making it stick to the bottom of the screen.
   4. app:menu: Links to a menu resource that defines the items shown in the navigation bar.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <androidx.fragment.app.FragmentContainerView
        android:id="@+id/navigation_host_fragment_activity_main"
        android:name="androidx.navigation.fragment.NavHostFragment"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        app:navGraph="@navigation/navigation_graph"
        app:defaultNavHost="true"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintBottom_toTopOf="@+id/navigation_view"
        app:layout_constraintStart_toStartOf="parent" />
```

```
    <com.google.android.material.bottomnavigation.BottomNavigationView
        android:id="@+id/navigation_view"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:background="?android:attr/windowBackground"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:menu="@menu/bottom_navigation_menu" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Figure 7. Main activity layout

Activity and user input handling

To create an activity, you should either use the functionality of the Android Studio IDE or create the corresponding files manually. I have written about this before.

Basically, user input is handled inside the setOnClickListener() method. For example, when you click the Create Task button, this method is triggered and performs data collection and transaction to another fragment. More details in the chapter on fragments.

Activity lifecycle management

Lifecycle methods are crucial for managing an app's state in Android. They help control what happens to an activity or fragment at different points during its life.

Why they are important:

1. **Resource management** - methods allow us to allocate and release resources properly. For example, we can load data in onCreate() and release resources in onDestroy(), preventing memory leaks.
2. **State preservation** - onSaveInstanceState() let us save the current state before an activity is destroyed (like during a configuration change). This way, we can restore the state later.
3. **UI updates** - Lifecycle methods help manage UI updates. Animations can be started or data can be loaded in onStart() and stopped in onStop().
4. **User experience** - Proper use of lifecycle methods enhances user experience. By managing the state effectively, developer avoid scenarios where users lose data or see unexpected behavior when they navigate away from your app and return.
5. **Background tasks** - Lifecycle methods help manage background tasks. For example, it is possible to pause a video in onPause() and resume it in onResume(), ensuring that the app behaves correctly based on user interaction.

One common example of using this kind of methods is overriding the onCreate() method in MainActivity, which you can see in Figure 8, where we declare the NavHostFragement and BottomNavigationView. After declaring, we provide a navController for the NavigationView and so on. This way, we set up the FragmentContainerView and the bottom navigation menu. We connect them. You also can see figure 2 with another onCreate() method.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    val navigationView: BottomNavigationView =
findViewById(R.id.navigation_view)
    val navigationHostFragment =
supportFragmentManager.findFragmentById(R.id.navigation_host_fragment_activit
y_main) as NavHostFragment
    val navigationController = navigationHostFragment.navController

    navigationView.setupWithNavController(navigationController)
    bottomNavItemChangeListener(navigationView, navigationController)

}
```

Figure 8. MainActivity.kt onCreate() method

Fragments and fragment lifecycle

Fragments are created by extending the Fragment class and defining their layout. They can be added to activities using XML or programmatically. We can navigate between fragments using a Navigation Controller. This allows for flexible and reusable UI components within app. Fragments renders and interchange each other in FragmentContainerView.

A typical example of fragments is NotificationsFragment.kt, which is responsible for notifications (Figure 9). OnCreateView() we fill the layout, and in onViewCreated() we set the data and display it.

```
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import java.sql.Timestamp

class NotificationsFragment : Fragment() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_notifications, container,
false)
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        val dataset =
arrayListOf(NotificationsRecyclerViewItem("holebas_sead", "liked your post",
Timestamp(124, 10, 15, 14, 48, 3, 15)),
            NotificationsRecyclerViewItem("antoniocassano", "commented your
post", Timestamp(124, 10, 12, 13, 37, 0, 28)),
            NotificationsRecyclerViewItem("dimitar_berbatov", "liked your
post", Timestamp(124, 8, 29, 20, 44, 5, 36)),
            NotificationsRecyclerViewItem("methew_bal", "commented your
post", Timestamp(124, 9, 3, 18, 58, 10, 69)),
```

