PREDAVANJE

JET BRAINS

“Exploring Kotlin Multiplatform Development”

ALEXANDER SYSOEV
SOFTWARE ENGINEER

ZLATIBOR,
20-23. NOVEMBAR

Kotlin

Exploring Kotlin Multiplatform Development

Alexander Sysoev
Software Developer at Kotlin Open Source Ecosystem

JET BRAINS

What is Kotlin?

What is Kotlin?

- Statically typed language

What is Kotlin?

- Statically typed language
- Developed by JetBrains and Open Source Contributors

JET
BRAINS

What is Kotlin?

- Statically typed language
- Developed by JetBrains and Open Source Contributors
- Cross-platform

JET
BRAINS

What is Kotlin?

- Statically typed language
- Developed by JetBrains and Open Source Contributors
- Cross-platform
- Official language for Android development

JET
BRAINS

What is Kotlin?

- Statically typed language
- Developed by JetBrains and Open Source Contributors
- Cross-platform
- Official language for Android development
- Awesome

JET
BRAINS

What is Kotlin?

- Multiplatform

What is Kotlin?

- Multiplatform
  - Jvm

What is Kotlin?

- Multiplatform
  - Jvm
  - Js

What is Kotlin?

- Multiplatform
  - Jvm
  - Js
  - Wasm

What is Kotlin?

- Multiplatform
  - Jvm
  - Js
  - Wasm
  - Native

What is Kotlin?

- Multiplatform
  - Jvm
  - Js
  - Wasm
  - Native
    - Apple (iOS, macOS, tvOS, watchOS)

What is Kotlin?

- Multiplatform
  - Jvm
  - Js
  - Wasm
  - Native
    - Apple (iOS, macOS, tvOS, watchOS)
    - Windows

What is Kotlin?

- Multiplatform
  - Jvm
  - Js
  - Wasm
  - Native
    - Apple (iOS, macOS, tvOS, watchOS)
    - Windows
    - Linux

What is Kotlin? 

- Multiplatform
- Mobile

What is Kotlin?

- Multiplatform
- Mobile
  - iOS
  - Android

What is Kotlin?

- Multiplatform
- Mobile
- Web

What is Kotlin?

- Multiplatform
- Mobile
- Web
  - Js
  - Wasm

What is Kotlin?

- Multiplatform
- Mobile
- Web
- Server-side

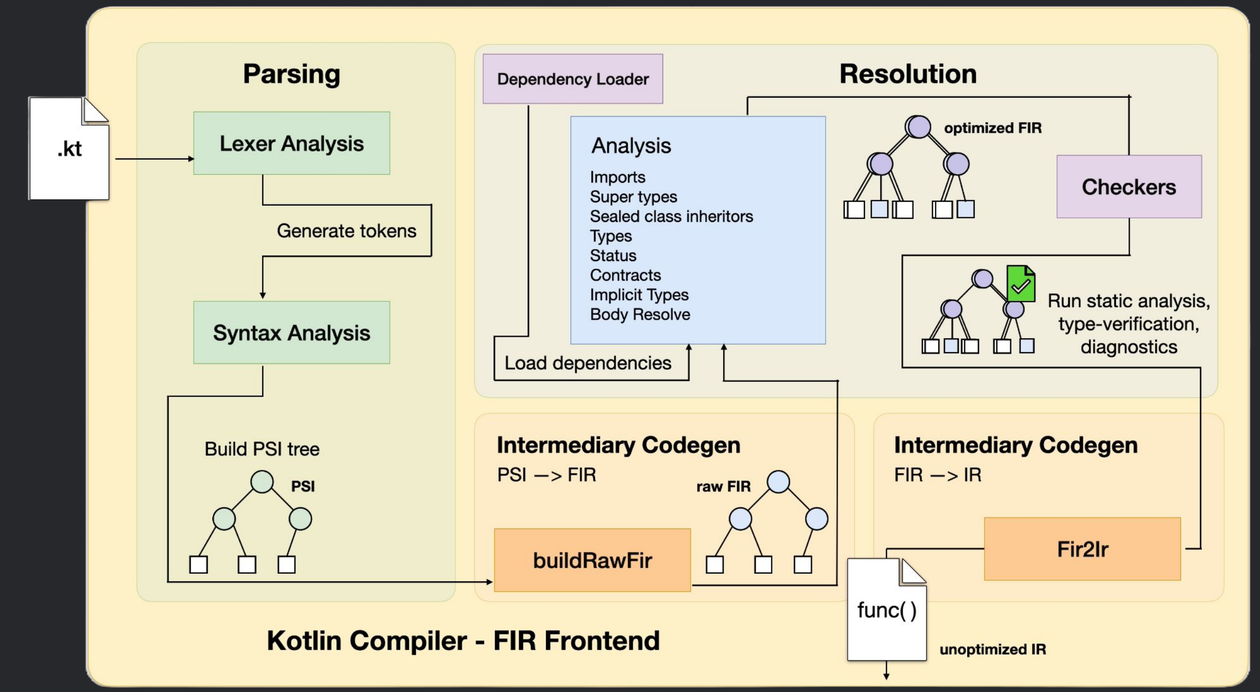
What is Kotlin?

- Multiplatform
- Mobile
- Web
- Server-side
- Data Science

Agenda

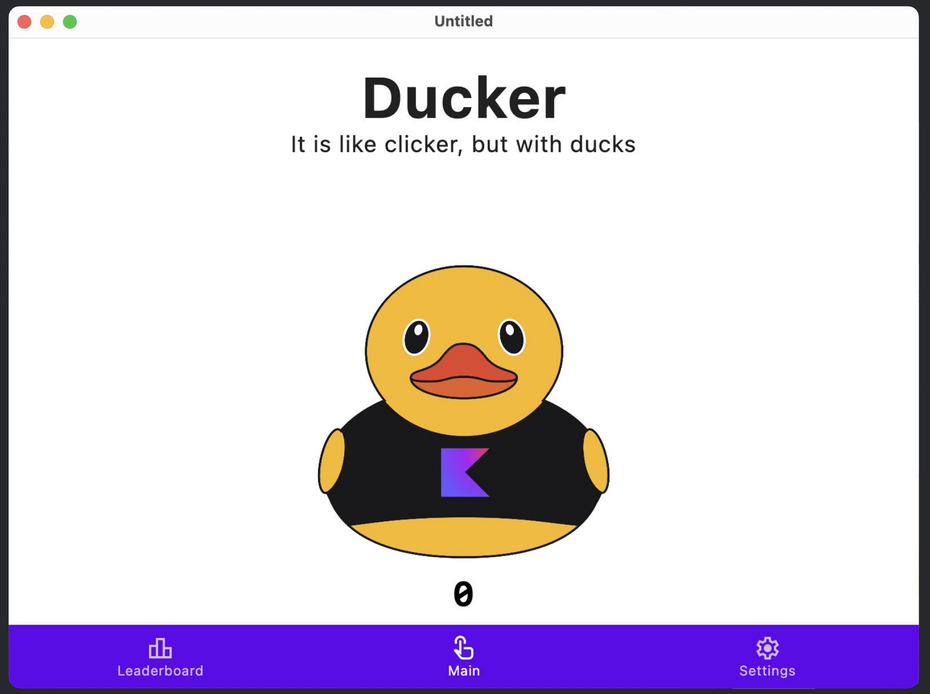
Agenda

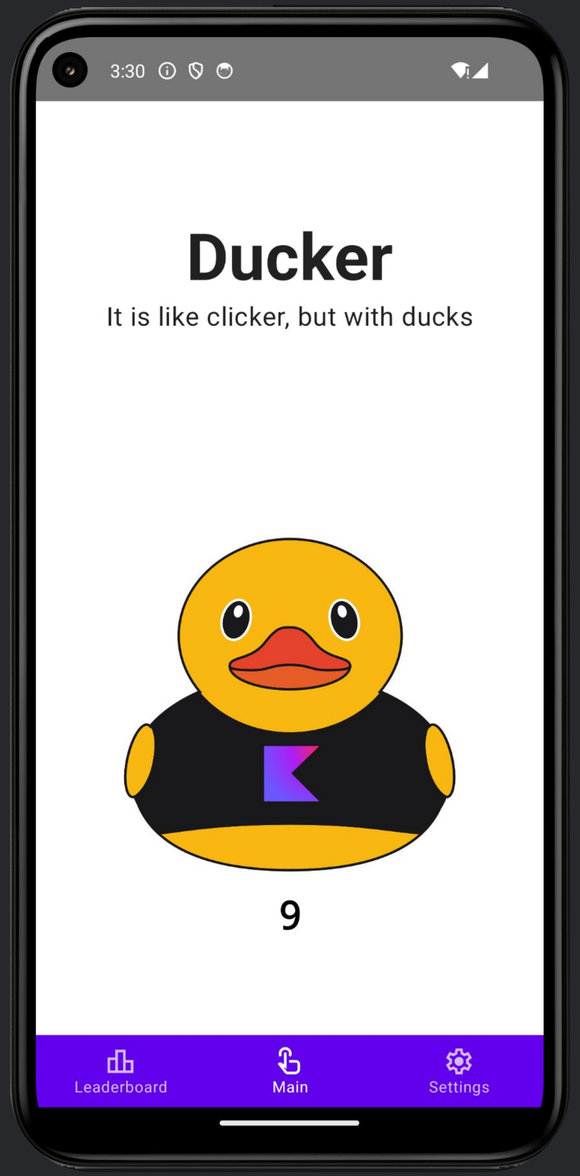
- Kotlin compiler structure overview
- Differences between K1 and K2 compilers
- New Frontend Intermediate Representation (FIR) overview
- Our first compiler plugin with multiplatform support!



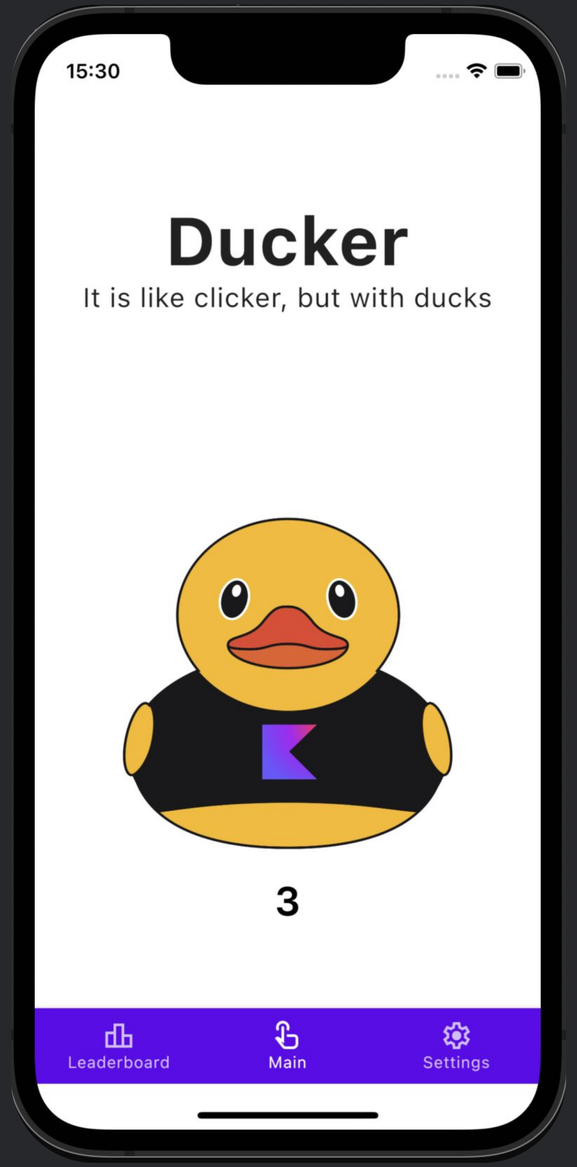
Agenda

- ~~Kotlin compiler structure overview~~
- ~~Differences between K1 and K2 compilers~~
- ~~New Frontend Intermediate Representation (FIR) overview~~
- ~~Our first compiler plugin with multiplatform support!~~
- Clicker App

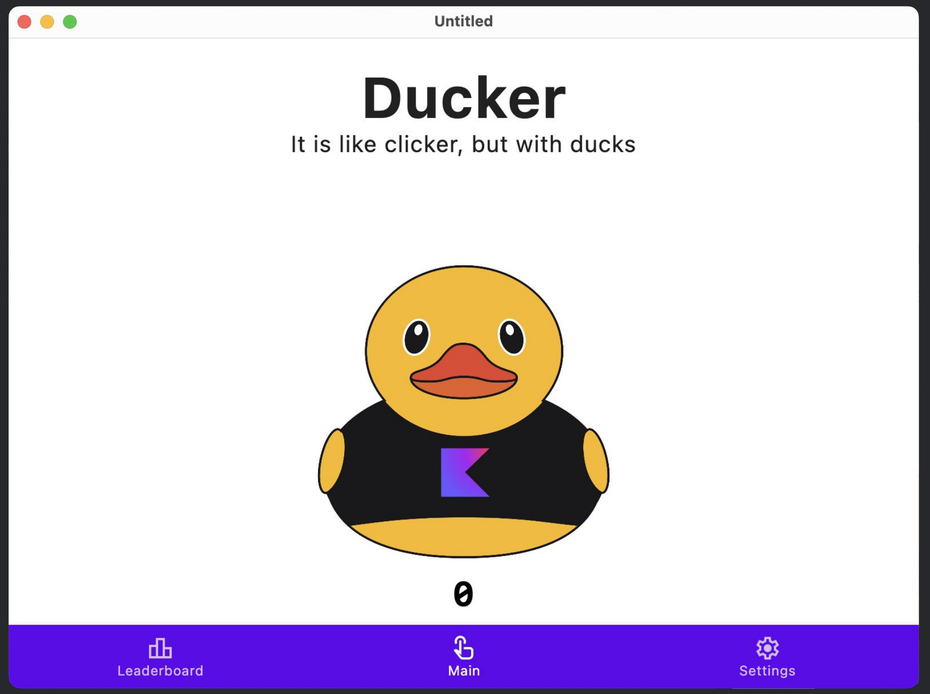




Android



iOS



Desktop

App Features

App Features

- Clickable Duck

App Features

- Clickable Duck
- Counter

App Features

- Clickable Duck
- Counter
- Many ducks!

App Features

- Clickable Duck
- Counter
- Many ducks!
- Leaderboard

App Features

- Clickable Duck
- Counter
- Many ducks!
- Leaderboard
- Android, iOS and Desktop, share 100% of code

How are we going to make it?

How are we going to make it?

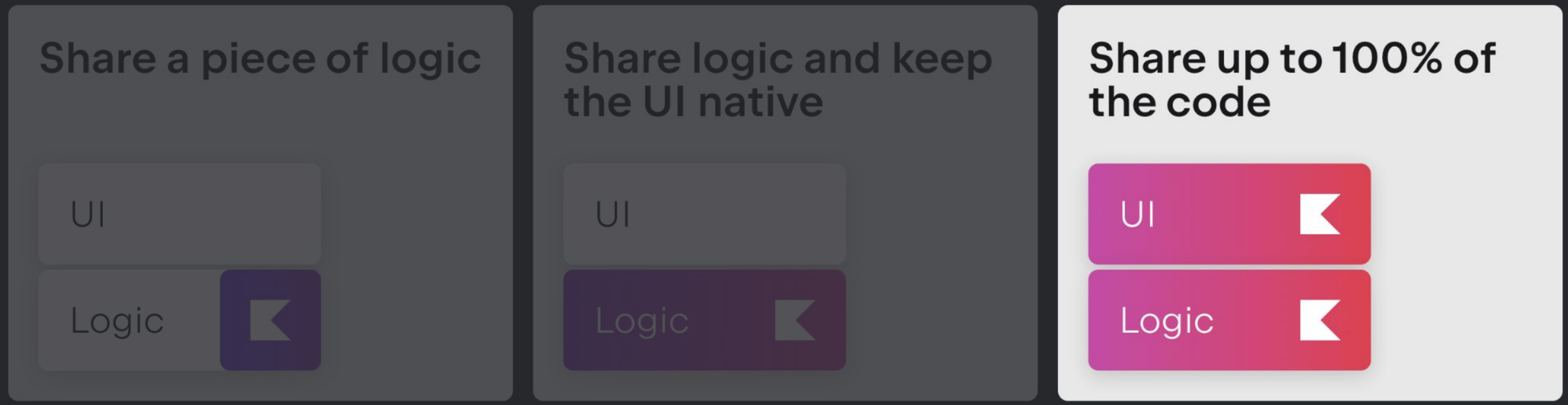
- Kotlin Multiplatform

How are we going to make it?

- Kotlin Multiplatform

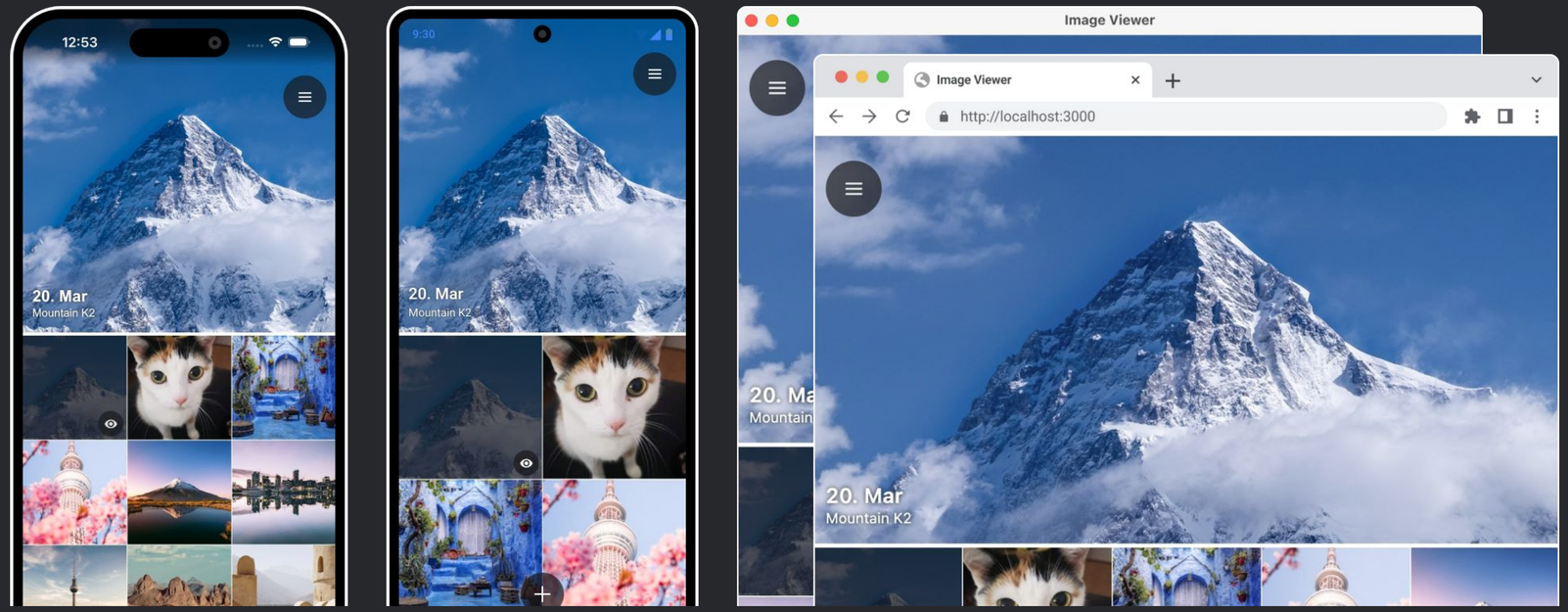
How are we going to make it?

- Kotlin Multiplatform



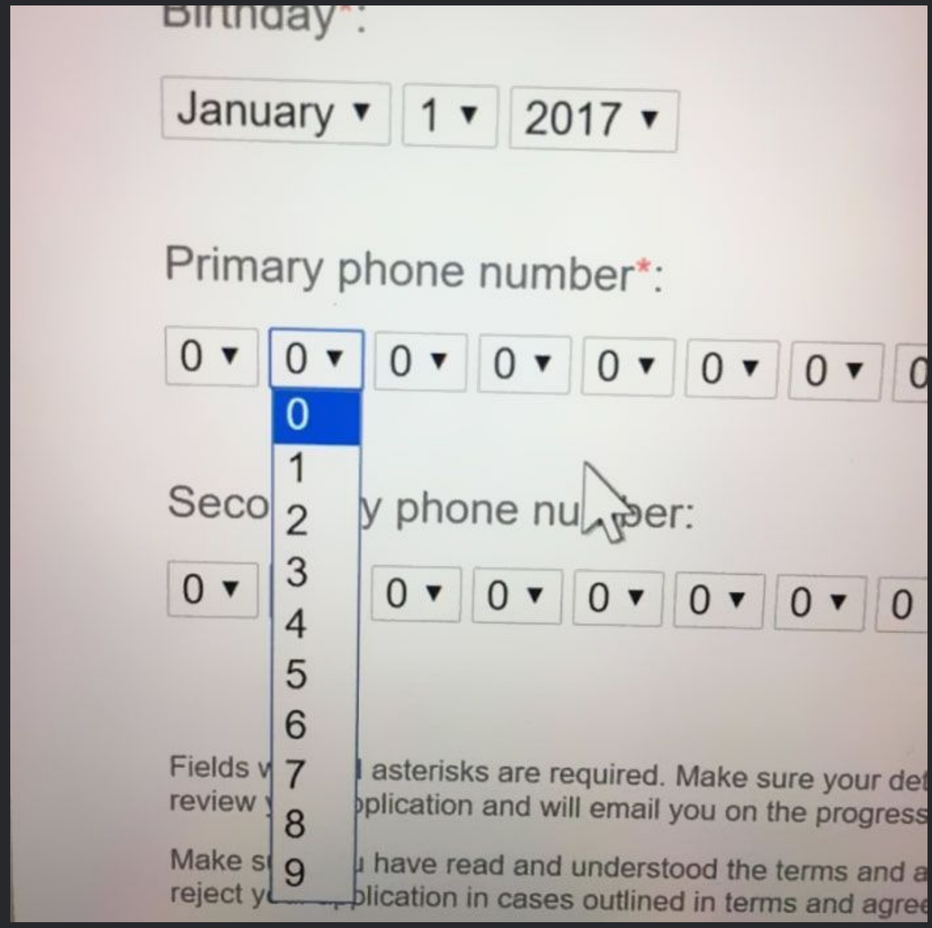
How are we going to make it?

- Kotlin Multiplatform
- Compose Multiplatform



How are we going to make it?

- Kotlin Multiplatform
- Compose Multiplatform
- No UI Design skills



How are we going to make it?

- Kotlin Multiplatform
- Compose Multiplatform
- No UI Design skills
- Use cool tooling!

How are we going to make it?

- Kotlin Multiplatform
- Compose Multiplatform
- No UI Design skills
- Use cool tooling!
- Experiment!

Development

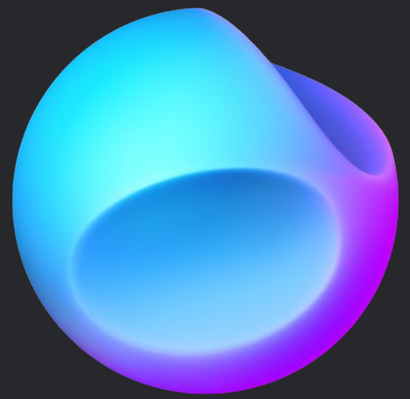
How to start?

How to start?

We need a place to write code!

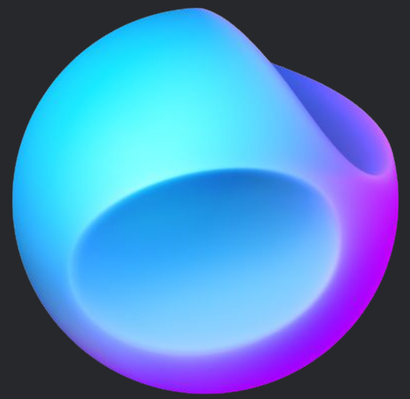
How to start?

We need a place to write code!



How to start?

We need a place to write code!



*

How to start?

We need a place to write code!

*

How to start?

We need a place to write code!

* Fleet's support for KMP projects is in early stages of development

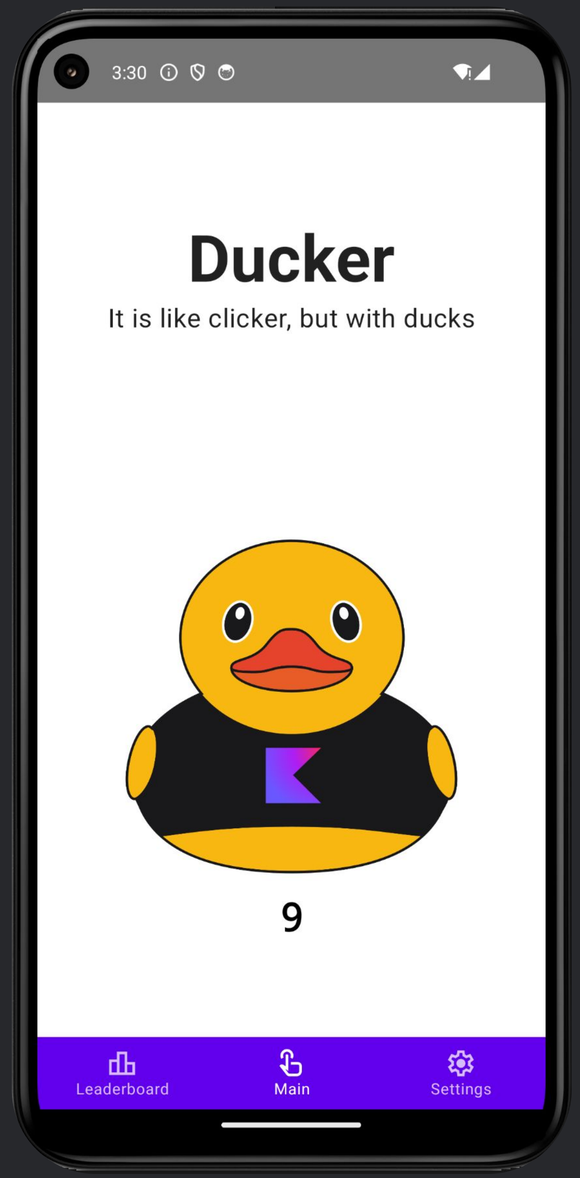
Basic setup 

Mobile emulators

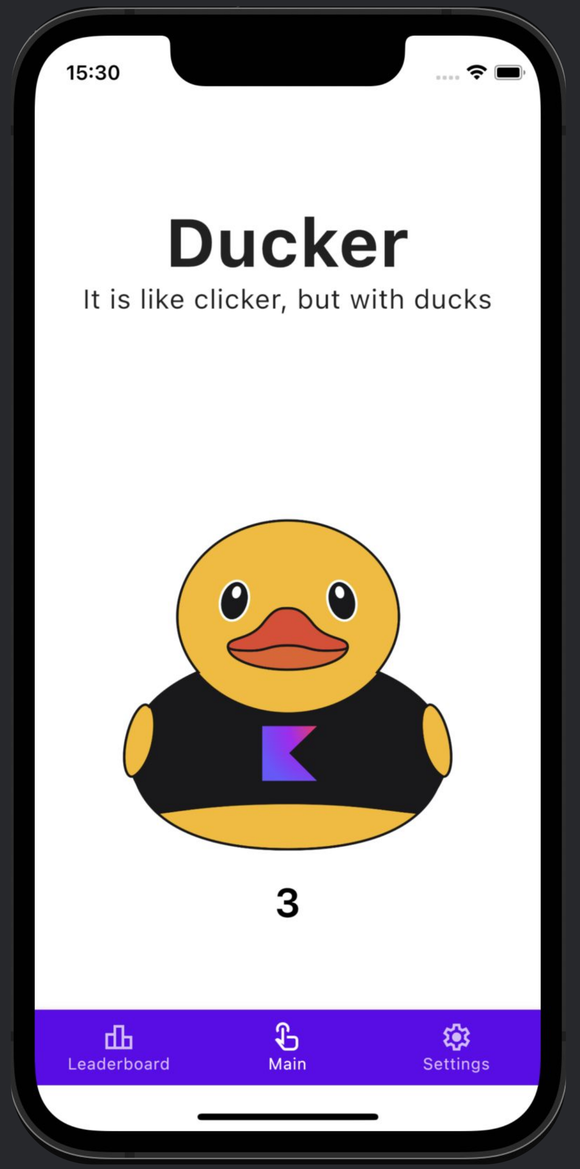
Basic setup

Mobile emulators

- Setup in Android Studio and Xcode



Android

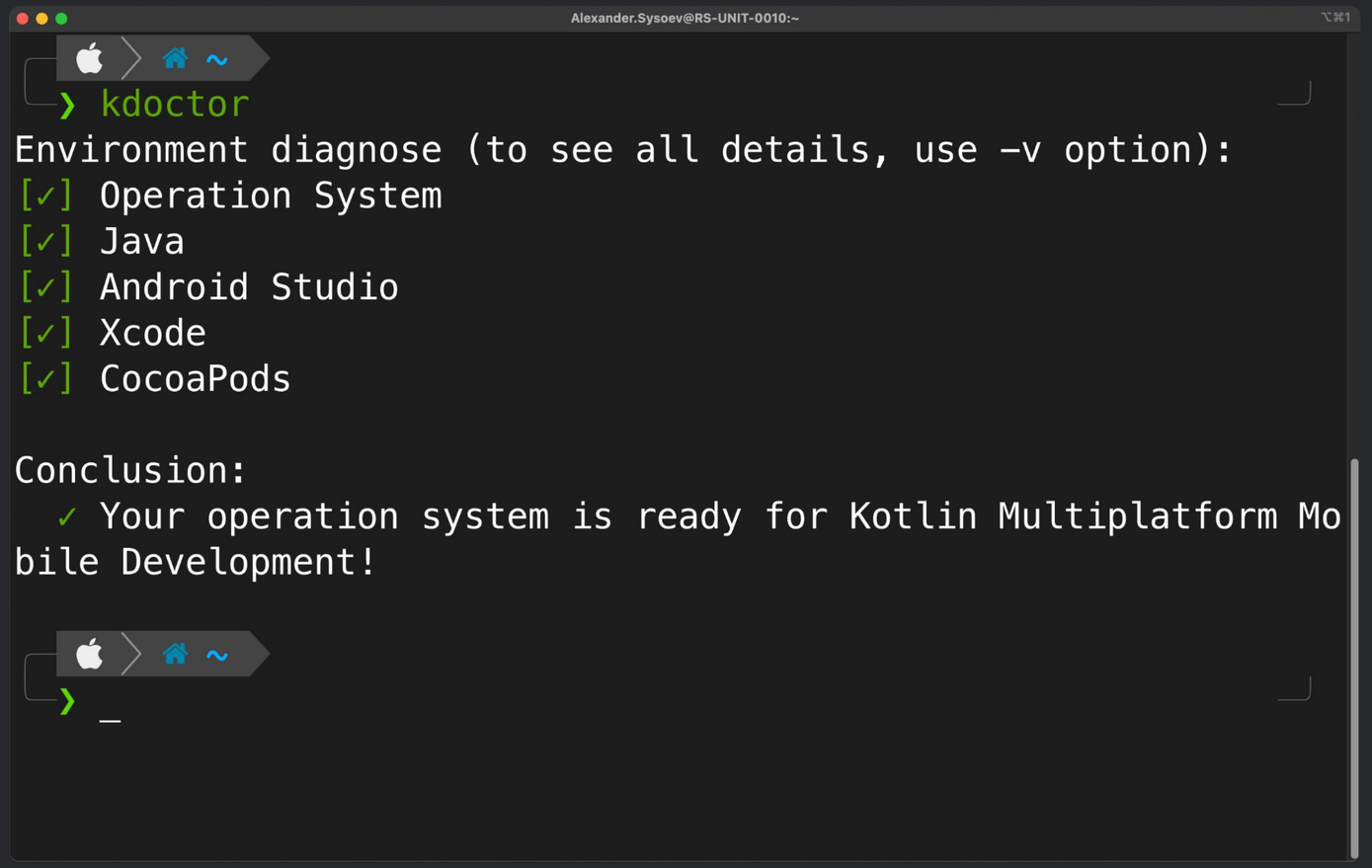


iOS

Basic setup

Mobile emulators

- Setup in Android Studio and Xcode
- Verify setup with ***kdoctor***

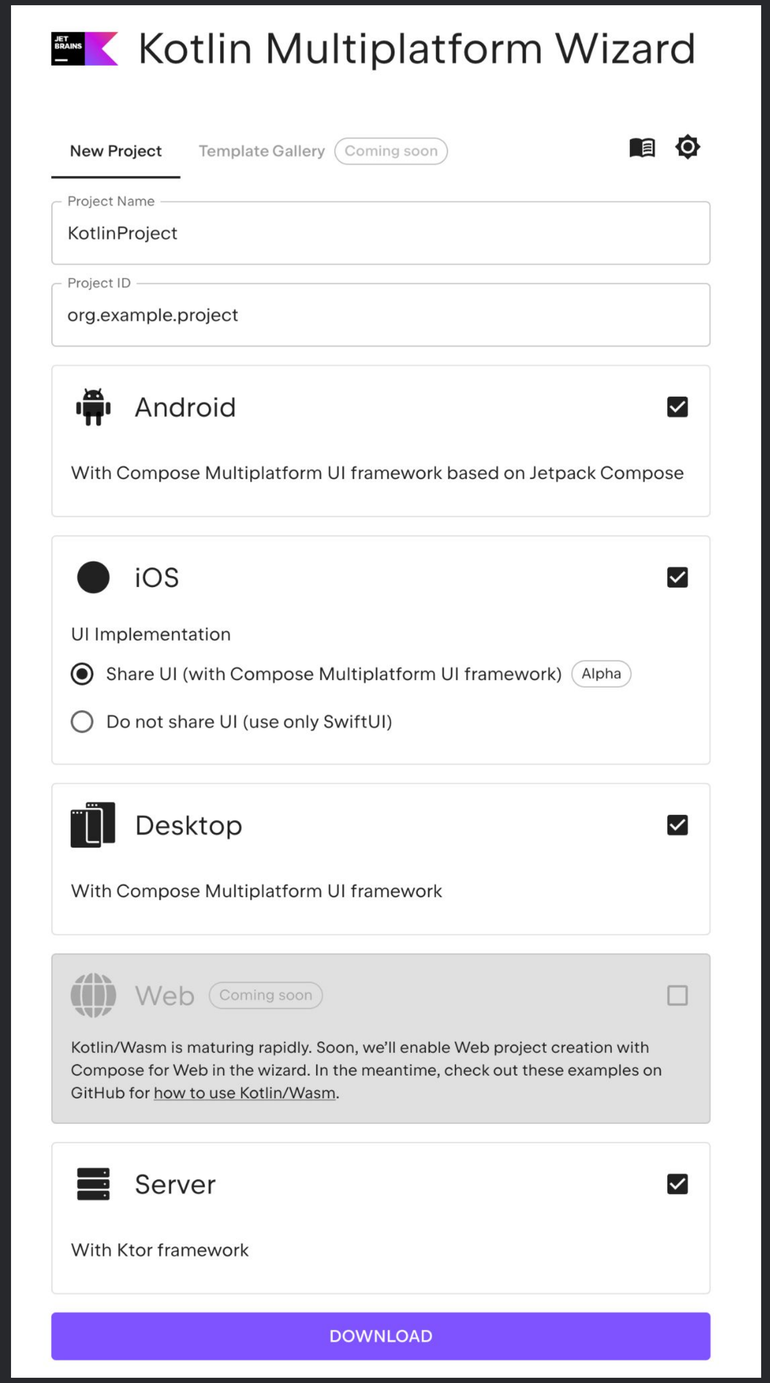


Start coding!

