HTEC GROUP

# Kotlin Multiplatform Mobile - An overview

**Nikola Mihalek**

October 2022

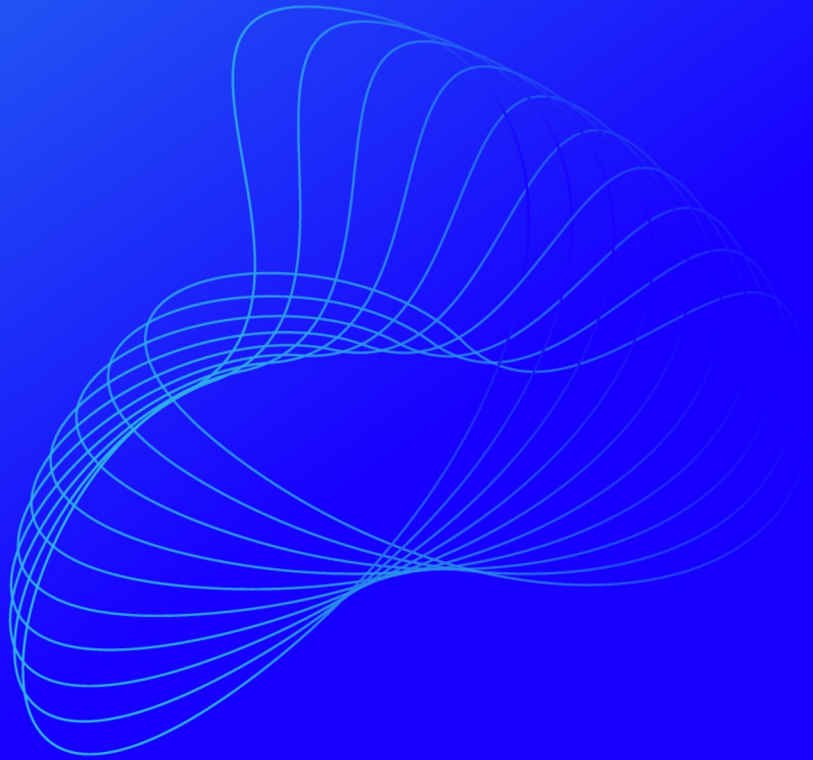
- **Intro**
- Kotlin Multi..what?
- Architecture
- Code walkthrough
- Expect/Actual
- Summary

- KMM = best framework ever?
- What about Google?
- What's KMM?
- What's it for?
- Structuring code
- Different / same
- Bitter XP, pros and cons

