

LIVESPORT



Martin Štrambach

iOS Developer at Livesport
martin.strambach@livesport.eu

April 2022
Cocoaheads Meetup

1. Tech Stack
2. Issues and Solutions



TECH STACK UNDER THE HOOD

LIVESPORT

- ▶ Swift vs Kotlin
- ▶ SwiftUI vs Jetpack Compose
- ▶ Concurrency vs Coroutines
- ▶ Combine vs Flow
- ▶ Kotlin Multiplatform Mobile (KMM)



WHO'S DEPLOYED KMM INTO PRODUCTION?

WHO'S DEPLOYED KMM INTO PRODUCTION?

vmware®

PHILIPS



down dog

Quizlet

Yandex

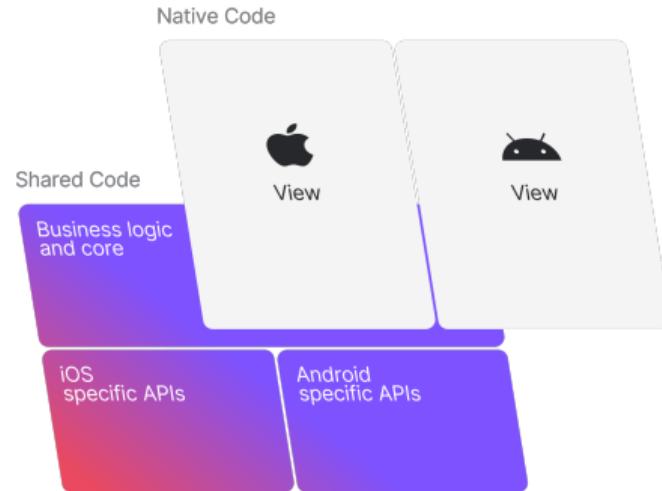
AUTODESK

NETFLIX

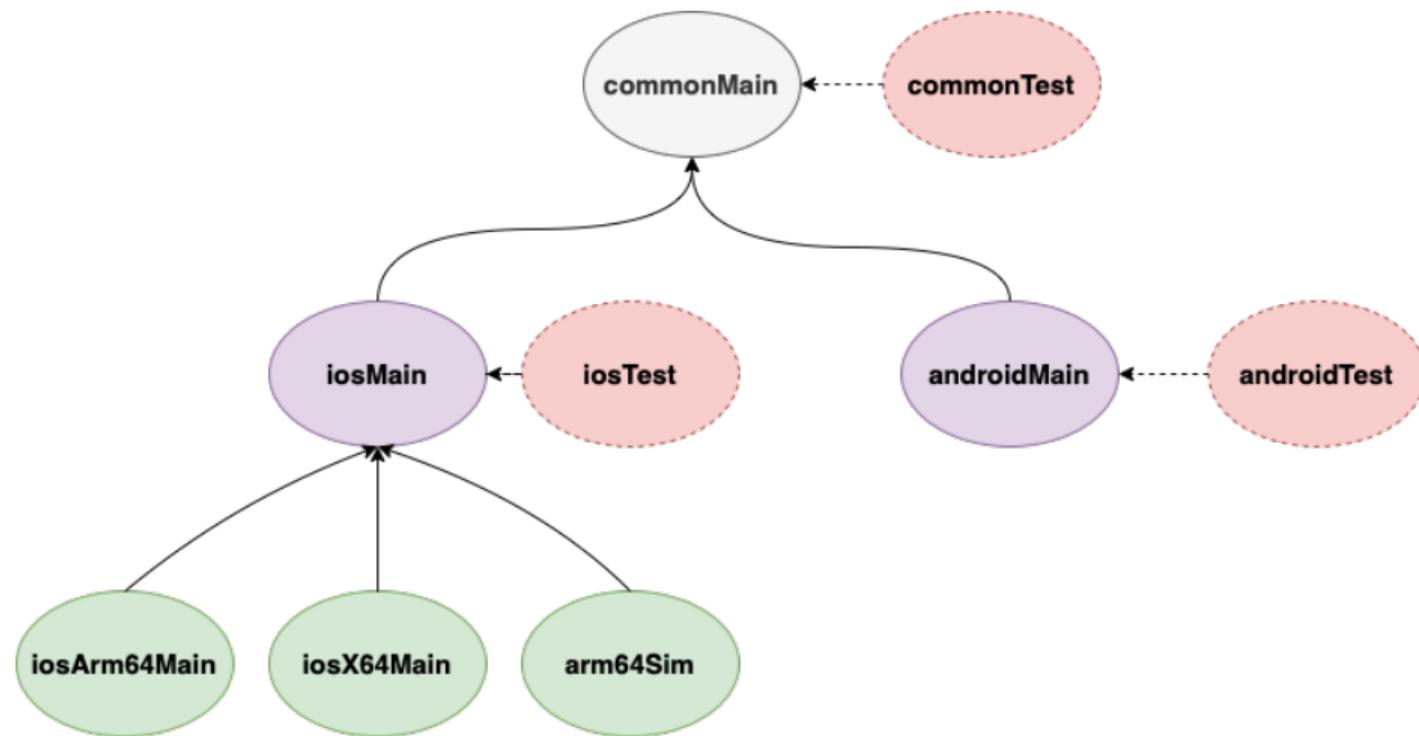
KMM QUICK OVERVIEW 1/2

LIVESPORT

- ▶ SDK for Android and iOS.
- ▶ Single codebase.
- ▶ Ideal for business logic.
- ▶ Lots of multiplatform libraries.
- ▶ Translated into ObjC.



KMM QUICK OVERVIEW 2/2



3 FUCKUPS == 3 LESSONS

LIVESPORT

3 FUCKUPS == 3 LESSONS

LIVESPORT

1. Flow vs Combine interoperability

3 FUCKUPS == 3 LESSONS

LIVESPORT

1. Flow vs Combine interoperability
2. Missing types in native code

3 FUCKUPS == 3 LESSONS

LIVESPORT

1. Flow vs Combine interoperability
2. Missing types in native code
3. Suspend functions in Swift code

FLOW VS COMBINE INTEROPERABILITY

COMBINE PUBLISHERS

LIVESPORT

```
public protocol Publisher {  
    /// The kind of values published by this publisher.  
    associatedtype Output  
    /// The kind of errors this publisher might publish.  
    associatedtype Failure : Error  
    /// Attaches the specified subscriber to this publisher.  
    /// Implementations of "Publisher" must implement this method.  
    /// - Parameter subscriber: The subscriber to attach to this  
    /// "Publisher", after which it can receive values.  
    func receive<S>(subscriber: S) where S : Subscriber,  
        Self.Failure == S.Failure,  
        Self.Output == S.Input  
}
```

ANDROID IMPLEMENTATION

LIVESPORT

```
class KotlinNativeFlowWrapper<T: Any>(private val flow: Flow<T>) {  
    fun subscribe(  
        scope: CoroutineScope,  
        onEach: (item: T) -> Unit,  
        onComplete: () -> Unit,  
        onThrow: (error: Throwable) -> Unit,  
