



11 JUNI, 2023

ANDROID (KOTLIN)
VOOR MOBILE APP DEVELOPMENT
STOLOS FLEET MANAGEMENT API

ADRIAN BIEDNY



Contents

NL Samenvatting	2
EN Summary	3
Motivatatie projectkeuze	4
Gebruikte technologieën.	5
Kotlin: “onder de loep”	5
DVM vs ART	5
Kotlin Mobile Multiplatform	5
Mijn aanpak.	6
Mijn code (snippets met uitleg).	7
Hoe werkt de app? Dev log.	14
Screenshots van de app.	14
V1	14
V2	19
Mijn ervaringen en wat ik geleerd heb.	21
Links	26

NL Samenvatting

Dit is het eindverslag als onderdeel van mijn graduaatsproef voor de graduaatsopleiding programmeren op HoGent. Dit verslag schijnt wat licht op volgende zaken:

- Waarom ik voor dit project koos.
- Uitleg over de specifieke kenmerken van mijn project (en andere relevante interessante technologieën).
- Hoe ik tot mijn eindresultaat kwam.
- Hoe mijn eindresultaat eruit ziet.
- Technische uitleg (hoe je de app kan starten en gebruiken / code snippet uitleg).

EN Summary

This is the final report/thesis as part of my final project for the associate degree programming at HoGent. This report sheds some light on the following matters:

- Why I chose this project.
- An explanation of the specifics of my project (and other relevant interesting technologies).
- How I arrived at my result.
- What my result looks like.
- Technical explanation (how to start and use the app / code snippet explanation).

Motivatie projectkeuze

Het vak “Projectwerk – Graduaatsproef” waarvoor dit verslag geschreven is (specifiek Graduaatsproef) heeft 2 onderdelen. Voor het Projectwerk gedeelte moesten we in groepen RESTful API's (C#.Net 7.0) als backend maken met React als frontend. Ik was in de groep “Stolos”. Wij hadden het bedrijf AllPhi als klant gekregen en we mochten voor hen een Fleet Management applicatie ontwikkelen. In Stolos was ik verantwoordelijk voor de backend (API, Business laag, Data laag, databank (MySQL), server (Linux Ubuntu 22.04)). Het server gedeelte was niet een van de vereiste. Ik had zelf een server ter beschikking en als groep hadden we gekozen dat een hands-on demo veel leuker zou zijn dan lokaal te hosten. De groeps-website is te vinden op <http://affiche.me:3000> en de groeps-API op <https://affiche.me:7144> het kan echter zijn dat na de graduaatsproef individuele presentaties deze niet meer zullen runnen op de server, in dat geval is het mogelijk om de source code van mijn GitHub repository te downloaden en zelf lokaal uit te proberen (<https://github.com/NotAffiche/>).

Dan nu voor het Graduaatsproef gedeelte. De code die we als groep hadden ontwikkeld was in 2 delen. De frontend en de backend. Voor onze individuele graduaatsproeven mochten we 1 van deze 2 delen kiezen en herschrijven in een taal of framework naar keuze (het moest wel een originele combinatie van taal en onderdeel zijn binnen onze groepjes). Daarom, ook al is dit project mijn graduaatsproef, werkte ik aan dit project niet met een “finished/polished product” attitude. Ik werkte hier aan met een R&D mindset. Zo veel mogelijk nieuwe technologie uitproberen en te leren.

Zoals eerder vermeld, in de groep werkte ik aan de backend. De reden hiervoor is: ik heb weinig tot geen frontend talent. Daarom wou ik mezelf een beetje een uitdaging geven door iets frontend (achtig) te maken. Er waren 2 technologieën die ik voor al een hele tijd wou uitproberen: Svelte (JS website framework) en Kotlin (& Kotlin MMP naar daarover meer in Kotlin: “onder de loep”). Uiteindelijk heb ik toch voor Kotlin gekozen want volgens mij is een mobile app toch een beetje minder frontend intensief (toch nog meer logica) dan een website. Daarbij is een React frontend website al onderdeel van het groeps gedeelte en het vak “Mobile Development” is pas voor volgend semester (en het is in React Native). Zelf heb ik ook al veel ervaring met Java (~ 9 jaar) maar ik heb er al een tijd niet mee gewerkt dus mis ik de JVM omgeving.