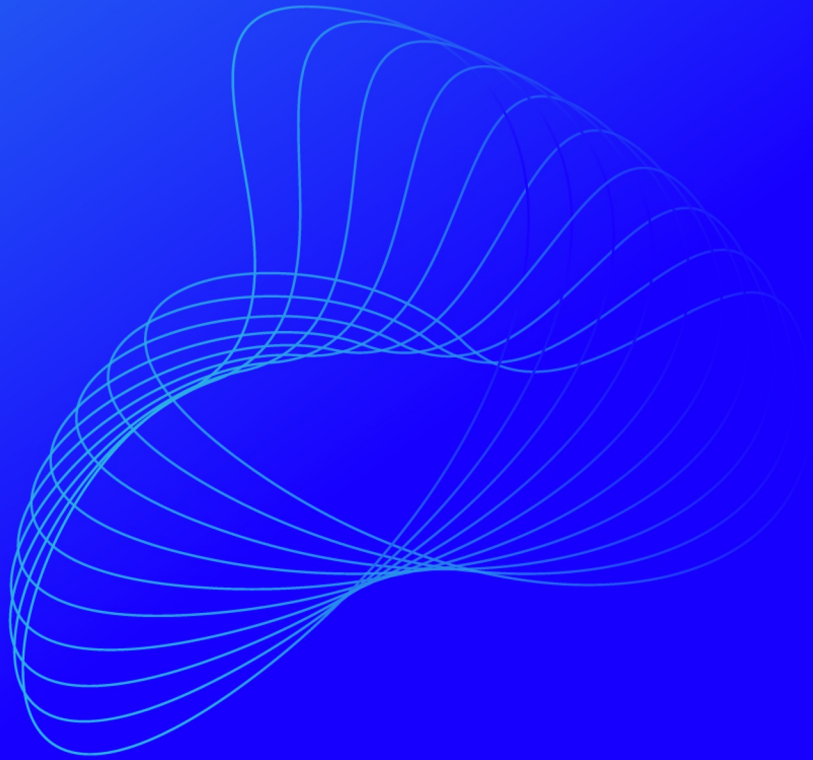
HTEC GROUP

# But before we do…

# DEMO TIME 🙏🙏🙏



# wow, I know!



# Kotlin multi..what?

- Was a fake wow.. but something else is wow
- App logic is the same for both
- Not just the same, identical
- Pure Kotlin
- Platform agnostic – perfect for the domain layer

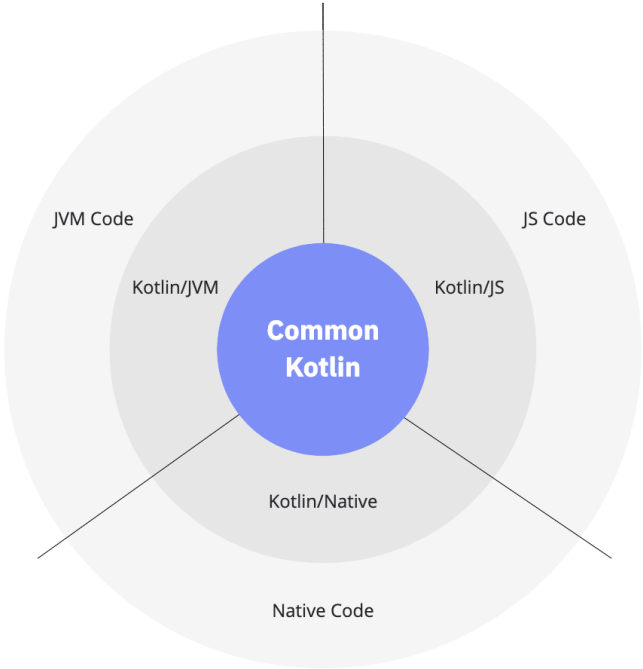
HTEC GROUP

# Kotlin multi..what?

- Kotlin Multiplatform Mobile
- Multiplatform framework
- Big shared codebase

HTEC GROUP

# Kotlin multi..what?



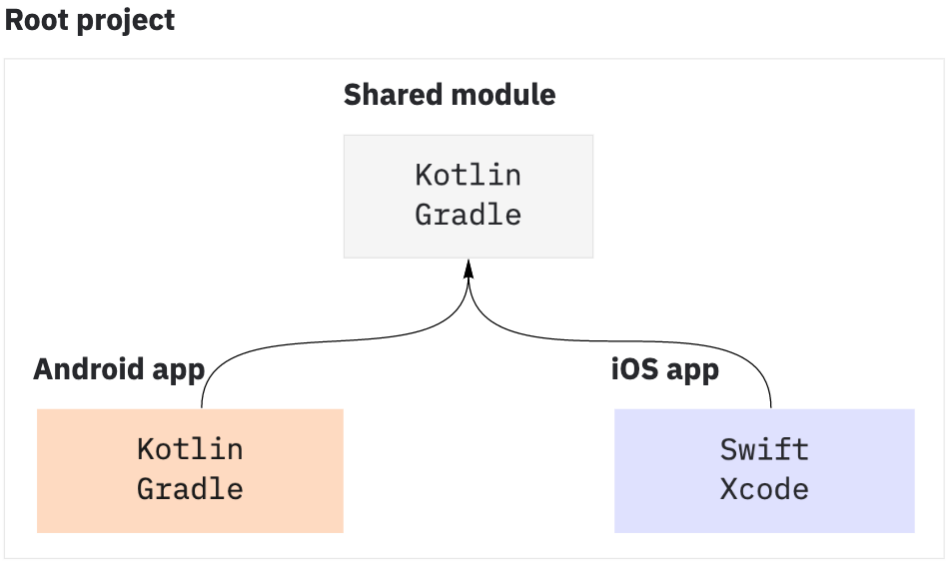
- Intro
- DEMO
- **Kotlin Multi..what?**
- Architecture
- Code walkthrough
- Expect/Actual
- Summary

