Android Template Code

Jacob Conner

September 22, 2021

Contents

1 Gradle

This section examines the build.gradle files in the Project and App folders and the project Build.settings files.

1.1 ProjectBuildGradle

```
// Top-level build file where you can add configuration options common to all sub-projects/modules.
buildscript {
    ext {
        compose_version = '1.0.1'
    }
    repositories {
        google()
        mavenCentral()
    }
    dependencies {
        classpath "com.android.tools.build:gradle:7.0.2"
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:1.5.21"

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

Listing 1: Project build.gradle file

All Android projects consist of at least two *build.gradle* files, a project level *build.gradle* file and an application level *build.gradle* file. The project level *build.gradle* file is shown in Listing 1. Generally this *build.gradle* file contains a `buildscript` tag that has a number of subsections. The section mostly likely to be editted is the `ext` section. In this section gradle environment variables can be defined particularly to keep track of versions of various dependencies. In this example the `compose_version` variable is defined with `compose_version = '1.0.1'`. Dependencies can be defined in this section but generally should be defined using the application level *build.gradle* file.

1.2 AppBuildGradle

```
plugins {
    id 'com.android.application'
    id 'kotlin-android'
    id 'kotlin-kapt'
    id("org.jetbrains.dokka") version "1.4.0"
}

android {
    compileSdk 31

    defaultConfig {
        applicationId "com.example.roomandapi"
        minSdk 21
        targetSdk 31
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.
    AndroidJUnitRunner"
        vectorDrawables {
            useSupportLibrary true
        }
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-
    optimize.txt'), 'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = '1.8'
        useIR = true
    }
    buildFeatures {
        compose true
    }
    composeOptions {
        kotlinCompilerExtensionVersion compose_version
        kotlinCompilerVersion '1.5.21'
    }
```

```
    packagingOptions {
        resources {
            excludes += '/META-INF/{AL2.0,LGPL2.1}'
        }
    }
}

tasks.named("dokkaHtml") {
    outputDirectory.set(buildDir.resolve("dokka"))
}

dependencies {

    implementation 'androidx.core:core-ktx:1.6.0'
    implementation 'androidx.appcompat:appcompat:1.3.1'
    implementation 'com.google.android.material:material:1.4.0'
    implementation "androidx.compose.ui:ui:$compose_version"
    implementation "androidx.compose.material:material:$compose_version
    "
    implementation "androidx.compose.ui:ui-tooling-preview:
    $compose_version"
    implementation 'androidx.lifecycle:lifecycle-runtime-ktx:2.3.1'
    implementation 'androidx.activity:activity-compose:1.3.1'
    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core
    :3.4.0'
    androidTestImplementation "androidx.compose.ui:ui-test-junit4:
    $compose_version"
    debugImplementation "androidx.compose.ui:ui-tooling:
    $compose_version"

    //Navigation()
    implementation "androidx.navigation:navigation-compose:2.4.0-
    alpha04"

    def room_version = "2.3.0"
    implementation "androidx.room:room-runtime:$room_version"
    annotationProcessor "androidx.room:room-compiler:$room_version"
    implementation "androidx.room:room-ktx:2.3.0"
    kapt "androidx.room:room-compiler:2.3.0"

    implementation "androidx.compose.runtime:runtime-livedata:
    $compose_version"
    implementation "androidx.lifecycle:lifecycle-livedata-ktx:2.3.1"

    implementation("io.coil-kt:coil-compose:1.3.1")
```

```
85      //API handling
86    //  implementation "org.jetbrains.anko:anko-common:0.10.8"
87      implementation 'com.google.code.gson:gson:2.8.7'
88      implementation 'com.squareup.retrofit2:retrofit:2.9.0'
89      implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
90
91  }
```

Listing 2: App build.gradle file

This app-level *build.gradle* file is used to define the various plugins, application dependencies and gradle scripts pertinent to the application. In the example file in Listing 2 the file starts with plugins section. The plugins section consists of four plugins.:

- com.android.application
- kotlin-android
- Kapt plugin - kotlin-kapt
- Dokka plugin - org.jetbrains.dokka

`com.android.application` and `kotlin-android` are all included in the application by default. This example has added the plugin `kotlin-kapt` to enable the program to load the room compiler dependency using the kapt command. The dokka plugin needed to create documentation using KDoc is added with the `id("org.jetbrains.dokka") version "1.4.0"`. Various gradle tasks should be added in the application gradle file.