Gebruikte technologieën.

Ik maakte gebruik van verschillende technologieën om dit project te maken.

- Android Studio
 - De IDE waarin ik het project gemaakt heb en emulator die ik gebruikte voor het debuggen van de app.
- Kotlin (JetBrains, n.d.)
 - De programmeertaal waarin de Android app geprogrammeerd is.
- Android 12 (Runtime [ART]) SDK/API 31 – Snow Cone
 - Verschillende componenten en de runtime environment waarin Kotlin draait (Kotlin SRC -> Kotlin Compiler -> Java Bytecode -> DEX Compiler -> Dalvik Bytecode -> Android Runtime (ART) -> Machine Code).
- Retrofit
 - Een code bibliotheek voor het consumeren van API's.
- RecyclerView
 - Een component uit de AndroidX code bibliotheek om op een efficiënte manier data te tonen (sneller, minder “memory hogging” dan ListView).
- 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

Kotlin: “onder de loep”

Er is te veel te vertellen over Kotlin dus hier zijn een paar interessante dingen waarvan Kotlin gebruik maakt:

DVM vs ART

Vanaf Android KitKat (API Level 19 / Versie 4.4) in 2013 (dus ook mijn project) maakt Android gebruik van ART (Android Runtime) in plaats van de Dalvik interpreter voor code compilatie. Zo is er Ahead-of-time [AOT] compilatie (bij ART), en niet Just-in-time [JIT] compilatie, zoals bij Dalvik. DEX compiler wordt wel gebruikt om Dalvik bytecode te creëren. Het is enkel de Runtime (ART i.p.v. DVM (Dalvik Virtual Machine)) die is veranderd. Dit zorgt wel voor veranderingen: betere prestatie, strakkere installatietijdverificatie, verbeterde garbage collection,...

Kotlin Mobile Multiplatform

Alhoewel heb ik de KMM plugin niet gebruikt voor dit project, bestaat deze (sinds augustus 2020). Vroeger waren mobile applicaties geschreven enkel voor Android maar door KMM is het mogelijk om Android en iOS applicaties te maken met Kotlin. Dit gebeurt door de applicatie te verdelen in verschillende projecten. Een “shared” gedeelte en de 2 frontend delen (1 voor Android en 1 voor iOS). De Android en shared delen kunnen beide in Kotlin geschreven worden terwijl het iOS deel in Swift geschreven wordt (de plugin maakt automatisch een XCODE project voor je aan dat je zelf dan in XCODE moet openen en daar de code kan schrijven en een iPhone of ander iOS device kan emuleren).

Mijn aanpak.

Ik ben begonnen met Kotlin in mijn vingers te krijgen (simpel console app project: wachtwoord generator). Ik was direct mee met de syntax en werking van Kotlin zelf dus verder met het volgende: de console wachtwoord generator vertalen naar een Android app wachtwoord generator. Zo heb ik de werking van Android zelf kunnen aanleren. Hoe het werkt met activiteiten en de syntax van XML een beetje opgefrist.

Nadien heb ik een andere simpele app geschreven om Retrofit uit te testen en kijken hoe ik een API moet consumeren (<https://jsonplaceholder.typicode.com/>).

Eenmaal dat ik Retrofit heb begrepen ben ik begonnen aan het eindproduct.

Verder in dit verslag zijn er code snippets en screenshots van de app te vinden. Het kan zijn dat deze niet meer up2date zijn aangezien dat dit verslag een deadline heeft van 11/06/2023 om 23:59 en we mogen doorwerken aan de app tot de dag voor de presentatie (in mijn geval: 14/06).

Ik zal proberen om screenshots van verschillende versies hier te plaatsen.

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 equivalent 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.

Hoe werkt de app? Dev log.

