

Kotlin For Everything



Алексей Гладков
Leroy Merlin

Алексей Гладков



Немного о себе

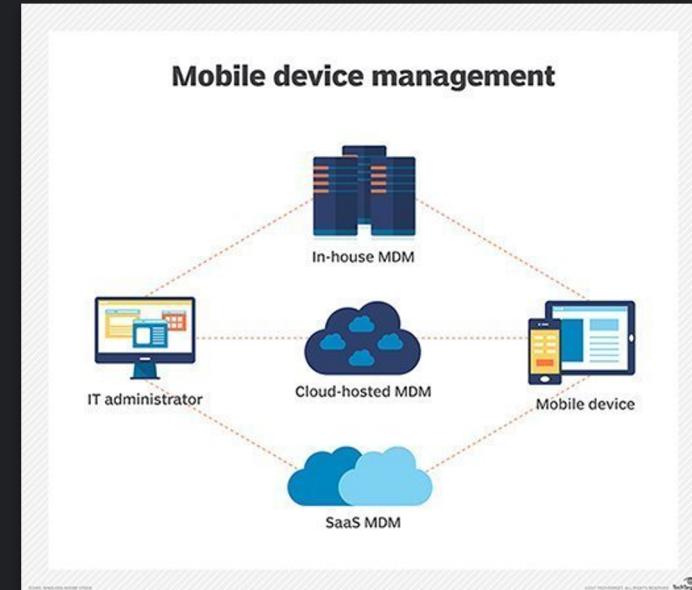
- 7 лет пишу нативно под iOS и под Android
- Преподаю Android-разработку в LoftSchool
- Автор канала **Mobile Developer**

Предпосылки

Предпосылки

Mobile Device Management

- > Много приложений
- > Удаленное управление



Проблема

- > Много приложений - легко запутаться

Проблема

- > Много приложений, легко запутаться
- > Отсутствие единого подхода к разработке

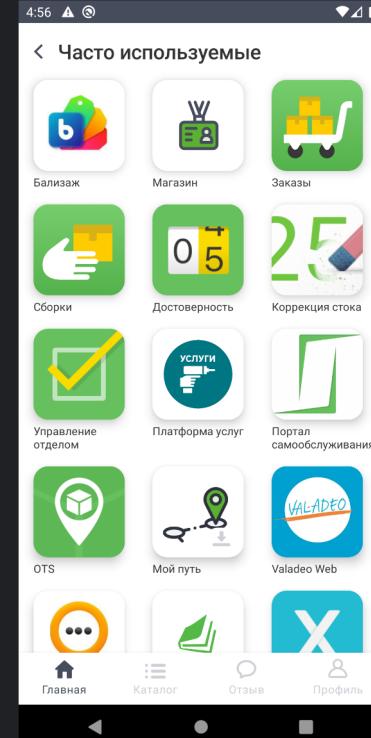
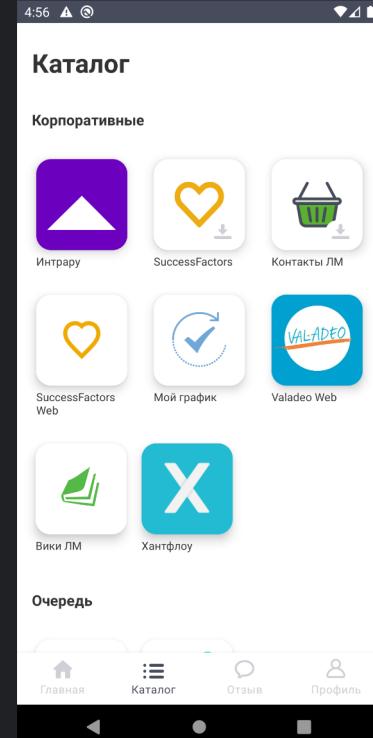
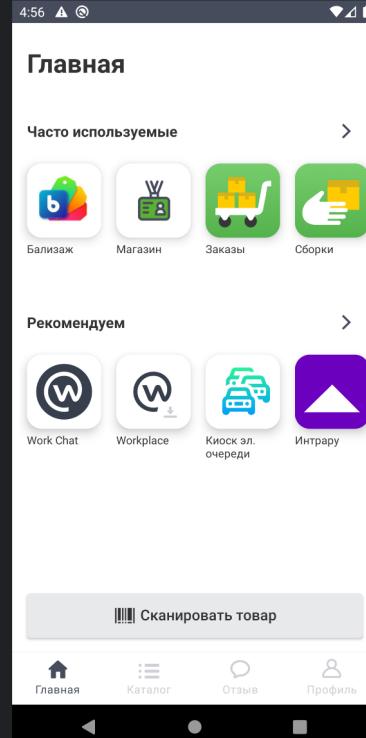
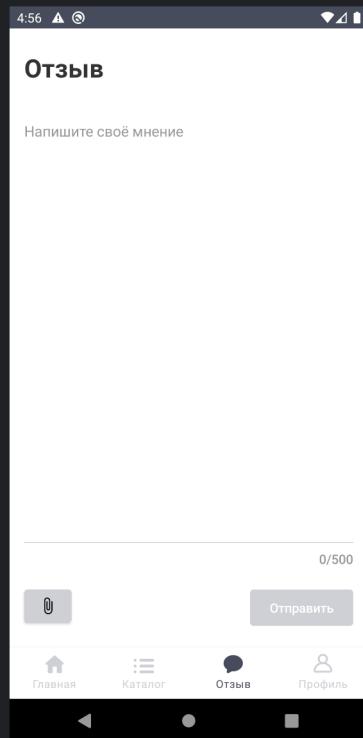
Проблема

- > Много приложений, легко запутаться
- > Отсутствие единого подхода к разработке
- > Отсутствие требований к внешним разработчикам

Проблема

- > Много приложений, легко запутаться
- > Отсутствие единого подхода к разработке
- > Отсутствие требований к внешним разработчикам
- > Плохой перфоманс, устаревший стек, бесконечный бэклог

LMWork



Что дальше?

Телефон
Планшет
Ноутбук



2015

Телефон
Планшет
Ноутбук

Телефон
Планшет
Ноутбук
Машина
ТВ
Часы
Холодильник
Стиралка
Лифт
Панели
...
Продолжение
следует

2015

2021

А что еще?

- > Новые ОС (Harmony, Aurora, Fuchsia)

А что еще?

- > Новые ОС (Harmony, Aurora, Fuchsia)
- > Различные киоски, автоматы и так далее

А что еще?

- > Новые ОС (Harmony, Aurora, Fuchsia)
- > Различные киоски, автоматы и так далее
- > Желание людей работать омниканально, не зависеть от обстоятельств

Задача

Android → → → Multiplatform

Почему Kotlin?