```
tasks.named("dokkaHtml"){
  outputDirectory.set(buildDir.resolve("dokka"))
```

Listing 3: Task to generate Dokka documentation

The task to generate dokka documentation is defined using the code in Listing 3 . Gradle documentation can be created using the terminal with the command `gradlew dokkaHtml`. Dokka will then parse all KDoc comment lines in Kotlin classes and generate the documentation in the build/dokka folder.

Dependencies are stored in the dependencies section of the gradle file. When a new empty compose application is created, a number of compose dependencies are included as well as various testing frameworks. The Jetpack compose navigation library and the jetpack compose lifecycle libraries are not included by default but are present in the example app *build.gradle* file. The project also makes use of RoomDatabase library so those libraries are included. In order to allow for the downloading and display of images from the internet the `io.coil-kt:coil-compose` library is included.

2 AndroidManifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.roomandapi">
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.
   ACCESS_NETWORK_STATE"/>
    <application
        android:usesCleartextTraffic="true"
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.RoomAndApi">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="@string/app_name"
            android:theme="@style/Theme.RoomAndApi.NoActionBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.
   LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Listing 4: Android Manifests File

The android manifests file shown in 4 provides information about the various permissions in the application.

```
<uses-permission android:name="android.permission.INTERNET" />
```

Listing 5: Line to give permission to allow internet access

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE
   "/>
```

Listing 6: Line to give permission to allow network access

The most common reason to edit this file is when an application needs to have access to a website or other resource. Internet access is enabled with the line shown in Listing 5. Network access is also needed to allow the application to access the web, and this is enabled with the line in Listing 6.

3 RoomDatabase

The RoomDatabase library is an object relational mapping (ORM) library in Android that serves as a layer for SQLite queries [**?**] .

3.1 Dependencies

The implementations are briefly discussed in Section 2, but a list of the required implementations in the *build.gradle* file are listed below.

- RoomDatabase Libraries
  - implementation ”androidx.room:room-runtime:$room_version”
  - annotationProcessor ”androidx.room:room-compiler:$room_version”
  - implementation ”androidx.room:room-ktx:2.3.0”
  - kapt ”androidx.room:room-compiler:2.3.0”
- Lifecycle Libraries
  - implementation ”androidx.compose.runtime:runtime-livedata:$compose_version”
  - implementation ”androidx.lifecycle:lifecycle-livedata-ktx:2.3.1”

3.2 Entities

The lowest level of RoomDatabase is the entity or data model. An entity is simply a data class that describes the schema for a time using Room annotations. The contents of each entity should correspond to the columns of a single SQLite table.

```
1  package com.example.roomandapi.entity
2
3  import androidx.room.ColumnInfo
4  import androidx.room.Entity
5  import androidx.room.PrimaryKey
6
7  @Entity(tableName = "team")
8  data class TeamMember(
9      @PrimaryKey(autoGenerate = true)
10     @ColumnInfo(name = "id")
11     var id: Int,
12
13     @ColumnInfo(name = "firstName")
14     var firstName: String,
15
16     @ColumnInfo(name = "lastName")
17     var lastName: String,
18
19     @ColumnInfo(name = "bio")
20     var bio: String,
21
22     @ColumnInfo(name = "imageUrl")
23     var imageUrl: String
24 )
```

Listing 7: Example entity class

In Listing 7, an example entity class `TeamMember` is provided for a table called team that stores information about members of a team. In this class, the first Room notation encountered is the `@Entity`notation. This notation is called before the data class is defined and is used to help the Room library identify where this model's data will reside in the SQLite database. In this case the table is called `Team` but the data class is called `TeamMember`. Next the data class is created using the `data class <nameOfClass>( <contents>)`. It is important to note that a data class in Kotlin does not contain any methods and is simply defined with the parameters using the parentheses that initialize it.