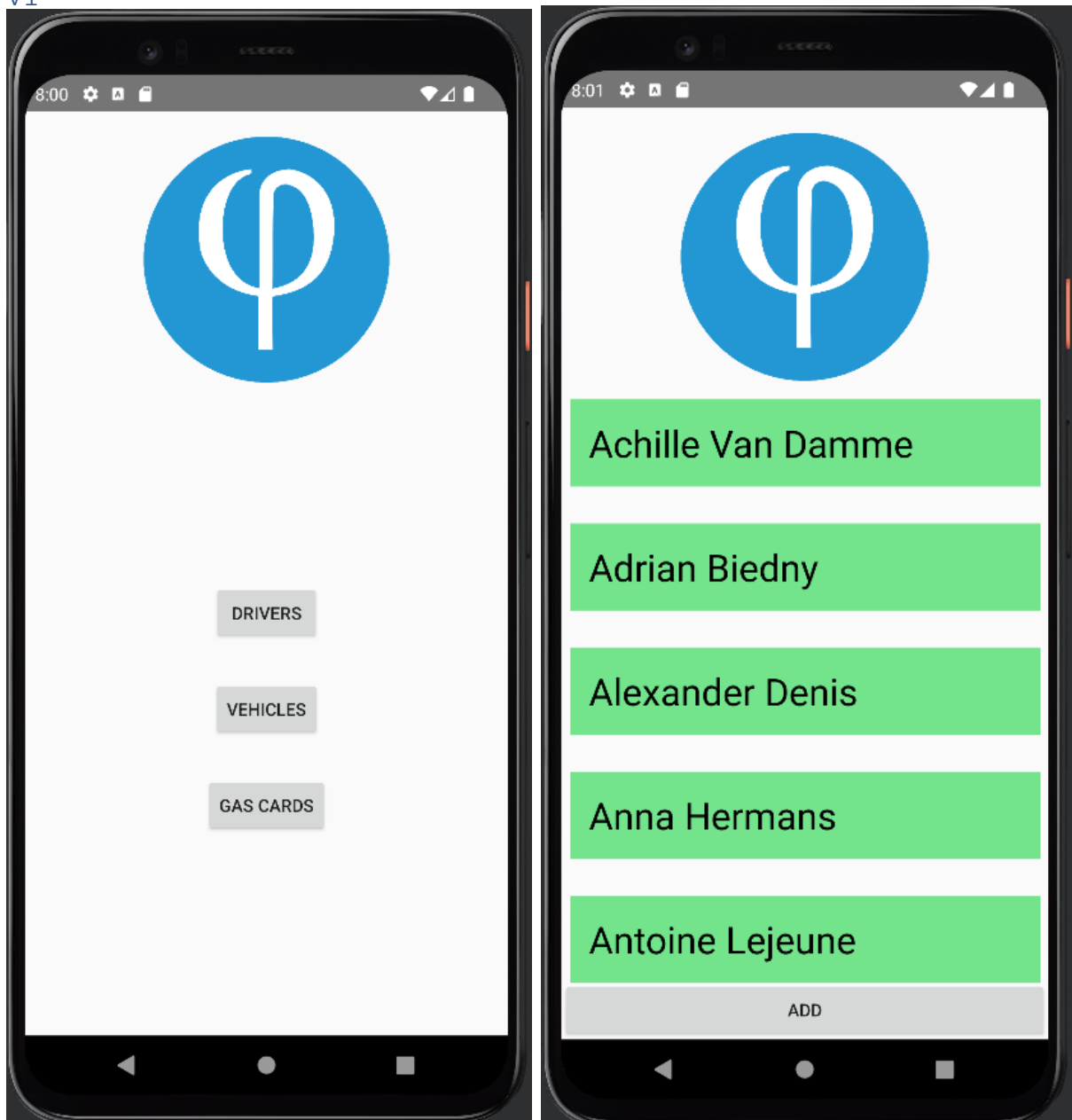
In het groepsproject (te vinden op GitHub) staat de lijst van specificaties die verwacht waren voor de frontend en backend van het groepsproject. Ik heb de werking van de app op dezelfde manier laten verlopen (bv. adres van bestuurders mag leeg zijn,...). Om zelf te debuggen, via Android Studio lanceren.

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 \leftrightarrow drivers \leftrightarrow 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].


Ondertussen zijn er al veel aanpassingen aangebracht, screenshots te vinden onder V2. Ik heb een kalender toegevoegd, multiselects en input validatie / formatting (voor rijksregisternummers). [10/06/2023]

Screenshots van de app.

