To do list application

Andrei Voicea

Abstract—This paper presents the installation steps, functionality, front-end and back-end, of the To do list application made in android studio using Kotlin and XML.

Contents

1. Introduction

Modern life stands out by setting and coordinating daily work and activities in order to be punctual but also efficient and productive on a daily basis. Thus having many activities to practice it is much easier for us to use an application in which we can put all the work we need to do that day, without having to consume pens and papers to make lists.

2. Installation

To install the application follow the steps:

1. Go to the application's GitHub page (https://github.com/AndreiVoicea2/TodoList.git)
2. Click on the button labeled "Code" -> "Download ZIP"
3. Open the project using Android Studio to modify the code, simulate the app or even get the apk for your mobile phone.

If after modifying the application code you want to run it on your mobile phone, follow these steps:

1. In the Android menu, go to Build > Build Bundle(s) / APK(s) > Build APK(s).
2. Android Studio will start building the APK for you. Once finished, a pop-up window in the bottom right will notify you of its completion. Click the "locate" button in this dialog.
3. The "locate" button should open File Explorer with the debug folder open, which contains a file called "app-debug.apk".
4. I personally use Google Drive to bring this apk to my mobile phone.

3. Functionallity

As soon as the start animation of the application takes place, the data saved in the database is loaded in parallel. Tasks are added to the list via the text box that says *Enter Todo Title*. This box does not accept tasks that already exist and if an input is added as empty it is replaced with the title *Default_Title*. The first button next to the text box allows you to select a date for the task. A calendar will be opened from which the date can be selected. By pressing the *Add Todo* button the new task is added to the list and saved in the database. These can be completed by checking the check box to the right of the task. They are also deleted from the database.

4. Software Architecture

4.1. Front-End

We start with the activity that deals with the start animation of the application. The order of execution of the activities has been modified in the android studio manifest.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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=".SplashScreenActivity">

    <ImageView
        android:layout_width="200dp"
        android:layout_height="200dp"
        android:src="@drawable/
    baseline_assignment_turned_in_24"
        android:layout_centerInParent="true"
        android:id="@+id/iv_note"  />

</RelativeLayout>
```

Code 1. activity_splash_screen.xml

This is the main layout container, using a **RelativeLayout** to allow relative positioning of child elements. RelativeLayout allows the placement of elements relative to other elements or relative to parent edges. **ImageView** displays an image centered inside the layout, with a fixed size of 200dp x 200dp.

```
<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:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#387047"
    tools:context=".MainActivity">
</androidx.constraintlayout.widget.ConstraintLayout>
```

Code 2. activity_main_layout

We move on to the activity that deals with the task list. This is the main container that organizes and aligns interior elements using constraints. ConstraintLayout is very flexible and allows positioning of elements relative to other elements or to the edge of the container.

```
<androidx.recyclerview.widget.RecyclerView
        android:id="@+id/rvTodoItems"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:background="#33994E"
        app:layout_constraintBottom_toTopOf="@+id/
    etTodoTitle"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.0"
        tools:ignore="MissingConstraints" />
```

Code 3. activity_main_RecyclerView

RecyclerView is a powerful and flexible widget for displaying a large list of data with high performance. It is used to display the list of To-Do items.

```
<EditText
        android:id="@+id/etTodoTitle"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:fontFamily="casual"
        android:hint="Enter Todo Title"
        android:textColor="#FFFFFF"
        android:textColorHint="#FFFFFF"
        app:layout_constraintBottom_toBottomOf="parent"
```

```
        app:layout_constraintEnd_toStartOf="@+id/datePickerButton"
        app:layout_constraintStart_toStartOf="parent"
        tools:ignore="MissingConstraints" />
```

Code 4. activity_main_EditText

EditText is a widget that allows the user to enter text. Used to add the task name.

```
    <Button
        android:id="@+id/btnAddTodo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:backgroundTint="#293D2E"
        android:fontFamily="casual"
        android:text="Add Todo"
        android:textColor="#F6F6F6"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        tools:ignore="MissingConstraints" />
```

Code 5. activity_main_AddButton

Button is a widget that allows the user to trigger an action. Used to add the task to the list.

```
    <Button
        android:id="@+id/datePickerButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:backgroundTint="#293D2E"
        android:fontFamily="casual"
        android:onClick=""
        android:text="Pick Date"
        android:textColor="#FFFFFF"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toStartOf="@+id/btnAddTodo"
        tools:ignore="MissingConstraints" />
```

Code 6. activity_main_DateButton

Button used to select the task date.

