5 MEI, 2023

Adrian Biedny

AndroiD (Kotlin)   
Voor mobile app development

Stolos Fleet Management API

Contents

[Waarom een Android app (met Kotlin)? 2](#_Toc134210025)

[Mijn aanpak. 2](#_Toc134210026)

[Gebruikte technologieën. 2](#_Toc134210027)

[Mijn code (snippets met uitleg). 3](#_Toc134210028)

[De app: aantrekkelijk, mogelijke veranderingen? 10](#_Toc134210029)

[Screenshots van de app. 10](#_Toc134210030)

# Waarom een Android app (met Kotlin)?

Aanvankelijk was ik van aan het twijfelen tussen een Android app (met Kotlin) of een front-end pagina met Svelte. Uiteindelijk heb ik ervoor gekozen om met Kotlin te gaan werken want ik ookal ben ik capabel om full-stack te werken, mijn front-end talenten toch wel beperkt zijn. Een Android app heeft ook natuurlijk een UI / front-end aspect, echter zit hier meer logica in dan creatieve en schone “looks”.

Ik heb zelf ook al 9.5 jaar met Java gewerkt dus ik heb al wat ervaring met de JVM. Rond 2018 heb ik gehoord van andere JVM-talen zoals Scala en Kotlin, maar ik had nooit de tijd of het excuus om er een project in te maken. Nu dat de kans zich presenteerde, sprong ik erop zonder twijfel.

Er zijn natuurlijk andere mobile talen waarin ik een app heb kunnen maken zoals React native bv, maar aangezien we die taal bij Mobile gaan leren was Kotlin “the way to go”.

# Mijn aanpak.

Ik ben begonnen met de basis van Kotlin en de Android libs eens te bekijken en in mijn vingers te krijgen door de simpelste applicatie ooit te maken: een wachtwoord generator.

Daarna ben ik gaan kijken naar de conventies en de meest gebruikte libs en frameworks bij het ontwikkelen van een API consuming app.

Dat werd dan op zich gevolgd door het programmeren, (manueel) testen, debuggen, bug-fixen, … Rinse and repeat.

Uiteindelijk heb ik toch wel een relatief schoon, werkend project.

# Gebruikte technologieën.

Ik maakte gebruik van verschillende technologieën om dit project gemaakt te krijgen.

* Kotlin
  + De programmeertaal die de Android app doet werken.
* Android
  + Verschillende componenten en de runtime environment waarin Kotlin in dit geval draait.
* Retrofit
  + Een lib voor Android voor het consumen van API’s.
* RecyclerView
  + Een lib voor AndroidX om op een efficiënte manier data te tonen (enkel de getoonde items in de view worden in de run memory gehouden).
* XML
  + Een equivalente aan WPF, om UI (componenten) te maken.
* Gradle
  + Een build-automatiseringstool (vaak gebruikt bij Java, komt als standaard voor Android apps in Kotlin). Vergemakkelijkt implementeren, testen, deployen, publishen, …
* C# RESTful API (ADO.net -> MySQL server als DB) [Ons groepsproject: Stolos Fleet Management API]
* Ubuntu 22.04 server

# Mijn code (snippets met uitleg).

dependencies **{** implementation 'androidx.core:core-ktx:1.8.0'  
 implementation 'androidx.lifecycle:lifecycle-runtime-ktx:2.3.1'  
 implementation 'androidx.activity:activity-compose:1.5.1'  
 implementation platform('androidx.compose:compose-bom:2022.10.00')  
 implementation 'androidx.compose.ui:ui'  
 implementation 'androidx.compose.ui:ui-graphics'  
 implementation 'androidx.compose.ui:ui-tooling-preview'  
 implementation 'androidx.compose.material3:material3'  
  
 implementation 'com.squareup.retrofit2:retrofit:2.8.1'  
 implementation 'com.squareup.retrofit2:converter-gson:2.8.1'  
 implementation 'com.squareup.okhttp3:okhttp:4.10.0'  
 implementation 'com.squareup.okhttp3:logging-interceptor:4.0.1'  
 implementation 'androidx.constraintlayout:constraintlayout:2.1.4'  
 implementation 'androidx.constraintlayout:constraintlayout-compose:1.0.1'  
 implementation 'com.google.android.material:material:1.5.0'  
 implementation 'androidx.recyclerview:recyclerview:1.3.0'  
  
 testImplementation 'junit:junit:4.13.2'  
 androidTestImplementation 'androidx.test.ext:junit:1.1.3'  
 androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'  
 androidTestImplementation platform('androidx.compose:compose-bom:2022.10.00')  
 androidTestImplementation 'androidx.compose.ui:ui-test-junit4'  
 debugImplementation 'androidx.compose.ui:ui-tooling'  
 debugImplementation 'androidx.compose.ui:ui-test-manifest'  
**}**