```
            NotificationsRecyclerViewItem("honeymad", "liked your post",
Timestamp(124, 6, 15, 14, 45, 23, 21)),
            NotificationsRecyclerViewItem("yasos_biba", "commented your
post", Timestamp(124, 9, 11, 9, 28, 30, 31)),
            NotificationsRecyclerViewItem("lena-golovach", "liked your post",
Timestamp(124, 10, 5, 12, 36, 32, 68)),
            NotificationsRecyclerViewItem("balotelli_mario", "commented your
post", Timestamp(124, 10, 7, 22, 25, 11, 22)),
            NotificationsRecyclerViewItem("ravin_zilberman", "liked your
post", Timestamp(124, 10, 8, 10, 14, 28, 99)),
            NotificationsRecyclerViewItem("m.martsinkevich", "commented your
post", Timestamp(124, 10, 9, 11, 3, 0, 55)),
            NotificationsRecyclerViewItem("holebas_sead", "liked your post",
Timestamp(124, 10, 15, 14, 48, 3, 15)),
            NotificationsRecyclerViewItem("antoniocassano", "commented your
post", Timestamp(124, 10, 12, 13, 37, 0, 28)),
            NotificationsRecyclerViewItem("dimitar_berbatov", "liked your
post", Timestamp(124, 8, 29, 20, 44, 5, 36)),
            NotificationsRecyclerViewItem("methew_bal", "commented your
post", Timestamp(124, 9, 3, 18, 58, 10, 69)),
            NotificationsRecyclerViewItem("honeymad", "liked your post",
Timestamp(124, 6, 15, 14, 45, 23, 21)),
            NotificationsRecyclerViewItem("yasos_biba", "commented your
post", Timestamp(124, 9, 11, 9, 28, 30, 31)),
            NotificationsRecyclerViewItem("lena-golovach", "liked your post",
Timestamp(124, 10, 5, 12, 36, 32, 68)),
            NotificationsRecyclerViewItem("balotelli_mario", "commented your
post", Timestamp(124, 10, 7, 22, 25, 11, 22)),
            NotificationsRecyclerViewItem("ravin_zilberman", "liked your
post", Timestamp(124, 10, 8, 10, 14, 28, 99)),
            NotificationsRecyclerViewItem("m.martsinkevich", "commented your
post", Timestamp(124, 10, 9, 11, 3, 0, 55)),
            NotificationsRecyclerViewItem("holebas_sead", "liked your post",
Timestamp(124, 10, 15, 14, 48, 3, 15)),
            NotificationsRecyclerViewItem("antoniocassano", "commented your
post", Timestamp(124, 10, 12, 13, 37, 0, 28)),
            NotificationsRecyclerViewItem("dimitar_berbatov", "liked your
post", Timestamp(124, 8, 29, 20, 44, 5, 36)),
            NotificationsRecyclerViewItem("methew_bal", "commented your
post", Timestamp(124, 9, 3, 18, 58, 10, 69)),
            NotificationsRecyclerViewItem("honeymad", "liked your post",
Timestamp(124, 6, 15, 14, 45, 23, 21)),
            NotificationsRecyclerViewItem("yasos_biba", "commented your
post", Timestamp(124, 9, 11, 9, 28, 30, 31)),
            NotificationsRecyclerViewItem("lena-golovach", "liked your post",
Timestamp(124, 10, 5, 12, 36, 32, 68)),
            NotificationsRecyclerViewItem("balotelli_mario", "commented your
post", Timestamp(124, 10, 7, 22, 25, 11, 22)),
            NotificationsRecyclerViewItem("ravin_zilberman", "liked your
post", Timestamp(124, 10, 8, 10, 14, 28, 99)),
            NotificationsRecyclerViewItem("m.martsinkevich", "commented your
post", Timestamp(124, 10, 9, 11, 3, 0, 55)),
            )
        val notificationsRecyclerViewAdapter:
NotificationsRecyclerViewAdapter = NotificationsRecyclerViewAdapter(dataset)
        val notificationsRecyclerView: RecyclerView =
view.findViewById(R.id.notifications_recycler_view)
        notificationsRecyclerView.layoutManager =
LinearLayoutManager(context)
        notificationsRecyclerView.adapter = notificationsRecyclerViewAdapter
    }
}
```

Figure 9. NotificationsFragment.kt

