Kotlin Multiplatform

BY MUHAMMED **SAFI**UL AZAM
MOBILE APPLICATION DEVELOPER
EMAIL: MUHAMMED.SAFIUL.AZAM@GMAIL.COM

Today's goal

- iOS
- Android

- Optimize development.
- Share common libraries.
- Capitalize knowledge.

What makes a great mobile application?

- **Smooth interactive and intuitive experiences.**
- **Looks and feels like integrated part of the device.**
- **Utilize users' existing knowledge on devices' interfaces.**
- **Take full advantages of devices' native features.**
- **Of course: runs fast + less bugs.**

Don’t judge a book by it’s cover?

- **YES!! We do judge by cover. 😄**
- **We judge a mobile application by it’s UI/UX. It doesn’t matter how it functions inside or how awesome is our architecture!**

How can we make a great mobile application?

- **Provide native UX design.**
- **Use native UI tools and libraries.**
- **Business logics and etc. ⟶ Native or other technologies.**
- **Summery: Don't mess with UI/UX design. Use native tools and libraries for developing native experiences.**

What can we share among platforms?

- **Keep native UI/UX design.**
- **Share business logics and more without interrupting native UI/UX design.**

No	Yes
UI	Architectures
UX	Networks (HTTP)
Platform specific things	Databases (SQL)
	Data Models + Serializations
	Threads / Coroutines
	Events / Dispatchers
	…

What can we achieve by sharing among platforms?

- Less codes = less bugs.
- Reduce confusions on specifications (among platforms).
- Reduces development time.

Off the record 😄

- Feeling not alone when things go down.
- iOS developers finally can point to Android developers.

Welcome to
Kotlin Multiplatform

😄 😃 😍

What is Kotlin Multiplatform?

- Experimental feature in Kotlin 1.2 and 1.3 (until now).
- Complies code and generate libraries according to platforms.
- Allow us to access libraries like simply we access other libraries.
- Allow us to share business logic, connectivity and more.
- Android application ⟵ Multiplatform Libraries ⟶ iOS application.

  Under same project (optional)

Very very famous quote:
"Free libraries don't bring happiness but it helps."
-- Albert Einstein

Kotlin/Native

- Technology for compiling Kotlin code to native binaries.
- Supports two-way interoperability with native platforms.
  - Compiler create libraries and frameworks (swift / objective-c) for platforms.
  - Supports interoperability to use existing libraries and frameworks (swift / objective-c) directly from Kotlin/Native.

iOS frameworks in Kotlin/Native

- Kotlin/Native 1.3.41 - CFNetwork [ios_x64]
- Kotlin/Native 1.3.41 - ClassKit [ios_x64]
- Kotlin/Native 1.3.41 - CloudKit [ios_x64]
- Kotlin/Native 1.3.41 - CommonCrypto [ios_x64]
- Kotlin/Native 1.3.41 - Contacts [ios_x64]
- Kotlin/Native 1.3.41 - ContactsUI [ios_x64]
- Kotlin/Native 1.3.41 - CoreAudio [ios_x64]
- Kotlin/Native 1.3.41 - CoreAudioKit [ios_x64]
- Kotlin/Native 1.3.41 - CoreBluetooth [ios_x64]
- Kotlin/Native 1.3.41 - CoreData [ios_x64]
- Kotlin/Native 1.3.41 - CoreFoundation [ios_x64]
- Kotlin/Native 1.3.41 - CoreGraphics [ios_x64]
- Kotlin/Native 1.3.41 - CoreImage [ios_x64]
- Kotlin/Native 1.3.41 - CoreLocation [ios_x64]
- Kotlin/Native 1.3.41 - CoreMedia [ios_x64]
- Kotlin/Native 1.3.41 - CoreMIDI [ios_x64]
- Kotlin/Native 1.3.41 - CoreML [ios_x64]
- Kotlin/Native 1.3.41 - CoreMotion [ios_x64]
- Kotlin/Native 1.3.41 - CoreNFC [ios_x64]
- Kotlin/Native 1.3.41 - CoreServices [ios_x64]

... many more.