Start coding!

Project wizard

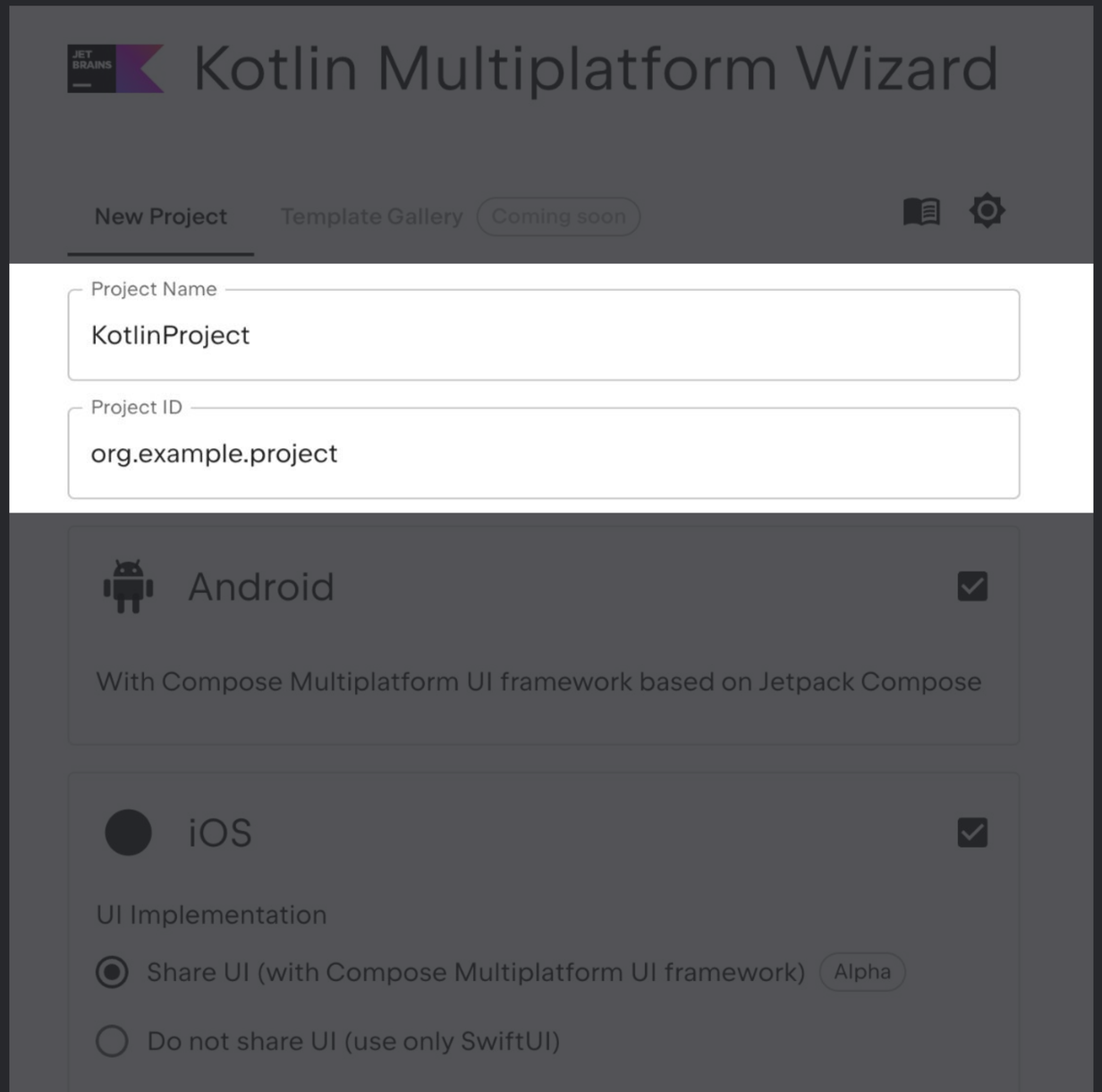
- https://kmp.jetbrains.com/



Start coding!

Project wizard

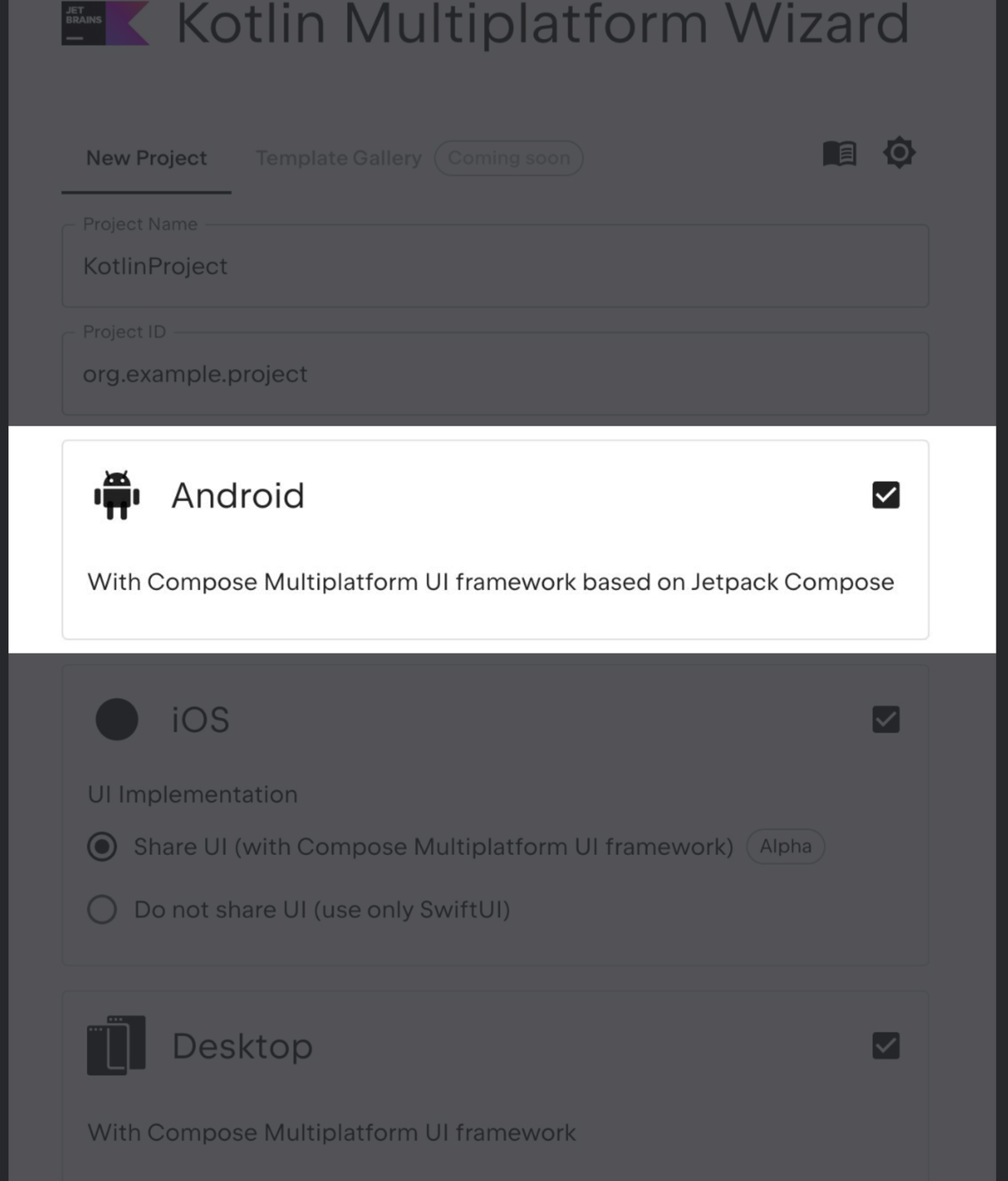
- https://kmp.jetbrains.com/



Start coding!

Project wizard

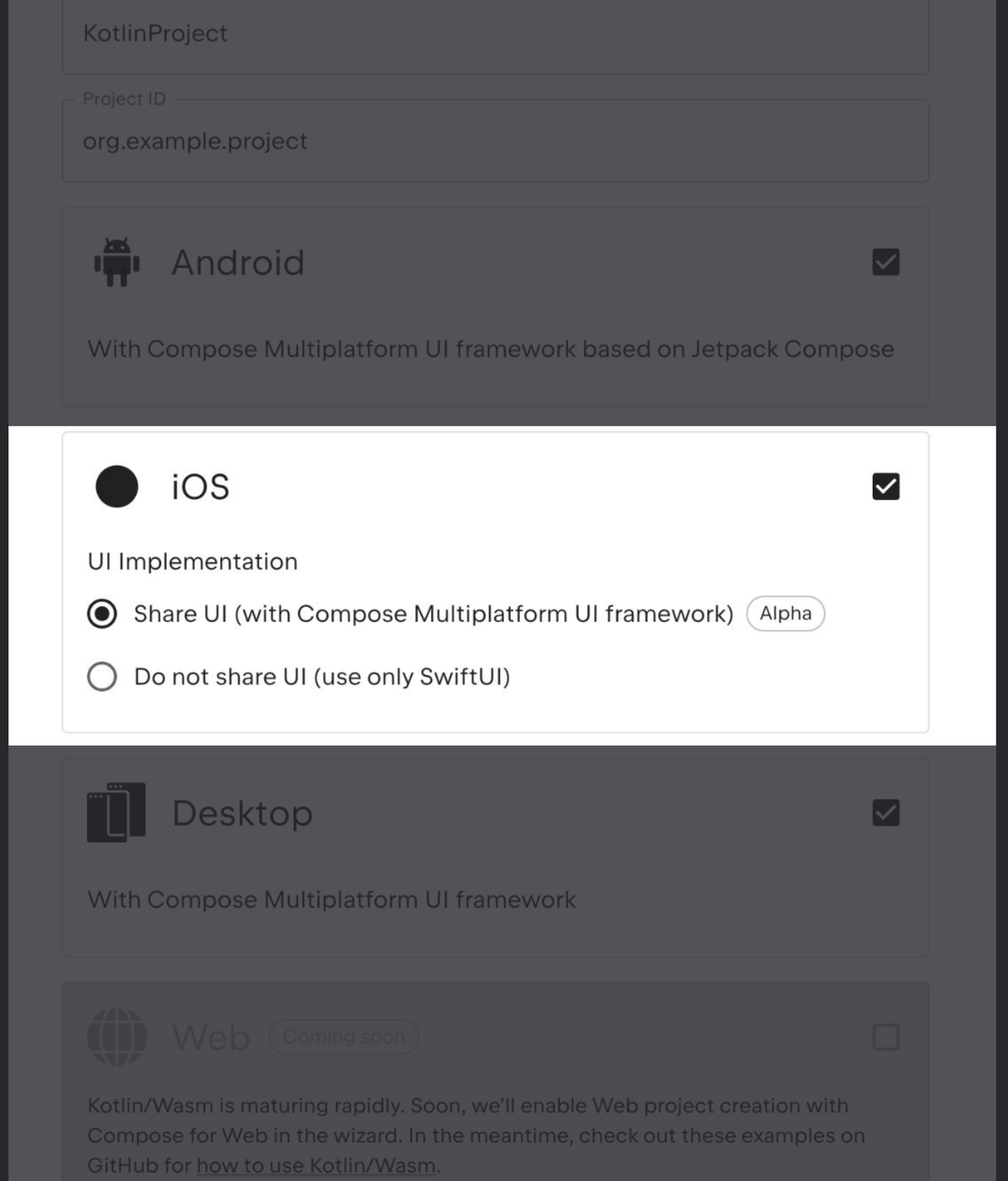
- https://kmp.jetbrains.com/
- Android



Start coding!

Project wizard

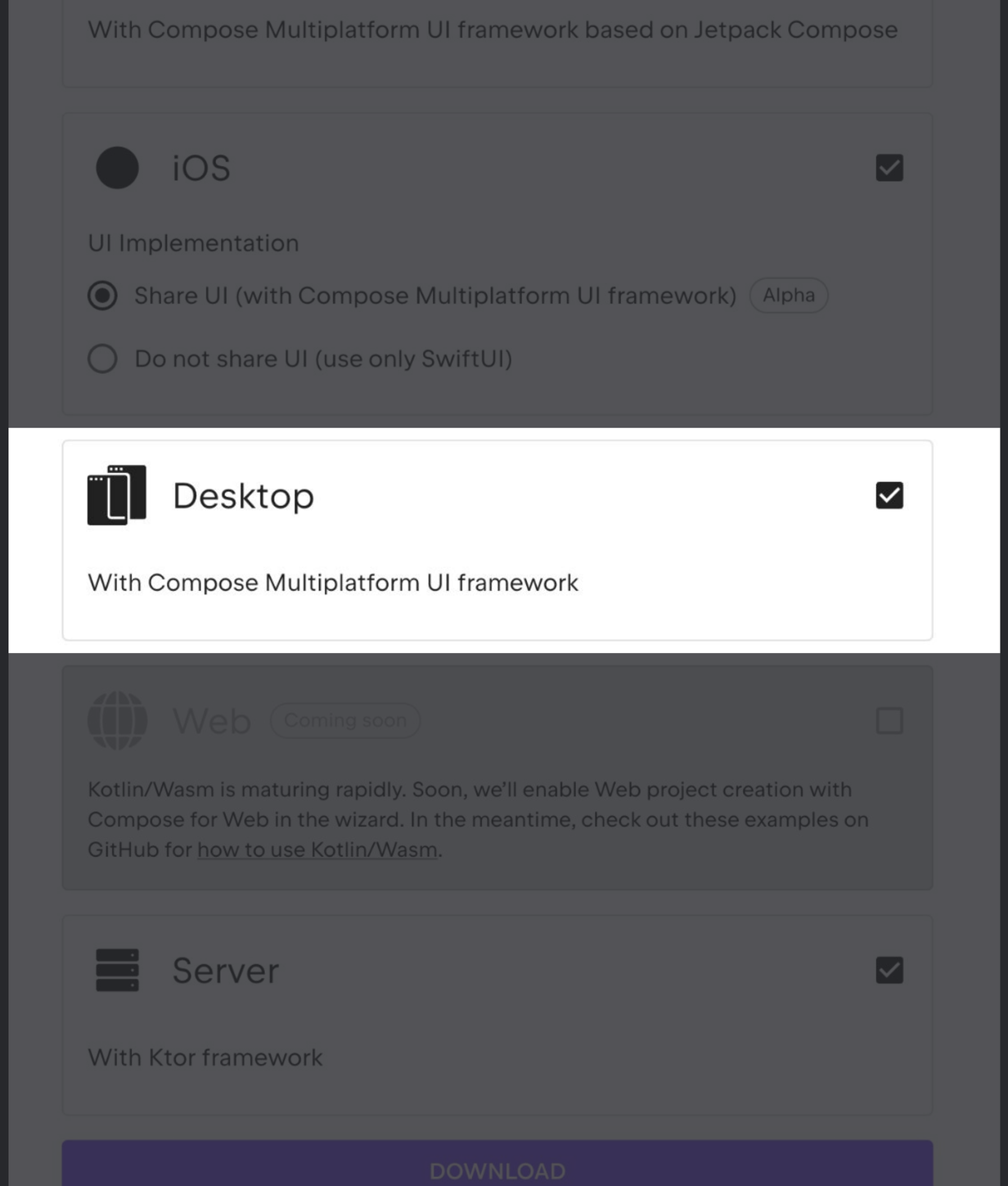
- https://kmp.jetbrains.com/
- Android
- iOS



Start coding!

Project wizard

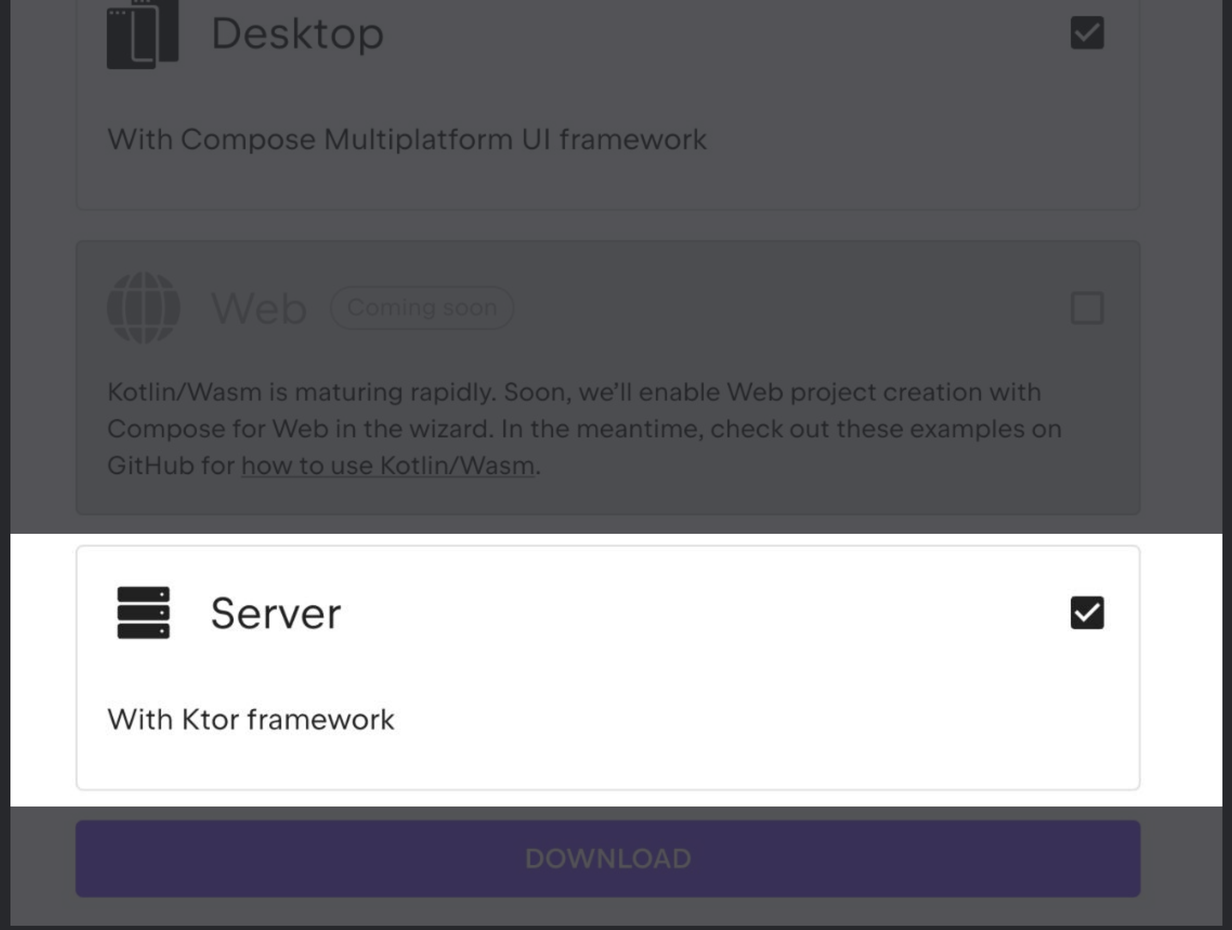
- https://kmp.jetbrains.com/
- Android
- iOS
- Desktop



Start coding!

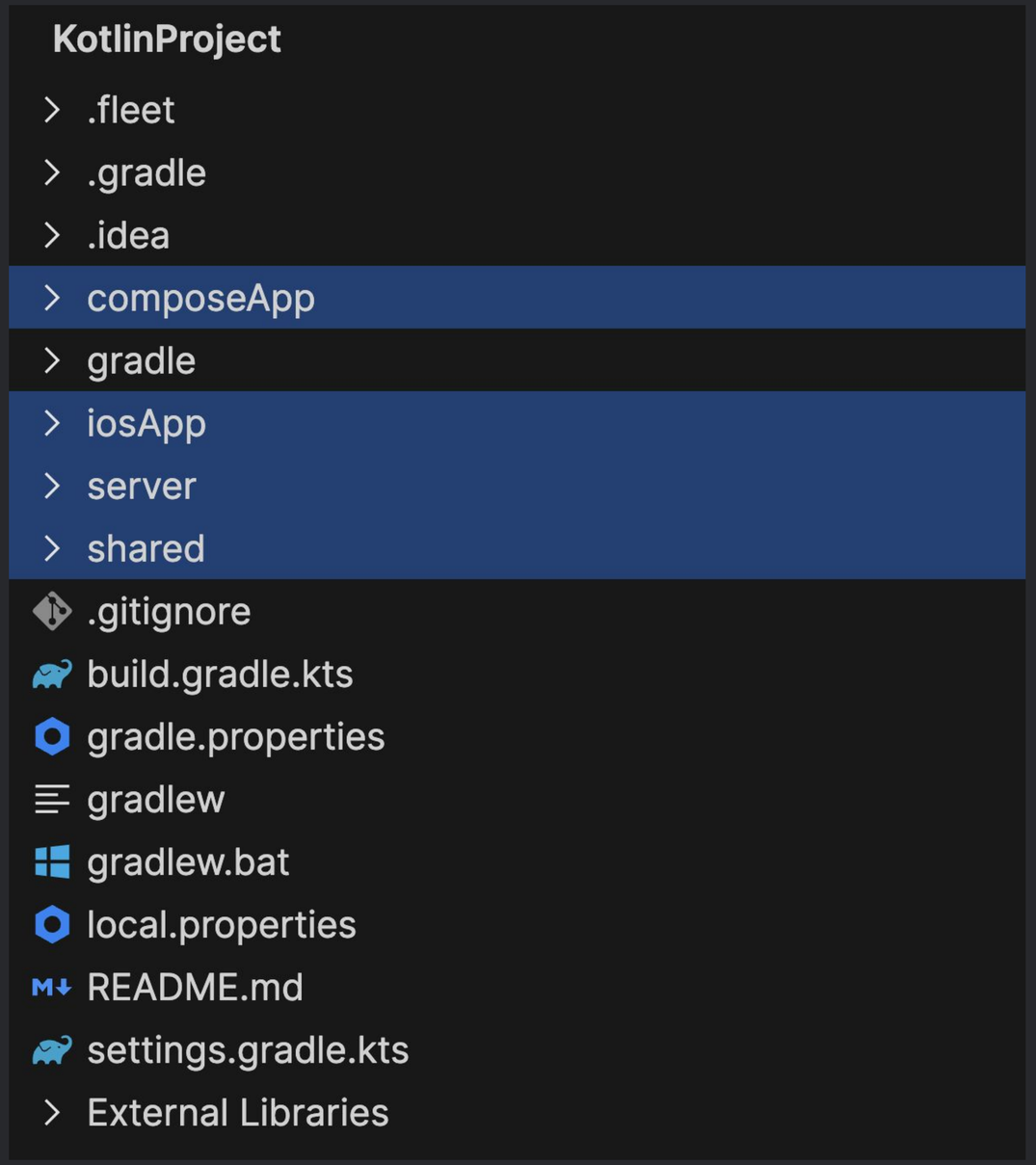
Project wizard

- https://kmp.jetbrains.com/
- Android
- iOS
- Desktop
- Server



Project tree

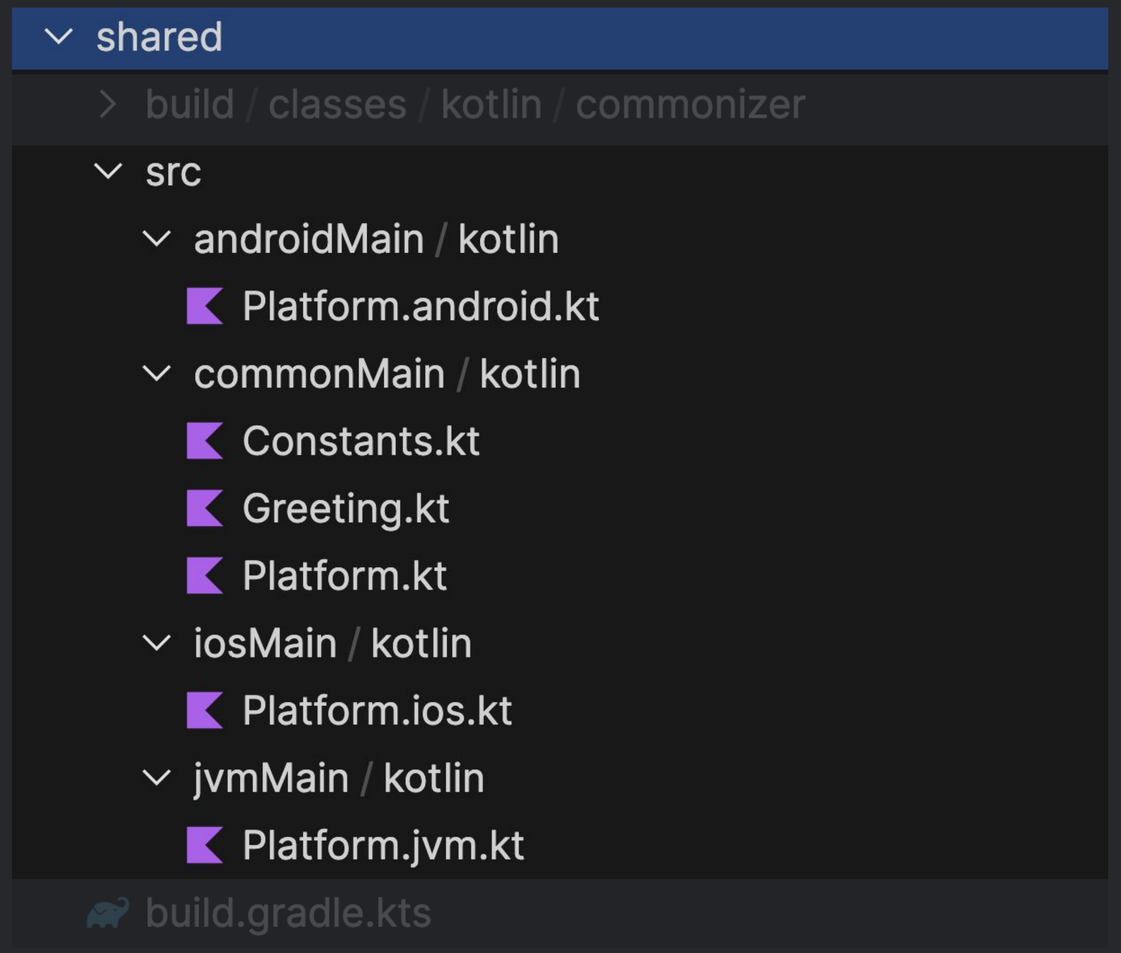
Modules



Project tree

Modules

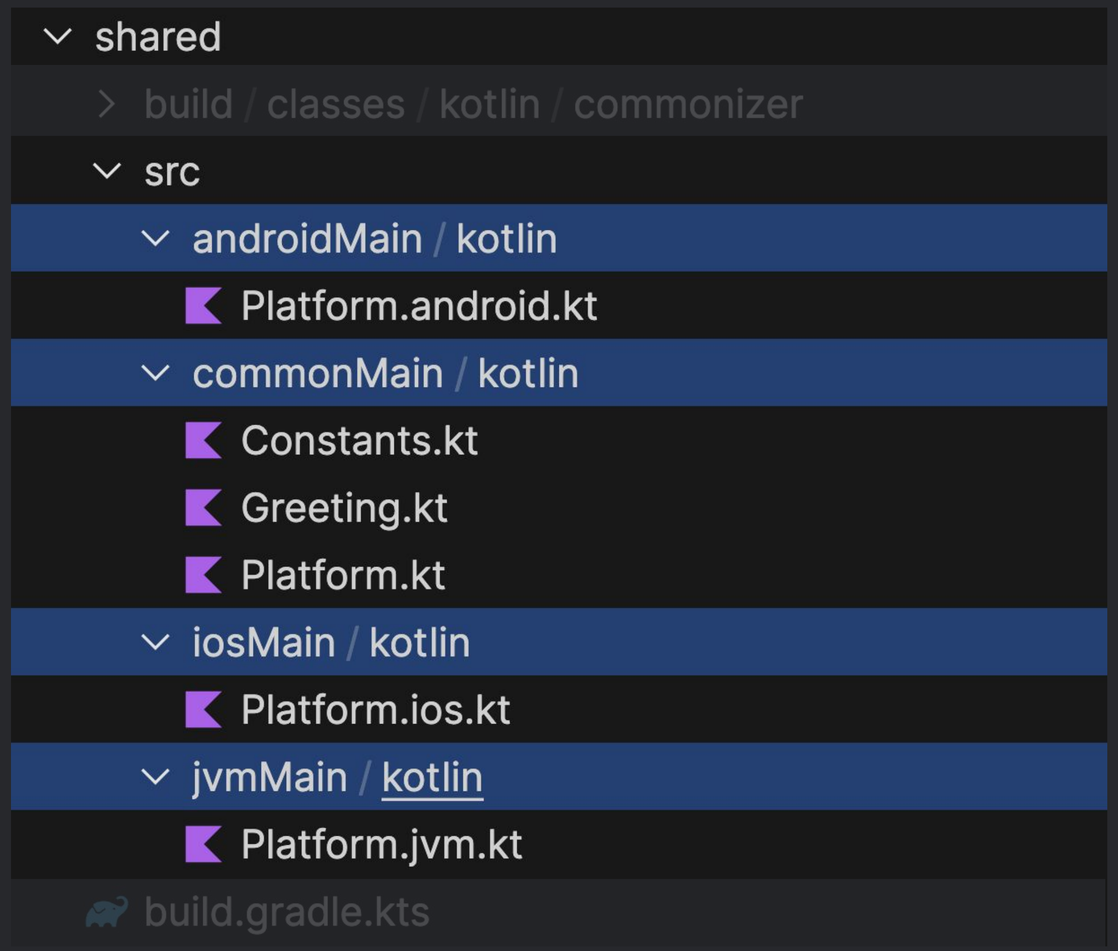
- shared



Project tree

Modules

- shared, source sets:
  - Common
  - Android
  - iOS
  - JVM



Project tree

Sharing code

```
// commonMain/Platform.kt
interface Platform {
    val name: String
}

expect fun getPlatform(): Platform
```

Project tree

Sharing code

```
// commonMain/Platform.kt
interface Platform {
    val name: String
}

expect fun getPlatform(): Platform
```

```
// jvmMain/Platform.jvm.kt
class JVMPlatform: Platform {
    override val name: String =
        "Java ${System.getProperty("java.version")}"
}

actual fun getPlatform(): Platform = JVMPlatform()
```

Project tree

Sharing code

```kotlin
// commonMain/Platform.kt
interface Platform {
    val name: String
}

expect fun getPlatform(): Platform
```