- It’s all KMM’s fault!
- One codebase to rule them all
- Written in Kotlin

HTEC GROUP

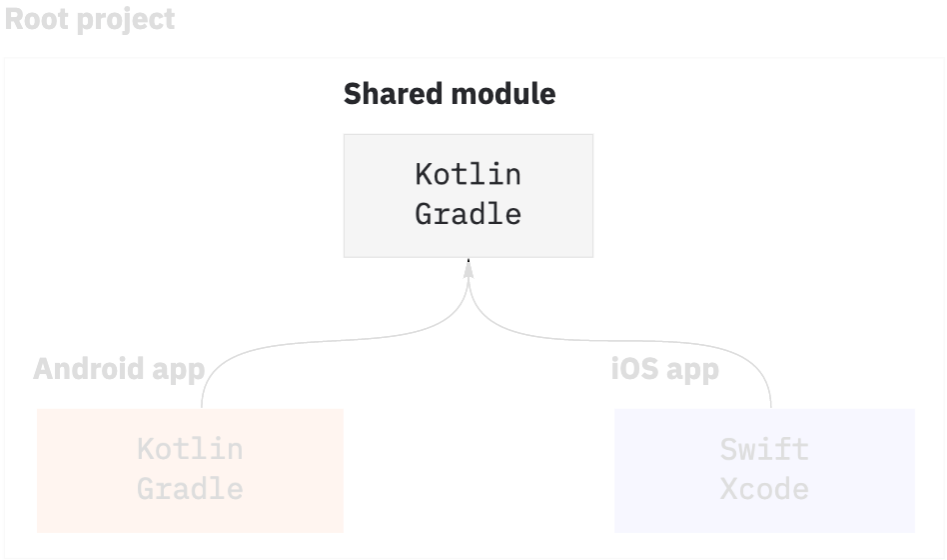
# Architecture

- For mobile, split into 3 parts
- Shared module most interesting



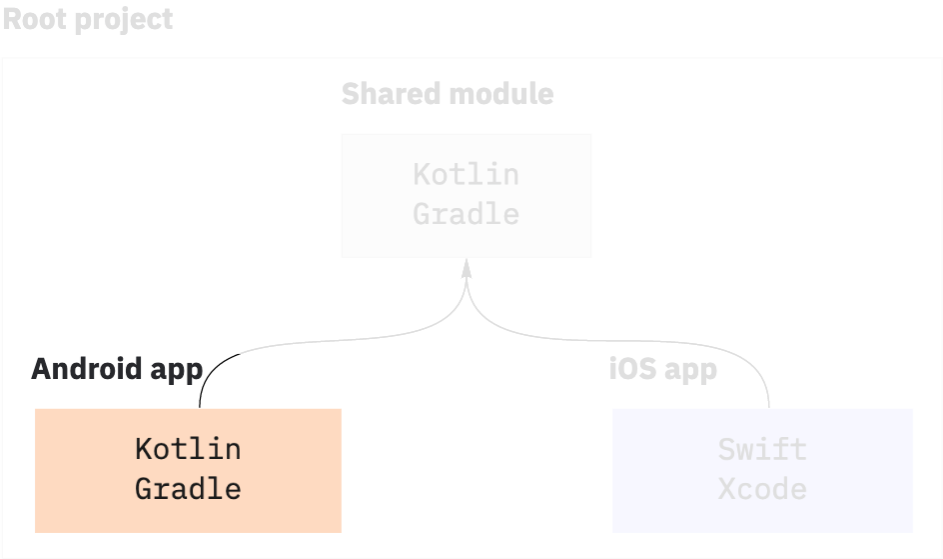
# Architecture

- Contains all the good stuff:
  - HTTP calls
  - Repositories
  - Use – Cases
  - Domain object mapping
- Almost everything!