Common libraries?

- **Most of the mobile applications use some essential libraries to function.**
- **Community actors already provide most of those essential libraries which are useable across platforms.**

Kotlin	Vendor	Essential
Kotlinx.Coroutines	JetBrains	Coroutines
SQLDelight	Square	Database
Kotlinx.Serialization	JetBrains	Serialization
Ktor.io	JetBrains	HTTP / Server / Client
…	…	…

Mechanism: expect and actual

- **Common codes to depend on platform-specific declarations.**
- **Common module can define *expected* declarations.**
- **Platform module can provide *actual* declarations corresponding to the expected ones.**

expect (Common)

```
expect object CouroutineUtils {
    // Dispatcher.
    val DISPATCHER: CoroutineDispatcher
}
```

actual (Android)

```
actual object CouroutineUtils {
    // Dispatcher.
    actual val DISPATCHER: CoroutineDispatcher = Dispatchers.Main
}
```

actual (iOS)

```
@ThreadLocal
actual object CouroutineUtils {
    // Dispatcher.
    @SharedImmutable
    actual val DISPATCHER: CoroutineDispatcher = NsQueueDispatcher(
        dispatch_get_main_queue()
    )

    internal class NsQueueDispatcher(private val dispatchQueue: dispatch_queue_t) : CoroutineDispatcher() {
        override fun dispatch(context: CoroutineContext, block: Runnable) {
            dispatch_async(dispatchQueue) {
                block.run()
            }
        }
    }
}
```

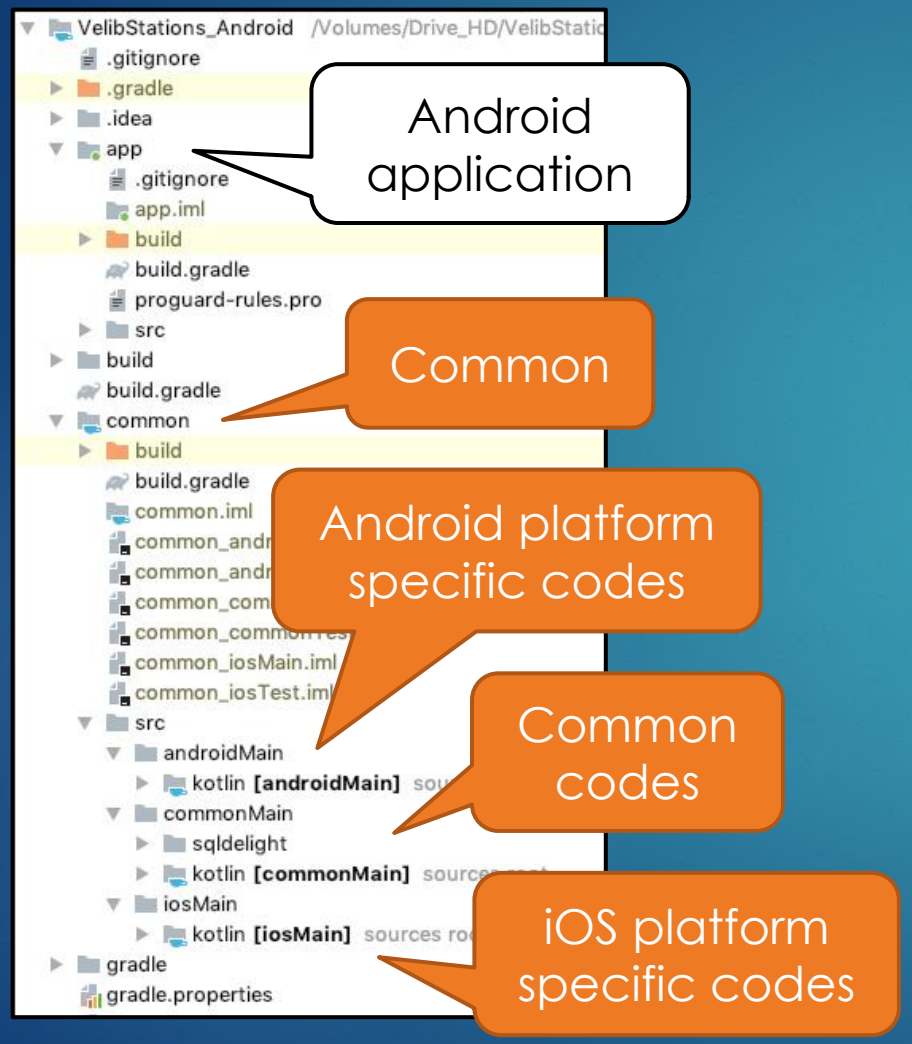
It's written in Kotlin ;)