Статья на
kotlinlang [EN]



Видео с
митапа [RU]



Common

JVM

JS

Native

Главная проблема

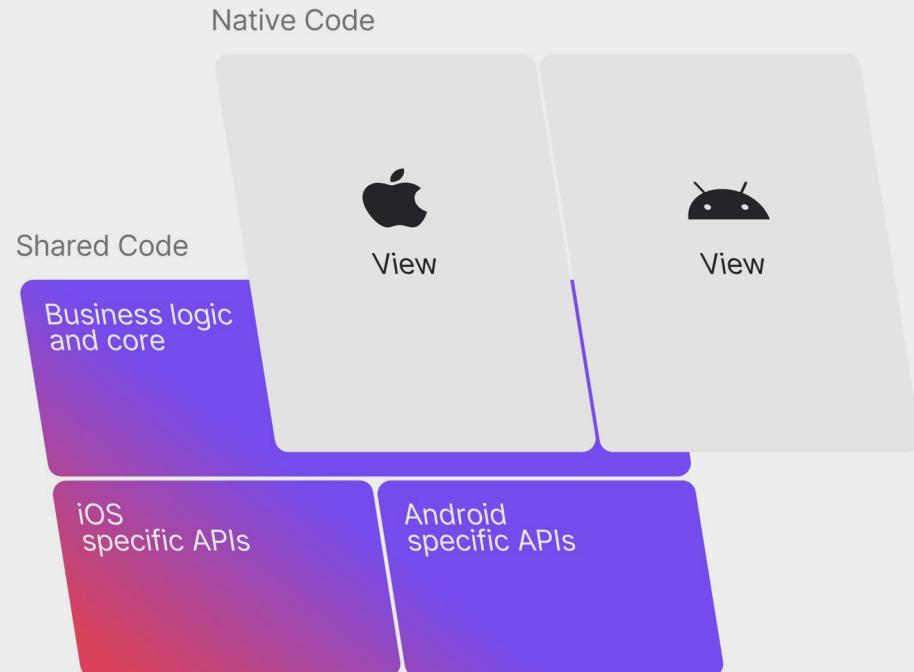
Отсутствие готовых решений. Малое количество библиотек



GitHub

[terrakok/kmm-awesome](#)

iOS & Android Shared Libraries



Задача

Android → → → Multiplatform

Из чего состоит мобильное приложение?



UI

Navigation

Architecture

DI

Resources

Data

Analytics

Device

Test



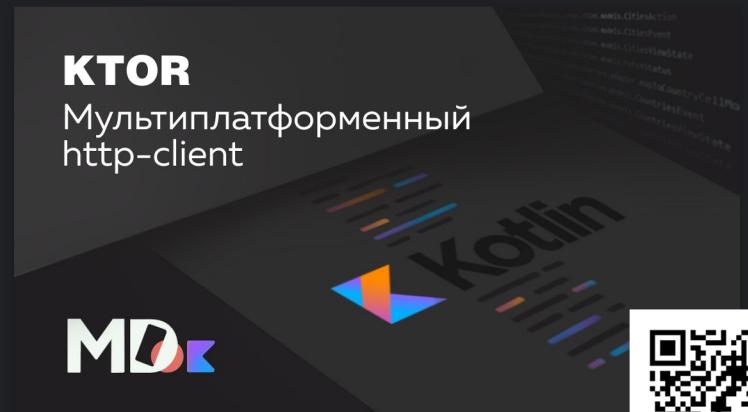
Data

Data .Remote

Ktor Client

Мультиплатформенный http-клиент

- iOS
- Android
- Desktop
- Web (Не протестировано)



Data.Local

SQL Delight

Мультиплатформенная DB



iOS



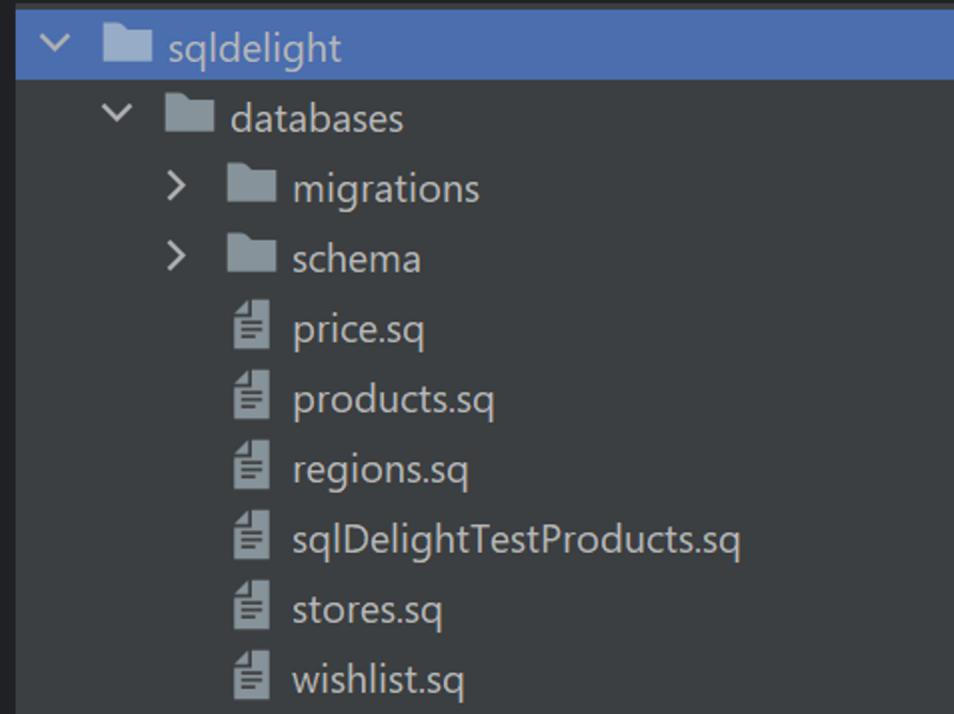
Android



Desktop (Не протестировано)



Web (Не протестировано)



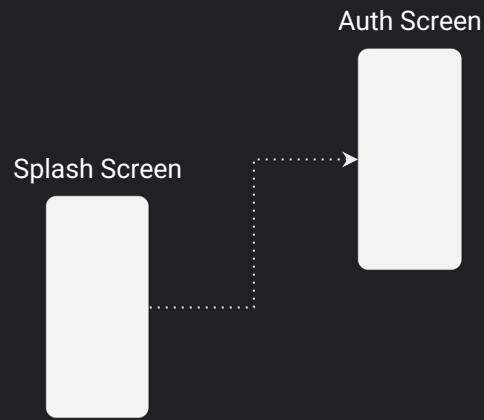
Architecture

Architecture.Android (Old)

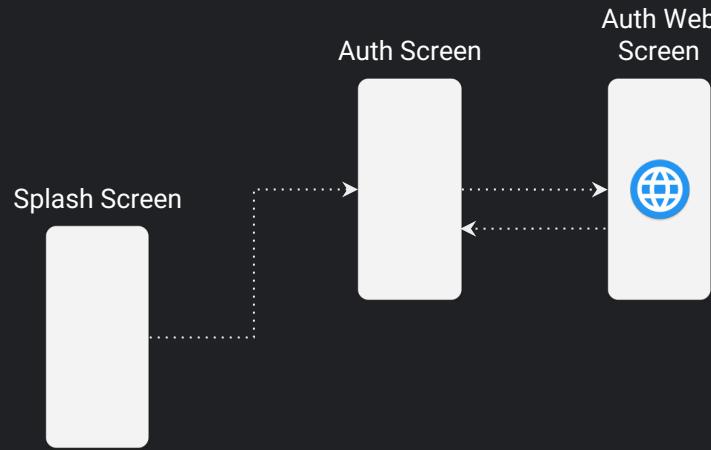
Splash Screen



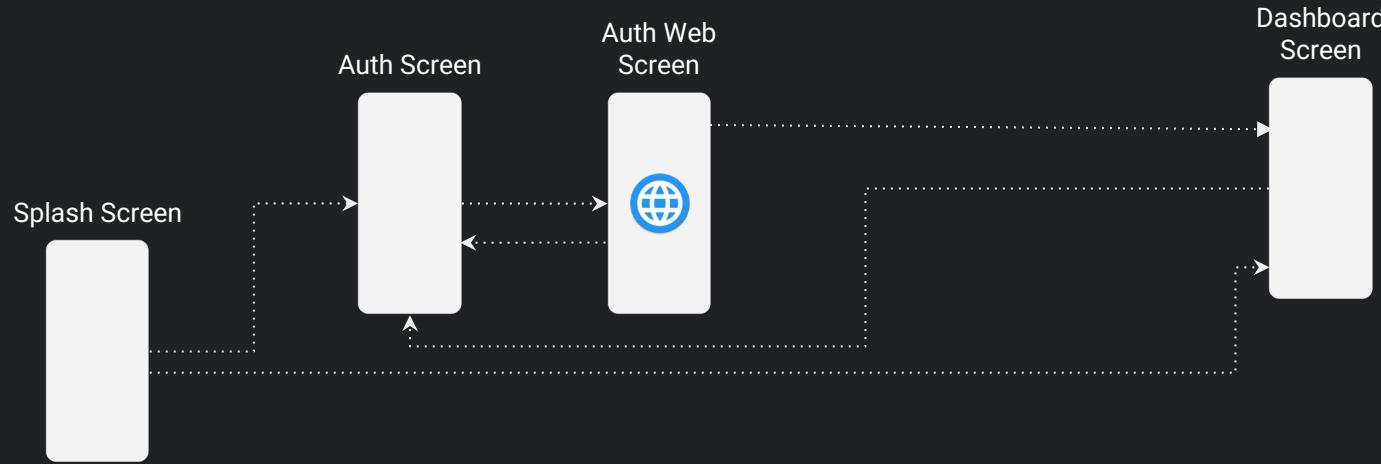
Architecture.Android (Old)