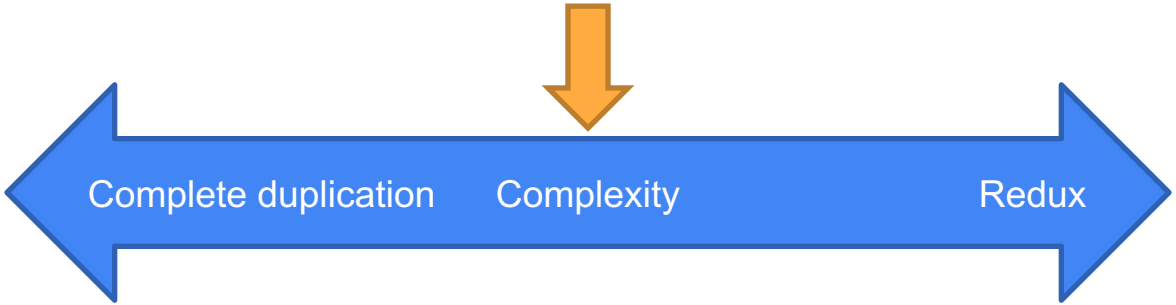
# Architecture

- Does just 1 thing – UI
- It's a true Android experience – coding and using



# Architecture

- Simple architecture
- Optimal ratio between complexity and DRY
- Can be more complex (Redux, state machine for navigation)
- Can be simpler (more code in platform modules, http calls, db, use-cases..)

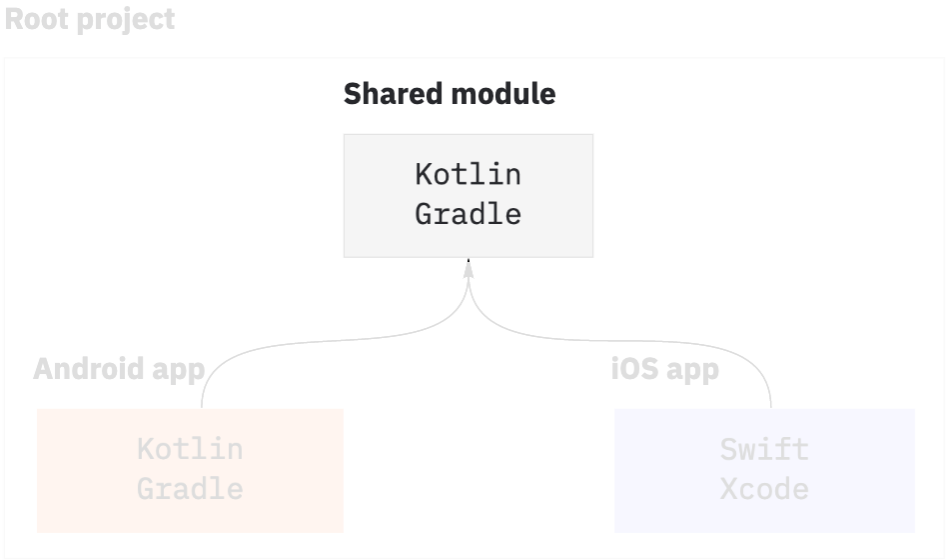


# Architecture – what I won’t show

- Navigation
- Shared or per platform
- Developer choice
- IMO – share it, I did!
- Sharing => mapper

# Architecture

- We haven't discussed some things
- Android and iOS platform code in shared
- Wait, aren't they separated?
- Yes, but..



- Intro
- DEMO
- Kotlin multi..what?
- **Architecture**
- Code walkthrough
- Expect/Actual
- Summary

- Weather app architecture is simple
- Optimal ratio of complexity and DRY
- No navigation today ☹
- No tests either! (but we can talk about them)
- More about shared a bit later

HTEC GROUP

# Code walkthrough - Android

- Android code very simple
- Compose to draw the view
- ViewModel to transform data
- Native UI and UX

# Code walkthrough - Android

- Compose drawing

```
TopAppBar {  this: RowScope
    Text(
        modifier = Modifier.fillMaxWidth(),
        textAlign = TextAlign.Center,
        text = "Platform: $platform"
    )
}
```

# Code walkthrough - Android

- Compose drawing

```
val weatherData = weatherUiState.data
Column(
    modifier = Modifier.align(Alignment.Center),
    verticalArrangement = Arrangement.spacedBy(15.dp)
) {  this: ColumnScope
    Text(text = "Temperature: ${weatherData.temperature} °C")
    Text(text = "Wind speed: ${weatherData.windSpeed} km/h")
    Text(text = "Measurement time: ${weatherData.lastUpdate}h")
}
```