In mijn build.gradle implementeer ik verschillende dependencies zoals bv. retrofit of recyclerview. Dit zijn libs/frameworks/componenten die ik gebruikt heb voor dit project.

<activity  
 android:name=".MainActivity"  
 android:exported="true"  
 android:label="@string/app\_name"  
 android:theme="@style/Theme.StolosAPFMA">  
 <intent-filter>  
 <action android:name="android.intent.action.MAIN" />  
  
 <category android:name="android.intent.category.LAUNCHER" />  
 </intent-filter>  
</activity>

In mijn AndroidManifest.xml zet ik de MainActivity als het startup object.

class MainActivity : ComponentActivity() {  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_main*)  
  
 *onBackPressedDispatcher*.addCallback(this, object : OnBackPressedCallback(true) {  
 override fun handleOnBackPressed() {  
 finishActivity(-1)  
 }  
 })  
  
 val btnDrivers = findViewById<Button>(R.id.*btnDrivers*)  
 val btnVehicles = findViewById<Button>(R.id.*btnVehicles*)  
 val btnFuelCards = findViewById<Button>(R.id.*btnFuelCards*)  
  
 btnDrivers.setOnClickListener **{** Toast.makeText(this, "Drivers", Toast.*LENGTH\_SHORT*).show()  
 startActivity(Intent(this, ActivityDrivers::class.*java*))  
 **}** btnVehicles.setOnClickListener **{** Toast.makeText(this, "Vehicles", Toast.*LENGTH\_SHORT*).show()  
 startActivity(Intent(this, ActivityVehicles::class.*java*))  
 **}** btnFuelCards.setOnClickListener **{** Toast.makeText(this, "Fuel cards", Toast.*LENGTH\_SHORT*).show()  
 startActivity(Intent(this, ActivityGasCards::class.*java*))  
 **}** }  
}

In de MainActivity zelf gebeurt er niets speciaals. Ik override het default “terug toets” gedrag door mijn eigen callback toe te voegen (dit heb ik in elke Activity gedaan om het gewenste gedrag te bekomen).

Voor de rest registreer ik onClickEvents om naar volgende Activities te gaan en maak ik Toasts om wat info te tonen. Een activity het equivalente van andere schermen/pagina’s in de applicatie. Een Toast is zoals een alert() in javascript, maar minder frustrerend want het forceert geen interactie (om het te sluiten bv).

val retrofit: Retrofit = Retrofit.Builder()  
 .baseUrl(BASE\_URL)  
 .client(okHttpClient)//okHttpClient defined in ACCEPT\_SPECIFIC\_TRUSTED\_CERTIFICATE//.client(okHttpClient)//okHttpClient defined in IGNORE\_UNTRUSTED\_HTTPS  
 .addConverterFactory(GsonConverterFactory.create())  
 .build()  
return retrofit

De RetrofitFactory klasse geeft eigenlijk dit stukje code terug dat op meerdere plaatsen in mijn app wordt aangesproken om verbinding te maken met de API. De BASE\_URL geeft de URL van de API, de GsonConverterFactory zorgt voor (de)serialisatie van Json objecten en in de okHttpClient heb ik gedefinieerd dat de verbinding met de API toegelaten mag worden (aangezien dat de API in Development loopt en geen trusted https ssl certificaat heeft. Dit heb ik opgelost op 2 verschillende manieren (zelf te kiezen door code in/uit commentaar te zetten):

1. Ik heb het ssl certificaat van de API/server meegegeven dus de app weet of dat de verbinding te vertrouwen is.
2. Je negeert alle niet vertrouwde certificaten.