V1



8:01



First Name: Adrian

Last Name: Biedny

Birth Date: 2001-08-09

RRN: 01.08.09-423.92


Licenses: AM,A1,A2,B

Address: Sint-niklaas

Vehicle VIN: 1122334455667
7011

GC Num: _____

8:04



First Name: _____

Last Name: _____

Birth Date: _____

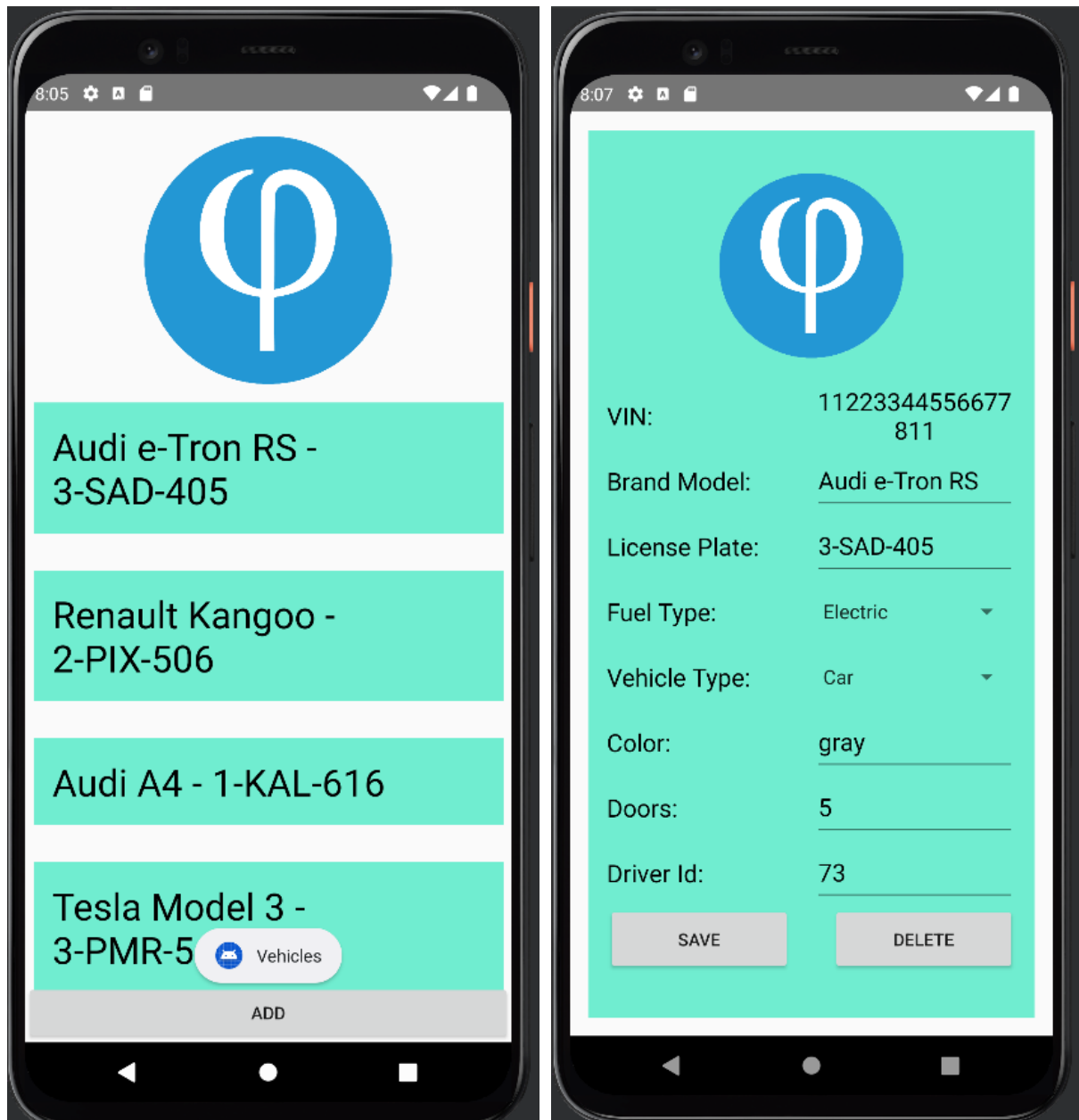
RRN: _____

Licenses: _____


Address: _____