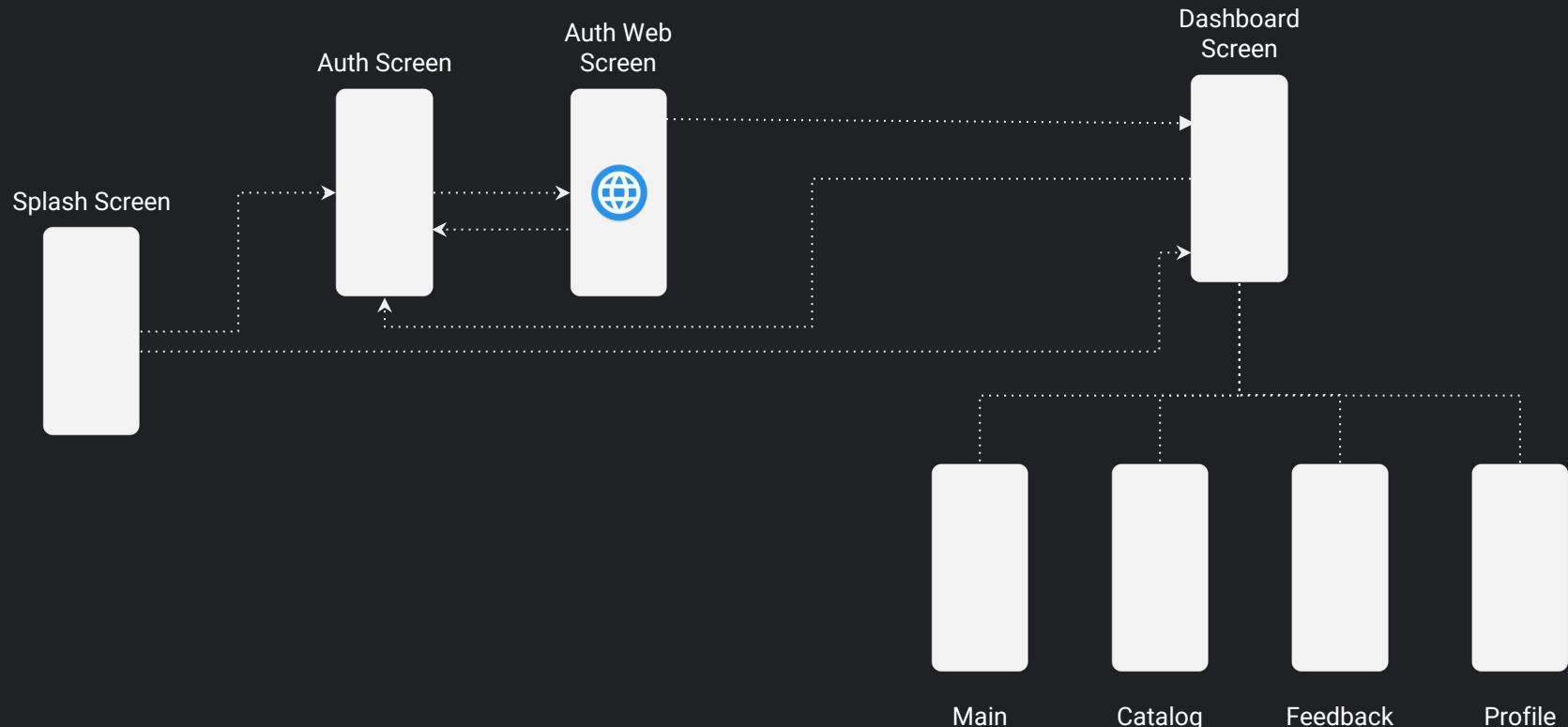
Architecture.Android (Old)



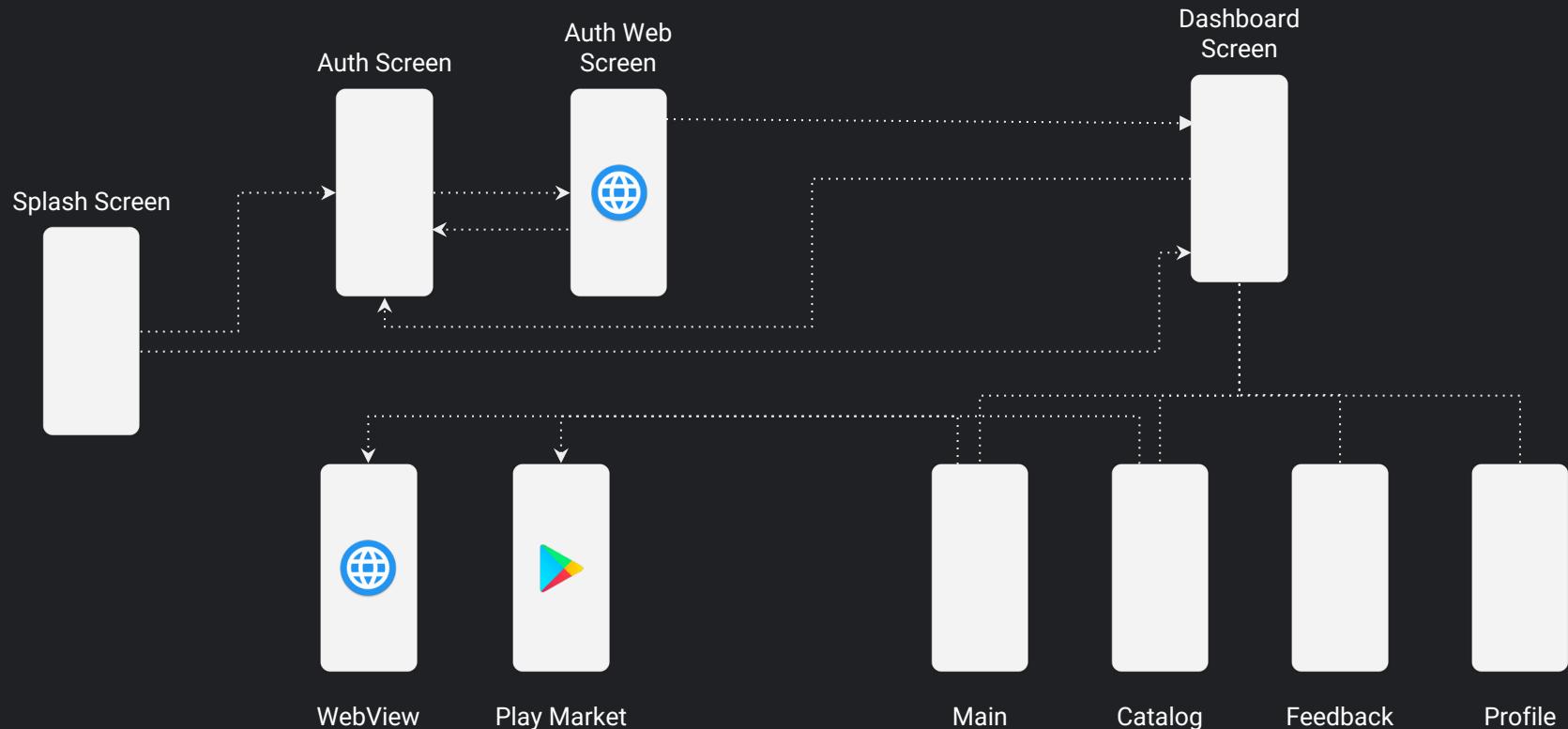
Architecture.Android (Old)