Platform independent codes?


```
class DatabaseManager : AddOn(), IDatabaseManager {

    private val mVelibDB: VelibDB by lazy {
        VelibDB(DatabaseUtils.VELIB_DB_DRIVER!!)
    }

    private val mVelibDatabase: IVelibDatabase by lazy {
        val velibDatabase = VelibDatabase(mVelibDB)
        velibDatabase.addAddOns(getAddOns())
        velibDatabase
    }

    override fun getVelibDatabase(): IVelibDatabase {
        return mVelibDatabase
    }
}
```

No platform-specific code = no expect / actual
Simply write common codes and use! 😃

Project hierarchy



Gradle (Common)

```
sourceSets["commonMain"].dependencies {
    implementation ("org.jetbrains.kotlin:kotlin-stdlib-common:$kotlin_stdlib_version")
    implementation ("org.jetbrains.kotlinx:kotlinx-coroutines-core-common:$kotlinx_coroutines_version")

    implementation ("io.ktor:ktor-client:$ktor_version")
    implementation ("io.ktor:ktor-client-json:$ktor_version")
    implementation ("io.ktor:ktor-client-serialization:$ktor_version")

    implementation ("org.jetbrains.kotlinx:kotlinx-serialization-runtime:$kotlin_serialization_version")
}

sourceSets["androidMain"].dependencies {
    implementation ("org.jetbrains.kotlin:kotlin-stdlib:$kotlin_stdlib_version")
    implementation ("org.jetbrains.kotlinx:kotlinx-coroutines-android:$kotlinx_coroutines_version")

    implementation("io.ktor:ktor-client-android:$ktor_version")
    implementation ("io.ktor:ktor-client-json-jvm:$ktor_version")
    implementation ("io.ktor:ktor-client-serialization-jvm:$ktor_version")

    implementation ("com.squareup.sqldelight:android-driver:$sqldelight_version")
}

sourceSets["iosMain"].dependencies {
    implementation ("org.jetbrains.kotlin:kotlin-stdlib:$kotlin_stdlib_version")
    implementation ("org.jetbrains.kotlinx:kotlinx-coroutines-core-native:$kotlinx_coroutines_version")

    implementation ("io.ktor:ktor-client-ios:$ktor_version")
    implementation ("io.ktor:ktor-client-json-native:$ktor_version")
    implementation ("io.ktor:ktor-client-serialization-native:$ktor_version")

    implementation ("org.jetbrains.kotlinx:kotlinx-serialization-runtime-native:$kotlin_serialization_version")

    implementation ("com.squareup.sqldelight:ios-driver:$sqldelight_version")
}
```

What to put where?