The next annotation used in this example class is `@PrimaryKey`. This annotation is used to define the column below as a primary key for the table, which means it has to be unique and cannot be null. The notation also has the possible parameter of `(autoGenerate = true)` which advises SQLite to autoincrement that column. The last annotation in the example entity class is `@ColumnInfo` which is used to define the name of the table column using the parameter

`name="<nameOfColumn>"`. Once the column name has been defined, a variable in kotlin is created for the model which is tied to the table column set. This process is repeated for all columns needed in the table and each column is separated by commas (”,”).

3.3 DAO

Once an entity has been created, a data abstraction object (DAO) has to be created. This is an interface for the entity that maps SQL queries to various methods that can be called.

```
1  package com.example.roomandapi.dao
2
3  import androidx.lifecycle.LiveData
4  import androidx.room.*
5  import com.example.roomandapi.entity.TeamMember
6
7  @Dao
8  interface TeamMemberDao {
9      @Query("SELECT * FROM team")
10     fun getAllTeamMembers(): LiveData<List<TeamMember>>
11
12     @Query("SELECT * FROM team WHERE id = :id")
13     suspend fun getById(id: Int): TeamMember
14
15     @Insert(onConflict = OnConflictStrategy.REPLACE)
16     suspend fun insert(item: List<TeamMember>)
17
18     @Update
19     suspend fun update(item: TeamMember)
20
21     @Delete
22     suspend fun delete(item: TeamMember)
23
24     @Query("DELETE FROM team")
25     suspend fun deleteAll()
26 }
```

Listing 8: Example DAO

An example of an DAO interface is shown in Listing 8. In this file, the interface is annotated as a DAO with the `@DAO` annotation. After the interface header is created with `interface <nameOfDAO>` the various SQL methods are defined with an annotation followed by a method. The first type of SQL Query annotation is `@Query\verb. This annotation takes a SQL query as a paramet` parameterized sql queries as shown in the `getById function` and the parameter in the query is used as a parameter in the method tied to that query. Room provides a few SQL query annotations that do not require you to specifically type out the SQL query. The `@Insert` annotation takes a list of entities and inserts them into a database using the method below. The option param-

eter `onConflict = OnConflictStrategy.REPLACE` allows you to specify the conflict strategy with foreign key constraints when a value is inserted. The `@Update` annotation is used to create an update query where an entity is provided and the values of the entity are used to update the value of the entity in the database. The annotation `@Delete` writes the query to delete the entity in the database that matches the entity provided in the method.

3.4 Repository

```
1  package com.example.roomandapi.repository
2
3  import androidx.lifecycle.LiveData
4  import com.example.roomandapi.dao.TeamMemberDao
5  import com.example.roomandapi.entity.TeamMember
6
7  class TeamMemberRepository (private val teamMemberDao: TeamMemberDao){
8      val readALlData: LiveData<List<TeamMember>> = teamMemberDao.
       getAllTeamMembers()
9
10     suspend fun addTeamMember(item: List<TeamMember>){
11         teamMemberDao.insert(item)
12     }
13
14     suspend fun updateTeamMember(item: TeamMember){
15         teamMemberDao.update(item)
16     }
17
18     suspend fun deleteTeamMember(item: TeamMember){
19         teamMemberDao.delete(item)
20     }
21
22     suspend fun deleteAll(){
23         teamMemberDao.deleteAll()
24     }
25 }
```

Listing 9: Example repository class

3.5 Database

```
1  package com.example.roomandapi.database
2
3  import android.content.Context
4  import androidx.room.Database
5  import androidx.room.Room
6  import androidx.room.RoomDatabase
7  import com.example.roomandapi.dao.TeamMemberDao
```

```
import com.example.roomandapi.entity.TeamMember

@Database(entities = [
    TeamMember::class
                      ],
    version = 1,
    exportSchema = false)
abstract class WCDatabase: RoomDatabase() {
    abstract fun teamMemberDao(): TeamMemberDao

    companion object{
        @Volatile
        private var INSTANCE: WCDatabase? = null

        fun getInstance(context: Context): WCDatabase {
            val sampleInstance = INSTANCE
            if(sampleInstance != null){
                return sampleInstance
            }
            synchronized(this){
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    WCDatabase::class.java,
                    "workout_companion_database"
                ).fallbackToDestructiveMigration().build()

                INSTANCE = instance
                return instance
            }


        }
    }
}
```