Een van deze twee oplossingen is veiliger dan de andere… 😉

interface ApiService {  
 @GET("api/drivers")  
 fun getDrivers(): Call<ArrayList<DriverModel>>  
 @GET("api/drivers/{id}")  
 fun getDriverById(@Path("id") driverId: Int): Call<DriverModel>  
 @POST("api/drivers")  
 fun addDriver(@Body driverModel: DriverModel): Call<Unit>  
 @PUT("api/drivers")  
 fun updateDriverById(@Body driverModel: DriverModel): Call<Unit>  
 @DELETE("api/drivers/{id}")  
 fun deleteDriverById(@Path("id") driverId: Int): Call<Unit>  
}

In de ApiService interface definieer ik de endpoints van de API. Hier zet ik ook de postfix van de BASE\_URL die zich in de RetrofitFactory bevindt. Ik zeg ook welke Call/Callback verwacht. In het geval van de GET’s bij onze API krijg je data terug (in ons geval, een DTO), maar bij de POST, PUT en DELETE krijg je bij ons niks terug qua data (wel natuurlijk een http-code).

data class DriverModel (  
 val driverID: Int?,  
 var firstName: String,  
 val lastName: String,  
 val birthDate: String,  
 val natRegNum: String,  
 val licenses: List<String>,  
 val address: String? = null,  
 val vehicleVin: String? = null,  
 val gasCardNum: String? = null  
)

Zo ziet het DriverModel eruit in mijn app. Het is een data class, d.w.z. dat het enkel data behoud, perfect voor binnen komende DTO’s.

val api: ApiService = RetrofitFactory(this).Retrofit().create(ApiService::class.*java*)

val rv: RecyclerView = findViewById(R.id.*rvDrivers*)

api.getDrivers().enqueue(object : Callback<ArrayList<DriverModel>> {  
 override fun onResponse(call: Call<ArrayList<DriverModel>>, response: Response<ArrayList<DriverModel>>) {  
 if(response.*isSuccessful*) {  
 rv.*apply* **{** *layoutManager* = LinearLayoutManager(this@ActivityDrivers)  
 *adapter* = DriverAdapter(context = *context*, response.body()!!)  
 **}** }  
 }  
 override fun onFailure(call: Call<ArrayList<DriverModel>>, t: Throwable) {  
 Log.e("ADBILOGSTOLOS", t.message.*toString*())  
 }  
})

In de ActivityDrivers is deze code terug te vinden. We maken een instantie van de ApiService d.m.v. Retrofit. Dit is een Call van een ArrayList van DriverModels. We versturen het als een call asynchroon naar de API/server en we melden een callback van een response (die succesvol (code 2xx-3xx) is of niet (4xx-5xx) of we melden een failure).

In deze code als het succesvol is, dan geven we de body van de response door naar de Adapter om het te vertalen naar zelfgemaakte .xml componenten die dan de RecyclerView in de activity\_drivers.xml gaan opvullen.

<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:background="#73E48B"  
 android:layout\_margin="15dp"  
 android:orientation="vertical"  
 android:padding="15dp">

<TextView  
 android:id="@+id/tvName"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:textColor="@color/black"  
 android:textSize="30sp"  
 tools:text="FirstName LastName"/>  
</LinearLayout>

Dit is een simpel card\_driver.xml component voor Driver (dit komt in de RecyclerView terecht via de Adapter).

<?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="match\_parent"  
 tools:context=".activities.ActivityDrivers">  
  
 <ImageView  
 android:id="@+id/ivLogo"  
 android:layout\_width="match\_parent"  
 android:layout\_height="200dp"  
 android:layout\_margin="20dp"  
 android:src="@drawable/logo"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent" />  
  
 <androidx.recyclerview.widget.RecyclerView  
 android:id="@+id/rvDrivers"  
 android:layout\_width="409dp"  
 android:layout\_height="500dp"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/ivLogo" />  
  
 <Button  
 android:id="@+id/btnAddDriver"  
 android:text="ADD"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent" />  
</androidx.constraintlayout.widget.ConstraintLayout>

Dit is de ActivityDrivers view. Het heeft enkel een logo, een knop om drivers toe te voegen en de RecyclerView die opgevuld is met card\_driver.xml componenten.