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

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

Project tree

Sharing code

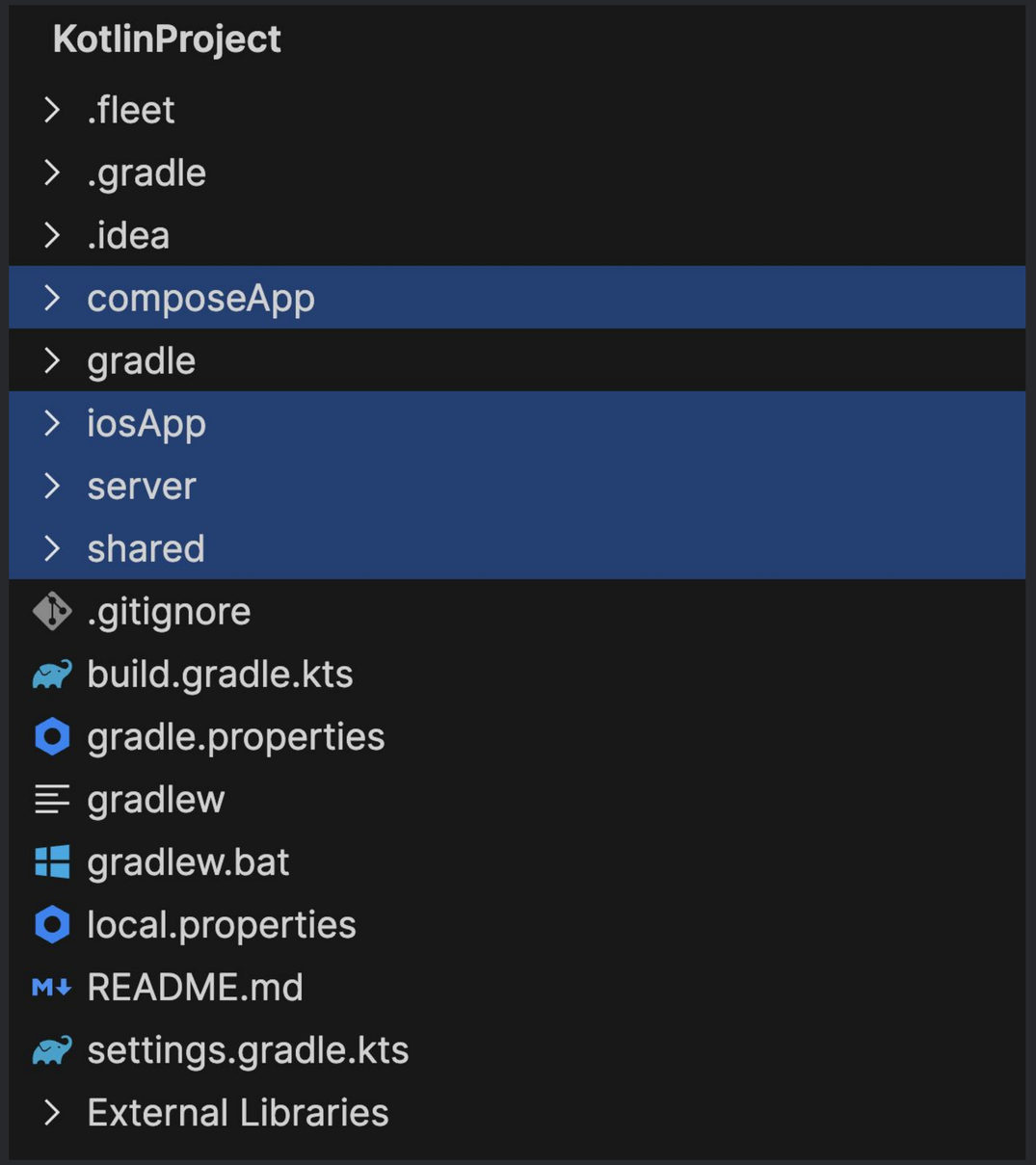
```
// commonMain/Platform.kt
interface Platform {
    val name: String
}

expect fun getPlatform(): Platform
```

```
// iosMain/Platform.ios.kt
class IOSPlatform: Platform {
    private val systemName =
        UIDevice.currentDevice.systemName()
    private val systemVersion =
        UIDevice.currentDevice.systemVersion
    override val name: String =
        "$systemName $systemVersion"
}
actual fun getPlatform(): Platform = IOSPlatform()
```

Project tree

Modules

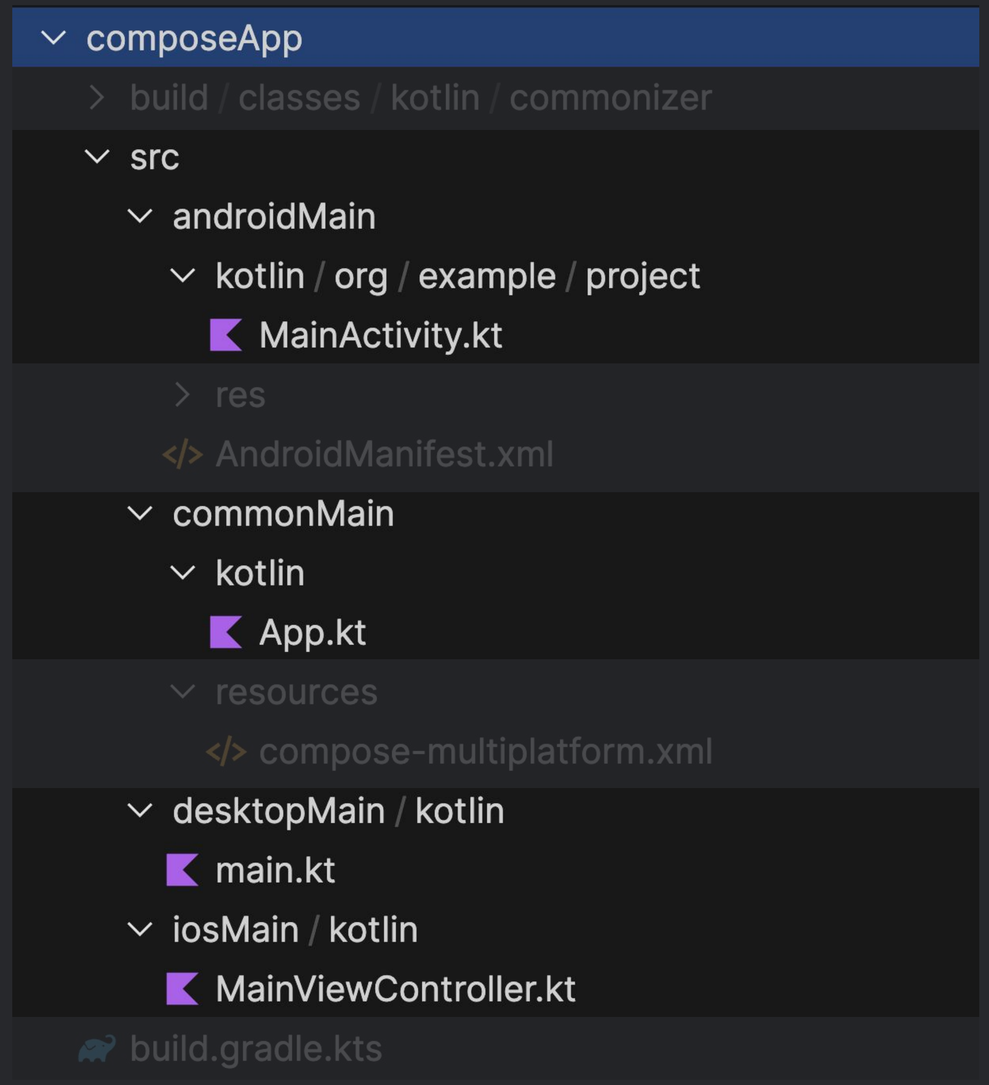
- shared



Project tree

Modules

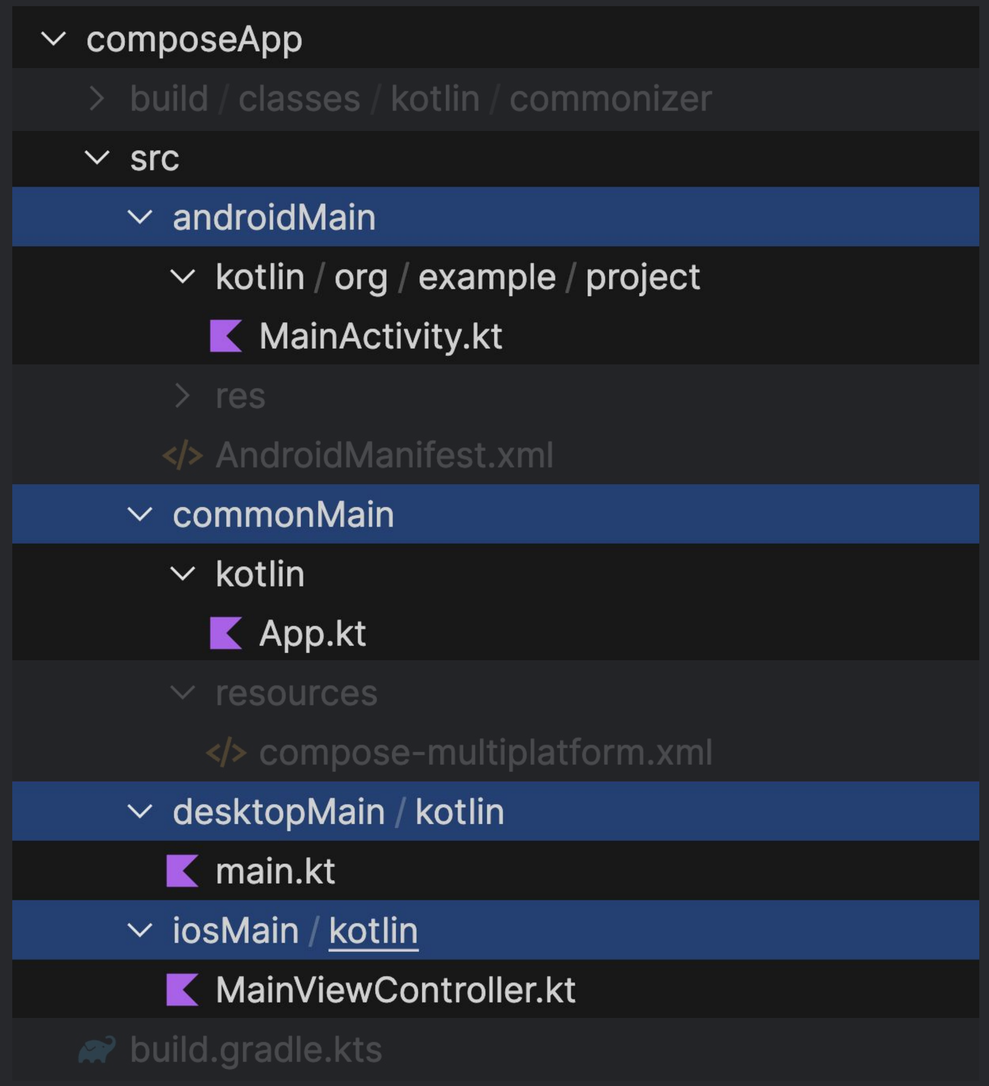
- shared
- composeApp



Project tree

Modules

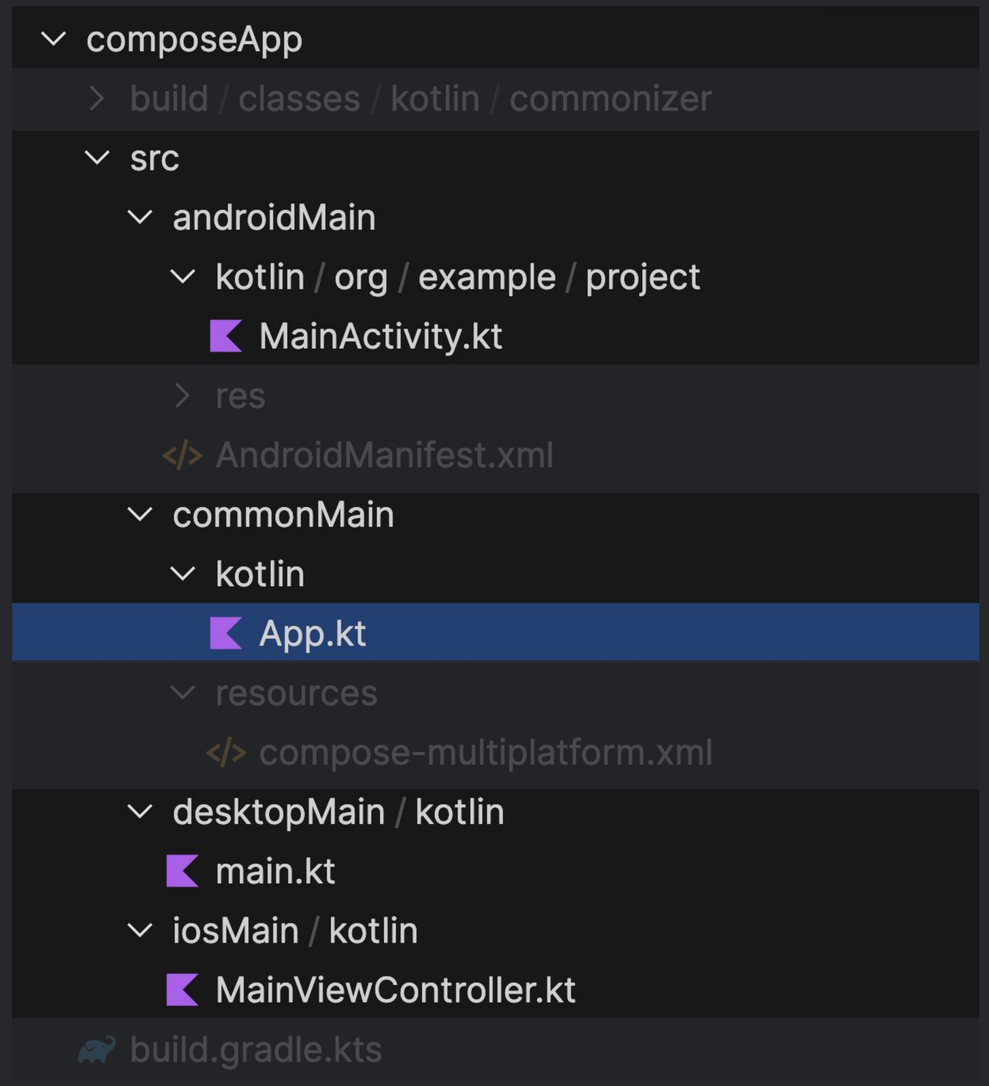
- shared
- composeApp, source sets:
  - Common
  - Android
  - iOS
  - Desktop



Project tree

Modules

- shared
- composeApp, source sets:
  - Common - Logic and UI of the App
  - Android
  - iOS
  - Desktop

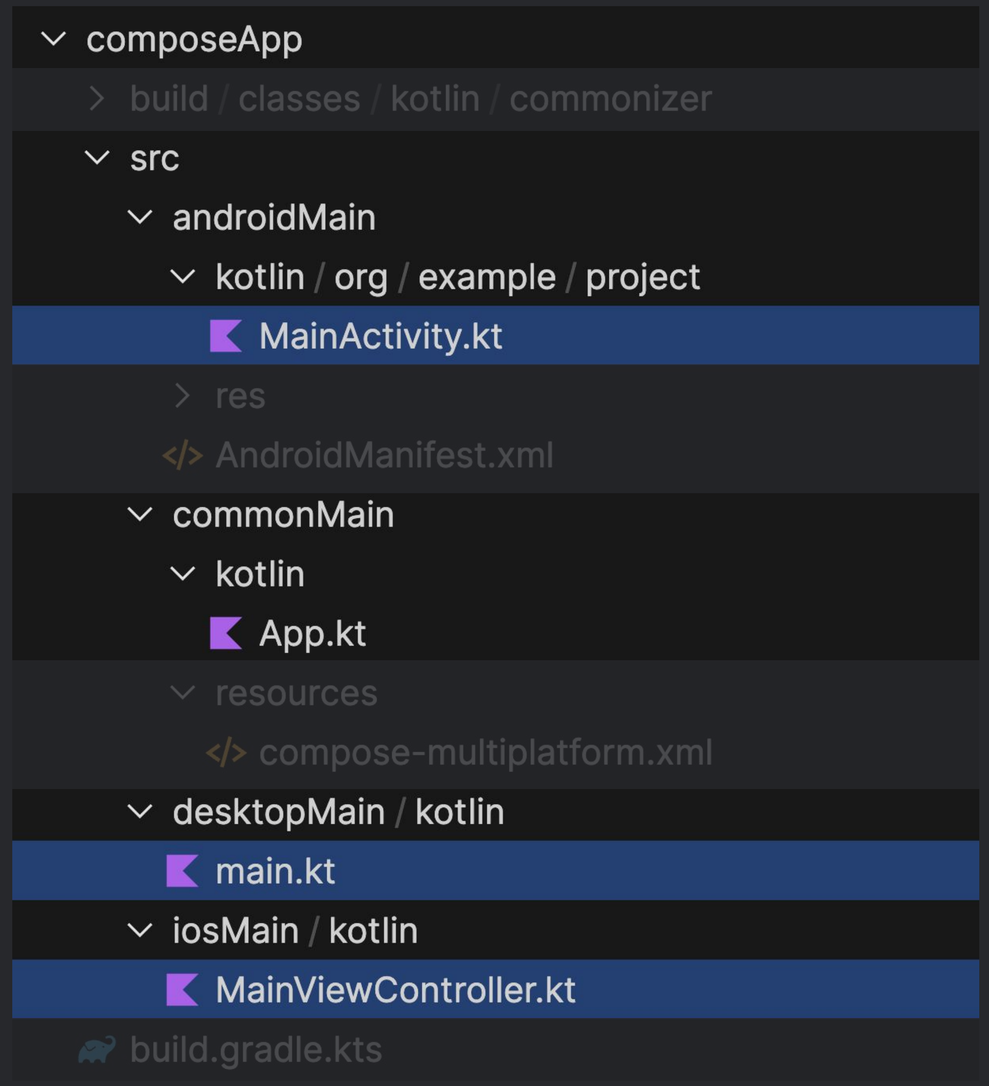


Project tree

Modules

- shared
- composeApp, source sets:
  - Common - Logic and UI of the App
  - Android
  - iOS
  - Desktop

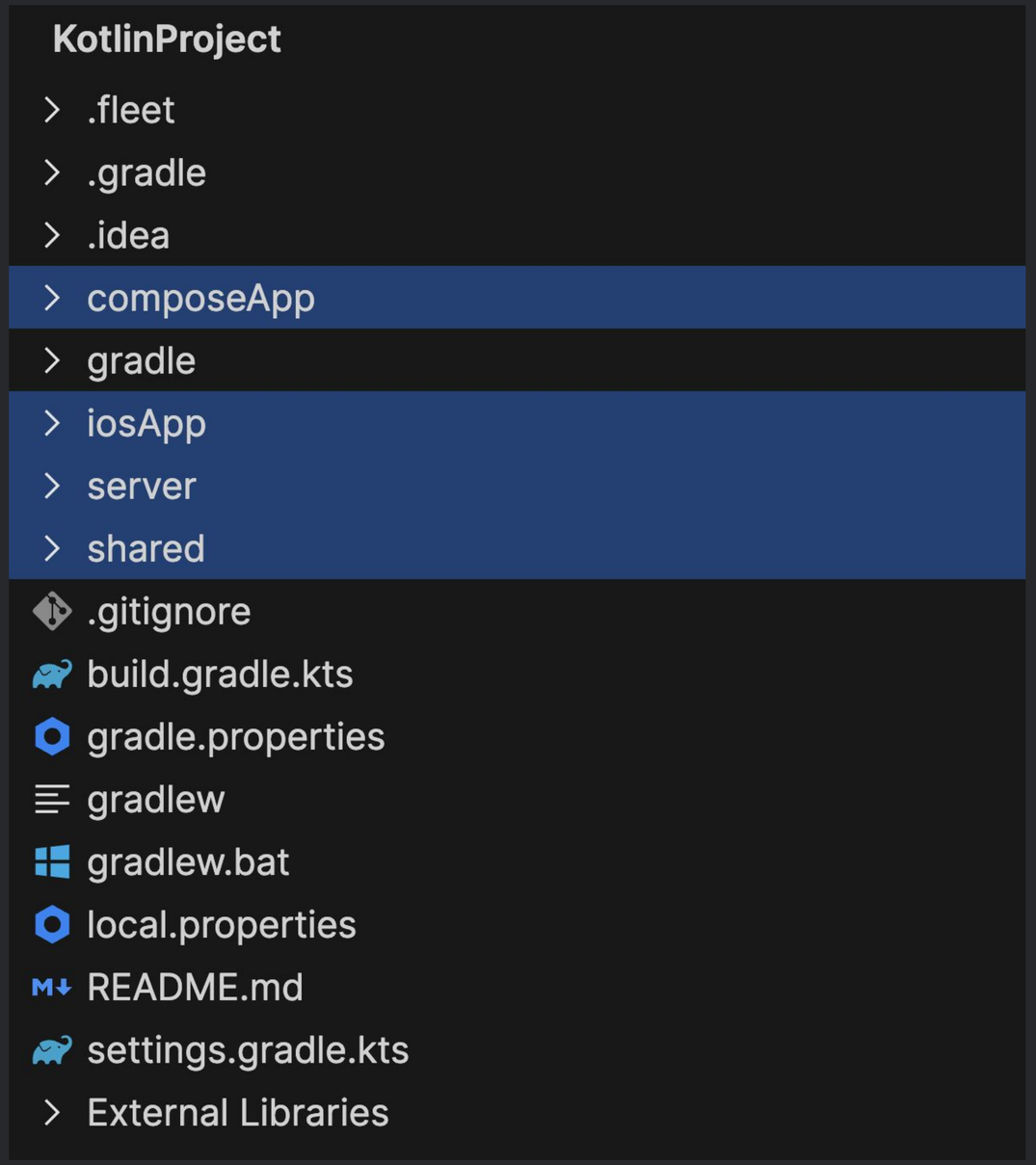
} How to run the App



Project tree

Modules

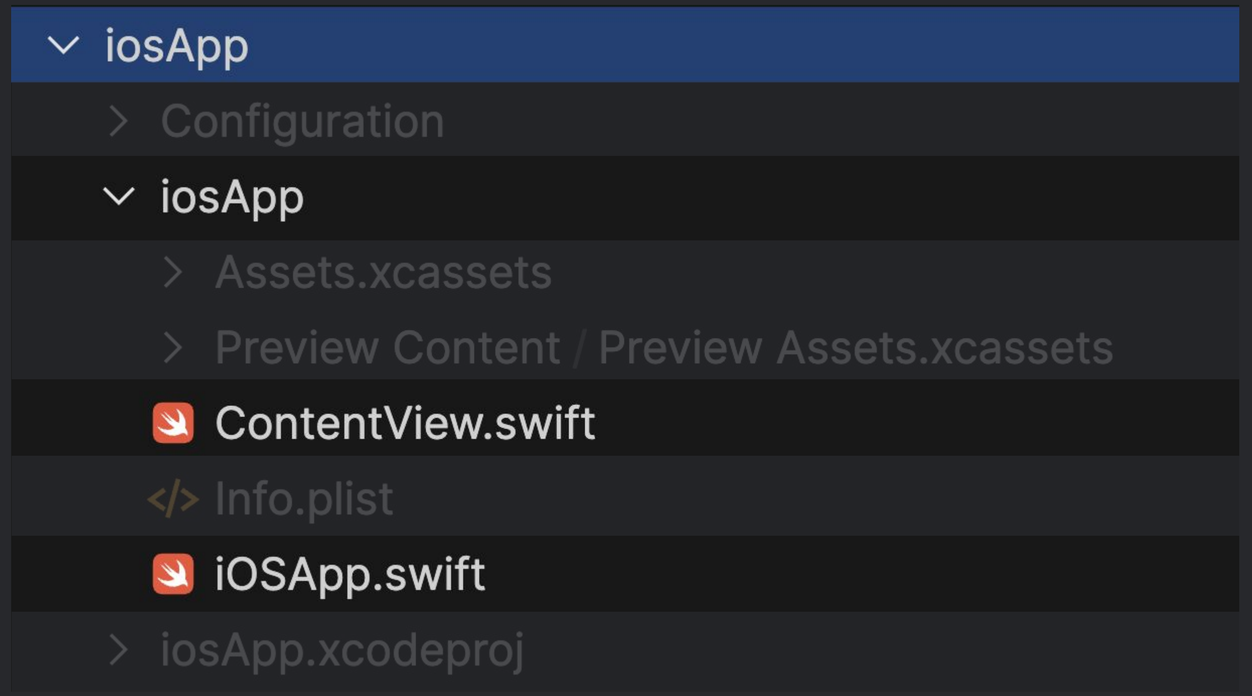
- shared
- composeApp



Project tree

Modules

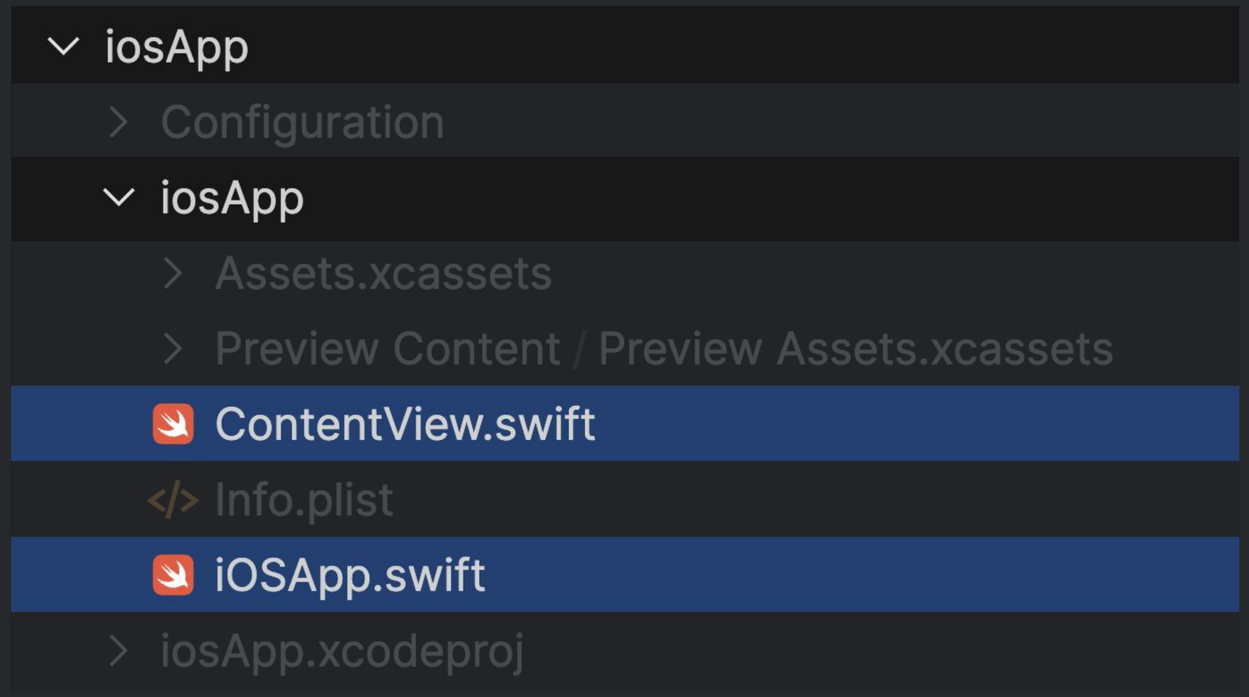
- shared
- composeApp
- iosApp



Project tree

Modules

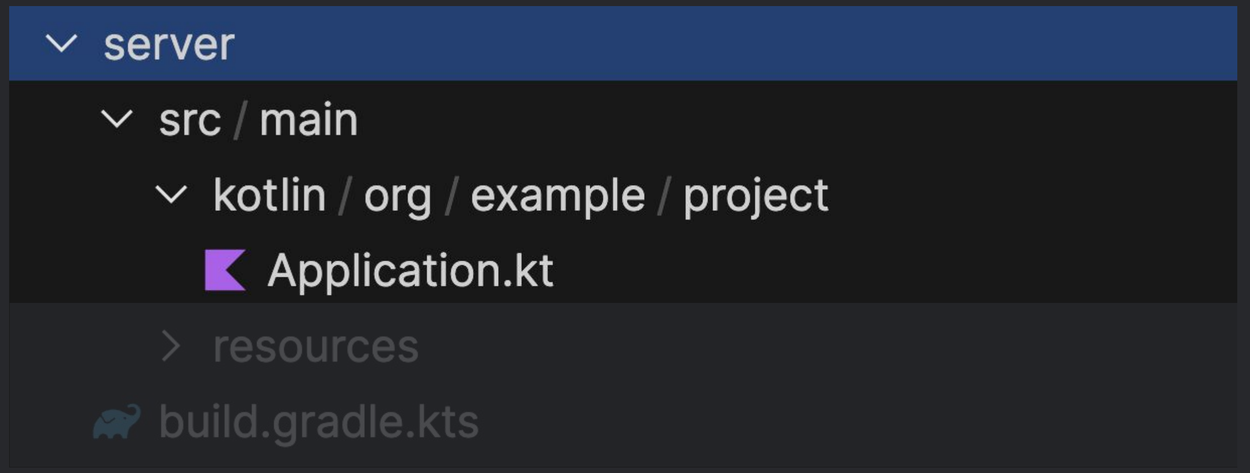
- shared
- composeApp
- iosApp - Xcode project folder



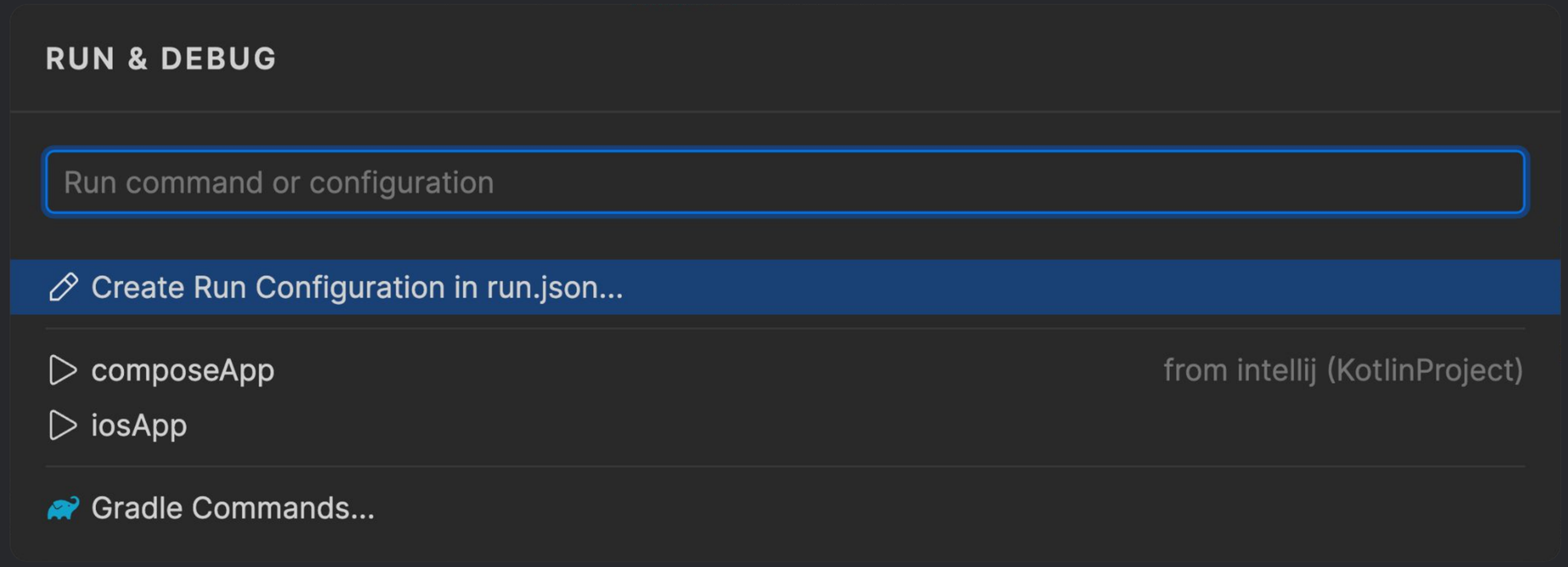
Project tree

Modules

- shared
- composeApp
- iosApp
- server



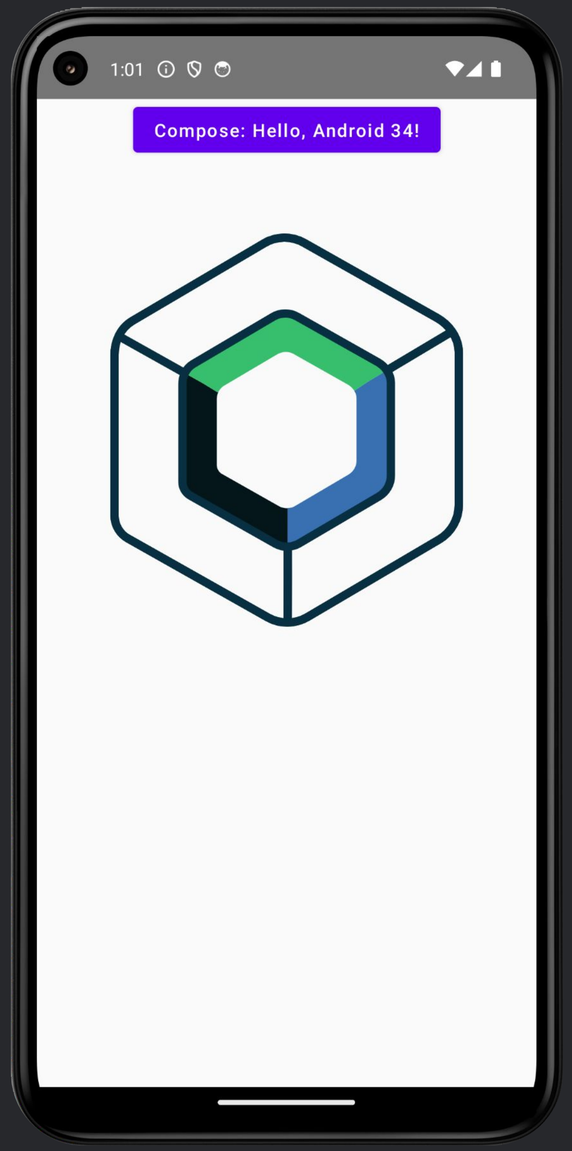
Run the App

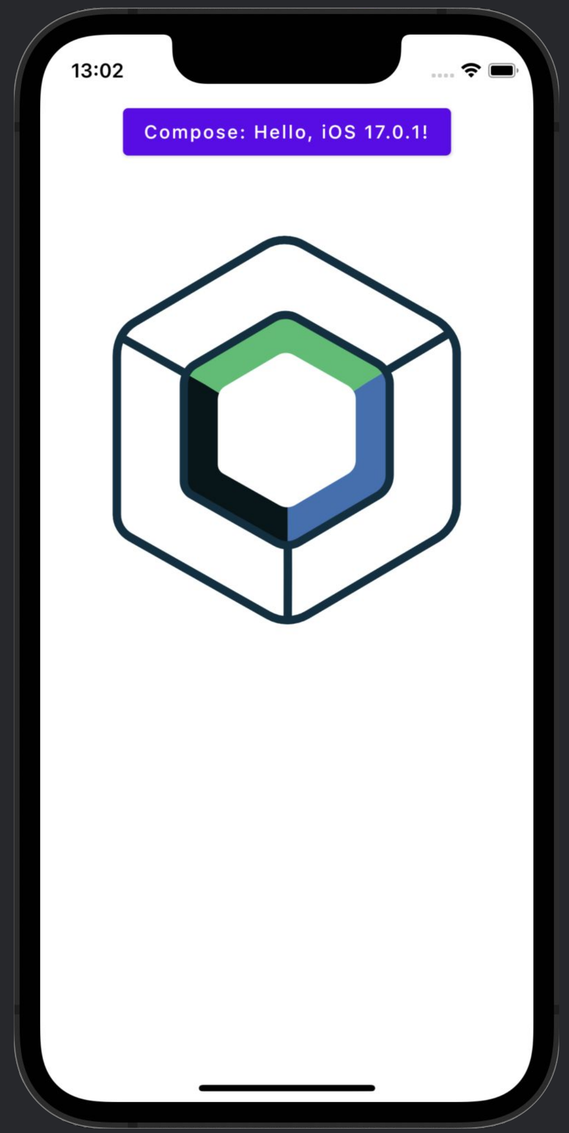

RUN & DEBUG
Run command or configuration
Create Run Configuration in run.json...
composeApp
from intellij (KotlinProject)
iosApp
Gradle Commands...