- **Declaring class, object, etc. with expect require their actual definition for platforms.**
- **Class, object, etc. with expect contain only signatures.**
- **Class, object, etc. with actual can contain extra methods.**

Class / Object / etc.	Package / Common
Platform independent	commonMain
expect declaration	commonMain
actual (Android) definition	androidMain
actual (iOS) definition	iosMain

iOS framework (libraries)


```
task packForXCode(type: Sync) {
    final File frameworkDir = new File(buildDir, "xcode-frameworks")
    final String mode = project.findProperty("XCODE_CONFIGURATION")?.toUpperCase() ?: 'DEBUG'
    final def framework = kotlin.targets.ios.binaries.getFramework("CommonKit", mode)

    inputs.property "mode", mode
    dependsOn framework.linkTask

    from { framework.outputFile.parentFile }
    into frameworkDir

    doLast {
        new File(frameworkDir, 'gradlew').with {
            text = "#!/bin/bash\nexport 'JAVA_HOME=${System.getProperty("java.home")}'\ncd '${rootProject.rootDir}'\n./gradlew \$@\n"
            setExecutable(true)
        }
    }
}
tasks.build.dependsOn packForXCode
```

CommonKit

Gradle
(Common)

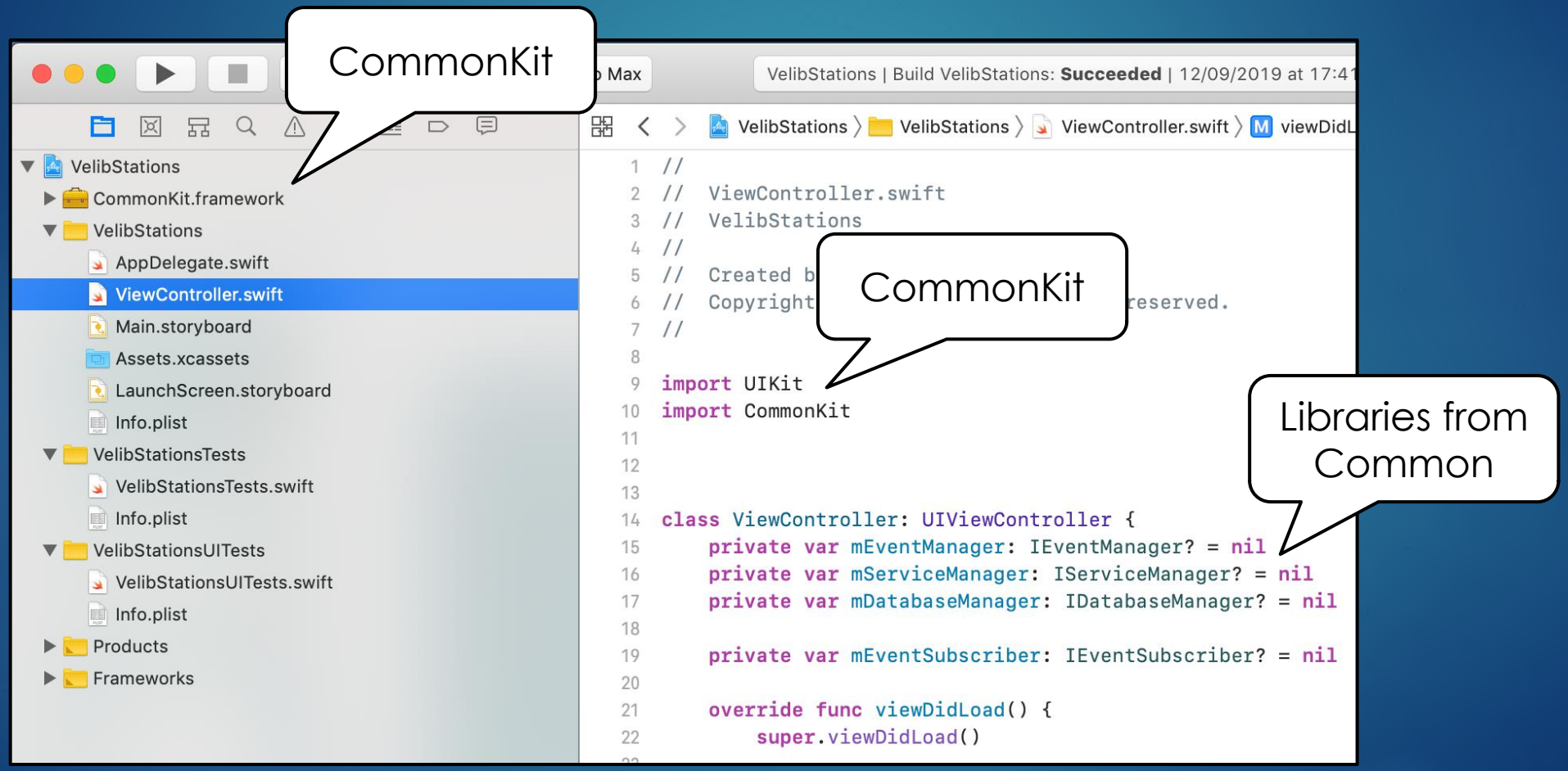
Frameworks, Libraries, and Embedded Content