# Code walkthrough - Android

- ViewModel

```
class HomeViewModel(
    private val getCurrentWeatherUseCase: GetCurrentWeatherUseCase,
    getPlatformUseCase: GetPlatformUseCase
): ViewModel() {

    var weatherState by mutableStateOf<Resource<WeatherUiState>>(Resource.Empty())

    val platform = getPlatformUseCase.execute()

    fun showCurrentWeather() {
        weatherState = Resource.Loading()
        viewModelScope.launch { this: CoroutineScope
            getCurrentWeatherUseCase.execute()
                .collectOrError(
                    onSuccess = { currentWeather ->
                        weatherState = Resource.Success(WeatherUiState(
                            temperature = currentWeather.temperature.roundToInt().toString(),
                            windSpeed = currentWeather.windSpeed.roundToInt().toString(),
                            lastUpdate = currentWeather.measurementTime.toString()
                        ))
                    },
                    onError = { it: Throwable?
                        weatherState = Resource.Error(it)
                    }
                )
        }
    }
}
```

# Code walkthrough - Android

- ViewModel public api

```
var weatherState by mutableStateOf<Resource<WeatherUiState>>(Resource.Empty())

val platform = getPlatformUseCase.execute()
```

# Code walkthrough - Android

- ViewModel data transform and use-case invocation

```
viewModelScope.launch {  this: CoroutineScope
    getCurrentWeatherUseCase.execute()
        .collectOrError(
            onSuccess = { currentWeather ->
                weatherState = Resource.Success(WeatherUiState(
                    temperature = currentWeather.temperature.roundToInt().toString(),
                    windSpeed = currentWeather.windSpeed.roundToInt().toString(),
                    lastUpdate = currentWeather.measurementTime.toString()
                ))
            },
```

# Code walkthrough - Shared

- Contains almost all the logic
- HTTP / database calls
- Use – Cases
- Repositories
- Domain model transformations

# Code walkthrough - Shared

- Get current weather

```
class GetCurrentWeatherUseCase(
    private val locationService: LocationService,
    private val weatherRepository: WeatherRepository
) {
    suspend fun execute(): Result<CurrentWeather> {
        val location = locationService.getCurrentLocation()
        return weatherRepository.getWeather(latitude = location.latitude, longitude = location.longitude)
            .map { it!!.currentWeather }
    }
}
```

# Code walkthrough - Shared

- Location service

```
class DummyLocationService: LocationService {

    private val locationNoviSad = Location( latitude: "52.52",  longitude: "13.41")

    override suspend fun getCurrentLocation(): Location = locationNoviSad

}
```

# Code walkthrough - Shared

- Weather repository

```
class SharedWeatherRepository(
    private val apiService: ApiService
): WeatherRepository, CoreRepository() {

    override suspend fun getWeather(latitude: String, longitude: String): Result<Weather> =
        safeApiCall {
            apiService.getCurrentWeatherForLocation(latitude = latitude, longitude = longitude)
                .toDomain()
        }
}
```

# Code walkthrough - Shared

- Weather repository

```
class SharedWeatherRepository(
    private val apiService: ApiService
): WeatherRepository, CoreRepository() {

    override suspend fun getWeather(latitude: String, longitude: String): Result<Weather> =
        safeApiCall {
            apiService.getCurrentWeatherForLocation(latitude = latitude, longitude = longitude)
                .toDomain()
        }
}
```

# Code walkthrough - Shared

- API service

```
override suspend fun getCurrentWeatherForLocation(latitude: String, longitude: String): WeatherRaw =
    httpClient.get( urlString: "https://api.open-meteo.com/v1/forecast") {  this: HttpRequestBuilder
        url {  this: URLBuilder
            parameters.append( name: "latitude", latitude)
            parameters.append( name: "longitude", longitude)
            parameters.append( name: "timezone",  value: "auto")
            parameters.append( name: "current_weather",  value: "true")
        }
    }.body()
```