class DriverAdapter(val context: Context, objects:ArrayList<DriverModel>) : RecyclerView.Adapter<DriverAdapter.DriverViewHolder>() {  
 private val drivers = objects  
 override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): DriverViewHolder {  
 val view = LayoutInflater.from(parent.*context*).inflate(R.layout.*card\_driver*, parent, false)  
 return DriverViewHolder(view)  
 }  
 override fun getItemCount(): Int {  
 return drivers.size  
 }  
 override fun onBindViewHolder(holder: DriverViewHolder, position: Int) {  
 val driver: DriverModel = drivers[position]  
 return holder.bindView(driver)  
 }  
 inner class DriverViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {  
 private val tvName: TextView = itemView.findViewById(R.id.*tvName*)  
 fun bindView(d: DriverModel) {  
 tvName.*text* = "${d.firstName} ${d.lastName}"  
 itemView.setOnClickListener**{** context.startActivity(Intent(context, ActivityDetailDriver::class.*java*).putExtra("driverID", d.driverID))  
 **}** }  
 }  
}

In de DriverAdapter geven we parameters we als bronnen van dat wat we hebben. Er zijn een paar override verplicht en we maken onze eigen Holder aan. In de DriverViewHolder definiëren welke data dat het toont en in de onCreateViewHolder van de Adapter zelf geven we de benodigde data mee. In de DriverViewHolder kunnen we ook events registreren zoals in dit geval, een onClickEvent. Hier opent het een detail scherm van een Driver en we gebruiken .putExtra(“driverID”, d.driverID) om de driverID mee te geven. Dit zal later gebruikt worden voor de getDriverById methode. Om de DriverDetailActivity op te vullen met een specifieke Driver die opgehaald zal worden van de API/server.

val driverId: Int = *intent*.getIntExtra("driverID", 0)  
  
if (driverId!=0) {

btnDelete.setVisibility(View.*VISIBLE*)

// REGISTER UPDATE

// REGISTER DELETE

} else {// code for creating new driver

btnDelete.setVisibility(View.*GONE*)

// REGISTER ADD

}

In de ActivityDetailDriver maken we verschillend gedrag mogelijk door te kijken of er een driverID meegegeven werd (vanuit de aangeklikte card\_driver.xml die gedrag had gekregen van de DriverViewHolder in de DriverAdapter).

//region REGISTER UPDATE  
btnSave.setOnClickListener(View.OnClickListener **{** var updatedD = DriverModel(driverId, evFirstName.*text*.toString(), evLastName.*text*.toString(), evBirthDate.*text*.toString(),  
 evRRN.*text*.toString(), evLicenses.*text*.toString().*split*(","), evAddress.*text*.toString(), evVehicle.*text*.toString(), evGasCard.*text*.toString())  
 val call: Call<Unit> = api.updateDriverById(updatedD)  
 call.enqueue(object : Callback<Unit> {  
 override fun onResponse(call: Call<Unit>, response: Response<Unit>) {  
 startActivity(Intent(this@ActivityDetailDriver, ActivityDrivers::class.*java*))  
 }  
 override fun onFailure(call: Call<Unit>, t: Throwable) {  
 Log.e("ADBILOGSTOLOS", t.message.*toString*())  
 }  
 })  
**}**)  
//endregion

Als er een driverID is en we drukken op update, willen we updaten.

//region REGISTER DELETE  
btnDelete.setOnClickListener **{** val call: Call<Unit> = api.deleteDriverById(driverId)  
 call.enqueue(object : Callback<Unit> {  
 override fun onResponse(call: Call<Unit>, response: Response<Unit>) {  
 if (response.*isSuccessful*) {  
 startActivity(Intent(this@ActivityDetailDriver, ActivityDrivers::class.*java*).putExtra("DELETED-INFO",  
 "${d.firstName} ${d.lastName}"))  
 } else {  
 Toast.makeText(*baseContext*, response.code().toString(), Toast.*LENGTH\_SHORT*)  
 Log.e("ADBILOGSTOLOS ${driverId}", response.code().toString())  
 }  
 }  
 override fun onFailure(call: Call<Unit>, t: Throwable) {  
 Toast.makeText(*baseContext*, response.code().toString(), Toast.*LENGTH\_SHORT*)  
 Log.e("ADBILOGSTOLOS", t.message.*toString*())  
 }  
 })  
**}**//endregion

Als er een driverID is en we drukken op delete, willen we verwijderen.