Architecture.Android (Old)



Architecture.Android (Old)



Architecture.Android (Old)

```
include ':feature-webview'  
include ':feature-splash'  
include ':feature-product-detail'  
include ':feature-splash'  
include ':feature-dashboard'  
include ':core-shared'  
include ':core-data-impl'  
include ':core-data-api'  
include ':core-utils'  
include ':feature-auth'  
include ':app'  
rootProject.name = "LeroyMerlin E"
```

Количество модулей: 11
Строк кода: 6999

Architecture.Multiplatform

```
include([
    ":common:common-main",
    ":common:common-root",
    ":common:common-dashboard",
    ":common:common-auth",
    ":common:common-splash",
    ":common:common-webview",
    ":common:common-markup",
    ":common:common-react_native_modules",
    ":common:common-data",
    ":common:common-compose",

    ":common:common-utils",
    ":common:compose-utils",

    ":common:navigation-compose",

    ":common:auth-compose",
    ":common:splash-compose",
    ":common:dashboard-compose",
    ":common:webview-compose",

    ":android",
    ":desktop"
])
```

Количество модулей: 22
Строк кода: 10420

Architecture.Multiplatform

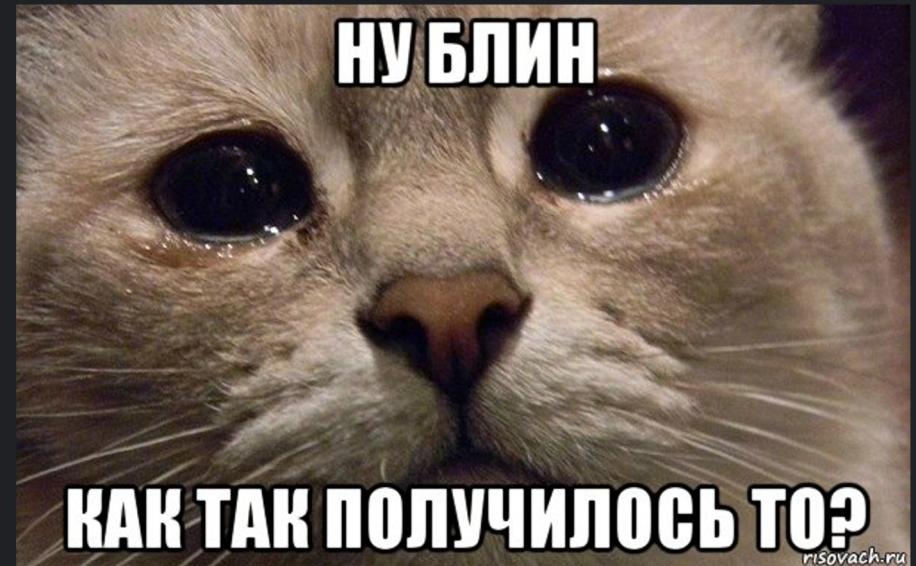
```
include(
    ":common:common-main",
    ":common:common-root",
    ":common:common-dashboard",
    ":common:common-auth",
    ":common:common-splash",
    ":common:common-webview",
    ":common:common-markup",
    ":common:common-react_native_modules",
    ":common:common-data",
    ":common:common-compose",

    ":common:common-utils",
    ":common:compose-utils",

    ":common:navigation-compose",

    ":common:auth-compose",
    ":common:splash-compose",
    ":common:dashboard-compose",
    ":common:webview-compose",

    ":android",
    ":desktop"
)
```

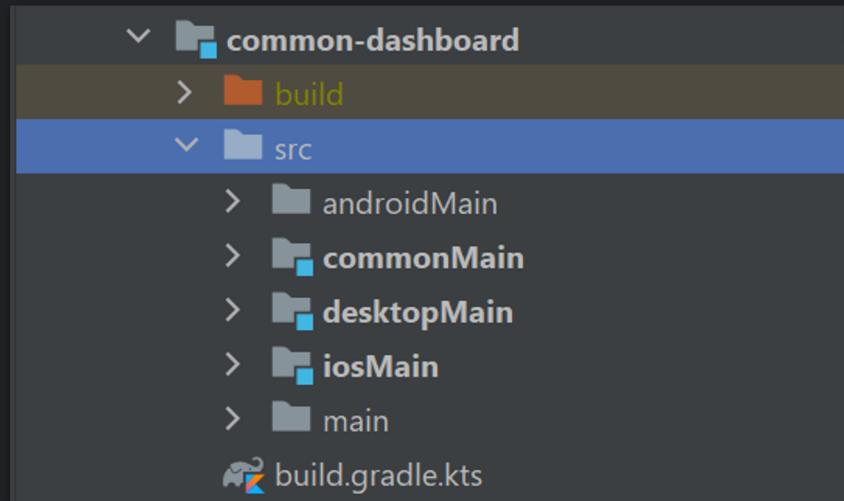


UI

Compose + iOS

Common

- > CommonMain. ViewModel
- > DesktopMain. Desktop UI
- > IOSMain. IOS UI
- > AndroidMain. Android UI



Compose + Common

- > Не собирается iOS
- > Магические ошибки в коде

```
plugins { this: PluginDependenciesSpecScope
    id( id: "multiplatform-setup")
    id( id: "multiplatform-compose-setup")
    id( id: "android-setup")
}
```

Compose + UI

- > Отдельный модуль под Compose
- > Отдельный модуль под сборку iOS
- > Отдельный модуль под сборку Compose

```
":common:common-dashboard",
":common:dashboard-compose",
":common:common-webview",
":common:webview-compose",
":common:common-scanner",
":common:scanner-compose",
```

ViewModel

```
abstract class KViewModel {

    private val coroutineTags = hashMapOf<String, CoroutineScope>()
    private val mainJobKey = "main.viewmodel.shared.coroutine.job"

    @Volatile
    private var isCleared = false

    val viewModelScope: CoroutineScope

    protected open fun onCleared() {}

    @MainThread
    fun clear()
        // Launch view model scope except you provide a new key
    fun launchNewScope(key: String = mainJobKey): CoroutineScope
}
```

```
abstract class KViewModel {

    private val coroutineTags = hashMapOf<String, CoroutineScope>()
    private val mainJobKey = "main.viewmodel.shared.coroutine.job"

    @Volatile
    private var isCleared = false

    val viewModelScope: CoroutineScope

    protected open fun onCleared() {}

    @MainThread
    fun clear()
        // Launch view model scope except you provide a new key
    fun launchNewScope(key: String = mainJobKey): CoroutineScope
}
```

```
abstract class BaseSharedViewModel<State, Action, Event>: KViewModel() {  
  
    fun viewStates(): CFlow<State?> = _viewStates.wrap()  
    fun viewActions(): CFlow<Action?> = _viewActions.wrap()  
  
    abstract fun obtainEvent(viewEvent: Event)  
}
```

```
abstract class BaseSharedViewModel<State, Action, Event>: KViewModel() {  
  
    fun viewStates(): CFlow<State?> = _viewStates.wrap()  
    fun viewActions(): CFlow<Action?> = _viewActions.wrap()  
  
    abstract fun obtainEvent(viewEvent: Event)  
}
```

```
abstract class BaseSharedViewModel<State, Action, Event>: KViewModel() {  
  
    fun viewStates(): CFlow<State?> = _viewStates.wrap()  
    fun viewActions(): CFlow<Action?> = _viewActions.wrap()  
  
    abstract fun obtainEvent(viewEvent: Event)  
}
```

```
abstract class BaseSharedViewModel<State, Action, Event>: KViewModel() {  
  
    fun viewStates(): CFlow<State?> = _viewStates.wrap()  
    fun viewActions(): CFlow<Action?> = _viewActions.wrap()  
  
    abstract fun obtainEvent(viewEvent: Event)  
}
```