Run the App

Mobile

The App runs out of the box! *

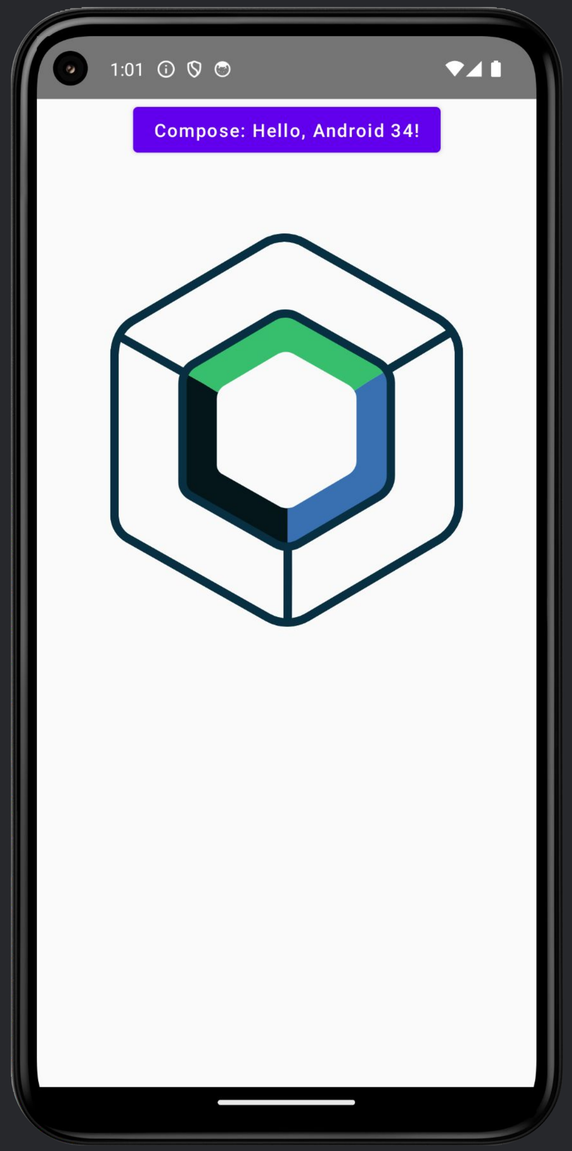


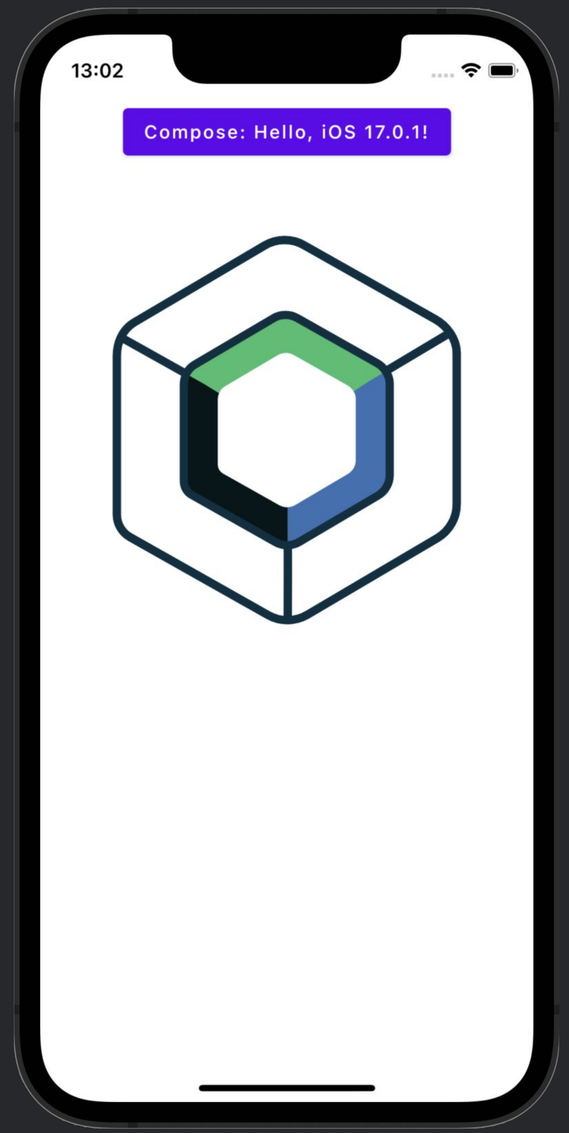


Run the App

Mobile

The App runs out of the box! *

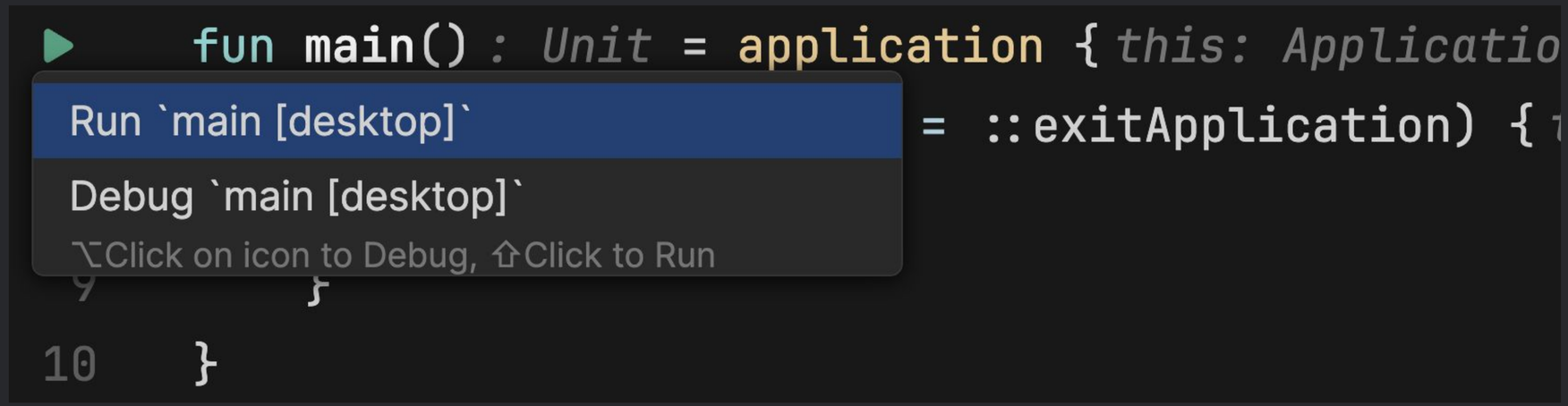




* if everything is correctly configured on your machine

Run the App

Desktop



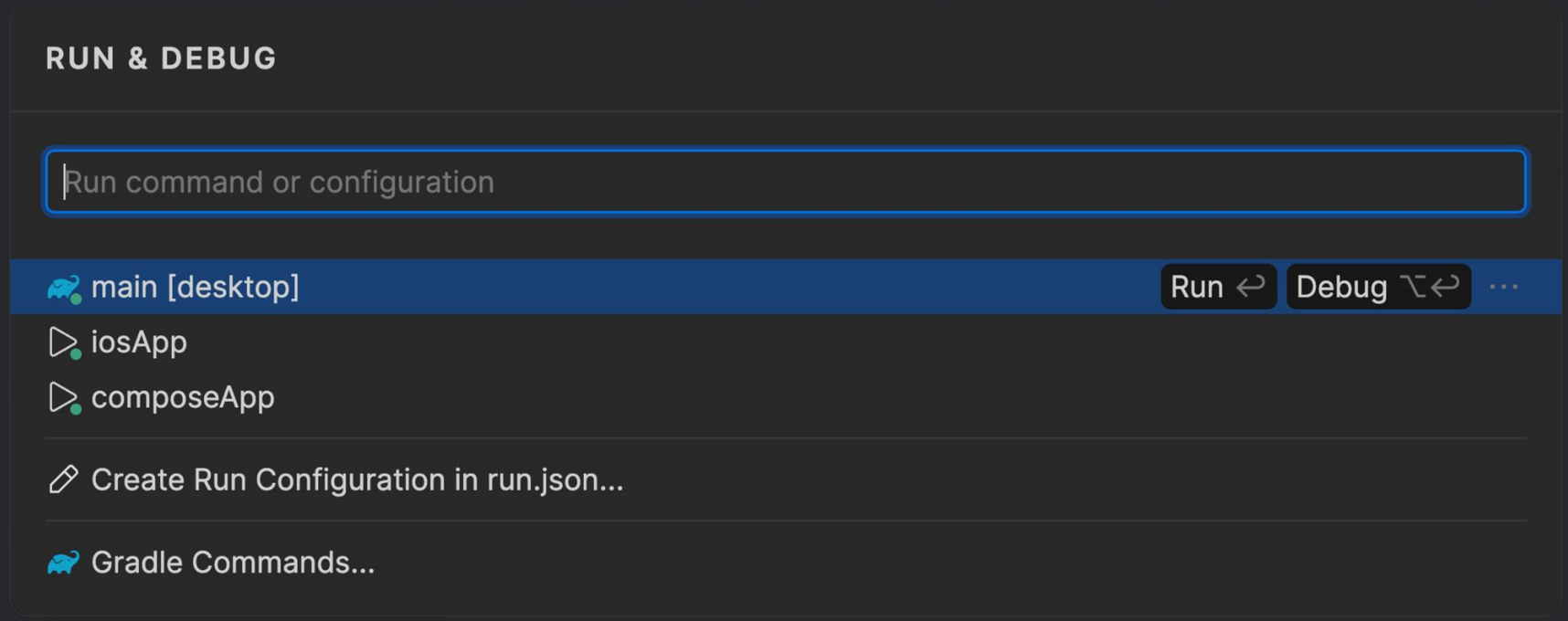
Run the App

Desktop

Again, the App runs out of the box!

Run the App

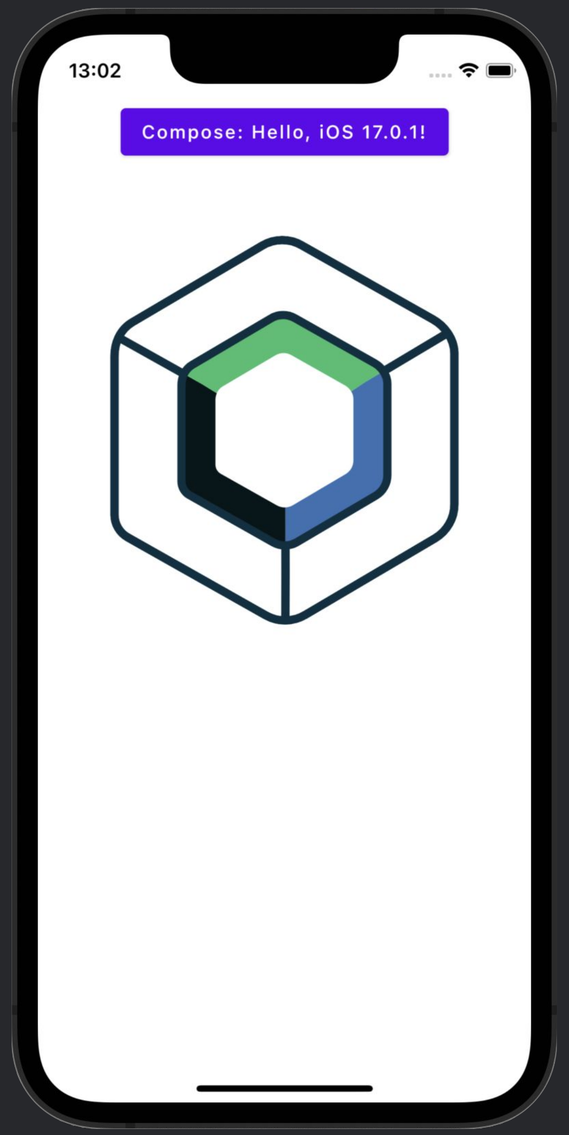
Desktop



Run the App

Caveats

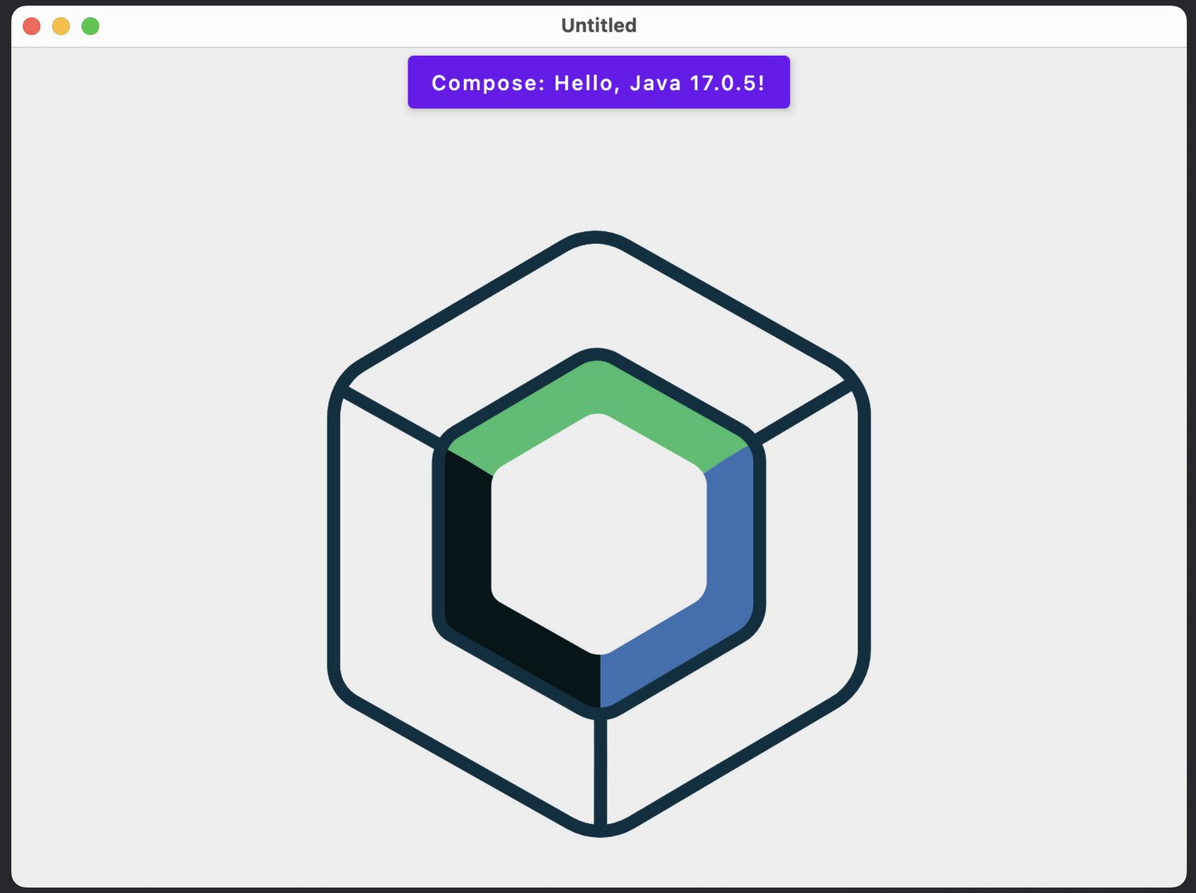
- iOS App worked at the first try! *



Run the App

Caveats

- iOS App worked at the first try! *
- Desktop - also first try!



Run the App

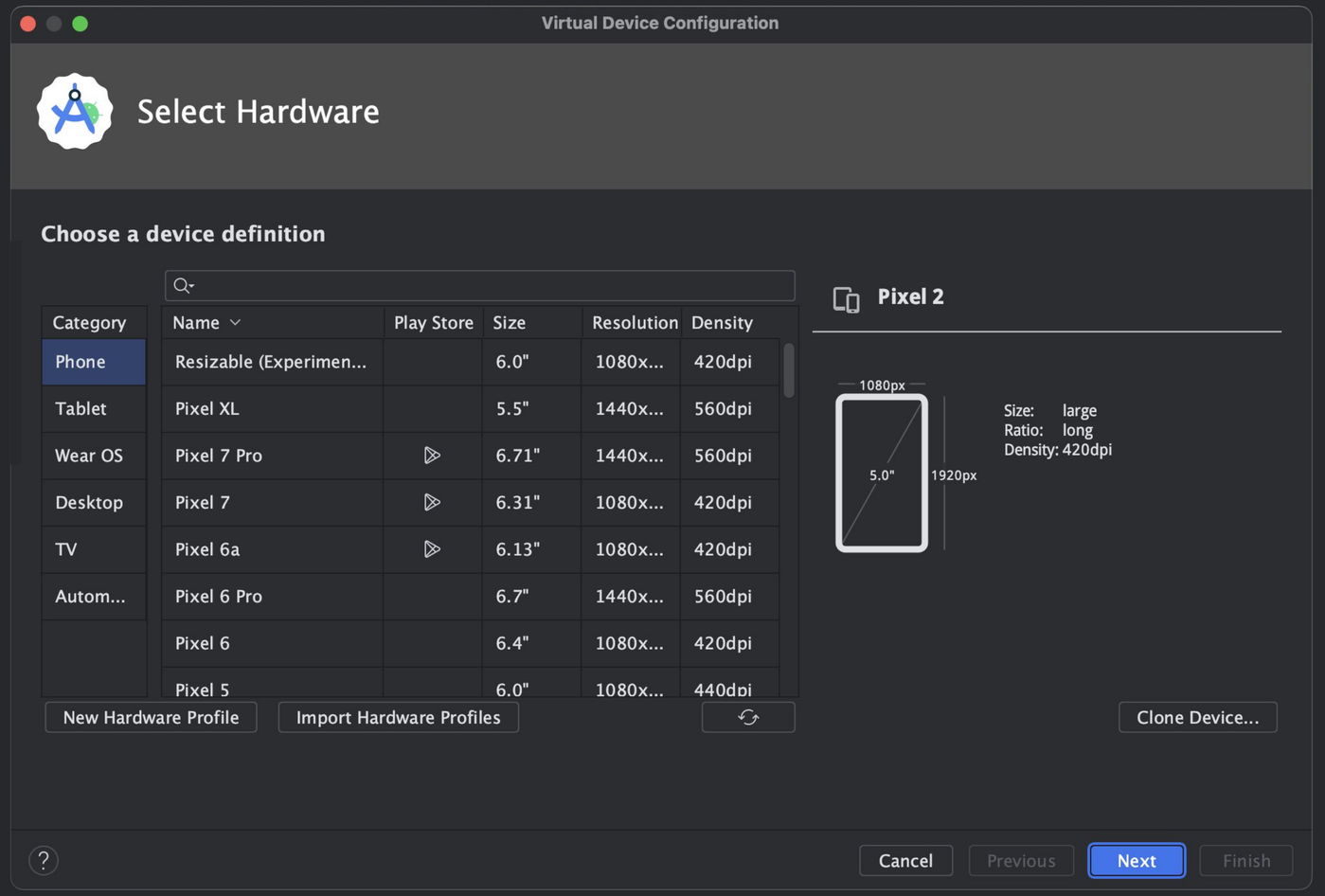
Caveats

- iOS App worked at the first try! *
- Desktop - also first try!
- Android - not so lucky

Run the App

Caveats

- iOS App worked at the first try! *
- Desktop - also first try!
- Android - not so lucky
  - Install emulator first



Run the App

Caveats

- iOS App worked at the first try! *
- Desktop - also first try!
- Android - on the second try!

Run the App

Caveats

- iOS App worked at the first try! *
- Desktop - also first try!
- Android - on the second try?..

Run the App

Caveats

- iOS App worked at the first try! *
- Desktop - also first try!
- Android - on the second try?..

```
w: Missing 'androidTarget()' Kotlin target in multiplatform project 'composeApp (:composeApp)'.
The Android Gradle plugin was applied without creating a corresponding 'android()' Kotlin Target:

```
plugins {
    id("com.android.application")
    kotlin("multiplatform")
}

kotlin {
    androidTarget() // <-- please register this Android target
}
```


FAILURE: Build completed with 2 failures.
```

Run the App

Caveats

- iOS App worked at the first try! *
- Desktop - also first try!
- Android
  - What went wrong?

Documentation:

- Install mobile emulators
- Verify with ***kdoctor***
- Restart Fleet

Run the App

Caveats

- iOS App worked at the first try! *
- Desktop - also first try!
- Android
  - What went wrong?

Documentation:

- Install mobile emulators
- Verify with ***kdoctor***
- Restart Fleet

Run the App

Caveats

- iOS App worked at the first try! *
- Desktop - also first try!
- Android
  - **Read the documentation!**

Run the App

Caveats

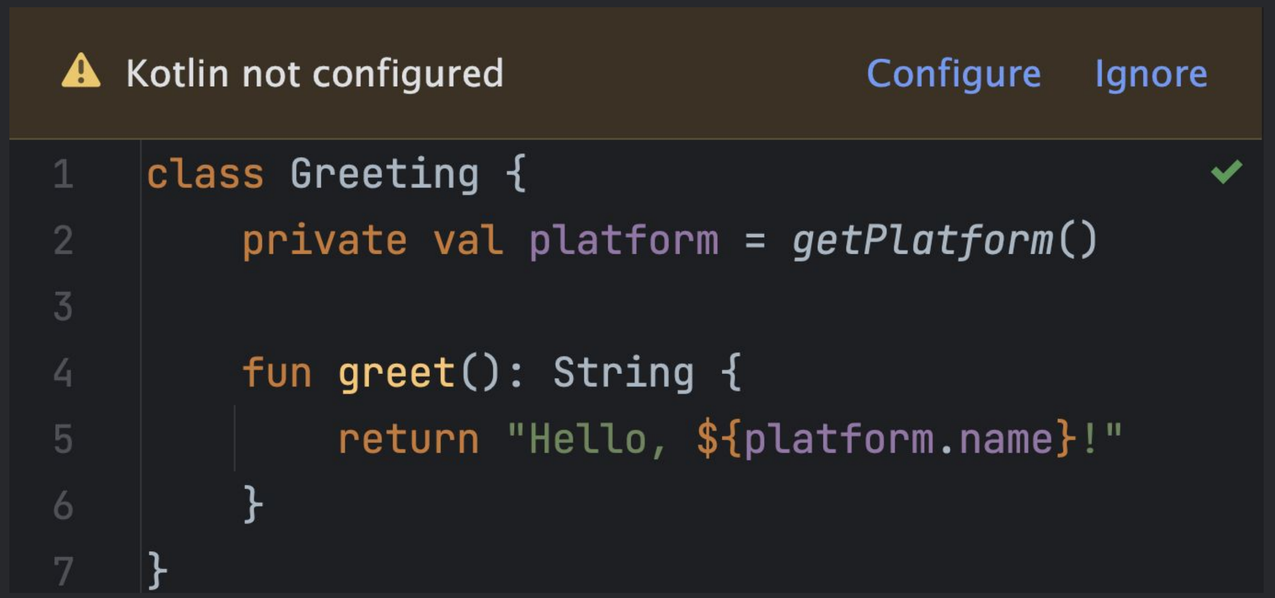
- iOS App worked at the first try! *
- Desktop - also first try!
- Android
  - What's with the error in the console?

Run the App

Caveats

- iOS App worked at the first try! *
- Desktop - also first try!
- Android
  - What's with the error in the console?

Android Studio

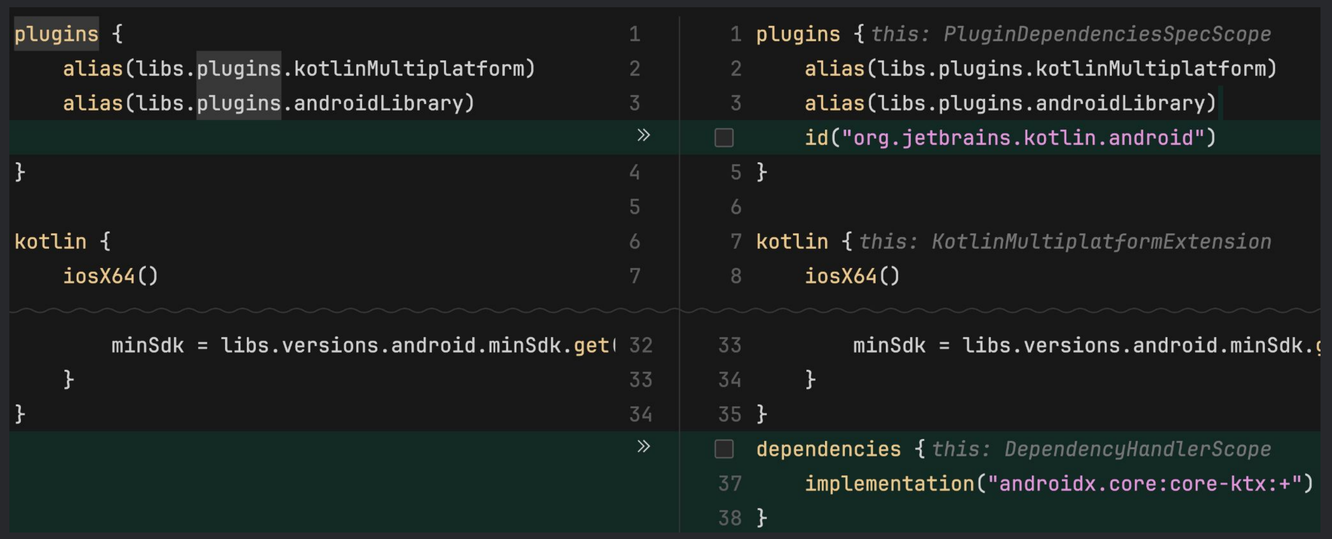


Run the App

Caveats

- iOS App worked at the first try! *
- Desktop - also first try!
- Android
  - What's with the error in the console?
  - Bad configuration

Git (Version Control System)



```
plugins {
    alias(libs.plugins.kotlinMultiplatform)
    alias(libs.plugins.androidLibrary)
}

kotlin {
    iosX64()
```

```
plugins { this: PluginDependenciesSpecScope
    alias(libs.plugins.kotlinMultiplatform)
    alias(libs.plugins.androidLibrary)
    id("org.jetbrains.kotlin.android")
}

kotlin { this: KotlinMultiplatformExtension
    iosX64()
```

```
        minSdk = libs.versions.android.minSdk.get
    }
}
```

```
        minSdk = libs.versions.android.minSdk.
    }
}
dependencies { this: DependencyHandlerScope
    implementation("androidx.core:core-ktx:+")
}
```

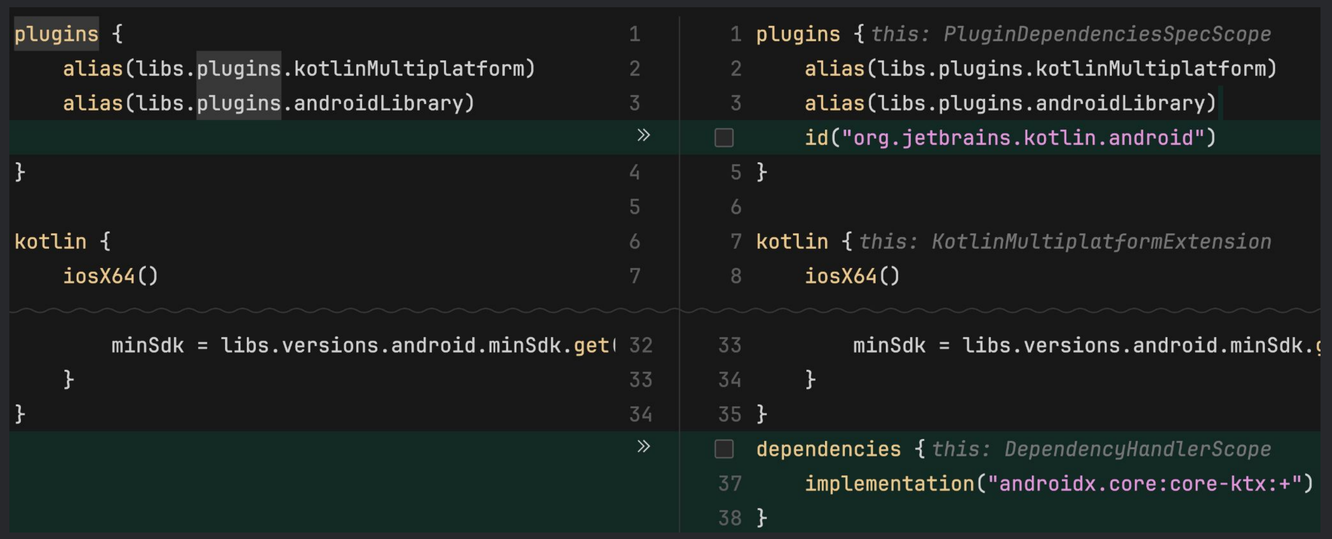
build.gradle.kts

Run the App

Caveats

- iOS App worked at the first try! *
- Desktop - also first try!
- Android
  - What's with the error in the console?
  - Bad configuration
  - Revert changes

Git (Version Control System)

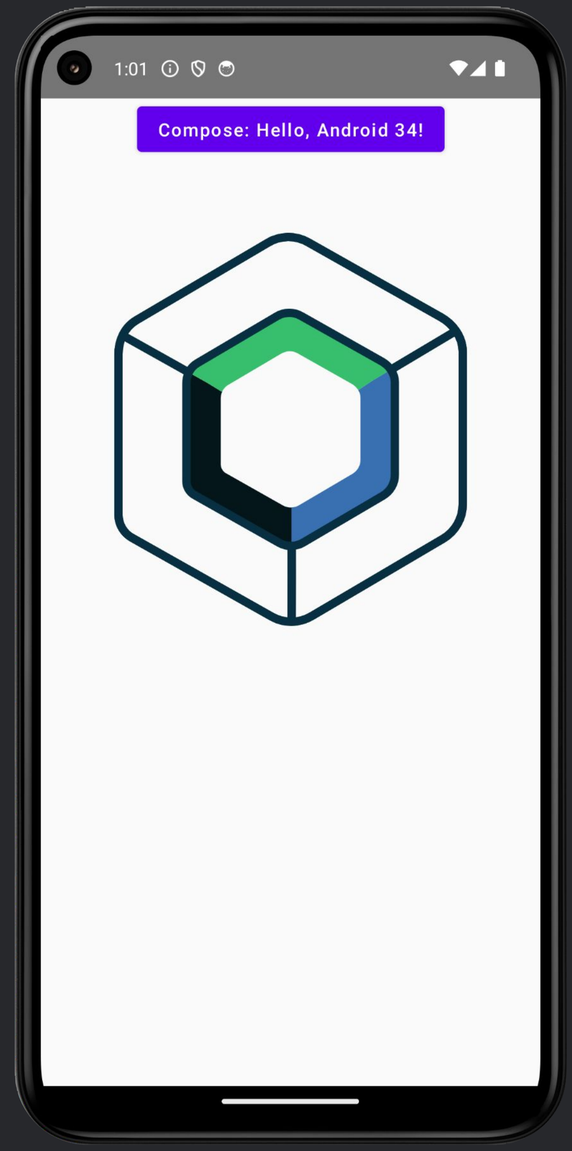


build.gradle.kts

Run the App

Caveats

- iOS App worked at the first try! *
- Desktop - also first try!
- Android - third time's a charm!