Vehicle VIN: _____

GC Num: _____



8:08



VIN: _____

Brand Model: _____

License Plate: _____

Fuel Type: Unknown ▾

Vehicle Type: Unknown ▾

Color: _____

Doors: _____

Driver Id: _____

SAVE

8:08



00860672806124688451

01445121330856533405

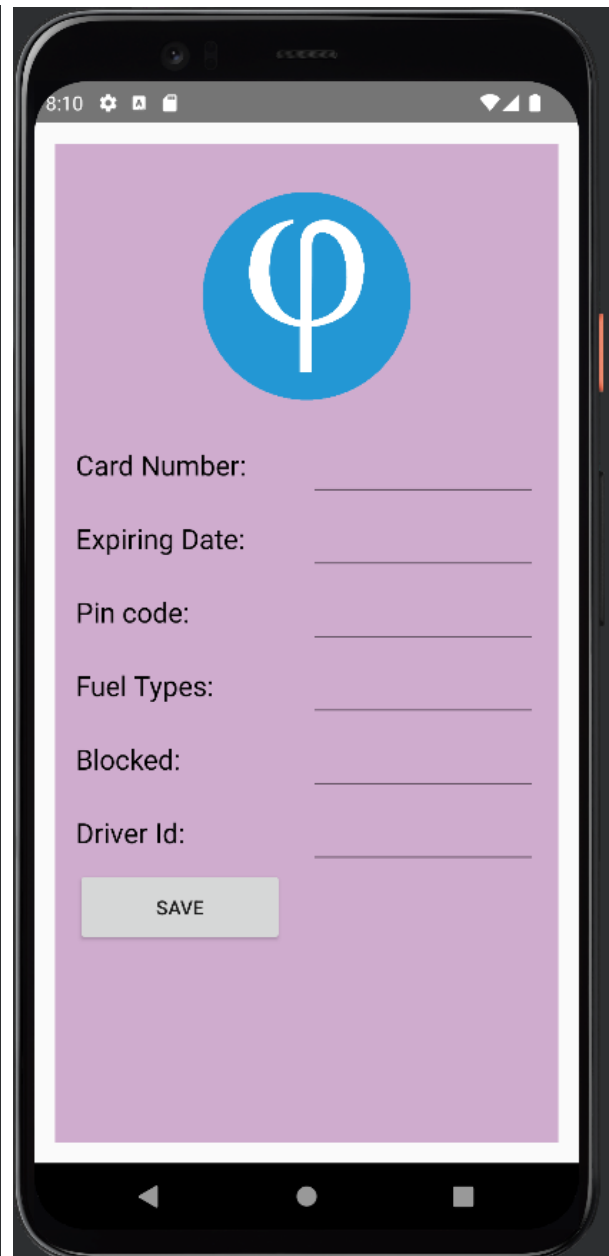
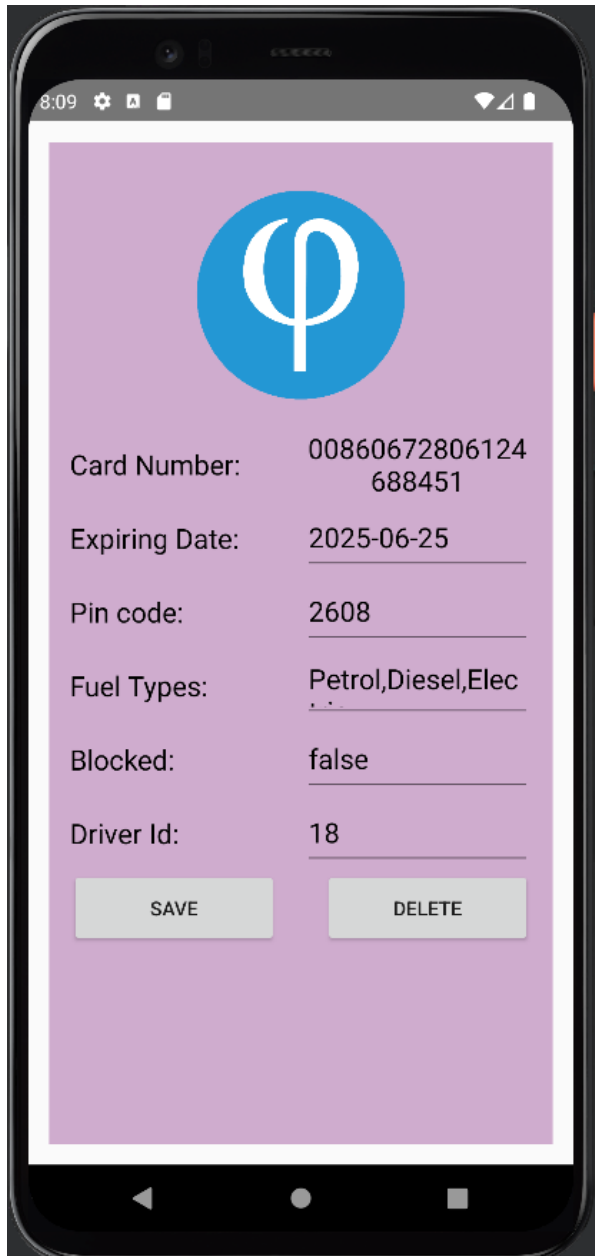
01550848657288216634