KViewModel + Compose

```
@Composable
fun <T : KViewModel> ViewModel(factory: () -> T, content: @Composable (T) -> Unit) {
    // Instantiate KViewModel to use
    val viewModel = remember { factory.invoke() }

    // Show content
    content(viewModel)

    DisposableEffect(Unit) {
        onDispose {
            viewModel.clear()
        }
    }
}
```

```
@Composable
fun <T : KViewModel> ViewModel(factory: () -> T, content: @Composable (T) -> Unit) {
    // Instantiate KViewModel to use
    val viewModel = remember { factory.invoke() }

    // Show content
    content(viewModel)

    DisposableEffect(Unit) {
        onDispose {
            viewModel.clear()
        }
    }
}
```

```
@Composable
fun <T : KViewModel> ViewModel(factory: () -> T, content: @Composable (T) -> Unit) {
    // Instantiate KViewModel to use
    val viewModel = remember { factory.invoke() }

    // Show content
    content(viewModel)

    DisposableEffect(Unit) {
        onDispose {
            viewModel.clear()
        }
    }
}
```

```
@Composable
fun <T : KViewModel> ViewModel(factory: () -> T, content: @Composable (T) -> Unit) {
    // Instantiate KViewModel to use
    val viewModel = remember { factory.invoke() }

    // Show content
    content(viewModel)

    DisposableEffect(Unit) {
        onDispose {
            viewModel.clear()
        }
    }
}
```

ViewState + Compose

```
@Composable
fun <T> CFlow<T>.observeAsState(initial: T? = null): State<T?> {
    val state = remember { mutableStateOf(initial) }
    val flow = this

    DisposableEffect(this) {
        val observer = flow.watch { state.value = it }
        onDispose {
            observer.close()
        }
    }

    return state
}
```

```
@Composable
fun <T> CFlow<T>.observeAsState(initial: T? = null): State<T?> {
    val state = remember { mutableStateOf(initial) }
    val flow = this

    DisposableEffect(this) {
        val observer = flow.watch { state.value = it }
        onDispose {
            observer.close()
        }
    }

    return state
}
```

```
@Composable
fun <T> CFlow<T>.observeAsState(initial: T? = null): State<T?> {
    val state = remember { mutableStateOf(initial) }
    val flow = this

    DisposableEffect(this) {
        val observer = flow.watch { state.value = it }
        onDispose {
            observer.close()
        }
    }

    return state
}
```

Использование

```
@Composable
fun AuthScreen() {
    ViewModel({ AuthViewModel() }) { authViewModel ->
        val viewState = authViewModel.viewStates().observeAsState()
        val viewAction = authViewModel.viewActions().observeAsState()

        viewState.value?.let {
            AuthView(viewState = it)
        }

        viewAction.value?.let { ... }
    }
}
```

```
@Composable
fun AuthScreen() {
    ViewModel({ AuthViewModel() }) { authViewModel ->
        val viewState = authViewModel.viewStates().observeAsState()
        val viewAction = authViewModel.viewActions().observeAsState()

        viewState.value?.let {
            AuthView(viewState = it)
        }

        viewAction.value?.let { ... }
    }
}
```

```
@Composable
fun AuthScreen() {
    ViewModel({ AuthViewModel() }) { authViewModel ->
        val viewState = authViewModel.viewStates().observeAsState()
        val viewAction = authViewModel.viewActions().observeAsState()

        viewState.value?.let {
            AuthView(viewState = it)
        }

        viewAction.value?.let { ... }
    }
}
```

```
@Composable
fun AuthScreen() {
    ViewModel({ AuthViewModel() }) { authViewModel ->
        val viewState = authViewModel.viewStates().observeAsState()
        val viewAction = authViewModel.viewActions().observeAsState()

        viewState.value?.let {
            AuthView(viewState = it)
        }

        viewAction.value?.let { ... }
    }
}
```

Использование в iOS

```
private let authViewModel = AuthViewModel()

override func viewDidLoad() {
    super.viewDidLoad()

    renderUI()

    authViewModel.viewStates().watch { [weak self] state in
        guard let self = self, let state = state else { return }

        // View state logic here
    }

    authViewModel.viewActions().watch { [weak self] action in
        guard let self = self, let action = action else { return }

        // View action logic here
    }
}
```

```
private let authViewModel = AuthViewModel()

override func viewDidLoad() {
    super.viewDidLoad()

    renderUI()

    authViewModel.viewStates().watch { [weak self] state in
        guard let self = self, let state = state else { return }

        // View state logic here
    }

    authViewModel.viewActions().watch { [weak self] action in
        guard let self = self, let action = action else { return }

        // View action logic here
    }
}
```

Navigation

Требования

- > Поддержка всех платформ - JVM, Native, JS

Требования

- > Поддержка всех платформ - JVM, Native, JS
- > Легкая интеграция с Compose Multiplatform

Требования

- > Поддержка всех платформ - JVM, Native, JS
- > Легкая интеграция с Compose Multiplatform
- > Простота использования

Требования

- > Поддержка всех платформ - JVM, Native, JS
- > Легкая интеграция с Compose Multiplatform
- > Простота использования
- > Простота расширения

Требования

- > Поддержка всех платформ - JVM, Native, JS
- > Легкая интеграция с Compose Multiplatform
- > Простота использования
- > Простота расширения
- > Поддержка мультибэкстека

Требования

- > Поддержка всех платформ - JVM, Native, JS
- > Легкая интеграция с Compose Multiplatform
- > Простота использования
- > Простота расширения
- > Поддержка мультибэкстека
- > Поддержка платформенного функционала (hardware back button)



GitHub

[arkivanov/Decompose](#)



arkivanov/ Decompose



Kotlin Multiplatform lifecycle-aware business logic components (aka BLoCs) with routing functionality and pluggable UI (Jetpack Compose, SwiftUI, JS React, etc.),...



8

Contributors



8

Issues



17

Discussions



670

Stars



28

Forks



Собственное решение

Подготовка

```
interface ScreenHost {  
    fun prepareForDrawing()  
    fun draw(destinationPoint: DestinationPoint)  
}
```

Подготовка

```
// Desktop  
abstract class DesktopScreenHost constructor(  
    private val window: JFrame  
) : ScreenHost
```

Подготовка

```
// Desktop
abstract class DesktopScreenHost constructor(
    private val window: JFrame
) : ScreenHost

// Android
abstract class AndroidScreenHost constructor(
    private val composeActivity: ComponentActivity
) : ScreenHost
```

Подготовка

```
// Desktop
abstract class DesktopScreenHost constructor(
    private val window: JFrame
) : ScreenHost

// Android
abstract class AndroidScreenHost constructor(
    private val composeActivity: ComponentActivity
) : ScreenHost

// iOS
abstract class AppleScreenHost constructor(
    private val viewController: UIViewController
) : ScreenHost
```

Генерация графа

```
fun RootControllerBuilder.generateGraph() {  
    // Простой экран  
    destination(screen = "splash")  
}
```

Генерация графа

```
fun RootControllerBuilder.generateGraph() {  
    // Для создания флоу навигации  
    flow(name = "auth") {  
        destination(screen = "login")  
        destination(screen = "loginTwoFactor")  
    }  
}
```

Генерация графа

```
fun RootControllerBuilder.generateGraph() {
    // Для создания мультистэка (например, bottom navigation)
    multistack(name = "main") {
        flow(name = "main") {
            destination(screen = "feed")
            destination(screen = "detail")
        }

        flow(name = "favorite") {
            destination(name = "flow")
        }
    }
}
```