Navigation is implemented using FragmentContainerView (inside MainActivity), NavComponent and navigation graph (navigation/navigation_graph.xml). In FragmentContainerView instances of Fragment class can dynamically replace each other. These replacements are handled by NavController and described in the navigation graph.

Using Actions and Argument (navigation_graph.xml - declaration, RecyclerViewAdapter - logic and implementation) I switched the fragment and passed data to them accordingly.

I used NavComponent because of the requirements of this assignment. I used NavComponent because of the requirements of this task. However, it allows you to navigate between Views and Activities while maintaining state, context, etc.

Activity – fragment interaction:

1. Fragments can communicate with the main activity using interfaces. The fragment defines an interface, and the activity implements it. This allows the fragment to send events or data back to the activity.
2. Fragments depend on the activity's lifecycle. When the activity is paused or stopped, the fragment also goes through its own corresponding lifecycle methods. This helps maintain the state and behavior consistent between the fragment and activity.
3. The activity manages fragment transactions (adding, replacing, or removing fragments) using FragmentManager. This allows dynamic changes to the UI based on user actions.

Conclusion

As a result, we received an application that fully complies with the requirements of this work, namely "simple task management application for Android using Kotlin. The app will allow users to create, view, update, and delete tasks, providing a user-friendly interface. The project will cover key concepts of Android development, Kotlin syntax, object-oriented programming, handling user input, and managing the activity and fragment lifecycle."

In this task I used basically the same methods as in the previous task. To further develop this application we can update the user interface and add more data storage functionality.

References

1. https://www.geeksforgeeks.org/android-image-picker-from-gallery-using-activityresultcontracts-in-kotlin/
2. https://www.geeksforgeeks.org/bottom-navigation-bar-in-android/
3. https://medium.com/@everydayprogrammer/implement-android-photo-picker-in-android-studio-3562a85c85f1
4. https://medium.com/@prasanth968/kotlin-image-picker-from-gallery-using-activityresultcontracts-9b5aa32e42d0
5. https://developer.android.com/training/data-storage/shared/photopicker#kotlin
6. https://proandroiddev.com/implementing-photo-picker-on-android-kotlin-jetpack-compose-326e33e83b85
7. https://medium.com/geekculture/searchable-recyclerview-e316289edc25
8. https://www.geeksforgeeks.org/gridview-in-android-with-example/
9. https://www.tutorialspoint.com/how-to-create-gridview-layout-in-an-android-app-using-kotlin
10. https://www.geeksforgeeks.org/android-gridview-in-kotlin/
11. https://stackoverflow.com/questions/20191914/how-to-add-gridview-setonitemclicklistener
12. https://developer.android.com/reference/kotlin/androidx/gridlayout/widget/GridLayout
13. https://abhiandroid.com/ui/gridview#gsc.tab=0
14. https://www.geeksforgeeks.org/gridview-in-android-with-example/
15. https://www.tutorialspoint.com/how-to-create-gridview-layout-in-an-android-app-using-kotlin
16. https://www.geeksforgeeks.org/android-gridview-in-kotlin/
17. https://www.geeksforgeeks.org/how-to-implement-android-searchview-with-example/
18. https://developer.android.com/reference/
19. https://kotlinlang.org/docs/home.html

Link to my GitHub repository (screenshots included):

- **https://github.com/BauyrzhanAzimkhanov/Mobile-programming-MSc.git**

Appendices