# Code walkthrough - iOS

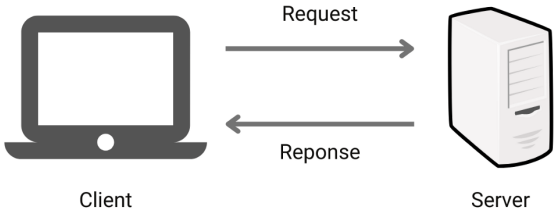
- iOS code

```
struct ContentView: View {

    @State var currentWeather = CurrentWeather(temperature: 0, windSpeed: 0, measurementTime: 0)
    @State var dataError = ""

    let platform = GetPlatformHelper().execute()
    let getCurrentWeather = GetCurrentWeatherHelper()
    var body: some View {
        Text("Platform: \(platform)")
            .offset(y: -230)

        Text("Temperature: \(Int(currentWeather.temperature.rounded())) °C")
        Text("Wind speed: \(Int(currentWeather.windSpeed.rounded())) km/h")
        Text("Measurement time: \(currentWeather.measurementTime)h")

        if (!dataError.isEmpty) {
            Text("Error" + dataError)
        }

        Button("Get current weather") {
            Task {
                await loadData()
            }
        }.buttonStyle(.bordered)
            .offset(y: 230)
    }

    func loadData() async {
        do {
            let result = try await getCurrentWeather.execute() as! ResultSuccess<CurrentWeather>
            currentWeather = result.data!
        } catch {
            dataError = error.localizedDescription
        }
    }
}
```

Where’s the code?

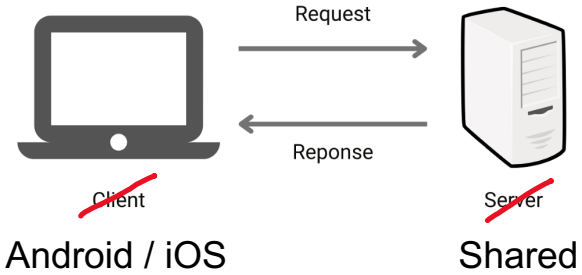
# Code walkthrough

- Client – Server architecture



# Code walkthrough

- Client – Server architecture



- Intro
- DEMO
- Kotlin multi..what?
- Architecture
- **Code walkthrough**
- Expect/Actual
- Summary

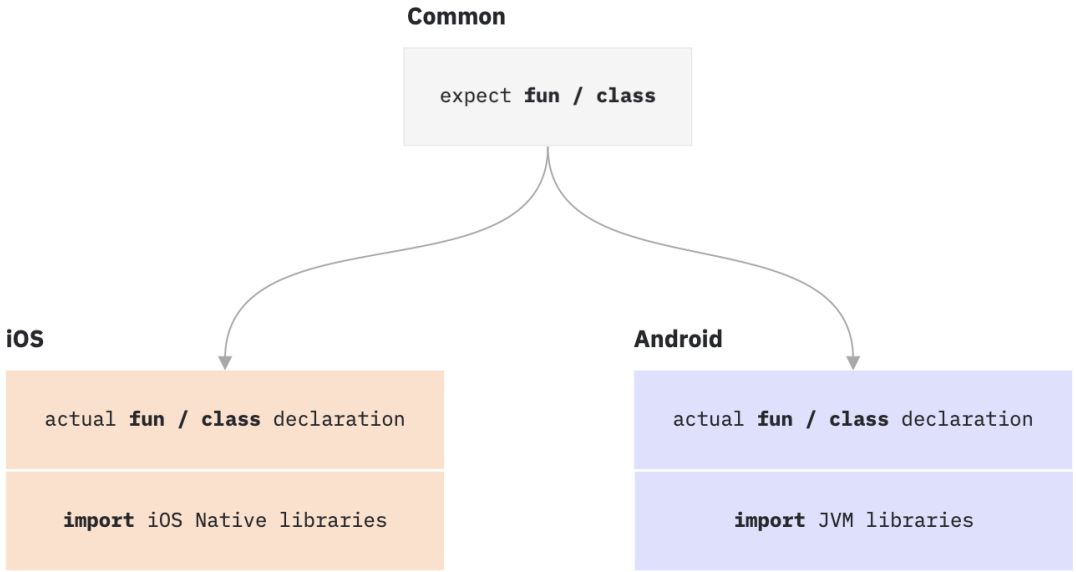
- Most logic is inside Shared
- Code is reused
- UI logic is in platform modules
- UX is native
- Kotlin for Android and Shared
- Swift / ObjC for iOS

# Expect / Actual

- Remember what we missed?