01614419286910013920

0227593 382741


Fuel cards

ADD



V2

19:21 14.0 KB/s 27%



First Name:

Last Name:

Birth Date:

RRN:


Licenses: ▼

Address:

Vehicle VIN:

GC Num:

00:25 8.00 KB/s 21%




1992
Tue, 18 Feb


< February 1992 >

M	T	W	T	F	S	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	

CANCEL OK

 DriverID: 25

00:24 0.78 KB/s 22%



Card Number: 0011223344556677889
9

Expiring Date: 2077-01-01

Pin code: 1234

Fuel Types: Petrol,Electric ▼


Blocked: ☒

Driver Id: 76

SAVE

DELETE

00:24 3.00 KB/s 22%



☐ Unknown

☒ Petrol

☐ Diesel

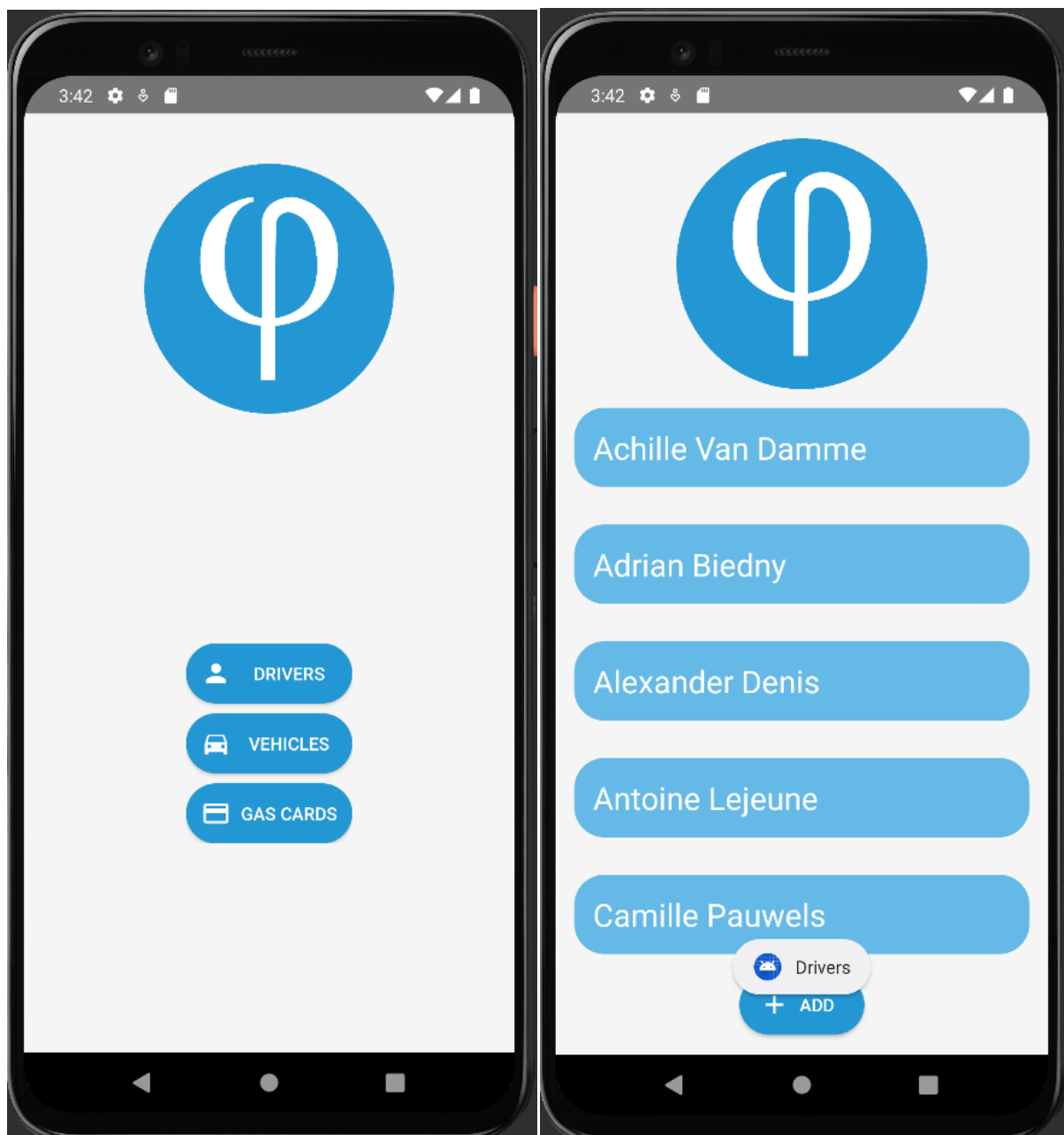
☒ Electric

☐ LPG

☐ PetrolHybrid


☐ DieselHybrid