```
<?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:layout_width="match_parent"
    android:layout_height="80dp">


    <TextView
        android:id="@+id/tvTodoTitle"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:fontFamily="casual"
        android:paddingStart="8dp"
        android:paddingEnd="8dp"
        android:text="Example"
        android:textColor="#F6F2F2"
        android:textSize="24sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toStartOf="@+id/tvTodoDate"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:ignore="MissingConstraints" />

    <TextView
        android:id="@+id/tvTodoDate"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:fontFamily="casual"
        android:gravity="center_vertical"
        android:paddingStart="8dp"
        android:paddingEnd="8dp"
        android:text=" 9/5/2024"
        android:textColor="#F8F5F5"
        android:textSize="20sp"
        android:textStyle="italic"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toStartOf="@+id/cbDone"
        app:layout_constraintTop_toTopOf="parent"
        tools:ignore="MissingConstraints" />

    <CheckBox
        android:id="@+id/cbDone"
        android:layout_width="wrap_content"
        android:layout_height="41dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:ignore="MissingConstraints" />

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

Code 7. item_todo.xml

This xml handles the design of the individual task to be added to the list. **ConstraintLayout**: Main container that uses constraints to arrange child elements. **TextView** for To-Do Title: Displays the title of a To-Do task, with specific stylizations (color, size, padding). **TextView** for To-Do Date: Displays the date of a To-Do task, with specific styling (color, size, italic style, padding). **CheckBox** for marking To-Do as completed: Allows the user to mark a task as completed.

4.2. Back-End

We start this section by presenting the functionality of the animation at the beginning of the application.

```
package com.example.todolist

import android.annotation.SuppressLint
import android.content.Intent
import android.os.Bundle
import android.widget.ImageView
import androidx.appcompat.app.AppCompatActivity
import kotlin.concurrent.thread


@SuppressLint("CustomSplashScreen")
class SplashScreenActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?)
    {

        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_splash_screen)

        val thread = thread(start = true)
        {

            TodoAdapter.loadFromDataBase()

        }

        val iv_note: ImageView = findViewById(R.id.iv_note)
        val fadingAnimationDuration: Long = 1500
        val splashScreenAlphaInitial = 0f
        val splashScreenAlphaMax = 1f

        iv_note.alpha = splashScreenAlphaInitial
        iv_note.animate().setDuration(fadingAnimationDuration).alpha(splashScreenAlphaMax)
            .withEndAction {

                val i = Intent(this, MainActivity::class.java)
                startActivity(i)
                thread.join()
                overridePendingTransition(android.R.anim.fade_in, android.R.anim.fade_out)
                finish()

            }
    }
}
```

Code 8. SplashScreenActivity.kt

Starting with line 18, I create and start a thread that deals with loading data from the database. The animation runs and waits for the data loading thread to finish its task. At the end the splash task is finished and the main activity is started.

Before analyzing the main activity we need to see the todo and todo adapter classes.

```
package com.example.todolist

import java.util.Calendar
import java.util.Date


data class Todo(private var title : String, private var
     date:Date = Calendar.getInstance().time, private
    var isChecked : Boolean = false)
{

    constructor() : this("")
    init{

        setTitle(title)
        setisChecked(isChecked)
        setDate(date)

    }
     fun getTitle(): String
    {

        return this.title

    }

    fun setTitle(title: String)
    {
        if(title.isBlank())
        {

            this.title = "Default_Title"


        }else
        {

            this.title = title


        }

    }

     fun getisChecked() : Boolean
    {

        return this.isChecked

    }

     fun setisChecked(isChecked: Boolean)
    {

        this.isChecked = isChecked

    }

    fun getDate(): Date
    {

        return this.date

    }

    fun setDate(date: Date)
    {

        this.date = date

    }


}
```

Code 9. Todo.kt

We define a data class that automatically provides methods such as toString, equals, hashCode, and copy. The class constructor has one mandatory and two optional parameters. If the date parameter is left blank it is automatically initialized with the date of the day the object was created. We also define a secondary constructor with no parameters. The init block is used to force the creation of the object to go through set functions. The following is the respect of the encapsulation principle using get and set functions.

```
object TodoAdapter : RecyclerView.Adapter<TodoAdapter.
    TodoViewHolder>()
{
    private val todos: MutableList<Todo> =
    mutableListOf()
    private var  database: DatabaseReference =
    FirebaseDatabase.getInstance().getReference("Todos
    ")
    class TodoViewHolder(itemView: View) : RecyclerView
    .ViewHolder(itemView)
    override fun onCreateViewHolder(parent: ViewGroup,
    viewType: Int): TodoViewHolder
    {
       return TodoViewHolder(LayoutInflater.from(parent
    .context).inflate(R.layout.item_todo, parent,
    false))
    }
```