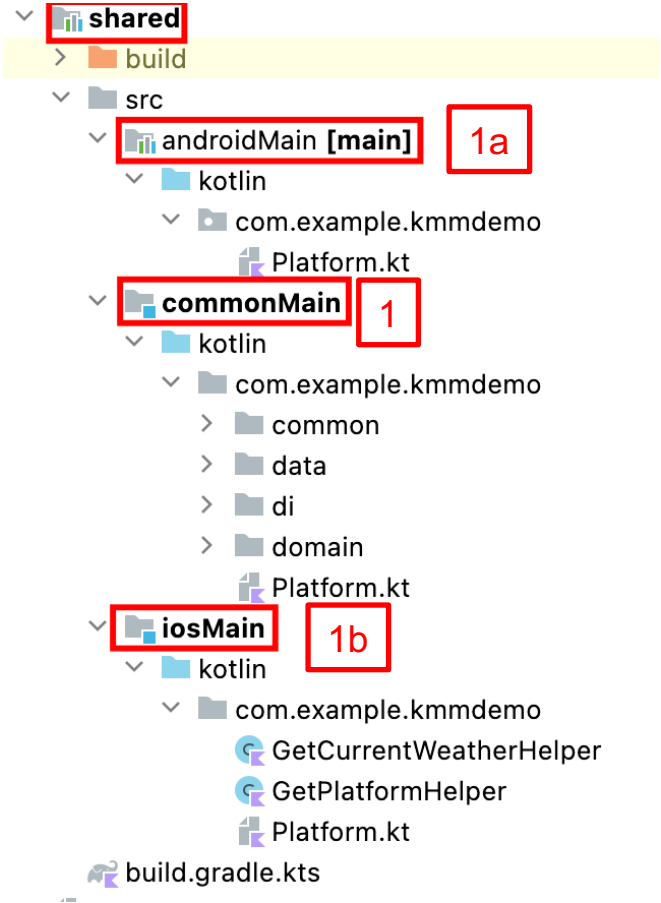
# Expect / Actual

- How did we get this platform information?



# Expect / Actual

- Shared module structure
- Module is split into 3 directories:
  - commonMain (1)
  - androidMain (1a)
  - iosMain (1b)
- Platform specific code can be accessed in 1a and 1b



# Expect / Actual

- It’s the same thing as Interface -> Class
- Only it’s done two [2] times!
- Expect / Actual can be applied to a **Function** or a **Class**
- Let’s take a look

# Expect / Actual - Shared

```
interface Platform {
    val name: String
}

internal expect fun getPlatform(): Platform
```

# Expect / Actual - Android

```
class AndroidPlatform : Platform {
    override val name: String = "Android ${android.os.Build.VERSION.SDK_INT}"
}

internal actual fun getPlatform(): Platform = AndroidPlatform()
```

# Expect / Actual - iOS

```
class IOSPlatform : Platform {
    override val name: String =
        UIDevice.currentDevice.systemName() + " " + UIDevice.currentDevice.systemVersion
}

internal actual fun getPlatform(): Platform = IOSPlatform()
```

HTEC GROUP

- Intro
- DEMO
- Kotlin multi..what?
- Architecture
- Code walkthrough
- **Expect/Actual**
- Summary

- Used to get platform specific things
- Things which need different implementations in Native and JDK
- DateTime implementation
- Library development (Ktor, Koin, SQLDelight)

- Intro
- DEMO
- Kotlin multi..what?
- Architecture
- Code walkthrough
- Expect/Actual
- **Summary**

- Weather app architecture is simple
- Optimal ratio of complexity and DRY
- No navigation today ☹
- More about shared a bit later

HTEC GROUP

# Summary – Bitter XP

- There’s no flavoring support!
  - But there’s BuildKonfig
- Firebase + BuildKonfig = HFIL
- Ktor configuration can be a huge pain.. Read the docs well!
- There’s only one DB.. SQLDelight, and you might not like it!
  - You need to write SQL
- No Dagger.. ☺
- No Hilt.. ☹

