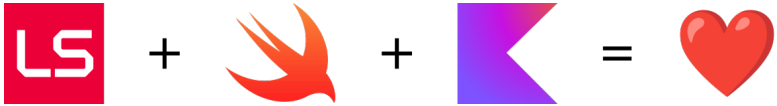
LIVESPORT



Martin Štrambach

iOS Developer at Livesport
martin.strambach@livesport.eu

April 2022
Cocoaheads Meetup

LIVESPORT

OUTLINE

1. Tech Stack
2. Issues and Solutions

TECH STACK UNDER THE HOOD

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

WHO'S DEPLOYED KMM INTO PRODUCTION?

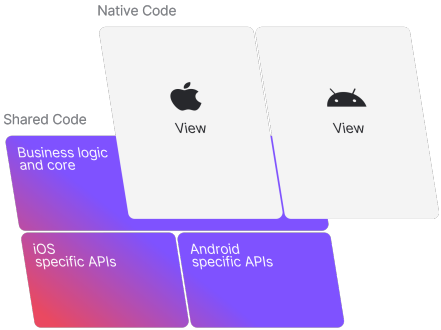
LIVESPORT

WHO'S DEPLOYED KMM INTO PRODUCTION?

LIVESPORT

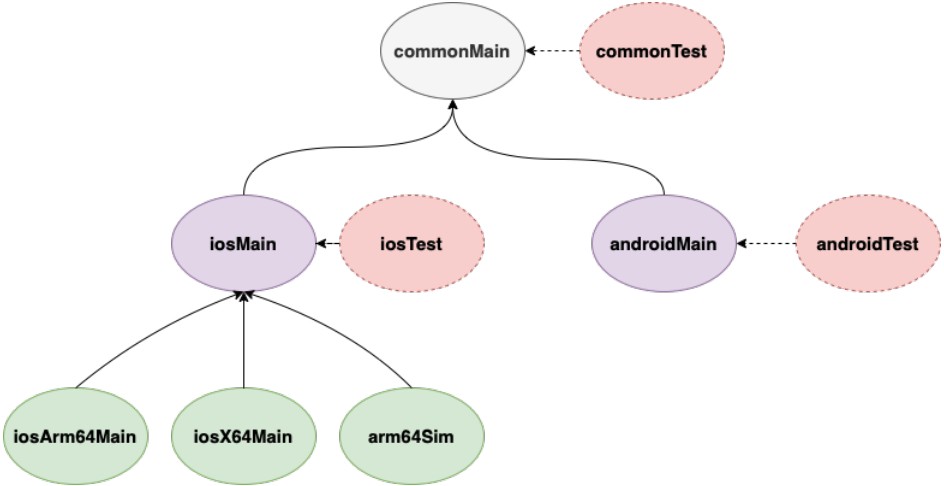
KMM QUICK OVERVIEW 1/2

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



KMM QUICK OVERVIEW 2/2

LIVESPORT



3 FUCKUPS

LIVESPORT

3 FUCKUPS == 3 LESSONS

LIVESPORT

3 FUCKUPS == 3 LESSONS

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

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

FUCKUP #1

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


```
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

FUCKUP #2

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

LIVESPORT

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?

Solution #1: Completely avoid using nested generic types.

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

WHAT TO DO?

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?

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])
```

FUCKUP #3

LIVESPORT

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

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?

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!

ONE MORE THING...

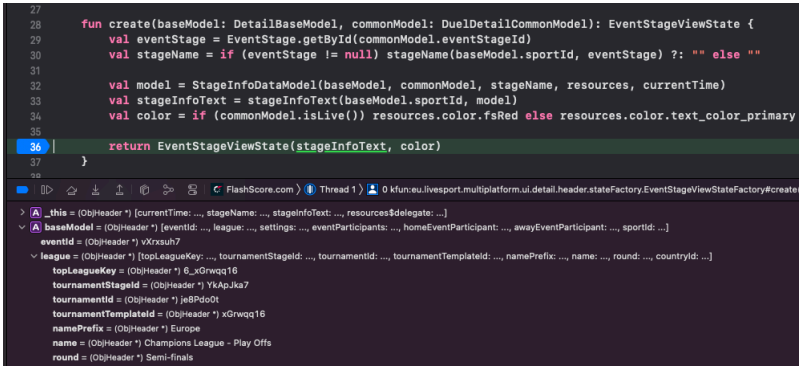
ONE MORE THING... KMM DEBUGGING

- 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.



WRAP UP!

- Xcode
- Android Studio

WRAP UP!

- Xcode
- Android Studio
- Official docs

WRAP UP!

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

WRAP UP!

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

CONTACTS

Martin Štrambach
martin.strambach@livesport.eu

April 2022
Cocoaheads Meetup