Run the App

Caveats

- iOS App worked at the first try! *
- Desktop - also first try!
- Android - third time's a charm!

Run the App

Caveats

- iOS App worked at the first try! *
  - * Xcode can also break the config
- Desktop - also first try!
- Android - third time's a charm!

Run the App

Caveats

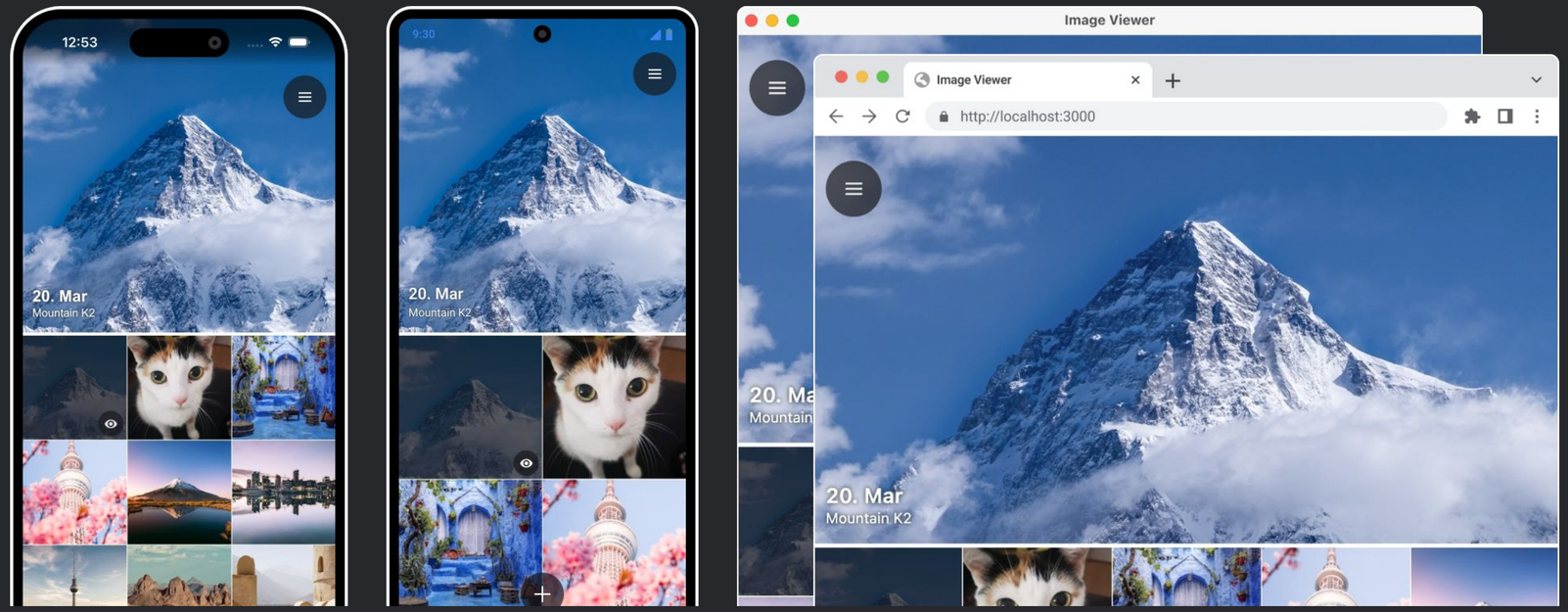
- iOS App worked at the first try! *
  - * Xcode can also break the config
  - Always use VCS!
- Desktop - also first try!
- Android - third time's a charm!

Coding!

Compose basics 

Compose basics

- Multiplatform UI framework



Compose basics 

- Multiplatform UI framework
- Initially developed by Google as Jetpack Compose for Android

Compose basics 

- Multiplatform UI framework
- Initially developed by Google as Jetpack Compose for Android
- Compose Multiplatform is based on Jetpack Compose

Compose basics 

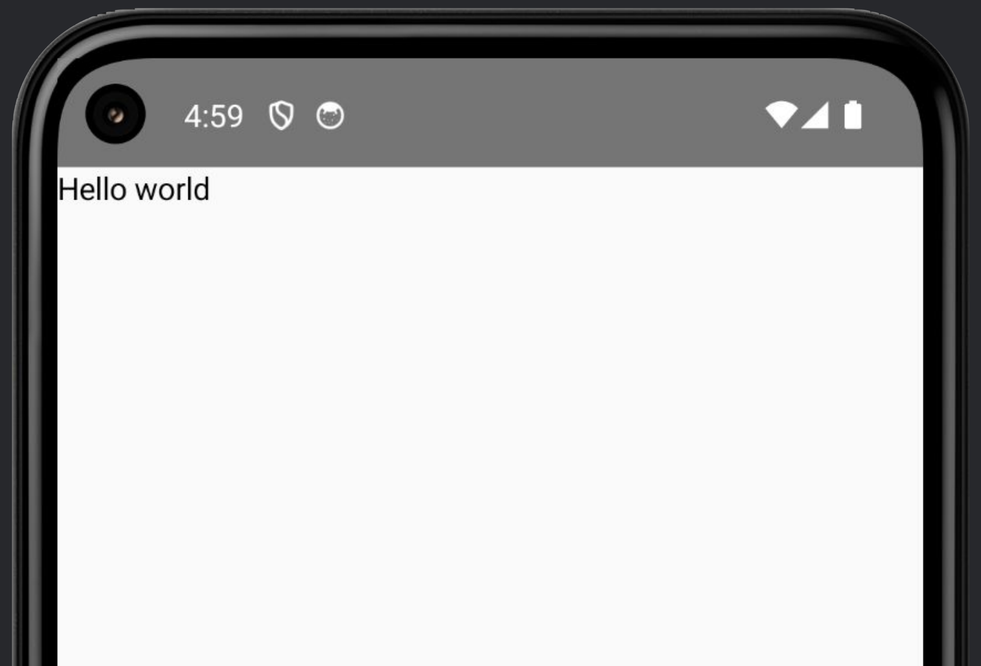
Structure

Compose basics

Structure

- Composables - building blocks

```
@Composable
fun App() {
    Text("Hello World")
}
```



Compose basics 

Structure

- Composables - building blocks

```
@Composable
fun App() {
    HelloWorld()
}

@Composable
fun HelloWorld() {
    Text("Hello World")
}
```

Compose basics 

Structure

- Composables - building blocks
  - Composables may only be called from a Composable

```
fun App() {
    HelloWorld()
}

@Composable
fun HelloWorld() {
    Text("Hello World")
}
```

Functions which invoke @Composable functions must be marked with the @Composable annotation

```
public fun App(): Unit
```

App.kt

KotlinProject.composeApp.commonMain

Compose basics 

Structure

- Composables - building blocks
  - Composables may only be called from a Composable
  - Lambdas can be Composable

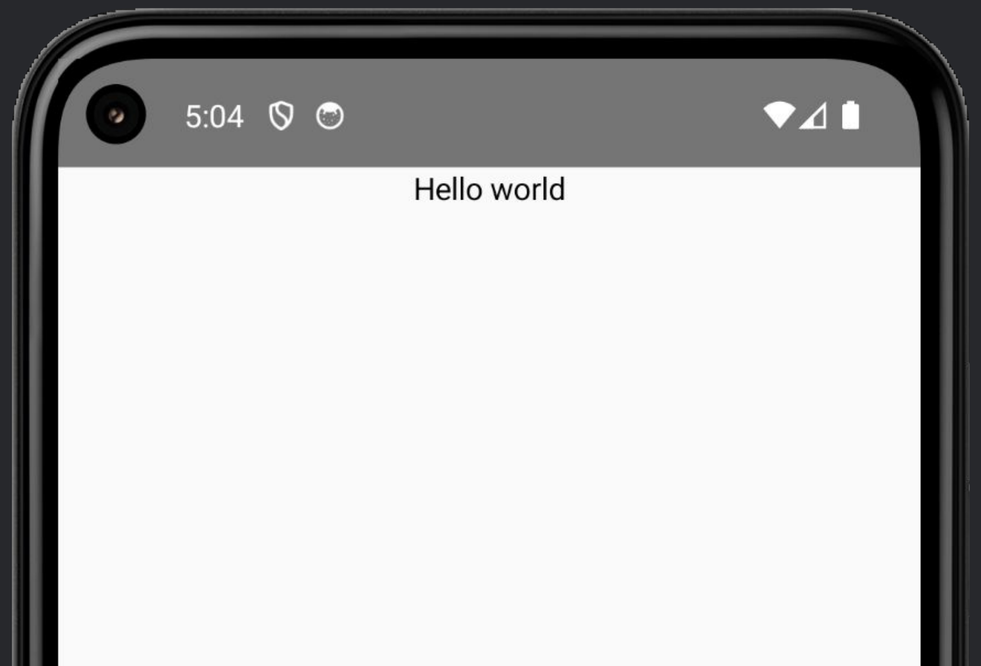
```
@Composable
fun App() {
    CenterHorizontally {
        Text("Hello world")
    }
}

@Composable
fun CenterHorizontally(
    content: @Composable () → Unit
) {
    Row(
        Modifier.fillMaxWidth(),
        Arrangement.Center
    ){
        content()
    }
}
```

Compose basics

Structure

- Composables - building blocks
  - Composables may only be called from a Composable
  - Lambdas can be Composable

```
@Composable
fun App() {
    CenterHorizontally {
        Text("Hello world")
    }
}
```



Compose basics 

Structure

- Composables - building blocks
- Managing state

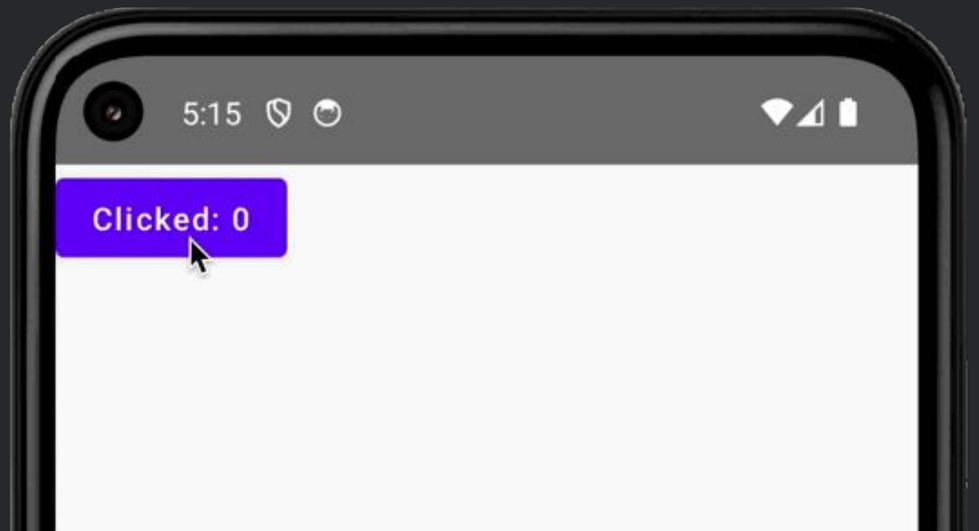
```
@Composable
fun App() {
    var counter = 0
    Button({
        counter++
    }) {
        Text("Clicked: $counter")
    }
}
```

Compose basics

Structure

- Composables - building blocks
- Managing state

```
@Composable
fun App() {
    var counter = 0
    Button({
        counter++
    }) {
        Text("Clicked: $counter")
    }
}
```



Compose basics 

Structure

- Composables - building blocks
- Managing state
  - Use remember function

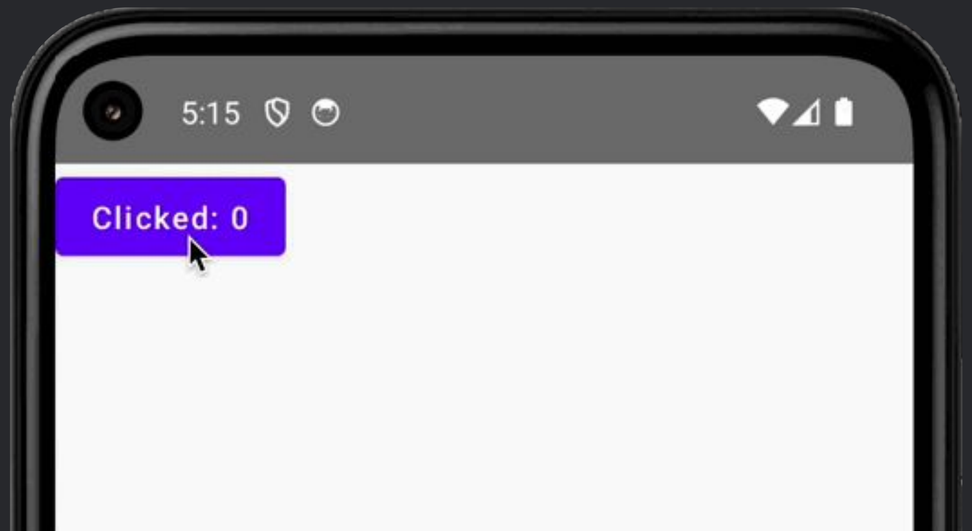
```
@Composable
fun App() {
    var counter = remember { 0 }
    Button({
        counter++
    }) {
        Text("Clicked: $counter")
    }
}
```

Compose basics

Structure

- Composables - building blocks
- Managing state
  - Use remember function

```
@Composable
fun App() {
    var counter = remember { 0 }
    Button({
        counter++
    }) {
        Text("Clicked: $counter")
    }
}
```



Compose basics 

Structure

- Composables - building blocks
- Managing state
  - Use remember function
  - Use mutableStateOf

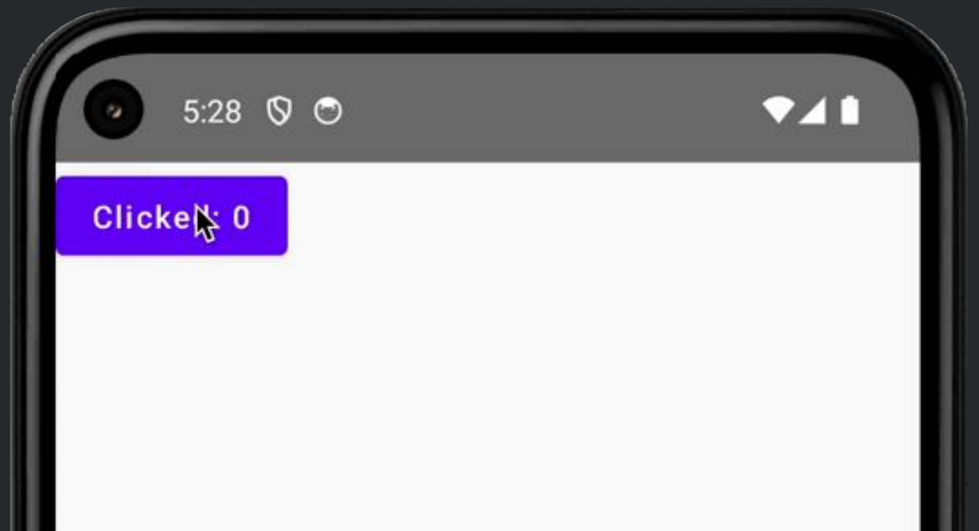
```
@Composable
fun App() {
    val counter = remember { mutableStateOf(0) }
    Button({
        counter.value++
    }) {
        Text("Clicked: ${counter.value}")
    }
}
```

Compose basics

Structure

- Composables - building blocks
- Managing state
  - Use remember function
  - Use mutableStateOf

```
@Composable
fun App() {
    val counter = remember { mutableStateOf(0) }
    Button({
        counter.value++
    }) {
        Text("Clicked: ${counter.value}")
    }
}
```

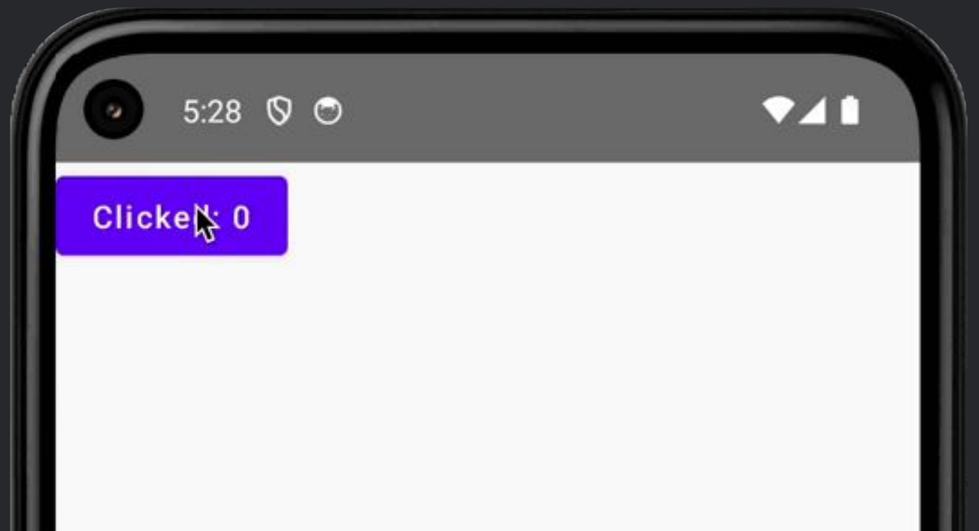


Compose basics

Structure

- Composables - building blocks
- Managing state
  - Use remember function
  - Use mutableStateOf
  - You can use delegates with state

```
@Composable
fun App() {
    var counter by remember {mutableStateOf(0)}
    Button({
        counter++
    }) {
        Text("Clicked: $counter")
    }
}
```



App UI

App UI

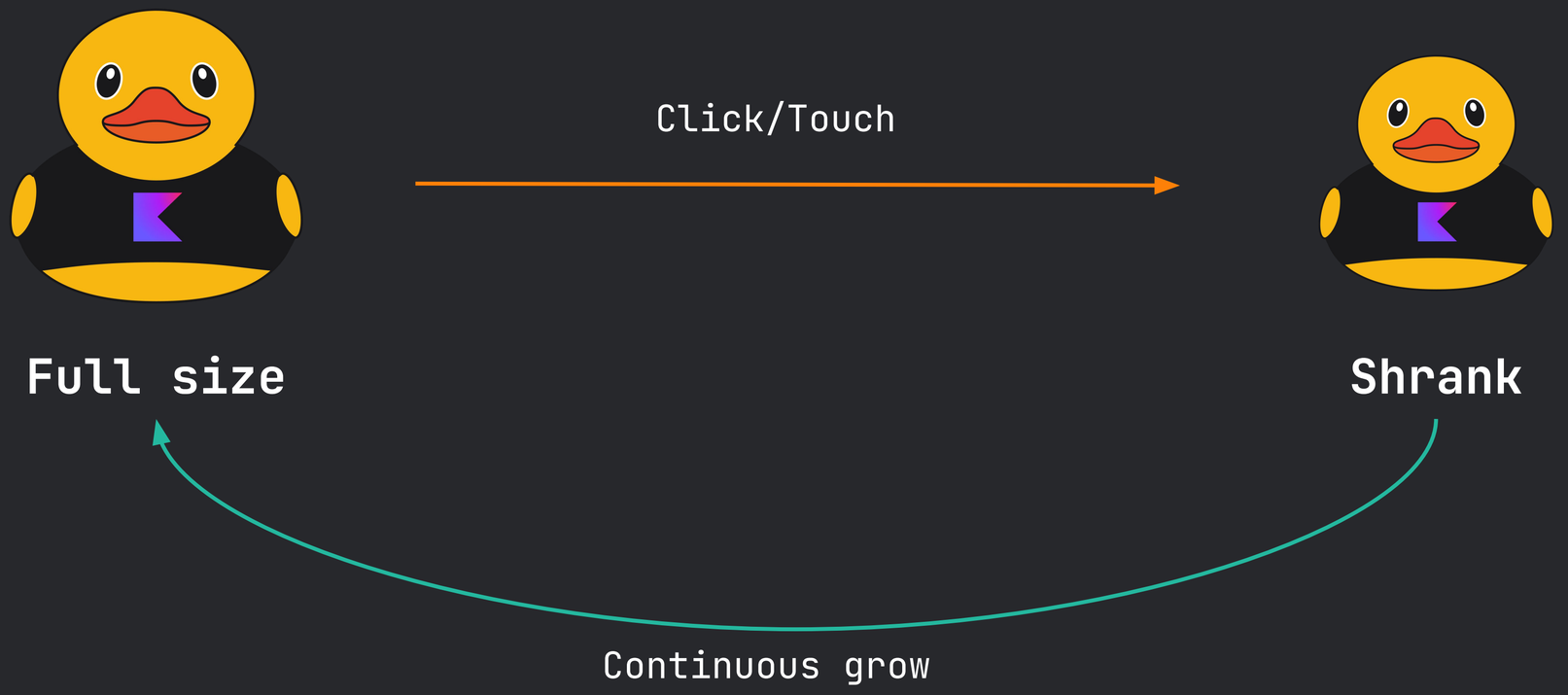
Clickable Duck

Full size

Shrank

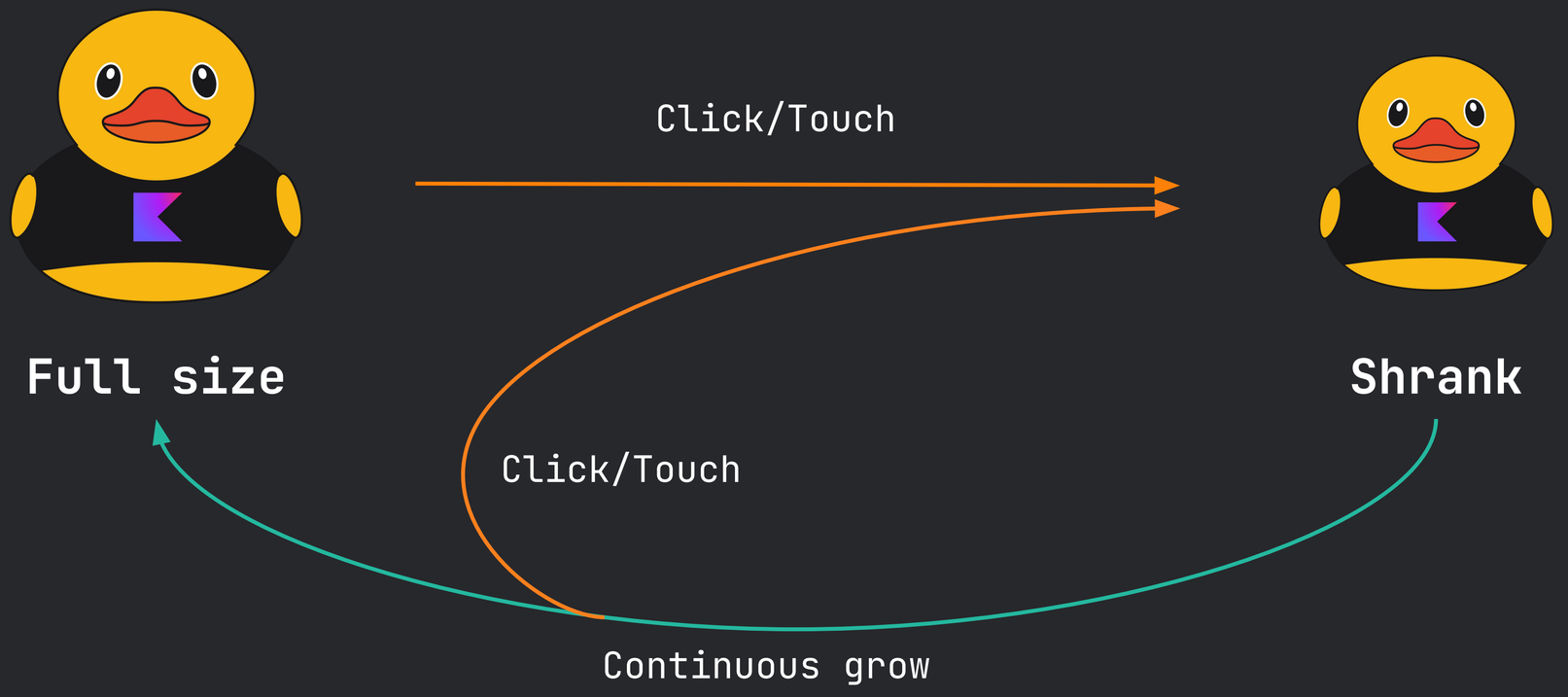
App UI

Clickable Duck



App UI

Clickable Duck



App UI

Clickable Duck

- Duck State

```
enum class ImageAnimationState(val targetSize: Float) {
    Shrank(330f), Full(360f)
}

// in Composable
var imageState by remember {
    mutableStateOf(ImageAnimationState.Full)
}
```

App UI

Clickable Duck

- Duck State
- Animatable

```
var imageState by remember {
    mutableStateOf(ImageAnimationState.Full)
}

val imageSize = remember {
    Animatable(imageState.targetSize)
}
```

App UI

Clickable Duck

- Duck State
- Animatable

```
var imageState by remember {
    mutableStateOf(ImageAnimationState.Full)
}

val imageSize = remember {
    Animatable(imageState.targetSize)
}

Image(
    painterResource(duck.resource()),
    contentDescription = "Duck",
    modifier = Modifier
        .size(imageSize.value.dp)
        .clickable {
            imageState = ImageAnimationState.Shrank
        }
)
```

App UI

Clickable Duck

- Duck State
- Animatable

```
var imageState by remember {
    mutableStateOf(ImageAnimationState.Full)
}

val imageSize = remember {
    Animatable(imageState.targetSize)
}

Image(
    painterResource(duck.resource()),
    contentDescription = "Duck",
    modifier = Modifier
        .size(imageSize.value.dp)
        .clickable {
            imageState = ImageAnimationState.Shrank
        }
)
```

App UI

Clickable Duck

- Duck State
- Animatable

```
var imageState by remember {
    mutableStateOf(ImageAnimationState.Full)
}

val imageSize = remember {
    Animatable(imageState.targetSize)
}

Image(
    painterResource(duck.resource()),
    contentDescription = "Duck",
    modifier = Modifier
        .size(imageSize.value.dp)
        .clickable {
            imageState = ImageAnimationState.Shrank
        }
)
```

App UI

Clickable Duck

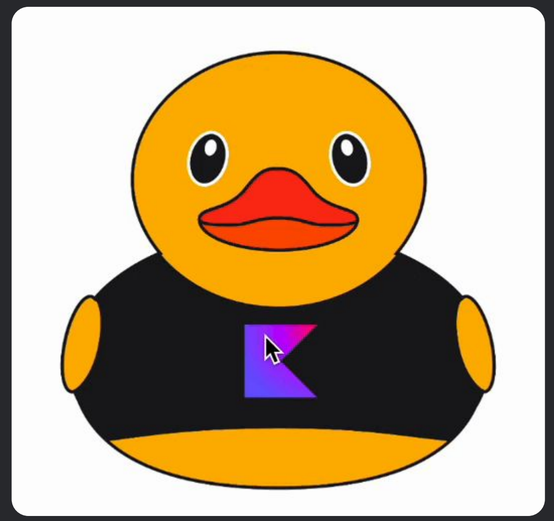
- Duck State
- Animatable
- LaunchedEffect

```
var imageState by remember {
    mutableStateOf(ImageAnimationState.Full)
}

val imageSize = remember {
    Animatable(imageState.targetSize)
}

LaunchedEffect(imageState) {
    imageSize.animateTo(
        targetValue = imageState.targetSize,
        animationSpec = tween(durationMillis = 70),
    ) {
        if (value == targetValue) {
            imageState = ImageAnimationState.Full
        }
    }
}
```

App UI

Clickable Duck

- Duck State
- Animatable
- LaunchedEffect



```
var imageState by remember {
    mutableStateOf(ImageAnimationState.Full)
}

val imageSize = remember {
    Animatable(imageState.targetSize)
}

LaunchedEffect(imageState) {
    imageSize.animateTo(
        targetValue = imageState.targetSize,
        animationSpec = tween(durationMillis = 70),
    ) {
        if (value == targetValue) {
            imageState = ImageAnimationState.Full
        }
    }
}
```

App UI

Click counter

App UI

Click counter

- Text formatting

```
@Composable
fun Counter(model: Model) {
    val value: Int by remember { model.counterValue }

    Text(
        text = "$value",
        fontStyle = FontStyle.Normal,
        fontWeight = FontWeight.Bold,
        fontFamily = FontFamily.Monospace,
        color = Color.Black,
        fontSize = 30.sp,
    )
}
```

App UI

Click counter

- Text formatting
- Model

```
@Composable
fun Counter(model: Model) {
    val value: Int by remember { model.counterValue }

    Text(
        text = "$value",
        fontStyle = FontStyle.Normal,
        fontWeight = FontWeight.Bold,
        fontFamily = FontFamily.Monospace,
        color = Color.Black,
        fontSize = 30.sp,
    )
}
```

App UI

Click counter

- Text formatting
- Model
  - Share state between Composables

```
@Composable
fun Counter(model: Model) {
    val value: Int by remember { model.counterValue }
    Text(...)
}

// Model.kt
class Model {
    val counterValue = mutableStateOf(0)
    val selectedDuck = mutableStateOf(DuckType.T_SHIRT)
}
```

App UI

Click counter

- Text formatting
- Model
  - Share state between Composables
  - Display in one Composable, update in another

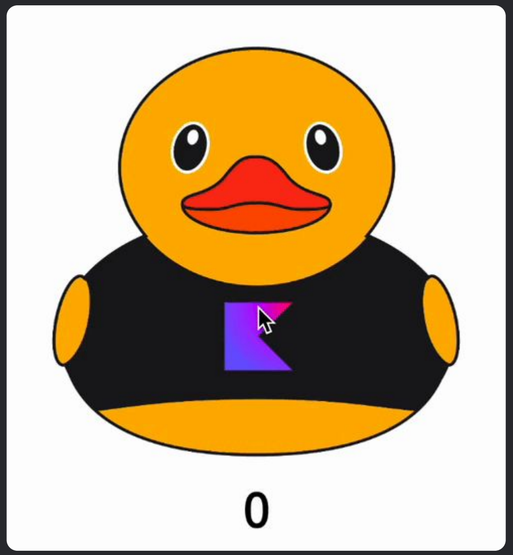
```
@Composable
fun Counter(model: Model) {
    val value: Int by remember { model.counterValue }
    Text(...)
}

// Model.kt
class Model {
    val counterValue = mutableStateOf(0)
    val selectedDuck = mutableStateOf(DuckType.T_SHIRT)
}

// ClickableDuck.kt Composable
Image(
    ...
    modifier = Modifier
        .clickable {
            ...
            model.counterValue.value++
        }
)
```