    ) = flow  
        .onEach { onEach(it) }  
        .catch { onThrow(it) }  
        .onCompletion { onComplete() }  
        .launchIn(scope)  
}
```

IOS RECEIVE FUNCTION IMPLEMENTATION

LIVESPORT

```
wrapFlow(flow: flow).subscribe(  
    scope: scope,  
    onEach: { result in  
        _ = subscriber.receive(result as! Output)  
    },  
    onComplete: { subscriber.receive(completion: .finished) },  
    onThrow: { error in  
        subscriber.receive(completion:  
            .failure(error.asError() as! Failure)  
    )  
}  
)
```

GITHUB REPOSITORY FOR YOU!

LIVESPORT



MISSING TYPES IN NATIVE CODE

COMPILED GENERIC KOTLIN CODE

LIVESPORT

A return value using a generic type.

```
typealias Description = Pair<String, Map<Int, String>>
```

COMPILED GENERIC KOTLIN CODE

A return value using a generic type.

```
typealias Description = Pair<String, Map<Int, String>>
```

A generic type definition.

```
public data class Pair<out A, out B>(
    public val first: A,
    public val second: B
)
```

COMPILED GENERIC KOTLIN CODE

A return value using a generic type.

```
typealias Description = Pair<String, Map<Int, String>>
```

A generic type definition.

```
public data class Pair<out A, out B>(
    public val first: A,
    public val second: B
)
```

A generated ObjC code.

```
(SharedKotlinPair<NSString*, NSDictionary<SharedInt*, NSString*>*>*)
    functionWithGenericReturnValue
```

COMPILED GENERIC KOTLIN CODE

A return value using a generic type.

```
typealias Description = Pair<String, Map<Int, String>>
```

A generic type definition.

```
public data class Pair<out A, out B>(
    public val first: A,
    public val second: B
)
```

A generated ObjC code.

```
(SharedKotlinPair<NSString*, NSDictionary<SharedInt*, NSString*>*>*)
    functionWithGenericReturnValue
```

Return value in Swift.

```
KotlinPair<NSString, NSDictionary>
```

WHAT TO DO?

LIVESPORT

Solution #1: Completely avoid using nested generic types.

```
data class DescriptionData(  
    val value1: String,  
    val value2: Map<Int, String>  
)
```

WHAT TO DO?