Name	Embed
CommonKit.framework	Embed & Sign

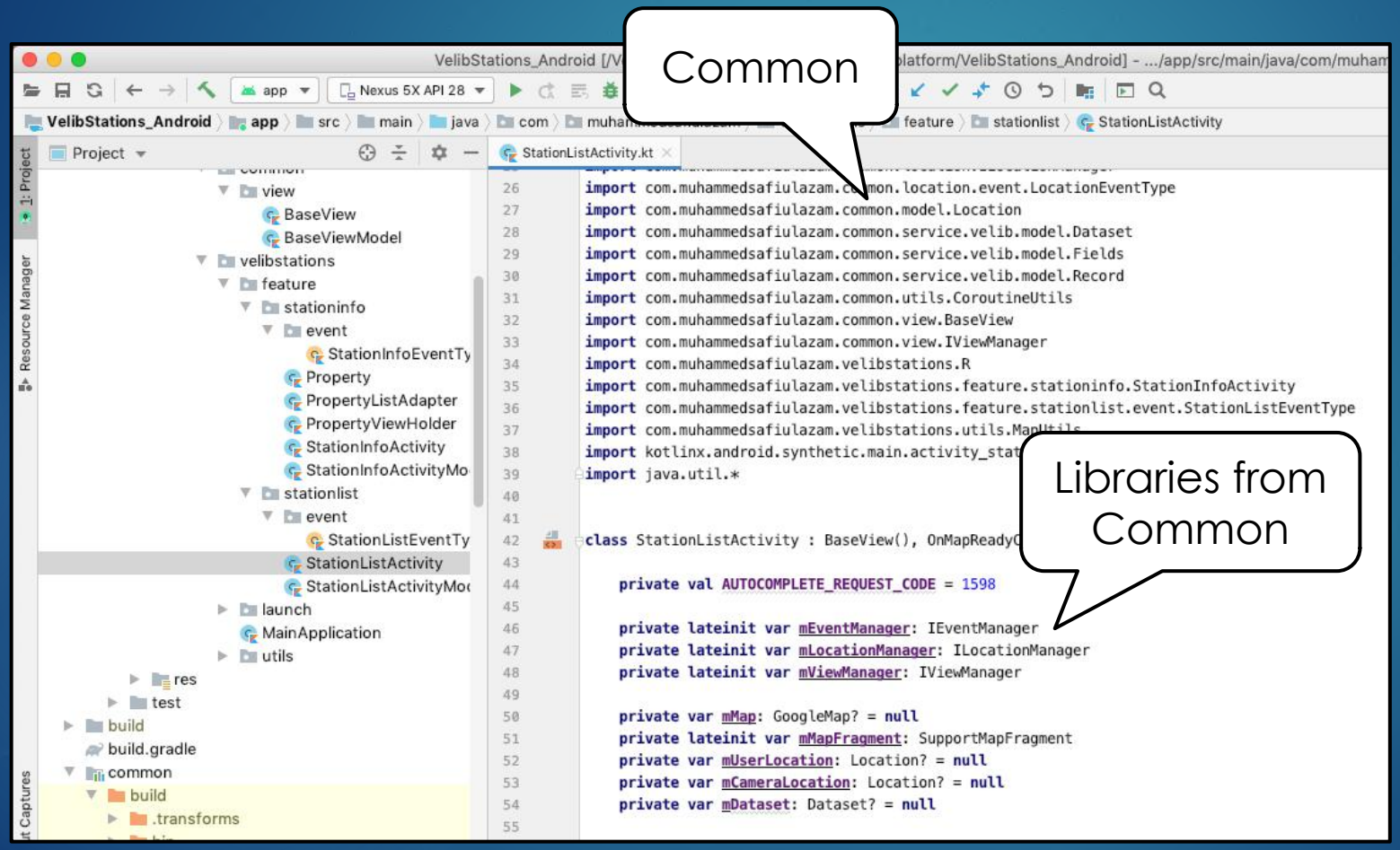
CommonKit

Project Settings
(iOS)

Common libraries in Swift



Common libraries in Kotlin



Just import libraries with package names.

Namespace in Swift?

- **Namespaces are implicit in Swift. Classes, etc are implicitly scoped by modules / frameworks.**
- **Kotlin uses underscore "_" to solve classes with same names which seemingly random.**
- **Solution: Don't create classes with same names or use typealias.**

Kotlin
com.x.y.z. Animal
com.a.b.c. Animal
com.a.b.c. Giraffe

After Compilation
→

Swift
CommonKit. Animal
CommonKit. Animal_
CommonKit. Giraffe