App UI

Click counter

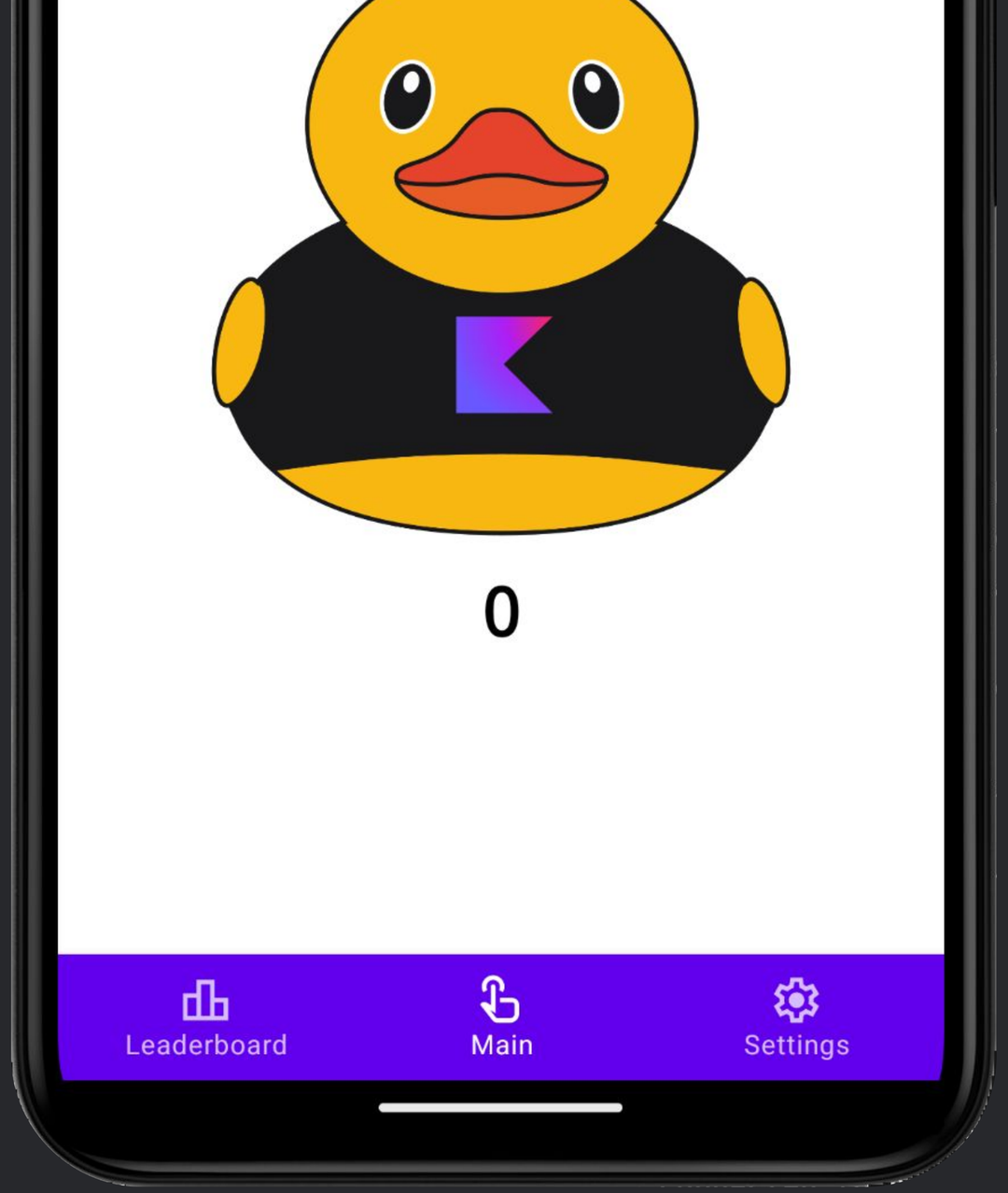
- Text formatting
- Model
  - Share state between Composables
  - Display in one Composable, update in another

```
@Composable
fun Counter(model: Model) {
    val value: Int by remember { model.counterValue }
    Text(...)
}
```



App UI

Bottom Navigation



App UI

Bottom Navigation

- BottomNavigation Composable

```
BottomNavigation {

}
```

App UI

Bottom Navigation

- BottomNavigation Composable
  - BottomNavigationItem as a building block

```
BottomNavigation {
    // selected, onCLick, icon - parameters
    BottomNavigationItem(false, {}, { Text("1") })
    BottomNavigationItem(true,  {}, { Text("2") })
    BottomNavigationItem(false, {}, { Text("3") })
}
```

App UI

Bottom Navigation

- BottomNavigation Composable
  - BottomNavigationItem as a building block

App UI

Bottom Navigation

- BottomNavigation Composable
  - BottomNavigationItem as a building block

Not exactly bottom

App UI

Bottom Navigation

- BottomNavigation Composable
  - BottomNavigationItem as a building block

Not exactly bottom

App UI

Bottom Navigation

- BottomNavigation Composable
  - BottomNavigationItem as a building block

```
@Composable
fun NavBar() {
    BottomNavigation {
        // selected, onCLick, icon - parameters
        BottomNavigationItem(false, {}, { Text("1") })
        BottomNavigationItem(true, {}, { Text("2") })
        BottomNavigationItem(false, {}, { Text("3") })
    }
}
```

App UI

Bottom Navigation

- BottomNavigation Composable
  - BottomNavigationItem as a building block

```
@Composable
fun NavBar() { ... }

// App.kt
@Composable
fun App() {
    AppBody(model) // ClickableDuck and Counter

    NavBar()
}
```

App UI

Bottom Navigation

- BottomNavigation Composable
  - BottomNavigationItem as a building block

```
@Composable
fun NavBar() { ... }

// App.kt
@Composable
fun App() {
    AppBody(model) // ClickableDuck and Counter

    NavBar()
}

// BottomNavigation.kt documentation

// TODO: b/149825331 add documentation references to
//   Scaffold here and samples for using
//   BottomNavigation inside a Scaffold
@Composable
fun BottomNavigation( ... )
```

App UI

Bottom Navigation

- BottomNavigation Composable
  - BottomNavigationItem as a building block
- Scaffold
  - Implements the basic visual layout structure

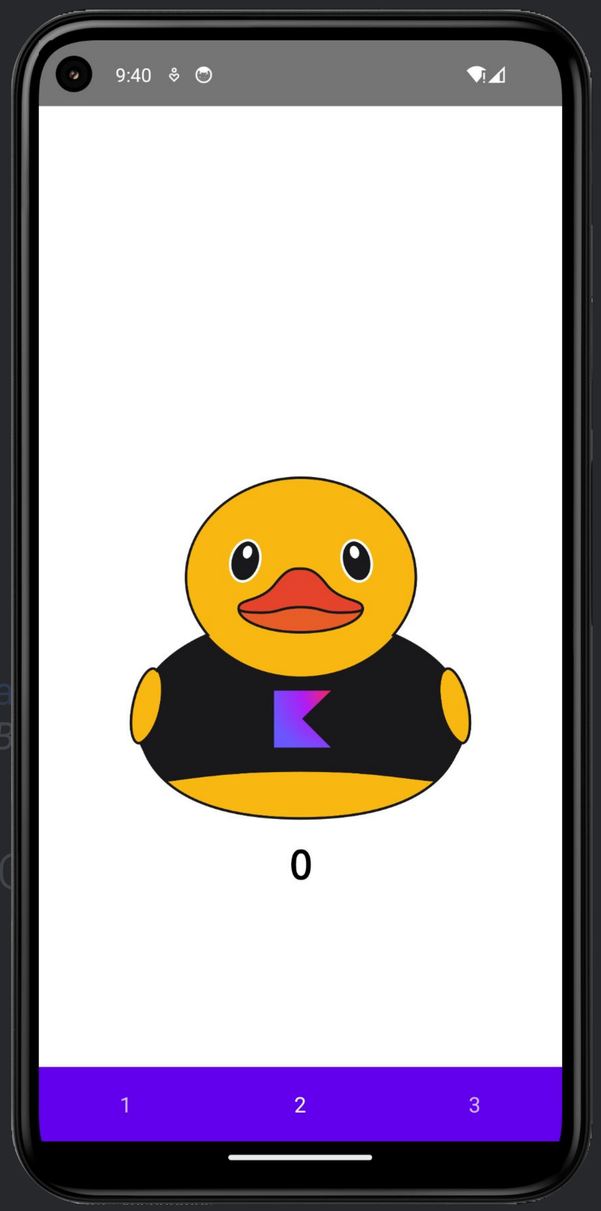
```
@Composable
fun NavBar() { ... }

// App.kt
@Composable
fun App() {
    Scaffold(
        bottomBar = {
            NavBar()
        }
    ) {
        AppBody(model)
    }
}
```

App UI

Bottom Navigation

- BottomNavigation Composable
  - BottomNavigationItem as a building block
- Scaffold
  - Implements the basic visual layout structure

```
@Composable
fun NavBar() {

// App.kt
@Composable
fun App() {
    Scaffold(
        bottomBa
            NavB
        }
    ) {
        AppBody(
    }
}
```

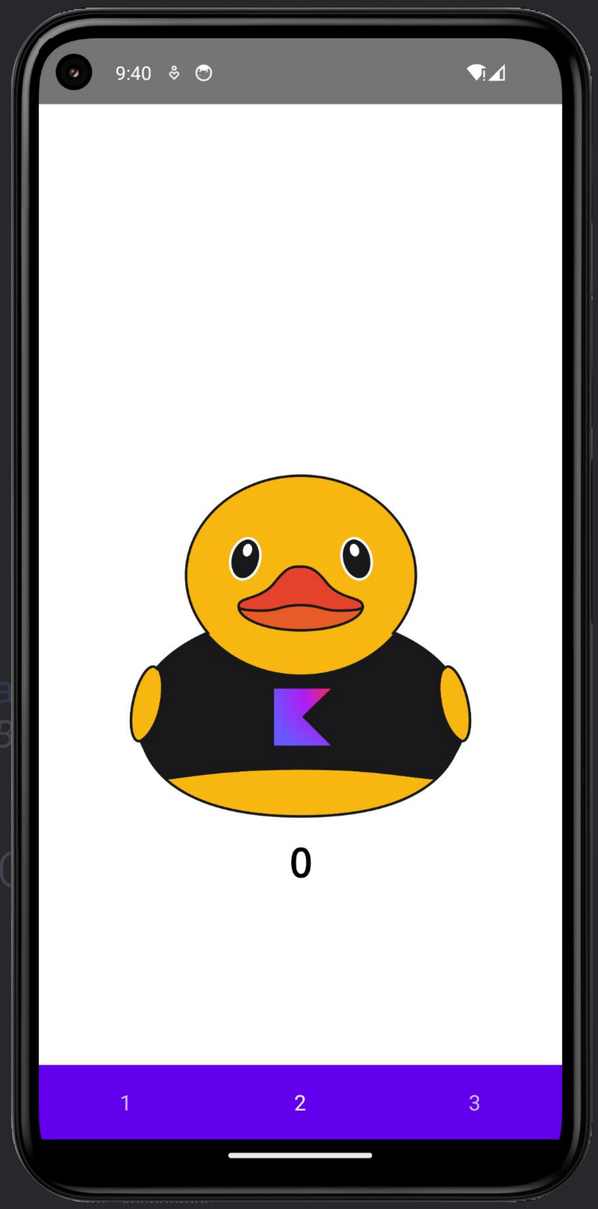


App UI

Bottom Navigation

- BottomNavigation Composable
  - BottomNavigationItem as a building block
- Scaffold
  - Implements the basic visual layout structure
- Compose Multiplatform inherits from Jetpack Compose

```
@Composable
fun NavBar() {

// App.kt
@Composable
fun App() {
    Scaffold(
        bottomBa
            NavB
        }
    ) {
        AppBody(
    }
}
```

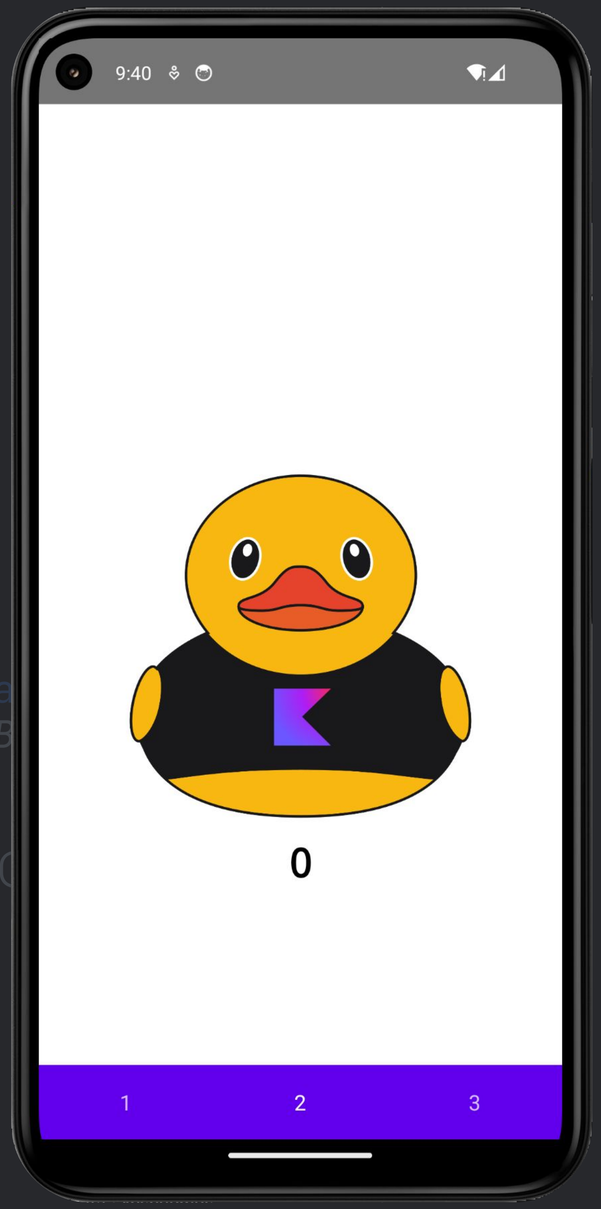


App UI

Bottom Navigation

- BottomNavigation Composable
- Scaffold
- Compose Multiplatform inherits from Jetpack Compose

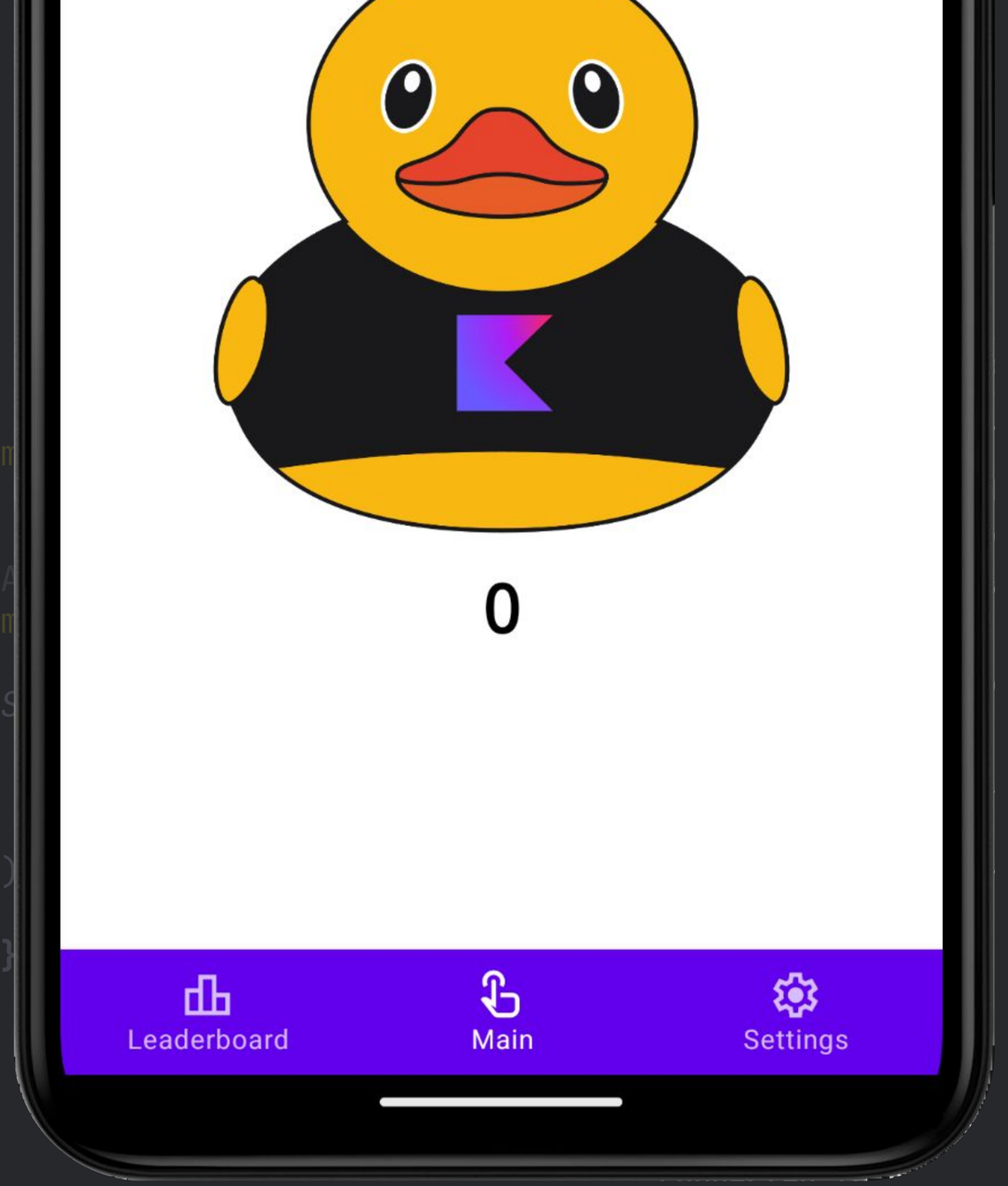
```
@Composable
fun NavBar() {

// App.kt
@Composable
fun App() {
    Scaffold(
        bottomBa
            NavB
        }
    ) {
        AppBody(
    }
}
```



App UI

Bottom Navigation

- BottomNavigation Composable
- Scaffold
- Compose Multiplatform inherits from Jetpack Compose
- Icons and text



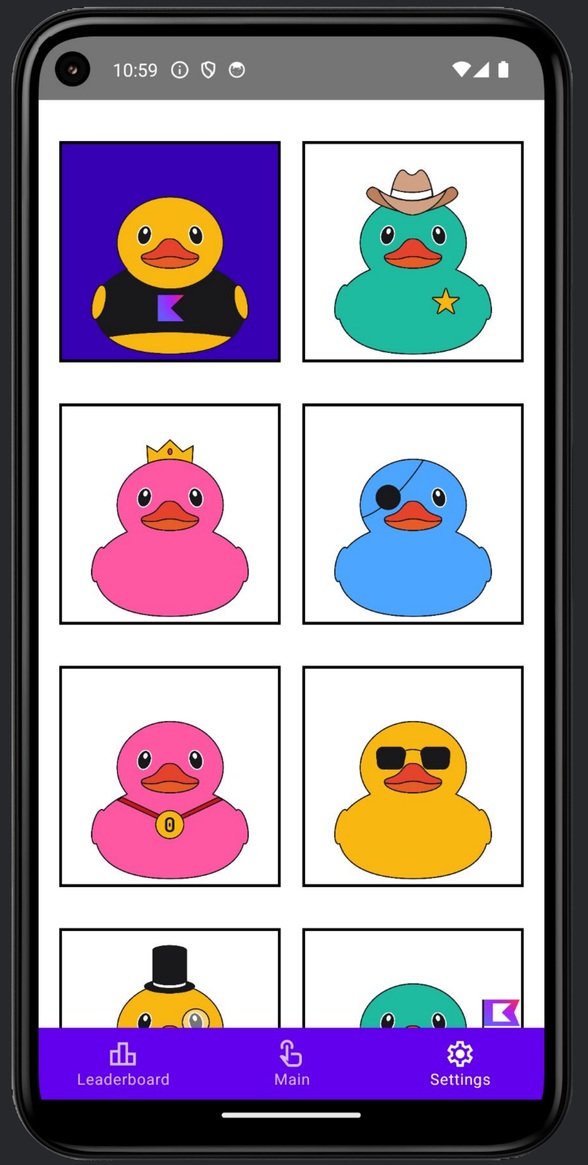
App UI

Settings Page

App UI

Settings Page

- Two column scrollable list of Ducks



App UI

Settings Page

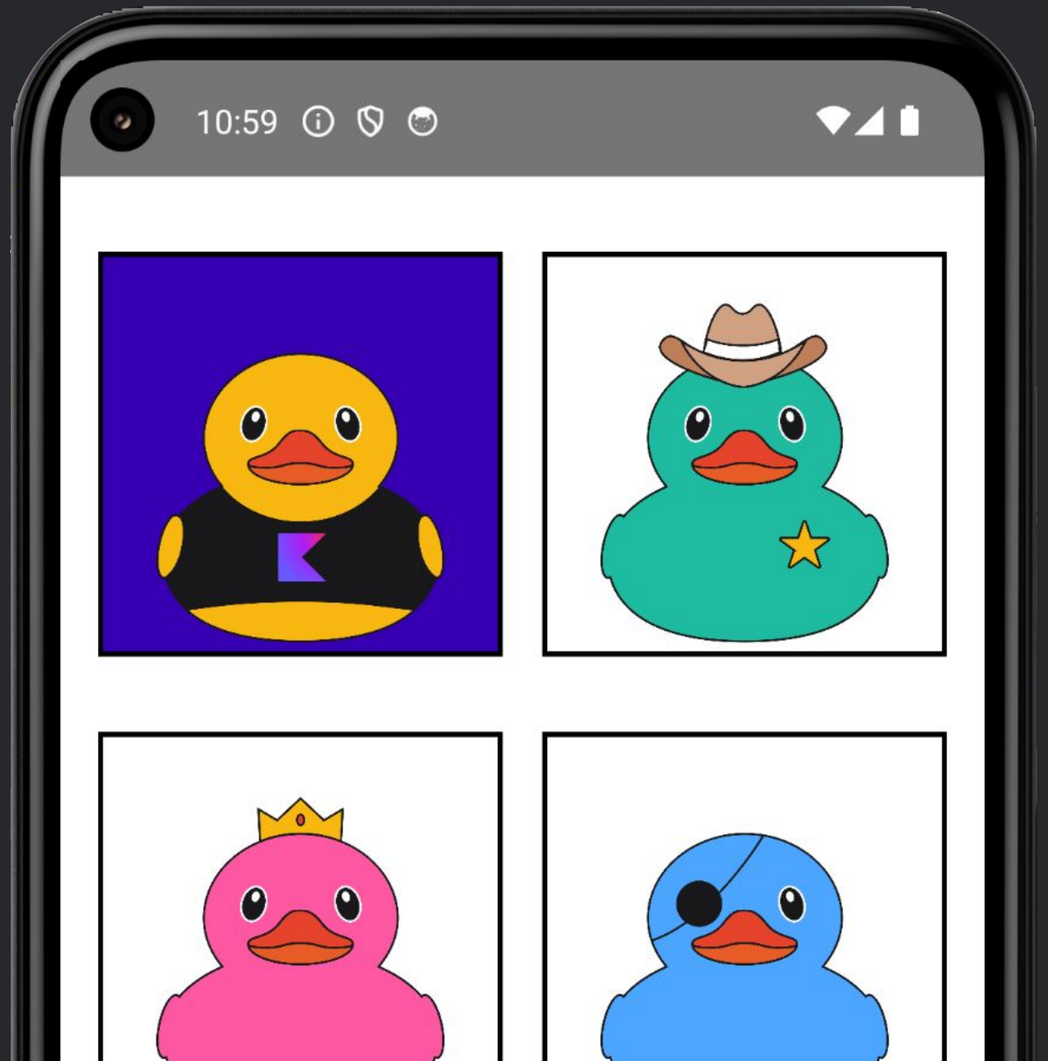
- Two column scrollable list of Ducks
- Scrollable Column

```
Column(
    modifier = Modifier
        .verticalScroll(rememberScrollState())
) { ... }
```

App UI

Settings Page

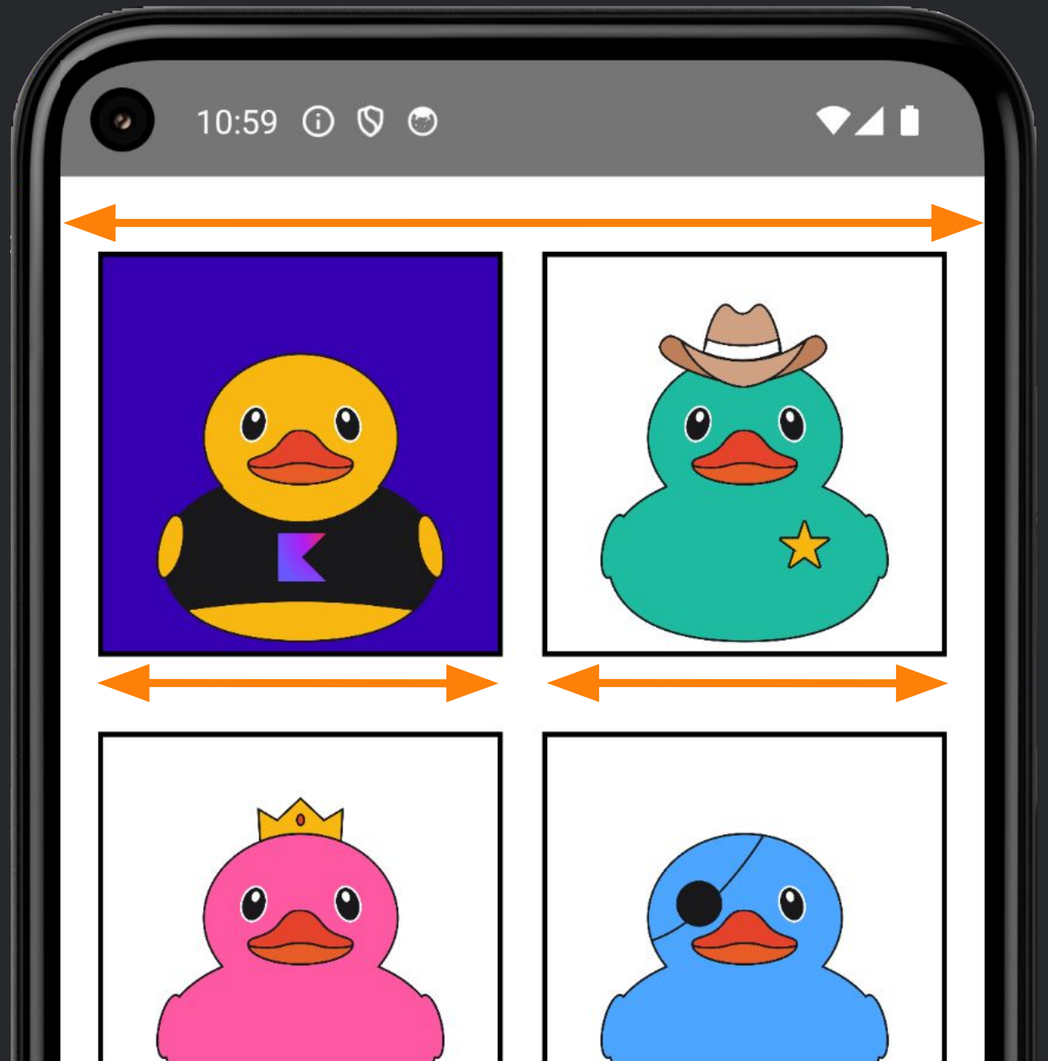
- Two column scrollable list of Ducks
- Scrollable Column
- Size of the screen



App UI

Settings Page

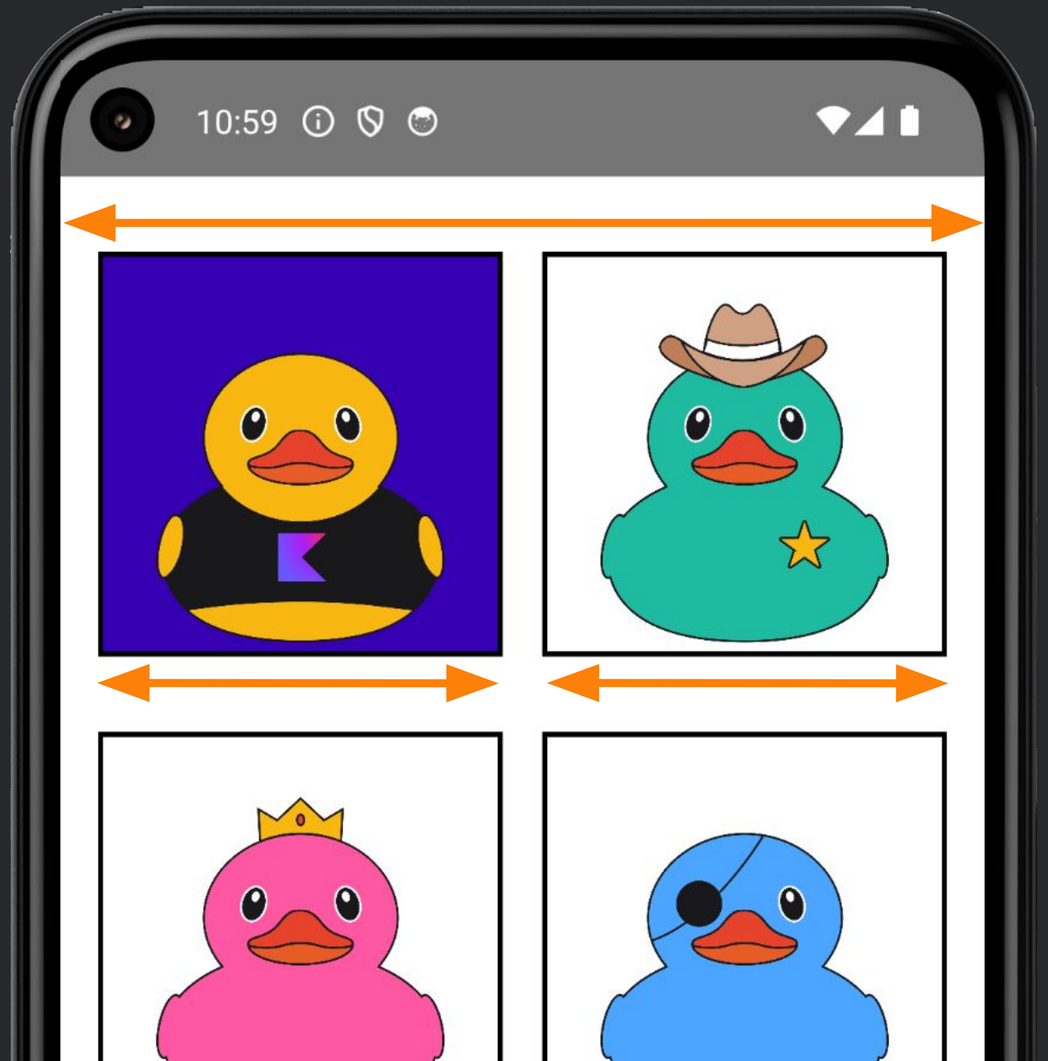
- Two column scrollable list of Ducks
- Scrollable Column
- Size of the screen



App UI

Settings Page

- Two column scrollable list of Ducks
- Scrollable Column
- Size of the screen
  - Unified way for Android, iOS and Desktop



App UI

Settings Page

- Two column scrollable list of Ducks
- Scrollable Column
- Size of the screen
  - Unified way for Android, iOS and Desktop
  - Use DP (device-independent pixels)

```
// WindowUtils.kt
@Composable
expect fun getScreenWidthDp(): Float
```

App UI

Settings Page

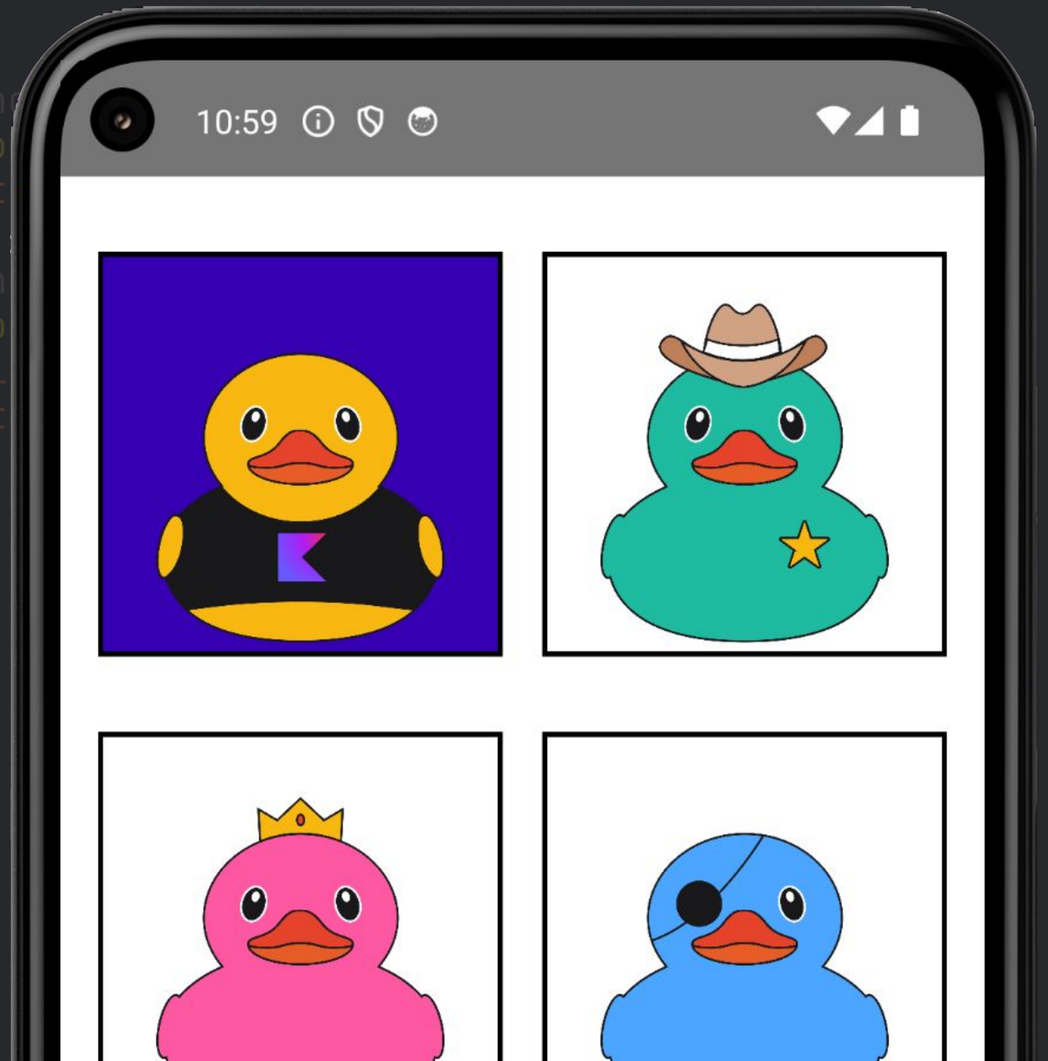
- Two column scrollable list of Ducks
- Scrollable Column
- Size of the screen
  - Unified way for Android, iOS and Desktop
  - Use DP (device-independent pixels)

```
// WindowUtils.kt
@Composable
expect fun getScreenWidthDp(): Float

// WindowUtils.android.kt
@Composable
actual fun getScreenWidthDp(): Float {
    return LocalConfiguration.current
        .screenWidthDp.toFloat()
}
```