Listing 10: Example database class

3.6 ViewModel

```
package com.example.roomandapi.viewmodel

import android.app.Application
import androidx.lifecycle.*
import com.example.roomandapi.database.WCDatabase
import com.example.roomandapi.entity.TeamMember
import com.example.roomandapi.repository.TeamMemberRepository
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
```

```
import java.lang.IllegalArgumentException

class TeamMemberViewModel(application: Application): AndroidViewModel(application) {
    val readAllTeamMembers: LiveData<List<TeamMember>>
    private val repository: TeamMemberRepository

    init{
        val teamMemberDao = WCDatabase.getInstance(application).teamMemberDao()
        repository = TeamMemberRepository(teamMemberDao = teamMemberDao)
        readAllTeamMembers = repository.readALlData
    }

    fun addTeamMember(item: List<TeamMember>){
        viewModelScope.launch(Dispatchers.IO){
            repository.addTeamMember(item = item)
        }
    }

    fun updateTeamMember(item: TeamMember){
        viewModelScope.launch(Dispatchers.IO){
            repository.updateTeamMember(item = item)
        }
    }

    fun deleteTeamMember(item: TeamMember){
        viewModelScope.launch(Dispatchers.IO){
            repository.deleteTeamMember(item = item)
        }
    }

    fun deleteAllTeamMembers(){
        viewModelScope.launch(Dispatchers.IO){
            repository.deleteAll()
        }
    }
}

class TeamMemberViewModelFactory(
    private val application: Application
): ViewModelProvider.Factory{
    override fun <T: ViewModel?> create(modelClass: Class<T>): T{
        @Suppress("UNCHECKED_CAST")
        if(modelClass.isAssignableFrom(TeamMemberViewModel::class.java)){
```

```
            return TeamMemberViewModel(application) as T
        }
        throw IllegalArgumentException("Unknown ViewModel class")
    }
}
```

Listing 11: Example ViewModel class

4 JetPack Compose

4.1 Navigation

```
package com.example.roomandapi.views

import android.app.Application
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material.Button
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.livedata.observeAsState
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextDecoration
import androidx.compose.ui.unit.sp
import androidx.lifecycle.viewmodel.compose.viewModel
import androidx.navigation.NavController
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import androidx.navigation.compose.rememberNavController


import com.example.roomandapi.entity.TeamMember

@Composable
fun appNavController() {

    val navController = rememberNavController()
    NavHost(navController, startDestination = "main") {
        composable(route = "main") {
```

```
            MainView(navController)
        }
        composable(route = "about") {
            AboutView(navController)
        }
    }
}


@Composable
fun MainView(navController: NavController) {
        Column(){
            Button(onClick = { navController.navigate("about") }) {
                Text("About Us")
            }
            Column(
                modifier = Modifier
                    .fillMaxSize()
                    .background(Color.White),
                horizontalAlignment = Alignment.CenterHorizontally,
                verticalArrangement = Arrangement.Center
            ) {
                Text(
                    text = "Workout Companion",
                    fontSize = 30.sp,
                    fontWeight = FontWeight.Bold,
                    textDecoration = TextDecoration.Underline,
                    fontFamily = FontFamily.Serif
                )
            }
        }
    }
```

Listing 12: Example database class

```
@Composable
fun appNavController() {

    val navController = rememberNavController()
    //default destination
    NavHost(navController, startDestination = "nameOfDefaultRoute") {
      //list routes
        composable(route = "<nameOfFirstRoute>") {
              //viewToLoad
      firstView(navController)
        }

        composable(route = "<nameOfSecondRoute>") {
```

```
      secondView(navController)
        }
    }
}
```

Listing 13: Function to create a simple Nav Controller

```
Button(onClick = { navController.navigate("<routeToNavigateTo") }) {
                //Label for button
                Text("Name of Button")
            }
}
```

Listing 14: Button to navigate to a view

5 Documenting with Dokka

test

List of Figures

List of Tables

Listings

References

[1] Google. Save data in a local database using room. Accessed on 2021-9-22.