Code 10. TodoAdapter_onCreateViewHolder

The TodoAdapter class works on the singleton model, because I wanted to have only one instance of this class, using the "object" declaration. For the database I used the libraries provided by Firebase. We use an inner class that extends RecyclerView.ViewHolder. The onCreateViewHolder function: creates and returns a new TodoViewHolder, which is an instance of the item_todo layout.

```
fun loadFromDataBase()
    {

        database.addValueEventListener(object :
    ValueEventListener {
            override fun onDataChange(snapshot:
    DataSnapshot) {

                todos.clear()

                for (data in snapshot.children) {
                    val todo = data.getValue(Todo::
    class.java)
                    todo?.let { addTodo(it) }
                }


                notifyDataSetChanged()
            }

            override fun onCancelled(error:
    DatabaseError) {
                Log.w("MainActivity", "loadTodos:
    onCancelled", error.toException())
            }
        })

    }
```

Code 11. TodoAdapter_loadFromDataBase

Items are removed from the todos list. Next the list is populated with items from the database. A warning message is returned if the process is interrupted.

```
    fun addTodo(todo : Todo)
    {
        var sameTitle = false

        for(eachTodo: Todo in todos)
        {

            if(todo.getTitle().trim().lowercase(Locale.
    getDefault()) == eachTodo.getTitle().trim().
```

```
    lowercase(Locale.getDefault()))
            {

                sameTitle = true

            }

            if(sameTitle)
            {

                break

            }

        }

        if(!sameTitle) {

            todos.add(todo)
            database.child(todo.getTitle()).setValue(
    todo)
            notifyItemInserted(todos.size - 1)

        }
    }
```

Code 12. TodoAdapter_addTodo

AddTodo function: adds a Todo to the todos list and to the Firebase database, if there is not already a Todo with the same title (ignoring spaces and upper/lower case differences).

```
    override fun onBindViewHolder(holder:
    TodoViewHolder, position: Int) {

        val curTodo = todos[position]
        holder.itemView.apply{

            val tvTodoTitle: TextView = findViewById(R.
    id.tvTodoTitle)
            val tvTodoDate: TextView = findViewById(R.
    id.tvTodoDate)
            val cbDone: CheckBox = findViewById(R.id.
    cbDone)


            tvTodoTitle.text = curTodo.getTitle()
            val myFormat = "dd-MM-yyyy"
            val sdf = SimpleDateFormat(myFormat, Locale
    .UK)
            tvTodoDate.text = sdf.format(curTodo.
    getDate())
            cbDone.isChecked = curTodo.getisChecked()


            cbDone.setOnCheckedChangeListener { _, _ ->


                database.child(curTodo.getTitle()).
    removeValue()
                todos.remove(curTodo)



            }

        }

    }
```

Code 13. TodoAdapter_onBindViewHolder

Function onBindViewHolder: binds Todo data to UI elements (TextView, CheckBox) for the specified position.

cbDone.setOnCheckedChangeListener: Removes Todo from Firebase and from the list when CheckBox is checked, and notifies the adapter about the removal of the item.

```
package com.example.todolist

import android.app.DatePickerDialog
import android.os.Bundle
import android.widget.Button
import android.widget.EditText
import androidx.activity.ComponentActivity
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import java.util.Calendar



class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?)
    {

        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

            val rvTodoItems: RecyclerView =
    findViewById(R.id.rvTodoItems)

            rvTodoItems.adapter = TodoAdapter
            rvTodoItems.layoutManager =
    LinearLayoutManager(this)

            setupUI()

    }

    private fun setupUI() {

        val btnAddTodo: Button = findViewById(R.id.
    btnAddTodo)
        val dateButton: Button = findViewById(R.id.
    datePickerButton)
        val myCalendar = Calendar.getInstance()

        val datePickerDialog = DatePickerDialog.
    OnDateSetListener { _, year, month, dayOfMonth ->
            myCalendar.set(Calendar.YEAR, year)
            myCalendar.set(Calendar.MONTH, month)
            myCalendar.set(Calendar.DAY_OF_MONTH,
    dayOfMonth)
        }

        dateButton.setOnClickListener {
            DatePickerDialog(this, datePickerDialog,
    myCalendar.get(Calendar.YEAR), myCalendar.get(
    Calendar.MONTH), myCalendar.get(Calendar.
    DAY_OF_MONTH)).show()
        }

        btnAddTodo.setOnClickListener {
            val etTodoTitle: EditText = findViewById(R.
    id.etTodoTitle)
            val todoTitle = etTodoTitle.text.toString()
            val todo = Todo(todoTitle, myCalendar.time)

            TodoAdapter.addTodo(todo)
            etTodoTitle.text.clear()
        }
    }

}
```

Code 14. MainActivity.kt

Finally we get to the main activity, that manages the user interface and user interaction. onCreate: Configures the activity on creation, initializes RecyclerView and calls setupUI. setupUI: Configures the UI, initializes the buttons, and manages the dialog for date selection and adding To-Do items.

5. Used Tools