App UI

Settings Page

- Two column scrollable list of Ducks
- Scrollable Column
- Size of the screen
  - Unified way for Android, iOS and Desktop
  - Use DP (device-independent pixels)



App UI

Settings Page

- Two column scrollable list of Ducks
- Scrollable Column
- Size of the screen
  - Unified way for Android, iOS and Desktop
  - Use DP (device-independent pixels)

```
// WindowUtils.kt
@Composable
expect fun getScreenWidthDp(): Float

// WindowUtils.android.kt
@Composable
actual fun getScreenWidthDp(): Float {
    return LocalConfiguration.current
        .screenWidthDp.toFloat()
}
```

App UI

Settings Page

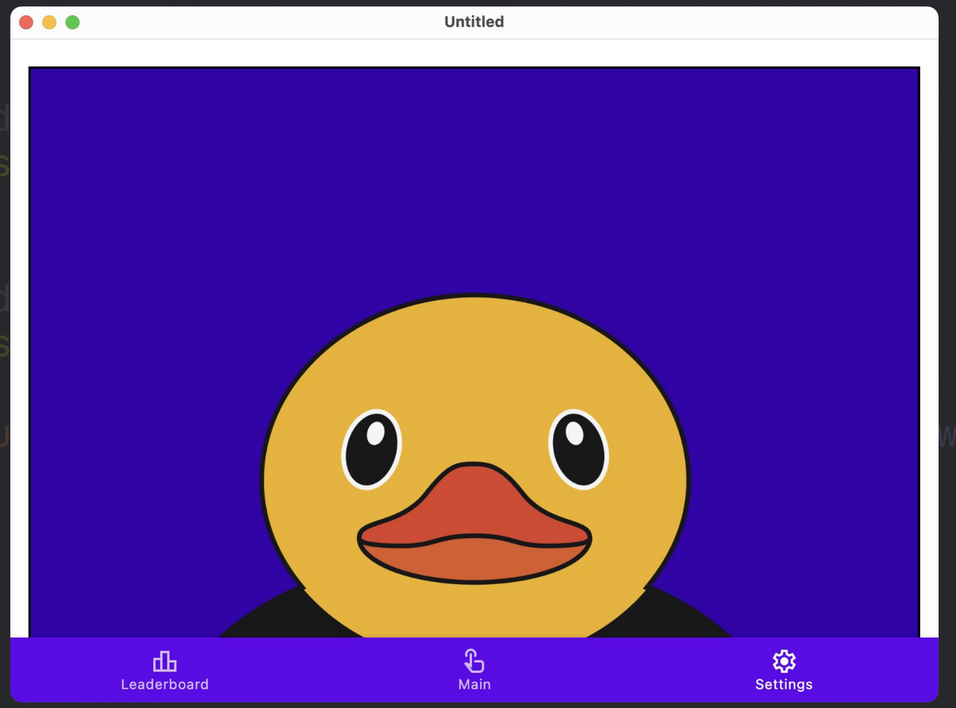
- Two column scrollable list of Ducks
- Scrollable Column
- Size of the screen
  - Unified way for Android, iOS and Desktop
  - Use DP (device-independent pixels)

```kotlin
// WindowUtils.kt
@Composable
expect fun getScreenWidthDp(): Float

// WindowUtils.ios.kt, WindowUtils.desktop.kt
@Composable
actual fun getScreenWidthDp(): Float {
    return LocalWindowInfo.current.containerSize.width
}
```

App UI

Settings Page

- Two column scrollable list of Ducks
- Scrollable Column
- Size of the screen
  - Unified way for Android, iOS and Desktop
  - Use DP (device-independent pixels)



App UI

Settings Page

- Two column scrollable list of Ducks
- Scrollable Column
- Size of the screen
  - Unified way for Android, iOS and Desktop
  - Use DP (device-independent pixels)

```kotlin
// WindowUtils.kt
@Composable
expect fun getScreenWidthDp(): Float

// WindowUtils.ios.kt, WindowUtils.desktop.kt
@Composable
actual fun getScreenWidthDp(): Float {
    return LocalWindowInfo.current.containerSize.width
}
```

App UI

Settings Page

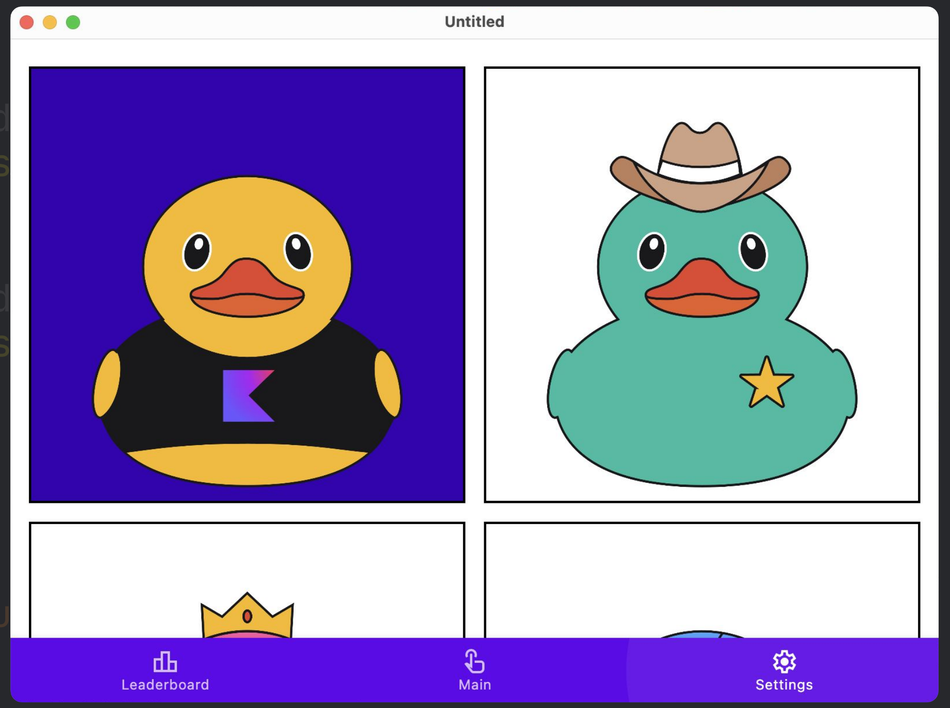
- Two column scrollable list of Ducks
- Scrollable Column
- Size of the screen
  - Unified way for Android, iOS and Desktop
  - Use DP (device-independent pixels)

```kotlin
// WindowUtils.kt
@Composable
expect fun getScreenWidthDp(): Float

// WindowUtils.ios.kt, WindowUtils.desktop.kt
@Composable
actual fun getScreenWidthDp(): Float {
    val density = LocalDensity.current.density
    val width = LocalWindowInfo.current
        .containerSize.width

    return width / density
}
```

App UI

Settings Page

- Two column scrollable list of Ducks
- Scrollable Column
- Size of the screen
  - Unified way for Android, iOS and Desktop
  - Use DP (device-independent pixels)



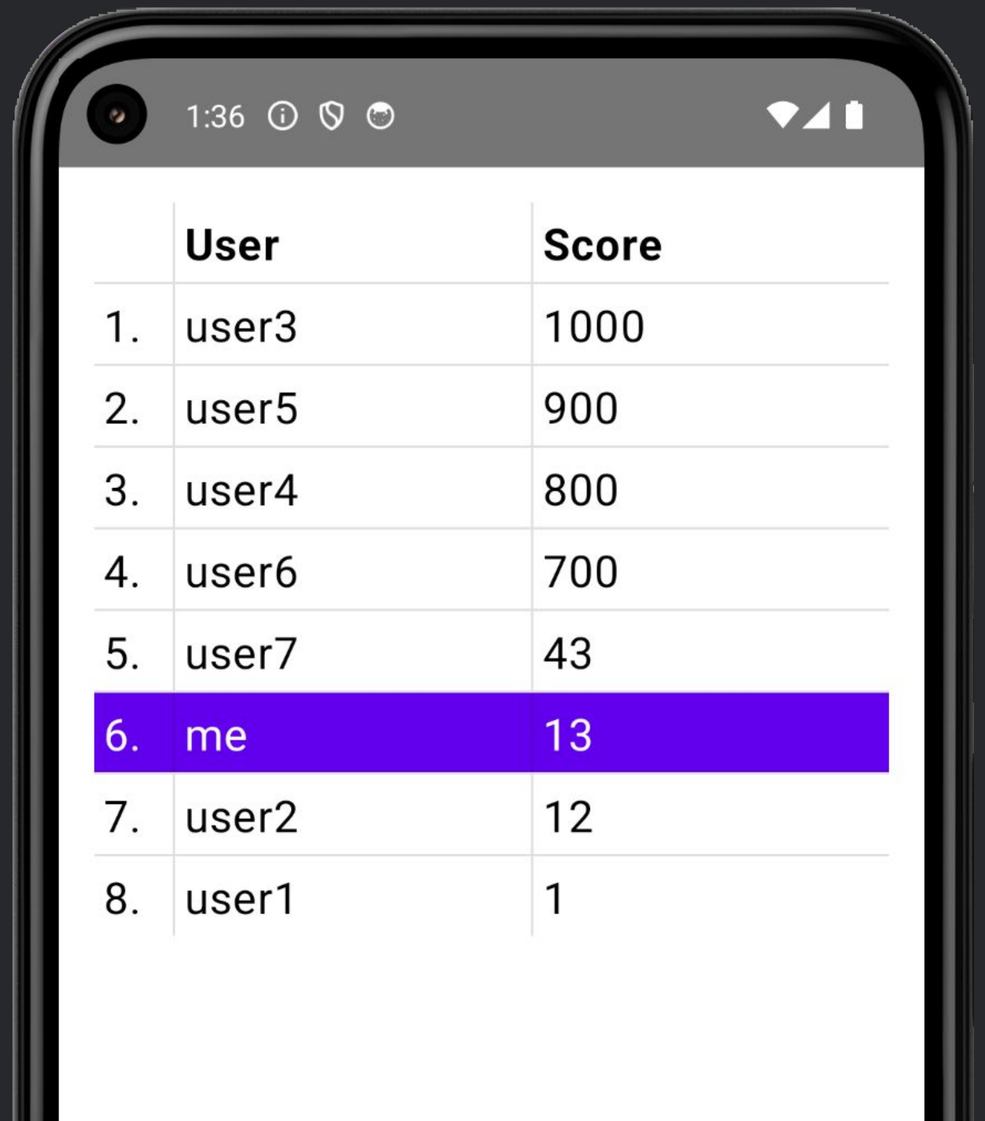
App UI

Leaderboard Page

App UI

Leaderboard Page

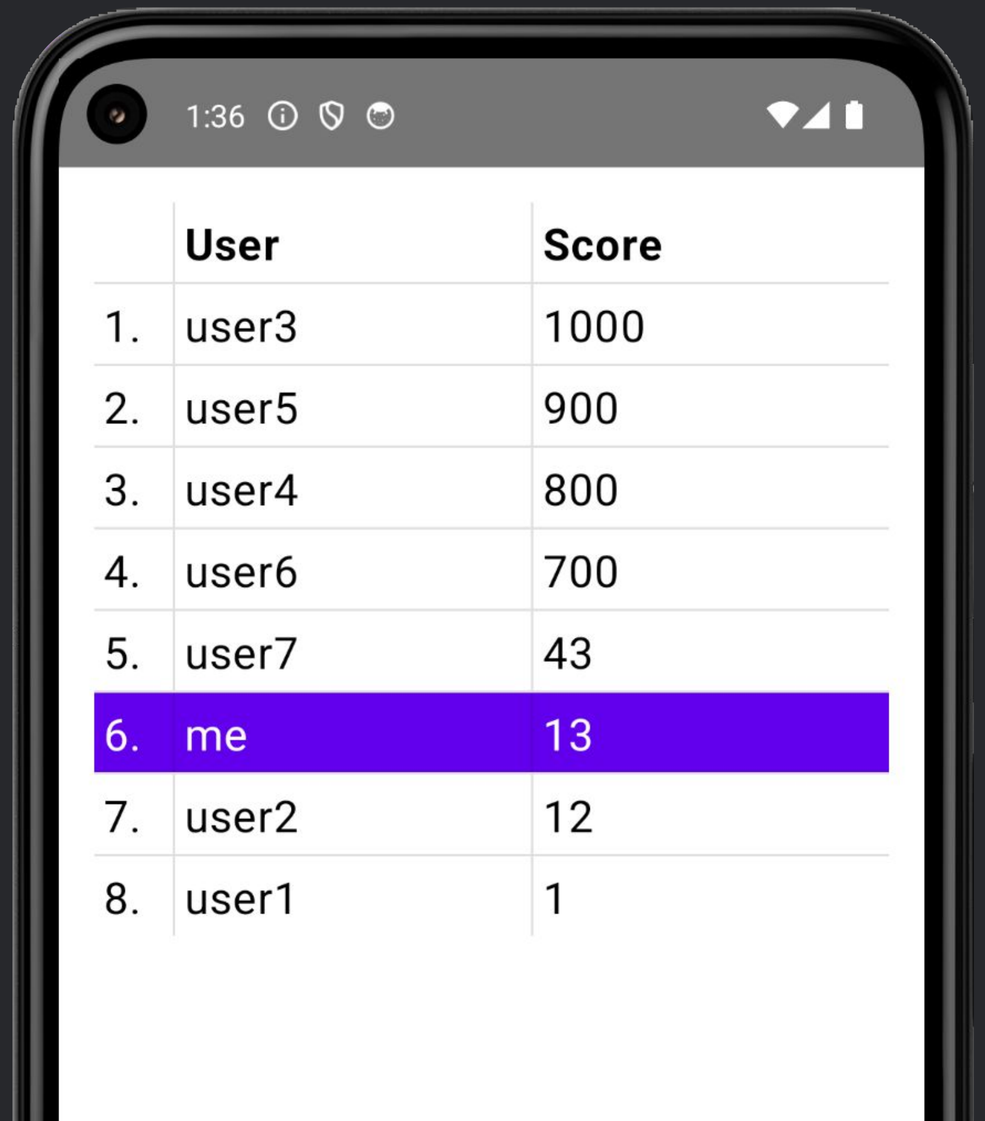
- Names and scores of all users



App UI

Leaderboard Page

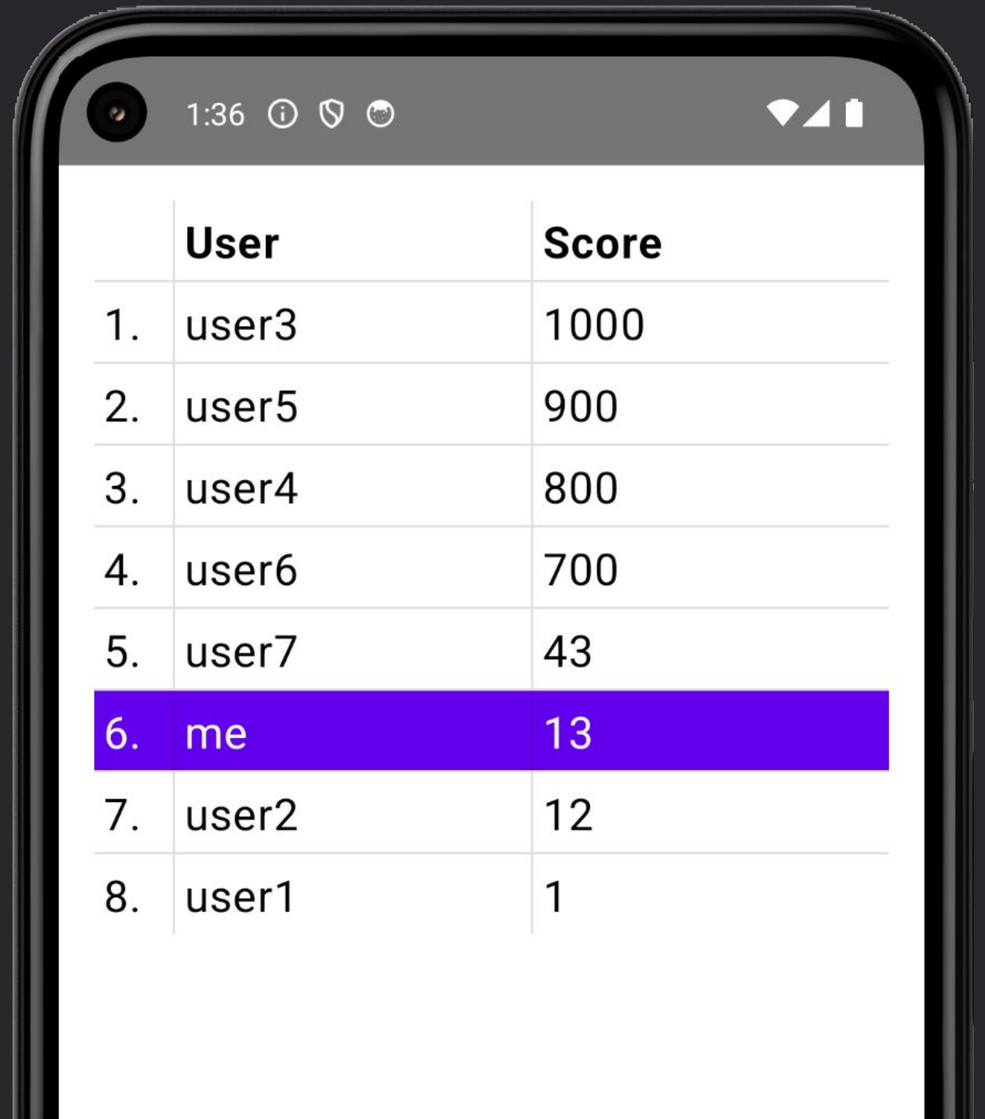
- Names and scores of all users
- But now all users are hardcoded



App UI

Leaderboard Page

- Names and scores of all users
- But now all users are hardcoded
  - Registration of users
  - Tracking users' scores



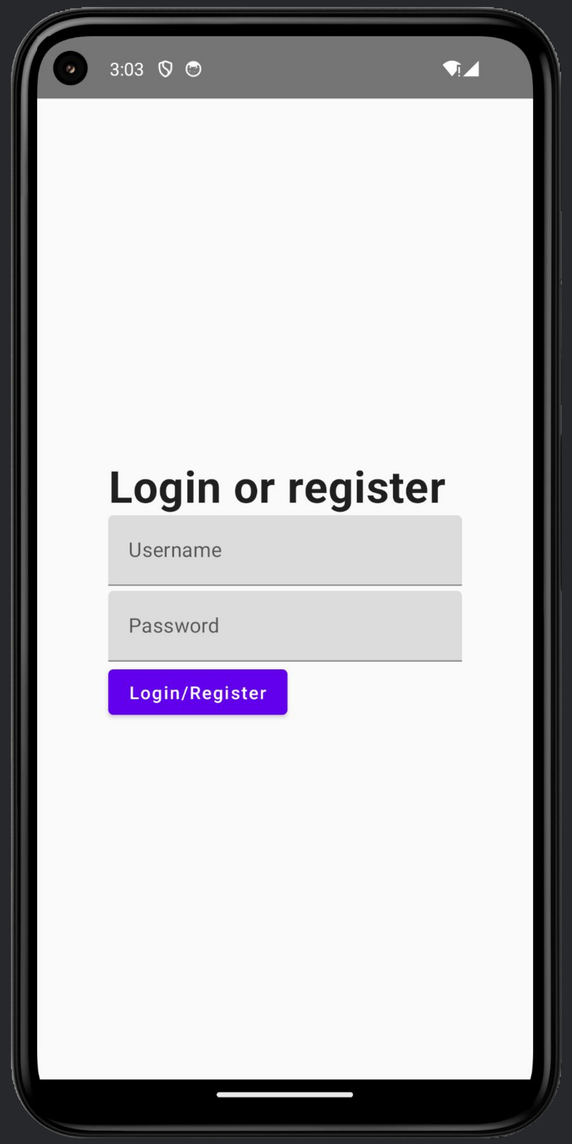
App UI

Login Page

App UI

Login Page

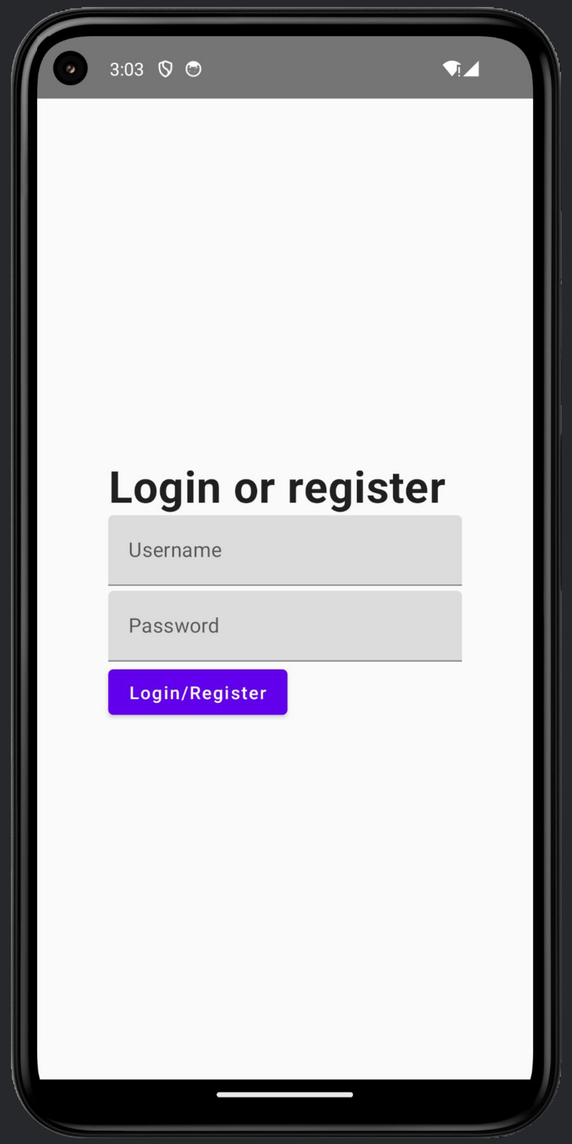
- If not logged in — show this page



App UI

Login Page

- If not logged in — show this page
- Store user credentials on a device
  - Allows instant log in



App UI

Login Page

- If not logged in — show this page
- Store user credentials on a device
  - Allows instant log in
- SQLDelight

App UI

Login Page

- If not logged in — show this page
- Store user credentials on a device
  - Allows instant log in
- SQLDelight
  - Multiplatform database driver

```
// sqldelight/User.sq

CREATE TABLE loggedUser (
   name TEXT PRIMARY KEY NOT NULL,
   password TEXT NOT NULL
);

login:
INSERT INTO loggedUser(name, password)
VALUES (?, ?);

logout:
DELETE FROM loggedUser WHERE 1=1;

getUser:
SELECT * FROM loggedUser;
```

App UI

Login Page

- If not logged in — show this page
- Store user credentials on a device
  - Allows instant log in
- SQLDelight
  - Multiplatform database driver
  - Generates API in Kotlin to make database queries

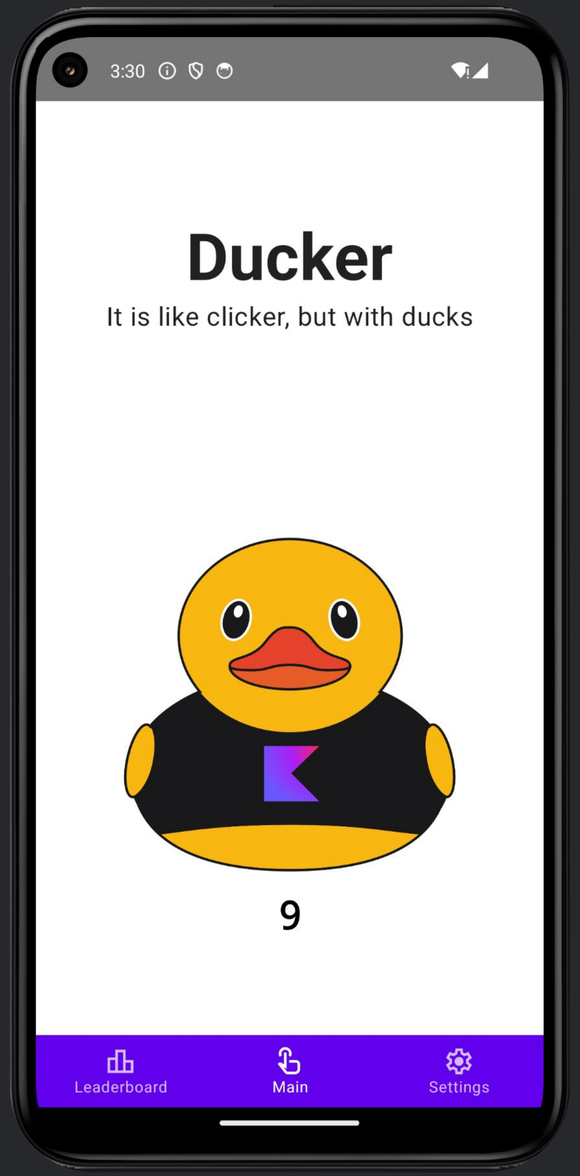
```
// Use generated Kotlin code to access database

database.userQueries.login(username, password)

database.userQueries.logout()

database.userQueries.getUser().executeAsOneOrNull()
```

App UI

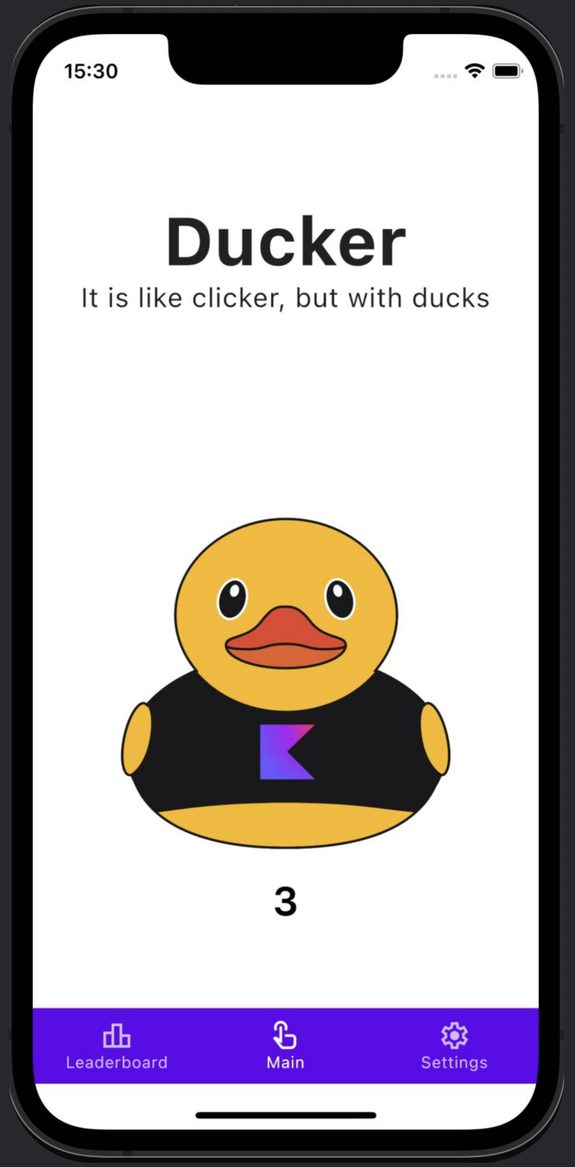
One final touch...

App UI

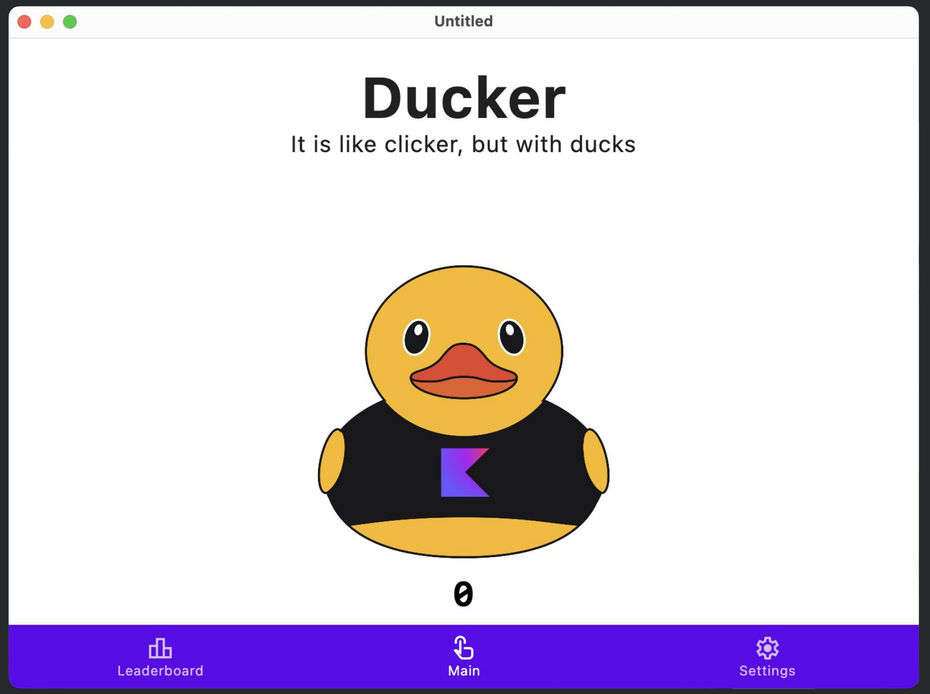
One final touch...