Генерация графа

```
fun RootControllerBuilder.generateGraph() {
    // Для создания мультистэка (например, bottom navigation)
    multistack(name = "main") {
        flow(name = "main") {
            destination(screen = "feed")
            destination(screen = "detail")
        }

        flow(name = "favorite") {
            destination(name = "flow")
        }
    }
}
```

Генерация графа

```
fun RootControllerBuilder.generateGraph() {
    // Для создания мультистэка (например, bottom navigation)
    multistack(name = "main") {
        flow(name = "main") {
            destination(screen = "feed")
            destination(screen = "detail")
        }

        flow(name = "favorite") {
            destination(name = "flow")
        }
    }
}
```

Использование

```
val screenHost = AppScreenHost(window)  
screenHost.prepareForDrawing()
```

```
val rootController = RootController(screenHost)  
rootController.setNavigationGraph { generateGraph() }  
rootController.launch(screen = "splash")
```

Navigation + Compose

Использование

```
class AppScreenHost(window: JFrame) : DesktopScreenHost(window) {  
  
    @Composable  
    override fun launchScreen(destinationPoint: DestinationPoint) {  
        val state = destinationPoint.rootController.backStackObserver.observeAsState()  
        state.value?.let { entry ->  
            when (entry.destination.destinationName()) {  
                // Draw root composables here  
  
                "splash" -> SplashScreen(entry.rootController)  
            }  
        }  
    }  
}
```

Использование

```
class AppScreenHost(window: JFrame) : DesktopScreenHost(window) {  
  
    @Composable  
    override fun launchScreen(destinationPoint: DestinationPoint) {  
        val state = destinationPoint.rootController.backStackObserver.observeAsState()  
        state.value?.let { entry ->  
            when (entry.destination.destinationName()) {  
                // Draw root composables here  
  
                "splash" -> SplashScreen(entry.rootController)  
            }  
        }  
    }  
}
```

```
@Composable
fun AuthFlow(rootController: RootController) {
    val flowRootController = rootController as FlowRootController
    val navigation = flowRootController.backStackObserver.observeAsState()
    val params = (navigation.value?.destination as? DestinationScreen)?.params

    navigation.value?.let { entry ->
        when (entry.destination.destinationName()) {
            "login" -> LoginScreen(entry.rootController)
            "twofactor" -> LoginCodeScreen(entry.rootController, params)
        }
    }
}
```

```
@Composable
fun AuthFlow(rootController: RootController) {
    val flowRootController = rootController as FlowRootController
    val navigation = flowRootController.backStackObserver.observeAsState()
    val params = (navigation.value?.destination as? DestinationScreen)?.params

    navigation.value?.let { entry ->
        when (entry.destination.destinationName()) {
            "login" -> LoginScreen(entry.rootController)
            "twofactor" -> LoginCodeScreen(entry.rootController, params)
        }
    }
}
```

```
@Composable
fun AuthFlow(rootController: RootController) {
    val flowRootController = rootController as FlowRootController
    val navigation = flowRootController.backStackObserver.observeAsState()
    val params = (navigation.value?.destination as? DestinationScreen)?.params

    navigation.value?.let { entry ->
        when (entry.destination.destinationName()) {
            "login" -> LoginScreen(entry.rootController)
            "twofactor" -> LoginCodeScreen(entry.rootController, params)
        }
    }
}
```

Call Place

```
@Composable
fun AuthScreen(
    rootController: RootController
) {
    ViewModel({ AuthViewModel() }) { authViewModel ->
        val viewAction = authViewModel.viewActions().observeAsState()

        // Нужно, чтобы выйти на уровень выше
        rootController.parentRootController?.launch("dashboard")

        // Нужно, чтобы запустить навигацию внутри данного графа
        rootController.launch("web", params = true)

        // Нужно, чтобы вернуться назад внутри данного графа
        rootController.popBackStack()
    }
}
```

```
@Composable
fun AuthScreen(
    rootController: RootController
) {
    ViewModel({ AuthViewModel() }) { authViewModel ->
        val viewAction = authViewModel.viewActions().observeAsState()
            // Нужно, чтобы выйти на уровень выше
        rootController.parentRootController?.launch("dashboard")

        // Нужно, чтобы запустить навигацию внутри данного графа
        rootController.launch("web", params = true)

        // Нужно, чтобы вернуться назад внутри данного графа
        rootController.popBackStack()
    }
}
```

```
@Composable
fun AuthScreen(
    rootController: RootController
) {
    ViewModel({ AuthViewModel() }) { authViewModel ->
        val viewAction = authViewModel.viewActions().observeAsState()
            // Нужно, чтобы выйти на уровень выше
        rootController.parentRootController?.launch("dashboard")

        // Нужно, чтобы запустить навигацию внутри данного графа
        rootController.launch("web", params = true)

        // Нужно, чтобы вернуться назад внутри данного графа
        rootController.popBackStack()
    }
}
```

```
@Composable
fun AuthScreen(
    rootController: RootController
) {
    ViewModel({ AuthViewModel() }) { authViewModel ->
        val viewAction = authViewModel.viewActions().observeAsState()

        // Нужно, чтобы выйти на уровень выше
        rootController.parentRootController?.launch("dashboard")

        // Нужно, чтобы запустить навигацию внутри данного графа
        rootController.launch("web", params = true)

        // Нужно, чтобы вернуться назад внутри данного графа
        rootController.popBackStack()
    }
}
```



GitHub

AlexGladkov/Odyssey



AlexGladkov/
Odyssey



Contributors



Issues



0

Stars



Forks



Resources



[icerockdev/moko-resources](https://github.com/icerockdev/moko-resources)



Style

Fronton

- > Единая дизайн-система
- > Доступна на всех платформах

Color

Базовая палитра цветов сочетает в себе вариативность в разных ситуациях и ограниченность использования.

Light mode Dark mode

Brand

Основные цвета бренда

primary #5A8030 rgb(90, 176, 48) Основной цвет	secondary #000000 rgb(0, 0, 0) Второстепенный	invert #FFFFFF rgb(255, 255, 255) Инвертированный

Text

Базовые цвета текста.

primary #333333 rgb(51, 51, 51) Основной текст	secondary #666666 rgb(102, 102, 102) Второстепенный текст	minor #999999 rgb(153, 153, 153) Незначительный текст	invert #FFFFFF rgb(255, 255, 255) Инвертированный

Typography

Интерфейс — это текст

Specification

Heading

Заголовки помогают привлечь внимание и донести основную мысль страницы, подзаголовки — структурируют и направляют.

Font Family	Size	Line height
Family Roboto (700)	28px	34px
Альтернативные шрифты:		
letter-spacing: auto;		
letter-spacing: +0.25px;		
font-weight: bold;		

h1

Family Roboto (700) Size 28px Line height 34px

Заголовок первого уровня — основной в интерфейсе.
line-height: 1.1em;
letter-spacing: auto;
font-weight: bold;

h2

Family Roboto (500) Size 22px Line height 27px

Заголовок второго уровня. Может использоваться как заголовок первого уровня, если не требуется сильный акцент или заголовок сам по себе.
line-height: 1.2em;
letter-spacing: auto;
font-weight: bold;

h3

Family Roboto (500) Size 16px Line height 24px

Заголовок третьего уровня.
line-height: 1.3em;
letter-spacing: -0.15px;
font-weight: bold;

Стройматериалы

Сухие смеси и грунтовки

Монтажные и кладочные смеси

Пескобетон

Fronton.Compose

```
FrontonTheme {  
    // Необходимо обернуть весь контент в тему  
}
```