CANCEL OK





3:43



VIN: 11223344556677815

Brand Model: TESLA

License Plate: ERR_404



Fuel Type: Petrol ▼

Vehicle Type: Car ▼


Color: PINK

Doors: 0

Driver Id: 78

 SAVE  DELETE

3:43




00112233445566778899


00860672806124688451

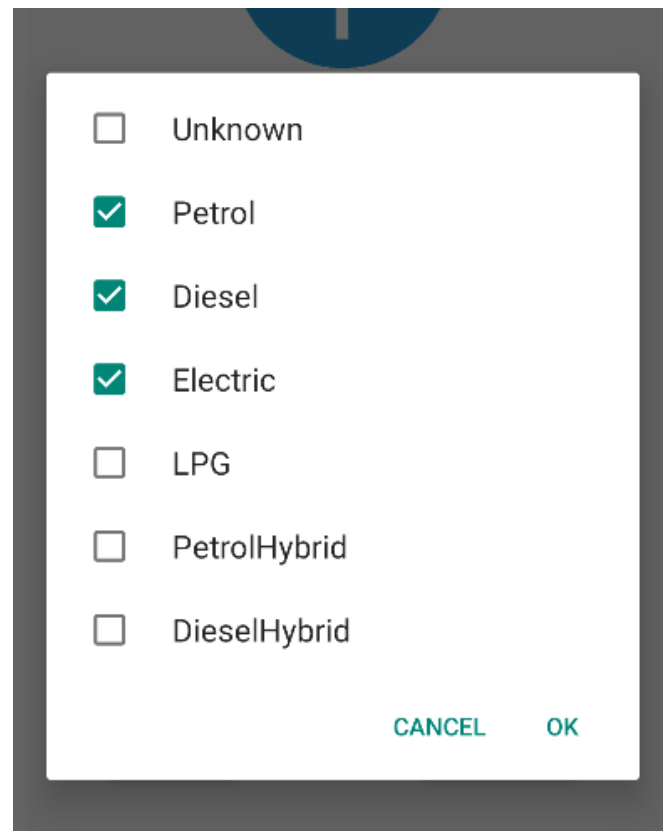
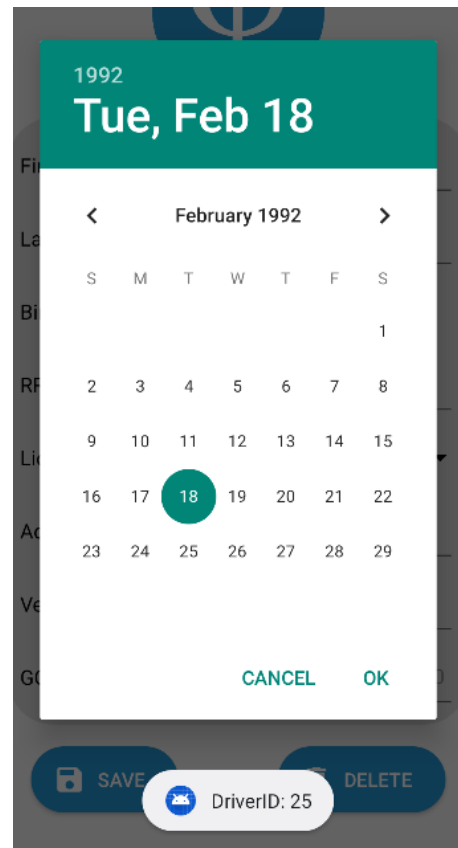
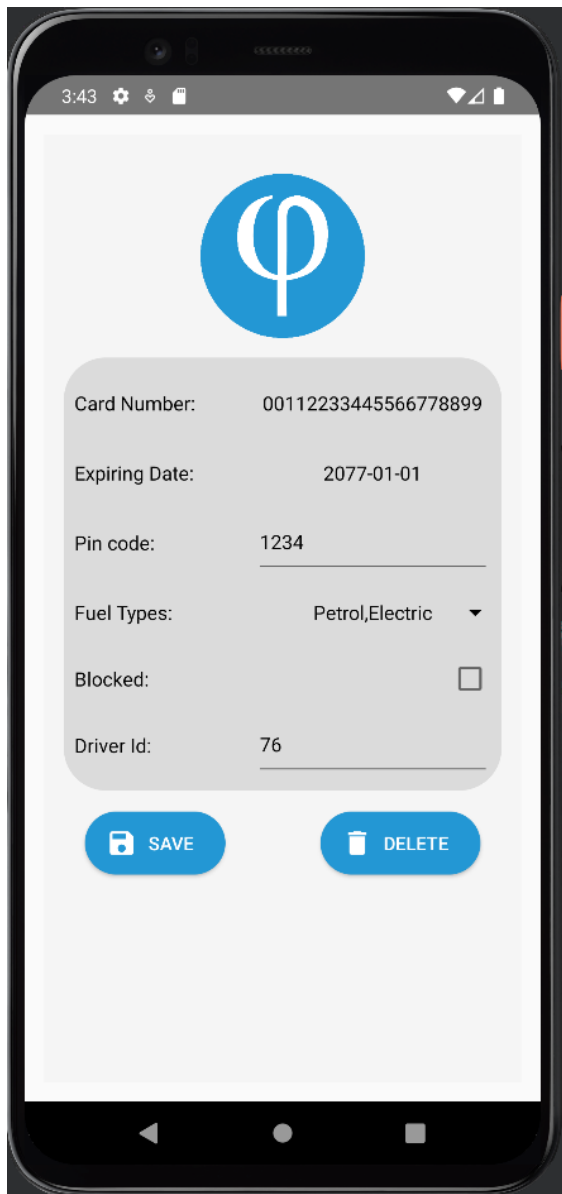
01445121330856533405

01550848657288216634

01614419286910013920

 Fuel cards

 ADD



Mijn ervaringen en wat ik geleerd heb.

Ik vond het geweldig om terug eens te programmeren in een JVM-taal (maar in dit geval in ART i.p.v. JVM). 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.

Nu dat KMMP ook in Beta is (full release binnenkort). Vind ik ook dat Kotlin voor Mobile development een grote(re) speler is. Door KMMP kan je shared code hebben en toch native code / componenten voor iOS en Android apart hebben die elk beter geschikt zullen zijn voor hun eigen omgevingen. Dus zou ik toch wel aanraden om dit als je

Links

Kotlin Documentation

<https://kotlinlang.org/docs/home.html>

Android Documentation

<https://developer.android.com/docs>

GitHub Source Code

<https://github.com/NotAffiche/StolosAPFMA/>

GitHub Source Code Group (API)

<https://github.com/NotAffiche/>

Test Project API

<https://jsonplaceholder.typicode.com/>