Android



iOS



Desktop

App UI

Caveats

App UI

Caveats

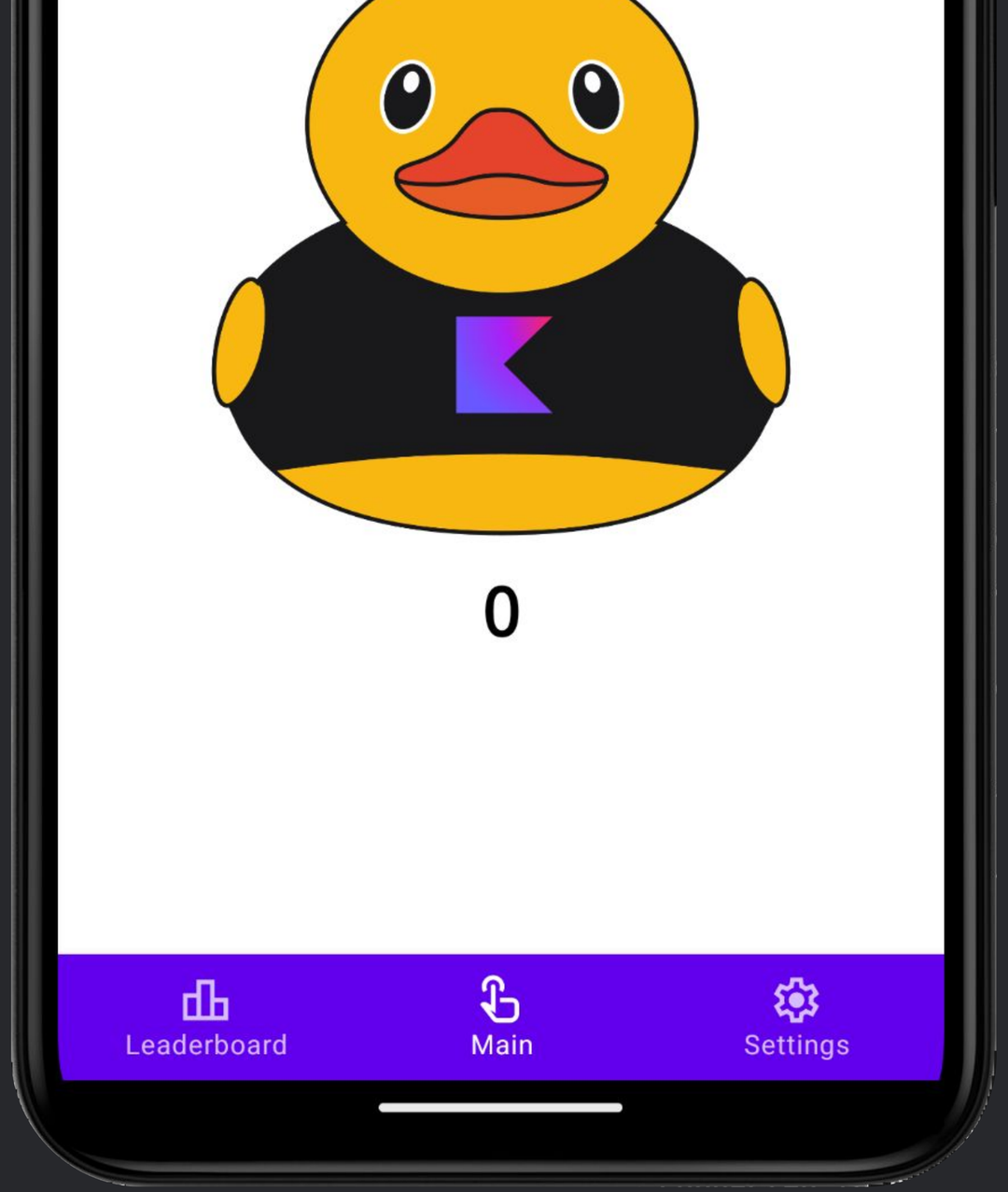
- Custom fonts are hard right now

Actual footage of me trying to add a custom font

App UI

Caveats

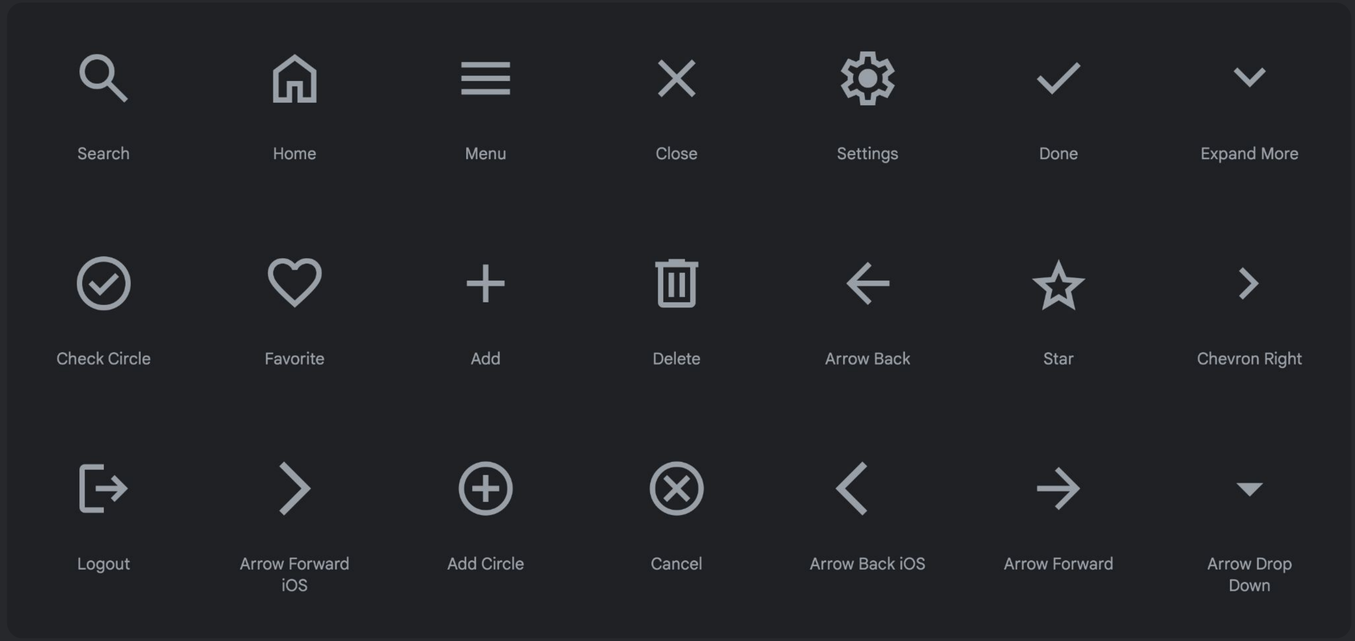
- Custom fonts are hard right now
- Icons from fonts.google.com/icons are broken*



App UI

Caveats

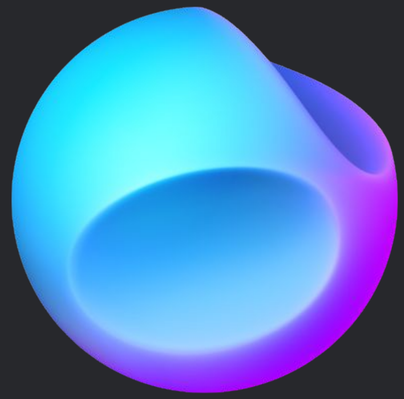
- Custom fonts are hard right now
- Icons from fonts.google.com/icons are broken*



App UI

Caveats

- Custom fonts are hard right now
- Icons from fonts.google.com/icons are broken*
  - The App just breaks on open and quits



App UI

Caveats

- Custom fonts are hard right now
- Icons from fonts.google.com/icons are broken*
  - The App just breaks on open and quits
  - Fleet does not have error logging right now

App UI

Caveats

- Custom fonts are hard right now
- Icons from fonts.google.com/icons are broken*
  - The App just breaks on open and quits

```
java.lang.IllegalArgumentException: Invalid color value
@android:color/white
```

App UI

Caveats

- Custom fonts are hard right now
- Icons from fonts.google.com/icons are broken*
  - The App just breaks on open and quits

```
java.lang.IllegalArgumentException: Invalid color value
@android:color/white

// settings_image.xml
<vector xmlns:android="..."
   android:width="24dp"
   android:height="24dp"
   android:viewportWidth="960"
   android:viewportHeight="960"
   android:tint="?attr/colorControlNormal">
 <path
     android:fillColor="@android:color/white"
     android:pathData="M419,880Q391,880 ..."/>
</vector>
```

App UI

Caveats

- Custom fonts are hard right now
- Icons from fonts.google.com/icons are broken*
  - The App just breaks on open and quits
  - But is fixable

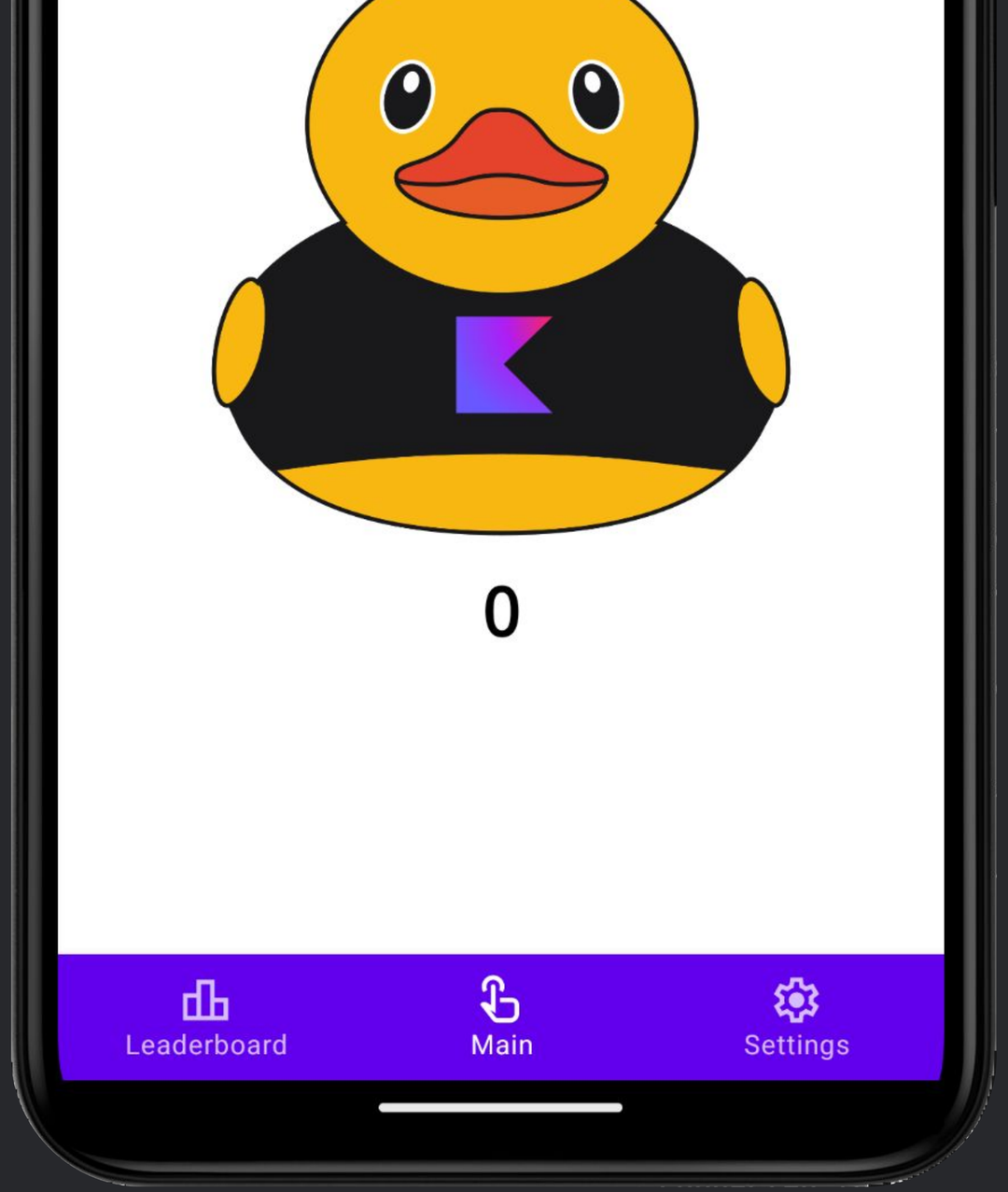
```
java.lang.IllegalArgumentException: Invalid color value
@android:color/white
```

```
// settings_image.xml
<vector xmlns:android="..."
   android:width="24dp"
   android:height="24dp"
   android:viewportWidth="960"
   android:viewportHeight="960"
   android:tint="?attr/colorControlNormal">
 <path
     android:fillColor="#FFFFFF"
     android:pathData="M419,880Q391,880 ..."/>
</vector>
```

App UI

Caveats

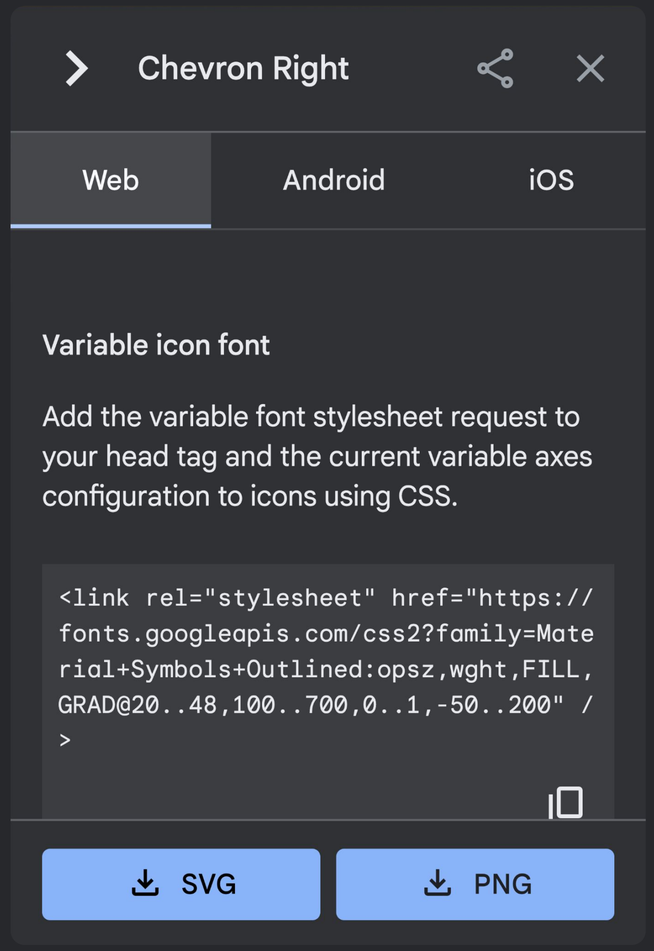
- Custom fonts are hard right now
- Icons from fonts.google.com/icons are broken*



App UI

Caveats

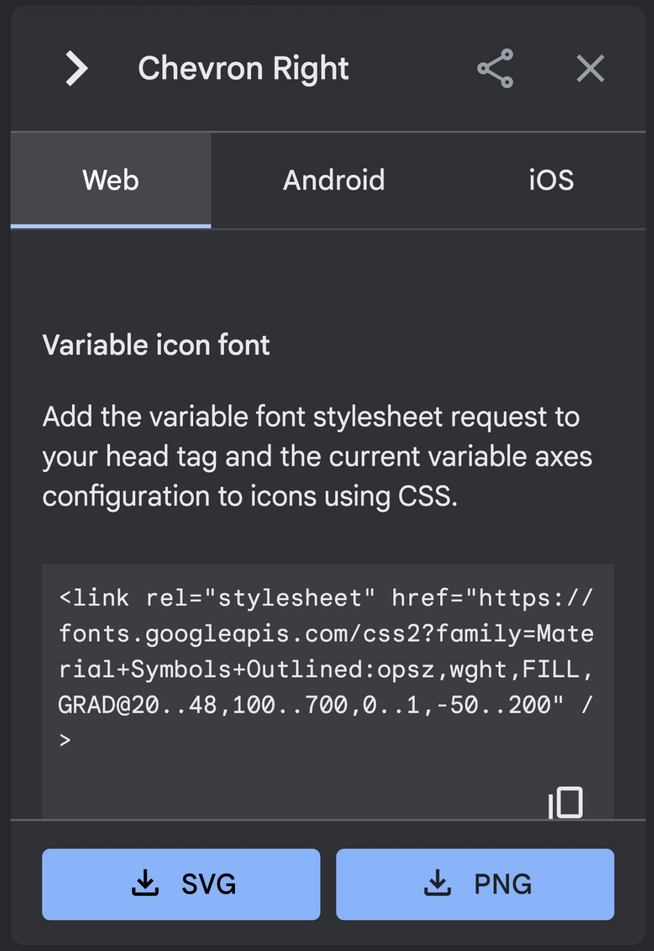
- Custom fonts are hard right now
- Icons from fonts.google.com/icons are broken*
  - *If downloaded for Android

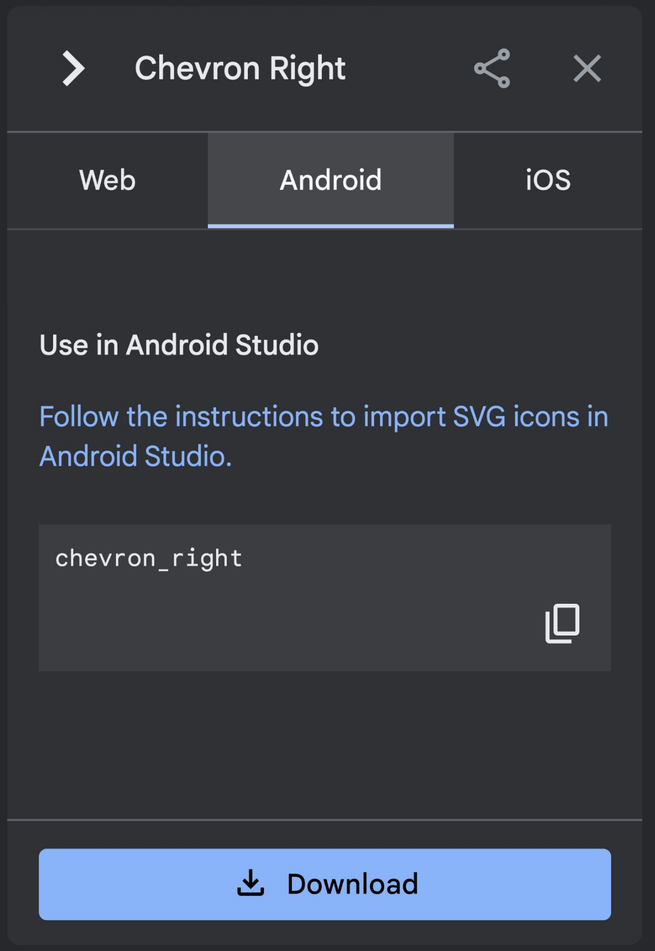


App UI

Caveats

- Custom fonts are hard right now
- Icons from fonts.google.com/icons are broken*
  - *If downloaded for Android





App UI

Caveats

- Custom fonts are hard right now
- Not everything that you download from the Internet is good (be careful with icons)

App UI

Caveats

- Custom fonts are hard right now
- Not everything that you download from the Internet is good (be careful with icons)
- SQLDelight does not break compilation if SQL is malformed

App UI

Caveats

- Custom fonts are hard right now
- Not everything that you download from the Internet is good (be careful with icons)
- SQLDelight does not break compilation if SQL is malformed

```
// Wrong
CREATE TABLE user (
    name VARCHAR(60) PRIMARY KEY NOT NULL,
    password VARCHAR(60) NOT NULL,
);
```

App UI

Caveats

- Custom fonts are hard right now
- Not everything that you download from the Internet is good (be careful with icons)
- SQLDelight does not break compilation if SQL is malformed
  - Fleet won't tell you that, AS with SQLDelight plugin will

```
// Wrong
CREATE TABLE user (
    name VARCHAR(60) PRIMARY KEY NOT NULL,
    password VARCHAR(60) NOT NULL,
);

// Right
CREATE TABLE loggedUser (
    name TEXT PRIMARY KEY NOT NULL,
    password TEXT NOT NULL
);
```

App UI

Caveats

- Custom fonts are hard right now
- Not everything that you download from the Internet is good (be careful with icons)
- SQLDelight on iOS is tricky (a bit)

App UI

Caveats

- Custom fonts are hard right now
- Not everything that you download from the Internet is good (be careful with icons)
- SQLDelight on iOS is tricky (a bit)
  - iOS 15.2+ is supported
  - `-lsqlite3` flag is needed

App UI

Caveats

- Custom fonts are hard right now
- Not everything that you download from the Internet is good (be careful with icons)
- Be careful with SQLDelight

Home stretch

Home stretch

Server

- Handling client calls

Home stretch

Server

- Handling client calls
  - Done ✅

Home stretch

Server

- Handling client calls
  - Done ✅
- Database persistence
  - Done ✅

https://github.com/Mr3zee/WATFOI-KMP-Ducker

Home stretch

Client calls for server

Home stretch

Client calls for server

- Use Ktor Client for multiplatform implementation of HTTP client

Home stretch

Client calls for server

- Use Ktor Client for multiplatform implementation of HTTP client
  - Configure client

```
expect val platformHost: String

val apiClient = HttpClient {
    install(ContentNegotiation) {
        json()
    }

    defaultRequest {
        url("http://$platformHost:$PORT/")
    }
}
```

Home stretch

Client calls for server

- Use Ktor Client for multiplatform implementation of HTTP client
  - Configure client

```
expect val platformHost: String

val apiClient = HttpClient {
    install(ContentNegotiation) {
        json()
    }

    defaultRequest {
        url("http://$platformHost:$PORT/")
    }
}
```

Home stretch

Client calls for server

- Use Ktor Client for multiplatform implementation of HTTP client
  - Configure client

```
expect val platformHost: String

val apiClient = HttpClient {
    install(ContentNegotiation) {
        json()
    }

    defaultRequest {
        url("http://$platformHost:$PORT/")
    }
}
```

Home stretch

Client calls for server

- Use Ktor Client for multiplatform implementation of HTTP client
  - Configure client

```
// Common
expect val platformHost: String

// Android
actual val platformHost: String = "10.0.2.2"

// iOS, Desktop
actual val platformHost: String = "0.0.0.0"
```

Home stretch

Client calls for server

- Use Ktor Client for multiplatform implementation of HTTP client
  - Configure client
  - Send requests

```
// Common for Client AND Server
@Serializable
data class User(
    val name: String,
    val password: String,
)

// Somewhere in Model on Client
suspend fun loginOrRegister(user: User) {
    apiClient.post("/api/loginOrRegister") {
        setBody(user)

        contentType(ContentType.Application.Json)
    }
}
```

Home stretch

Client calls for server

- Use Ktor Client for multiplatform implementation of HTTP client
  - Configure client
  - Send requests

```
// Common for Client AND Server
@Serializable
data class User(
    val name: String,
    val password: String,
)

// Somewhere in Model on Client
suspend fun loginOrRegister(user: User) {
    apiClient.post("/api/loginOrRegister") {
        setBody(user)

        contentType(ContentType.Application.Json)
    }
}
```

Home stretch

Client calls for server

- Use Ktor Client for multiplatform implementation of HTTP client
  - Configure client
  - Send requests
  - Use kotlinx.serialization

```
// Common for Client AND Server
@Serializable
data class User(
    val name: String,
    val password: String,
)

// Somewhere in Model on Client
suspend fun loginOrRegister(user: User) {
    apiClient.post("/api/loginOrRegister") {
        setBody(user)

        contentType(ContentType.Application.Json)
    }
}
```

Home stretch

Client calls for server

- Use Ktor Client for multiplatform implementation of HTTP client
  - Configure client
  - Send requests
  - Use kotlinx.serialization

```
// Common for Client AND Server
@Serializable
data class User(
    val name: String,
    val password: String, // NEVER EVER EVER DO THIS
)

// Somewhere in Model on Client
suspend fun loginOrRegister(user: User) {
    apiClient.post("/api/loginOrRegister") {
        setBody(user)

        contentType(ContentType.Application.Json)
    }
}
```

Home stretch

Client calls for server

- Use Ktor Client for multiplatform implementation of HTTP client
- Disable platform security checks for local machine

Home stretch

Client calls for server

- Use Ktor Client for multiplatform implementation of HTTP client
- Disable platform security checks for local machine

```
expect val platformHost: String

val apiClient = HttpClient {
    defaultRequest {
        url("http://$platformHost:$PORT/")
    }
}
```

Home stretch

Client calls for server

- Use Ktor Client for multiplatform implementation of HTTP client
- Disable platform security checks for local machine

```
expect val platformHost: String

val apiClient = HttpClient {
    defaultRequest {
        url("http://$platformHost:$PORT/")
    }
}
```

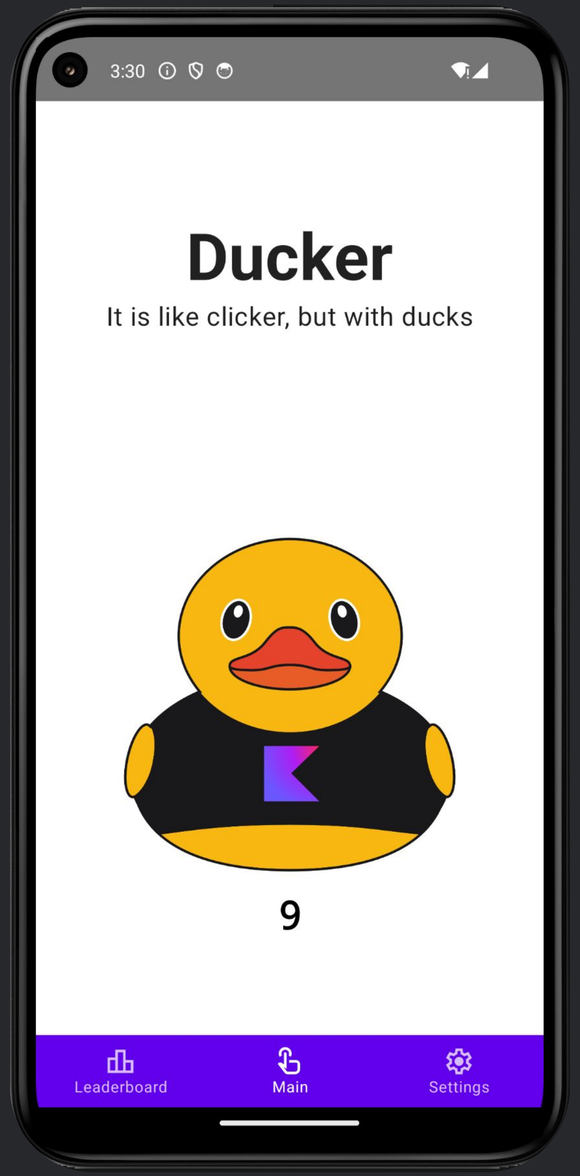
Home stretch

Client calls for server

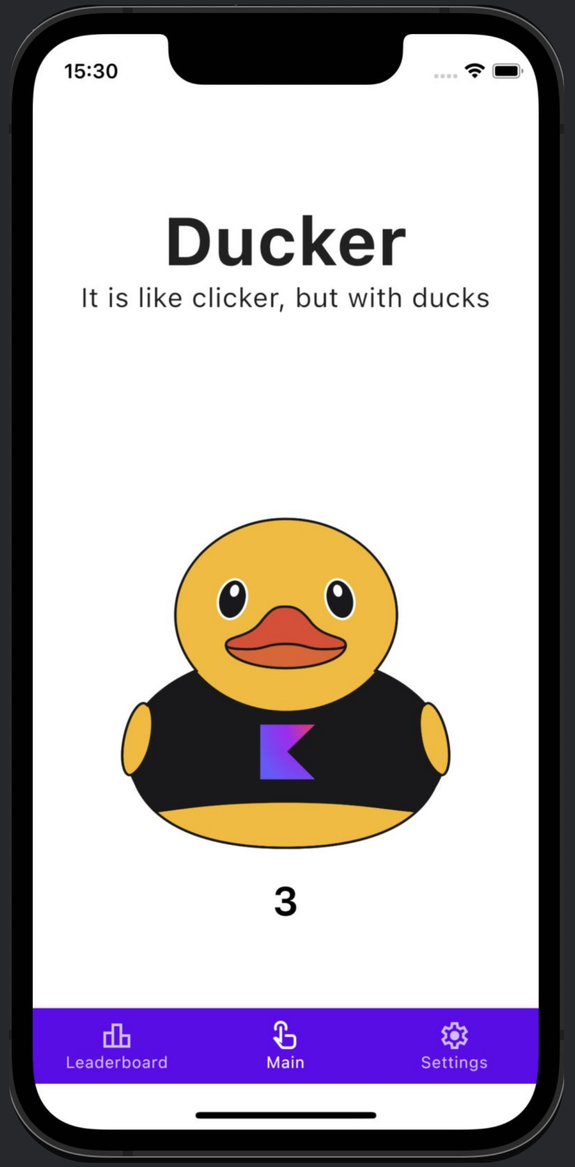
- Use Ktor Client for multiplatform implementation of HTTP client
- Disable platform security checks for local machine
  - Specifics are on the project's Github

```
expect val platformHost: String

val apiClient = HttpClient {
    defaultRequest {
        url("http://$platformHost:$PORT/")
    }
}
```

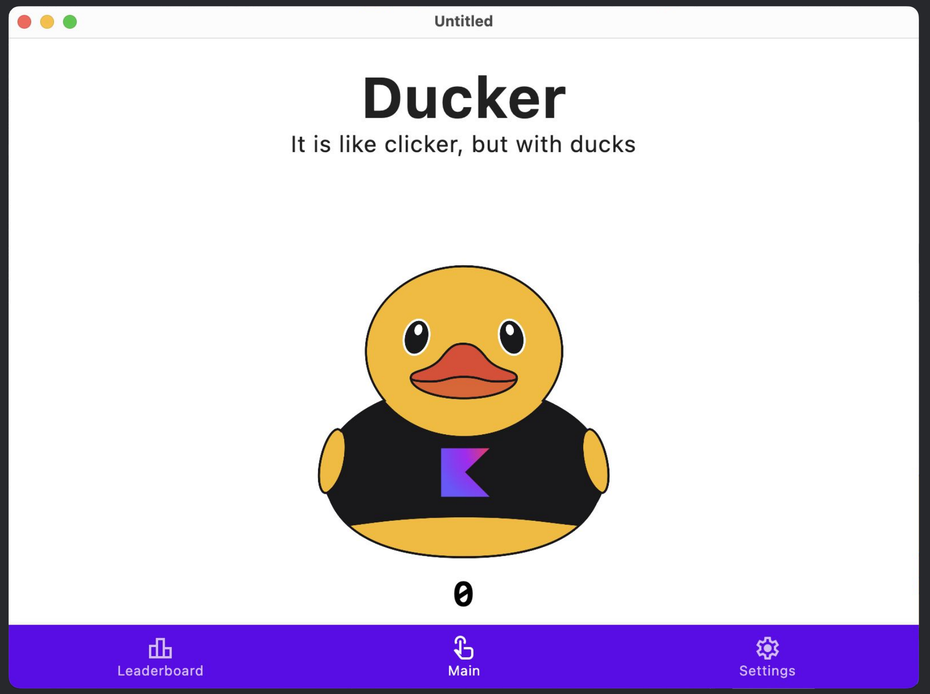
We are done!



Android



iOS



Desktop

1:36

	User	Score
1.	user3	1000
2.	user5	900
3.	user4	800
4.	user6	700
5.	user7	43
6.	me	13
7.	user2	12
8.	user1	1

Leaderboard Main Settings

3:30

Ducker

It is like clicker, but with ducks

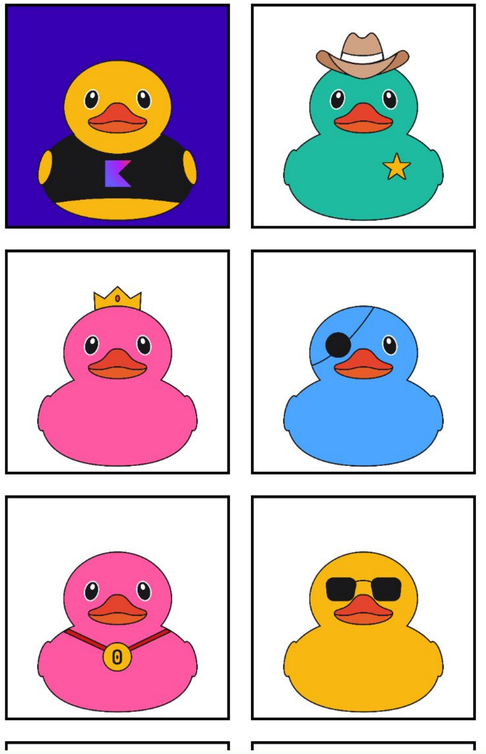
9

Leaderboard Main Settings

7:03

Logout

Select your duck



Leaderboard Main Settings

Conclusions

Conclusions

- The App is done
  - Clickable Duck
  - Leaderboard
  - Settings
  - Login
  - iOS, Android and Desktop

Conclusions

- The App is done
  - Clickable Duck
  - Leaderboard
  - Settings
  - Login
  - iOS, Android and Desktop
- Shared 100% of the UI

Conclusions

- The App is done
  - Clickable Duck
  - Leaderboard
  - Settings
  - Login
  - iOS, Android and Desktop
- Shared 100% of the UI
- Has local and server persistence

Conclusions

- The App is done
  - Clickable Duck
  - Leaderboard
  - Settings
  - Login
  - iOS, Android and Desktop
- Shared 100% of the UI
- Has local and server persistence
- Fought Fleet, Xcode and Android Studio

Conclusions

- The App is done
  - Clickable Duck
  - Leaderboard
  - Settings
  - Login
  - iOS, Android and Desktop
- Shared 100% of the UI
- Has local and server persistence
- Fought Fleet, Xcode and Android Studio
- Learned something new!

Conclusions

- The App is done
  - Clickable Duck
  - Leaderboard
  - Settings
  - Login
  - iOS, Android and Desktop
- Shared 100% of the UI
- Has local and server persistence
- Fought Fleet, Xcode and Android Studio
- Learned something new!

- Replace some UI elements with native ones

Conclusions

- The App is done
  - Clickable Duck
  - Leaderboard
  - Settings
  - Login
  - iOS, Android and Desktop
- Shared 100% of the UI
- Has local and server persistence
- Fought Fleet, Xcode and Android Studio
- Learned something new!

- Replace some UI elements with native ones
- Skipped all project configurations
  - Gradle
  - Android Manifest
  - xcodeproj

Conclusions

- The App is done
  - Clickable Duck
  - Leaderboard
  - Settings
  - Login
  - iOS, Android and Desktop
- Shared 100% of the UI
- Has local and server persistence
- Fought Fleet, Xcode and Android Studio
- Learned something new!

- Replace some UI elements with native ones
- Skipped all project configurations
  - Gradle
  - Android Manifest
  - xcodeproj
- Skipped some too technical details
  - Notes on that could be found on Github

Conclusions

- The App is done
  - Clickable Duck
  - Leaderboard
  - Settings
  - Login
  - iOS, Android and Desktop
- Shared 100% of the UI
- Has local and server persistence
- Fought Fleet, Xcode and Android Studio
- Learned something new!

- Replace some UI elements with native ones
- Skipped all project configurations
  - Gradle
  - Android Manifest
  - xcodeproj
- Skipped some too technical details
  - Notes on that could be found on Github
- Some code is demonstrative, not suitable for real life applications

Thank you!

Alexander Sysoev
Email: alexander.sysoev@jetbrains.com
Github: https://github.com/Mr3zee

JetBrains Novi Sad Discord server

WE ARE FUTURE OF IT

PAUZA

15 MINUTA

STUDENTSKA KONFERENCIJA
WATFOI 23