Fronton.Compose

```
FrontonTheme {  
    // Необходимо обернуть весь контент в тему  
}  
  
object Fronton {  
    val colors: FrontonColors // Используется для цветов  
        @Composable  
        get() = LocalFrontonColors.current  
  
    val typography: FrontonTypography // Используется для типографии  
        @Composable  
        get() = LocalFrontonTypography.current  
}
```

Fronton.Compose

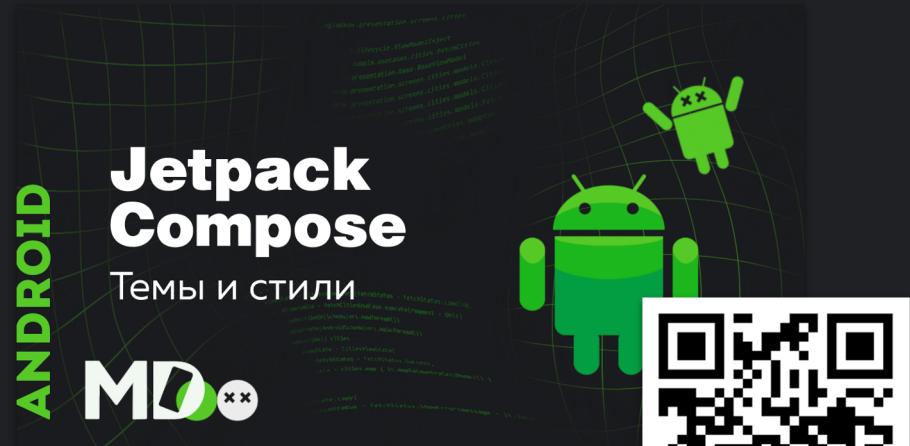
```
data class FrontonColors(  
    val textPrimary: Color,  
    val textSecondary: Color,  
    val textMinor: Color,  
    val textInvert: Color  
)
```

```
data class FrontonTypography(  
    val headings: Headings,  
    val body: Body,  
    val minor: Minor  
)
```

Fronton.Compose

```
data class FrontonColors(  
    val textPrimary: Color,  
    val textSecondary: Color,  
    val textMinor: Color,  
    val textInvert: Color  
)
```

```
data class FrontonTypography(  
    val headings: Headings,  
    val body: Body,  
    val minor: Minor  
)
```



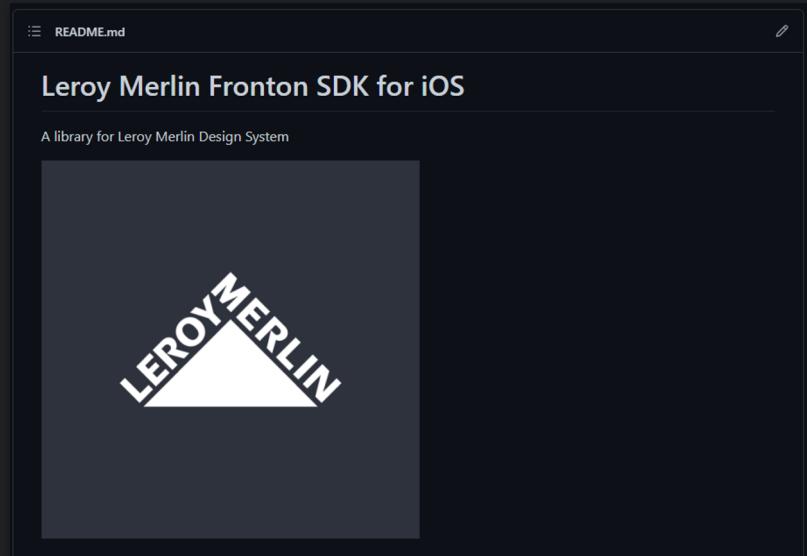
* Видео про Android

Fronton.iOS

```
target 'YourTarget' do
    pod 'Fronton-iOS', '~> 1.2.6'
// use latest release here
end

import Fronton_iOS
```

```
Fronton.Typography
Fronton.Colors
```



Strings

In Progress (Hardcode)

Images

Images.Compose

* Взято из <https://github.com/JetBrains/compose-jb>

```
@Composable  
expect fun imageResource(res: String): ImageBitmap
```

```
@Composable  
expect fun vectorResource(res: String): ImageVector
```

Images.Compose.Android

* Взято из <https://github.com/JetBrains/compose-jb>

```
@Composable
actual fun imageResource(res: String): ImageBitmap {
    val id = drawableId(res)
    return ImageBitmap.imageResource(id)
}
```

```
@Composable
actual fun vectorResource(res: String): ImageVector {
    val id = drawableId(res)
    return ImageVector.vectorResource(id)
}
```

Images.Compose.Android

* Взято из <https://github.com/JetBrains/compose-jb>

```
private fun drawableId(res: String): Int {  
    val imageName = res.substringAfterLast("/") .substringBeforeLast("." )  
    val drawableClass = R.drawable::class.java  
    val field = drawableClass.getDeclaredField(imageName)  
    val idValue = field.get(drawableClass) as Integer  
    return idValue.toInt()  
}
```

Images.Compose.Android

* Взято из <https://github.com/JetBrains/compose-jb>

```
private fun drawableId(res: String): Int {  
    val imageName = res.substringAfterLast("/") .substringBeforeLast("." )  
    val drawableClass = R.drawable::class.java  
    val field = drawableClass.getDeclaredField(imageName)  
    val idValue = field.get(drawableClass) as Integer  
    return idValue.toInt()  
}
```

Images.Compose.Android

* Взято из <https://github.com/JetBrains/compose-jb>

```
private fun drawableId(res: String): Int {  
    val imageName = res.substringAfterLast("/") .substringBeforeLast("." )  
    val drawableClass = R.drawable::class.java  
    val field = drawableClass.getDeclaredField(imageName)  
    val idValue = field.get(drawableClass) as Integer  
    return idValue.toInt()  
}
```

Images.Compose/Desktop

* Взято из <https://github.com/JetBrains/compose-jb>

```
@Composable  
actual fun imageResource(res: String) =  
    bitmapImage(res)
```

```
@Composable  
actual fun vectorResource(res: String): ImageVector =  
    vectorXmlResource(res)
```

Images.iOS

In Progress (Native)

DI



<https://github.com/Kodein-Framework/Kodein-DI>



Kodein-Framework/ Kodein-DI

Painless Kotlin Dependency Injection



40

Contributors

9

Issues

3k

Stars

159

Forks



Setup

Setup.Android + Desktop

```
CoreSharedSDK.init(  
    configuration = Configuration(  
        platformConfiguration = PlatformConfiguration(  
            androidContext = applicationContext  
        ),  
        platform = Configuration.Platform.Android,  
        version = "1.0",  
        build = "1",  
        isLoggingEnabled = true,  
    )  
)
```

Setup.iOS

```
CoreSharedSDK().doInit(  
    configuration:  
        Configuration(  
            platformConfiguration: .init(),  
            platform: Configuration.PlatformiOS.init(),  
            version: "1.0",  
            build: "1",  
            isLoggingEnabled: false  
        )  
)
```

Using

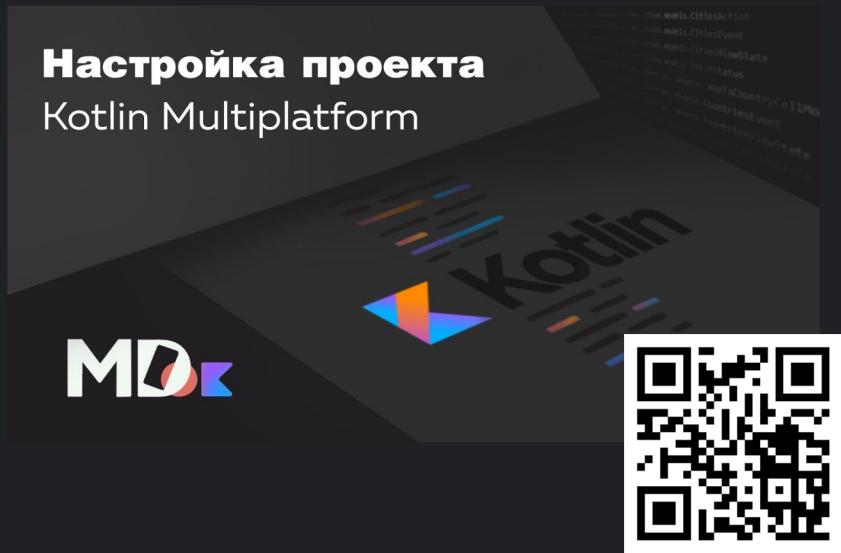
Using

```
private val authRepository = CoreSharedSDK.auth.authRepository  
// Android + Desktop
```

```
private let authRepository = CoreSharedSDK().auth.authRepository  
// IOS
```