Me you… no… you me…

Calling VelibService from Android (Kotlin) application

```
val serviceManager: IServiceManager = AddOnManager.getAddOn(AddOnType.SERVICE_MANAGER) as IServiceManager
serviceManager.getVelibService().getData( latitude: 48.85341, longitude: 2.3488,  radius: 1000.0,  index: 0,  count: 100)
```

Calling VelibService from iOS (Swift) application

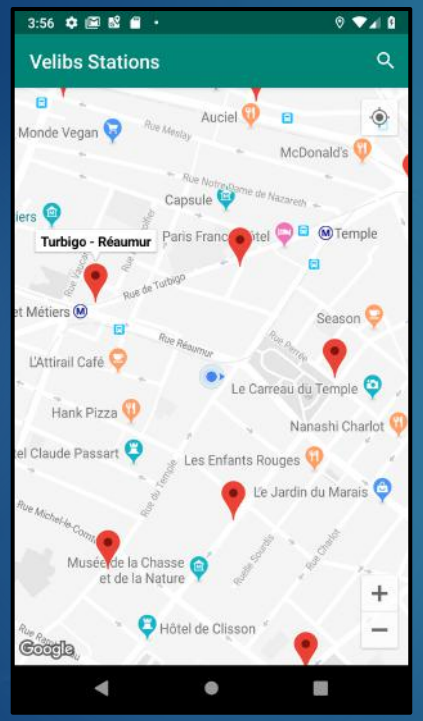
```
let serviceManager: IServiceManager = AddOnManager().getAddOn(type: AddOnType().SERVICE_MANAGER) as! IServiceManager
serviceManager.getVelibService().getData(latitude: 48.864716, longitude: 2.349014, radius: 1000.0, index: 0, count: 100)
```