//region REGISTER ADD  
btnSave.setOnClickListener**{** var licenses = ArrayList<String>()  
 for (l in evLicenses.*text*.toString().*split*(",")) {  
 licenses.add("$l")  
 }  
 var cDriver = DriverModel(  
 null,  
 evFirstName.*text*.toString(),  
 evLastName.*text*.toString(),  
 evBirthDate.*text*.toString(),  
 evRRN.*text*.toString(),  
 licenses,  
 evAddress.*text*.toString(),  
 evVehicle.*text*.toString(),  
 evGasCard.*text*.toString()  
 )  
  
 Log.i("ADBILOGSTOLOS", cDriver.toString())  
 api.addDriver(cDriver).enqueue(object : Callback<Unit> {  
 override fun onResponse(call: Call<Unit>, response: Response<Unit>) {  
 startActivity(Intent(this@ActivityDetailDriver, ActivityDrivers::class.*java*))  
 }  
 override fun onFailure(call: Call<Unit>, t: Throwable) {  
 Log.e("ADBILOGSTOLOS", t.message.*toString*())  
 }  
 })  
**}**//endregion

En natuurlijk als er geen driverID is en we drukken op add (in ActivityDrivers dan, niet in ActivityDetailDriver), dan willen we toevoegen. Bij onze Stolos Fleet Management API slaan we gebruikers op in de DB met een auto increment id, deze kunnen we zelf niet zetten, daarom dat er geen driverID is.

# De app: aantrekkelijk, mogelijke veranderingen?

Op zich ziet de app er zeker niet slecht uit. Er zijn nog een paar dingen dat ik zou kunnen aanpassen zoals bv een spinner voor het linken van gascards🡨🡪drivers🡨🡪vehicles zodat je kan kiezen uit elk object dat nog niet gelinkt is met een ander object, ook zou dit handig zijn voor een spinner van meerdere elementen zoals rijbewijzen of brandstof types; op dit moment zal ik dit niet implementeren want ik heb nog een hele boel andere projecten om uit te werken [05/05/2023].

# A screenshot of a phone Description automatically generated with medium confidenceA screenshot of a cell phone Description automatically generated with low confidenceScreenshots van de app.

A picture containing text, monitor, electronics, screenshot

Description automatically generatedA screenshot of a phone

Description automatically generated with low confidence

A screenshot of a phone

Description automatically generated with low confidenceA screenshot of a phone

Description automatically generated with medium confidence

Graphical user interface, application, chat or text message

Description automatically generatedA screenshot of a cell phone

Description automatically generated with low confidence

A picture containing text, electronics, screenshot, display

Description automatically generatedA screen shot of a cell phone

Description automatically generated with low confidence

# Mijn bevindingen en ervaringen

Ik vond het geweldig om terug eens te programmeren in een JVM-taal. Ik mis het dagelijks gebruik van Java, er is ook een heleboel nostalgie voor mij daar. Echter vind ik wel dat het programmeren in Kotlin vlotter ging dan ik mij kan herinneren in Java. Kotlin lijkt er natuurlijk zeer hard op met het feit dat het ook een “Factory hell is” maar doordat type-inferred is lijkt het sneller te gaan? Ik kan mijn vinger er niet exact op leggen maar het was zeer aangenaam om in Kotlin te werken. Alles van Kotlin zelf, maar ook de Android libs was goed en leesbaar gedocumenteerd. Retrofit en RecyclerView zijn ook 2 van de meest gebruikte libs voor Android dus het was zeer makkelijk om oplossingen te vinden als er problemen waren. En dan natuurlijk XML en Gradle, zo groot en bekend, elk probleem dat je maar kan hebben, heeft iemand ooit al eens gehad dus er zijn gegarandeerd bronnen online met oplossing voor jouw specifiek probleem.

Ik kan het ontwikkelen van een mobile app in Kotlin niet echt vergelijken met andere mobile / cross-platform talen, aangezien dat ik geen mobile apps in andere talen gemaakt heb.

Ik kan wel zeggen dat vanuit mijn ervaring, Kotlin als programmeertaal (niet alleen voor mobile apps) goed werkt, maar beter nog (voor programmeurs dan) goed aanvoelt om in te werken.