# Summary – Cons

- It’s cutting edge!
- Missing features – flavoring, and others
- Coroutines only just got fixed.. And it’s still early days
- Going against Google
- Everything is more difficult

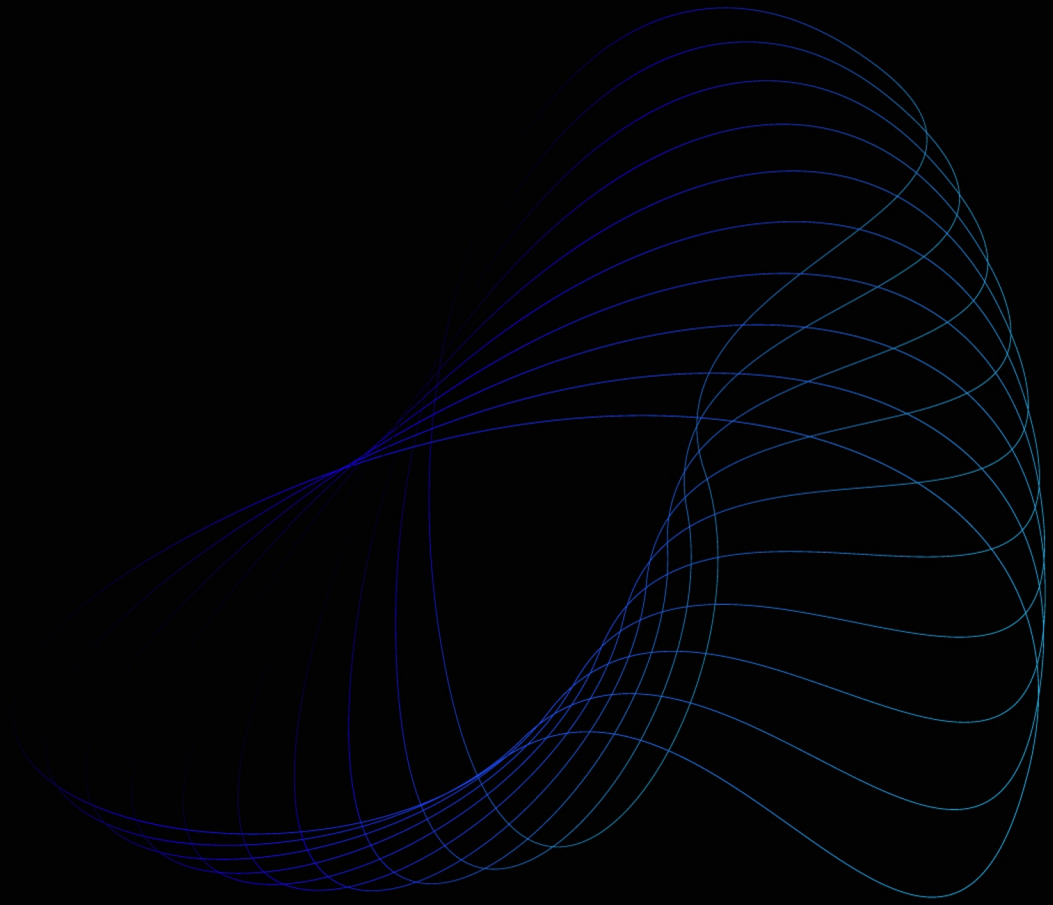
# Summary – Pros

- It’s cutting edge!
- Kotlin the whole time.. No Dart, C#, or dirty JavaScript
- Easier to maintain parity between iOS and Android
- iOS people can get on board – they have XCode
- You have XML and Compose
- Framework forces Clean Architecture and separation of concerns

# References

- https://kotlinlang.org/docs/multiplatform.html
- https://kotlinlang.org/docs/multiplatform-mobile-getting-started.html
- https://github.com/terrakok/kmm-awesome
- https://kotlinlang.org/docs/whatsnew1720.html

Hope you enjoyed and learned something today!



# Q&A

**HTEC group |** USA, San Francisco | 535 Mission St, 14th floor | +1.415.490.8175 | office-sf@htecgroup.com | htecgroup.com