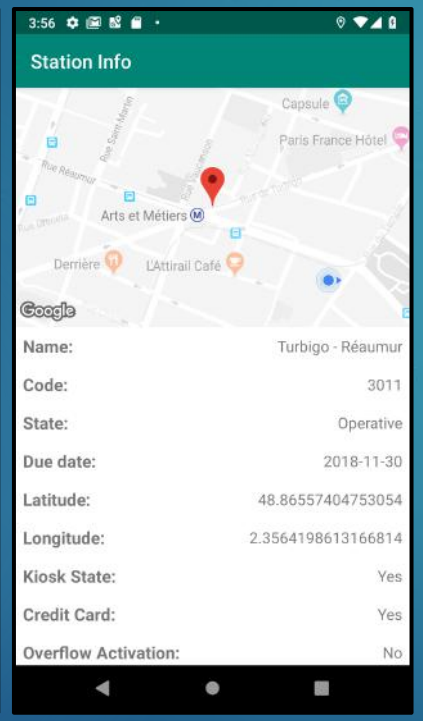
Example 🤓

- **Velib Stations:** https://github.com/muhammedsafiulazam/velibstations
- **Common libraries:**
  - **Domains: Event, AddOn (Architecture), Service, Database, Location, Coroutine, Model, Serializer, etc.**
- **Platform specific libraries:**
  - **Domains - Android: View (Activity), ViewModel, etc.**
  - **Domains - iOS: View (UIViewController), ViewModel, etc.**

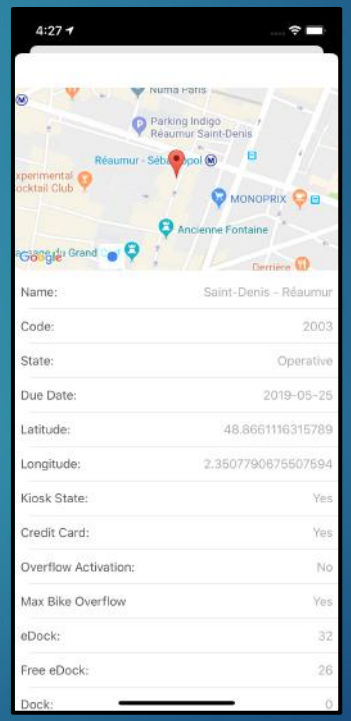
Demo / Velib Stations

Android Application





iOS Application



Findings?

- Kotlin's objects in Swift ⟶ Use @ThreadLocal and @SharedImmutable during declarations. https://kotlinlang.org/docs/reference/native/immutability.html
- If Kotlin's objects are derived from classes with mutable properties, Swift doesn't allow access to those mutable properties (runtime error).
- iOS ⟶ KotlinxSerializer require mappers for data models and serializers manually (once and easy).
- Android ⟶ kotlinx.serialization.Serializable is not derived from java.io.Serializable. So it's not possible to put in intent directly.

Oh dayum

- **Mixing Kotlin and Objective-C supertypes is not supported yet.** https://kotlinlang.org/docs/reference/native/objc_interop.html

```
class BaseView : UIViewController, IBaseView {
```

👎

```
class BaseView : UIViewController {
    @OverrideInit
    constructor(coder: NSCoder) : super (coder)
}
```

👍

IBaseView ⟶ Interface written in Kotlin.
UIViewController ⟶ Class written in Objective-C.

Note: These are my observations until now. I'm trying to find out better solutions of these issues. 😄

Kotlin singletons (objects)

Kotlin singletons

Kotlin singleton (made with an `object` declaration, including `companion object`) is imported to Swift/Objective-C as a class with a single instance. The instance is available through the factory method, i.e. as `[MySingleton mySingleton]` in Objective-C and `MySingleton()` in Swift.

References

- https://github.com/muhammedsafiulazam/velibstations
- https://kotlinlang.org/docs/reference/multiplatform.html
- https://www.raywenderlich.com/1022411-kotlin-multiplatform-project-for-android-and-ios-getting-started
- https://github.com/Kotlin/kotlinx.coroutines
- https://github.com/Kotlin/kotlinx.serialization
- https://github.com/cashapp/sqldelight
- https://github.com/ktorio/ktor

What's next?

Kotlin Multiplatform

Hands-on

(Hands-on explanations on **Velib Stations'** codes)

Questions?

Wake **UP**!?!

🥺 😮 😄