LIVESPORT

Solution #1: Completely avoid using nested generic types.

```
data class DescriptionData(  
    val value1: String,  
    val value2: Map<Int, String>  
)
```

A generated ObjC code.

```
@interface SharedDescriptionData : SharedBase  
NSString *value1;  
NSDictionary<SharedInt *, NSString *> *value2;  
@end;
```

WHAT TO DO?

LIVESPORT

Solution #1: Completely avoid using nested generic types.

```
data class DescriptionData(  
    val value1: String,  
    val value2: Map<Int, String>  
)
```

A generated ObjC code.

```
@interface SharedDescriptionData : SharedBase  
NSString *value1;  
NSDictionary<SharedInt *, NSString *> *value2;  
@end;
```

A return value in Swift.

```
DescriptionData(value1: String, value2: [KotlinInt : String])
```

SUSPEND FUNCTIONS IN SWIFT CODE

KOTLIN'S SUSPEND FUNCTIONS

LIVESPORT

A suspend function in Kotlin.

```
class SuspendFunctionManager {  
    suspend fun insert(value: Int) {  
        mutex.withLock {  
            values.add(value)  
        }  
    }  
}
```

KOTLIN'S SUSPEND FUNCTIONS

LIVESPORT

A suspend function in Kotlin.

```
class SuspendFunctionManager {  
    suspend fun insert(value: Int) {  
        mutex.withLock {  
            values.add(value)  
        }  
    }  
}
```

A generated code in Swift.

```
let manager = SuspendFunctionManager()  
manager.insert(  
    value: Int32  
    completionHandler: (KotlinUnit?, Error?) -> Void  
)
```

WHAT TO DO?

LIVESPORT

Solution #1: Wrap code in iOS Main part.

```
class SuspendFunctionWrapper(val scope: CoroutineScope) {  
    val suspendFunctionManager = SuspendFunctionManager()  
  
    fun insert(value: Int) {  
        scope.launch { suspendFunctionManager.insert(value) }  
    }  
}
```

WHAT TO DO?

LIVESPORT

Solution #1: Wrap code in iOS Main part.

```
class SuspendFunctionWrapper(val scope: CoroutineScope) {  
    val suspendFunctionManager = SuspendFunctionManager()  
  
    fun insert(value: Int) {  
        scope.launch { suspendFunctionManager.insert(value) }  
    }  
}
```

- ▶ Solution #2: Using new Concurrency framework

WRAP UP!

LIVESPORT

ONE MORE THING...

LIVESPORT

ONE MORE THING... KMM DEBUGGING

LIVESPORT

- ▶ Xcode Plug-In.
- ▶ Good performance with Kotlin 1.6.0.
- ▶ Stepping in code with breakpoints.
- ▶ Variables inspection and stack traces.

ONE MORE THING... KMM DEBUGGING

LIVESPORT

- ▶ Xcode Plug-In.
- ▶ Good performance with Kotlin 1.6.0.
- ▶ Stepping in code with breakpoints.
- ▶ Variables inspection and stack traces.

The screenshot shows a debugger interface with a code editor and a variable pane. The code editor displays a Kotlin function named `create` with several local variables and their values:

```
27
28     fun create(baseModel: DetailBaseModel, commonModel: DuelDetailCommonModel): EventStageViewState {
29         val eventStage = EventStage.getById(commonModel.eventStageId)
30         val stageName = if (eventStage != null) stageName(baseModel.sportId, eventStage) ?: "" else ""
31
32         val model = StageInfoDataModel(baseModel, commonModel, stageName, resources, currentTime)
33         val stageInfoText = stageInfoText(baseModel.sportId, model)
34         val color = if (commonModel.isLive()) resources.color.fsRed else resources.color.text_color_primary
35
36         return EventStageViewState(stageInfoText, color)
37     }
38
39
```

The variable pane shows the state of `_this` and `baseModel`. The `_this` object contains properties like `currentTime`, `stageName`, `stageInfoText`, and `resources$delegate`. The `baseModel` object contains properties like `eventId`, `league`, `settings`, `eventParticipants`, `homeEventParticipant`, `awayEventParticipant`, and `sportId`.

WRAP UP!

LIVESPORT

- ▶ Xcode
- ▶ Android Studio

WRAP UP!

LIVESPORT

- ▶ Xcode
- ▶ Android Studio
- ▶ Official docs

WRAP UP!

LIVESPORT

- ▶ Xcode
- ▶ Android Studio
- ▶ Official docs
- ▶ John O'Reilly's blog

WRAP UP!

LIVESPORT

- ▶ Xcode
- ▶ Android Studio
- ▶ Official docs
- ▶ John O'Reilly's blog



CONTACTS

Martin Štrambach
martin.strambach@livesport.eu

April 2022
Cocoaheads Meetup