DI

Дополнительная информация



Analytics + Platform

Analytics.Common

```
interface AnalyticsTracker {  
    fun trackEvent(event: AnalyticsEvent)  
}
```

```
interface AnalyticsEvent
```

Analytics.Common.Firebase

```
abstract class AnalyticsEventFB : AnalyticsEvent {
    abstract val name: String
    open var params: Map<String, Any> = emptyMap()

    override fun toString(): String {
        return "AnalyticsEventFB(name='$name', params=$params)"
    }
}
```

Analytics.Common.Firebase

```
abstract class AnalyticsEventFB : AnalyticsEvent {
    abstract val name: String
    open var params: Map<String, Any> = emptyMap()

    override fun toString(): String {
        return "AnalyticsEventFB(name='$name', params=$params)"
    }
}
```

Analytics.Common.Firebase

```
sealed class WebViewScreenEvents(  
    override val name: String,  
    override var params: Map<String, Any> = emptyMap()  
) : AnalyticsEventFB() {  
  
    data class WebApplicationInitialized(...) : WebViewScreenEvents(  
        name = "web_app_initialized",  
        params = hashMapOf(...)  
    )  
}
```

Analytics.Common.Firebase

```
sealed class WebViewScreenEvents(  
    override val name: String,  
    override var params: Map<String, Any> = emptyMap()  
) : AnalyticsEventFB() {  
  
    data class WebApplicationInitialized(...) : WebViewScreenEvents(  
        name = "web_app_initialized",  
        params = hashMapOf(...)  
    )  
}
```

Analytics.Common.Firebase

```
sealed class WebViewScreenEvents(
    override val name: String,
    override var params: Map<String, Any> = emptyMap()
) : AnalyticsEventFB() {

    data class WebApplicationInitialized(...) : WebViewScreenEvents(
        name = "web_app_initialized",
        params = hashMapOf(...)
    )
}
```

Analytics.Android

```
class FirebaseTracker @Inject constructor(  
    private val application: Application  
) : AnalyticsTracker {  
  
    private val firebaseAnalytics: FirebaseAnalytics by lazy {  
        FirebaseAnalytics.getInstance(application.applicationContext)  
    }  
  
    fun track(event: AnalyticsEventFB) {  
        // Здесь идет отправка в Firebase  
    }  
}
```

Analytics.iOS
Same

Using

Using

```
// Common
expect class RootContainer {
    fun prepareWithScreenHost()
}
```

Using

```
// Common
expect class RootContainer {
    fun prepareWithScreenHost()
}

// Android
actual class RootContainer(
    val composeActivity: ComponentActivity,
    val analyticsTracker: AnalyticsTracker,
    val performanceTracker: PerformanceTracker,
    val platformConfiguration: PlatformConfiguration
)
```

Using

```
// Common
expect class RootContainer {
    fun prepareWithScreenHost()
}

// Desktop
actual class RootContainer(
    val window: JFrame,
    val analyticsTracker: AnalyticsTracker,
    val performanceTracker: PerformanceTracker
)

// Android
actual class RootContainer(
    val composeActivity: ComponentActivity,
    val analyticsTracker: AnalyticsTracker,
    val performanceTracker: PerformanceTracker,
    val platformConfiguration: PlatformConfiguration
)
```

Using

```
// Common
expect class RootContainer {
    fun prepareWithScreenHost()
}

// Desktop
actual class RootContainer(
    val window: JFrame,
    val analyticsTracker: AnalyticsTracker,
    val performanceTracker: PerformanceTracker
)

// Android
actual class RootContainer(
    val composeActivity: ComponentActivity,
    val analyticsTracker: AnalyticsTracker,
    val performanceTracker: PerformanceTracker,
    val platformConfiguration: PlatformConfiguration
)

// iOS
actual class RootContainer(
    val window: JFrame,
    val analyticsTracker: AnalyticsTracker,
    val performanceTracker:
        PerformanceTracker
)
```

Device In Progress (Platform)

Testing

In Progress (Platform + Common Tests)

Testing.Common

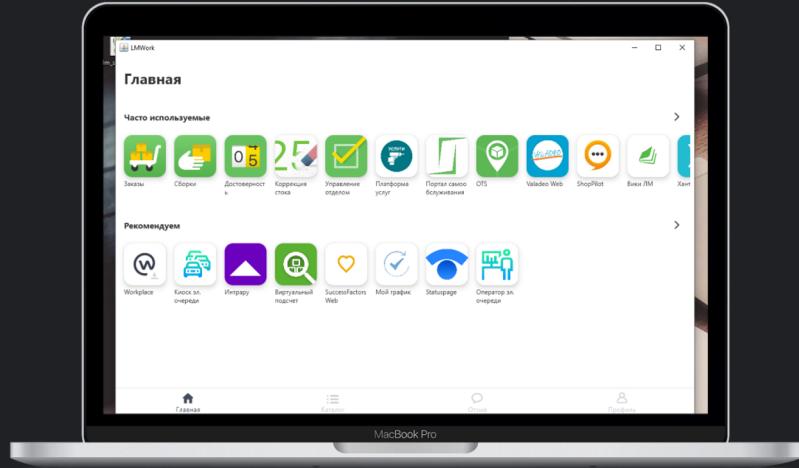
DIY Mobile Day в формате live coding

The graphic features a green header with white text. Below it, four circular portraits of speakers are arranged horizontally. Each portrait has a name below it. To the right of the speakers is a white smartphone displaying a product image of a cordless drill. A QR code is located at the bottom right.

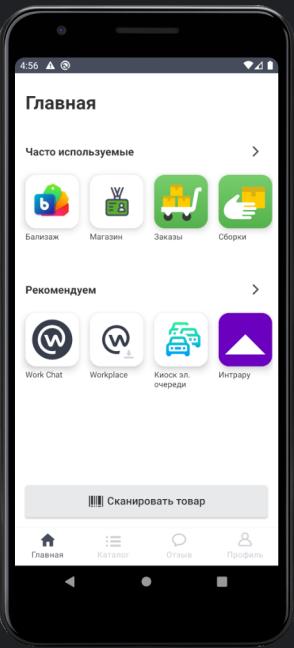
Speaker	Description
Мурагер Жайлхан	
Вячеслав Корниенко	
Алексей Гладков	
Катя Петрова	

A QR code is located at the bottom right of the graphic.

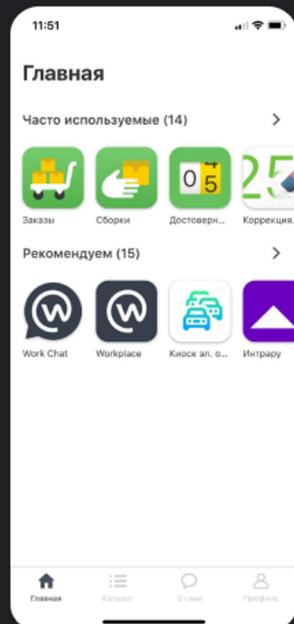
Результат



Desktop



Android



iOS

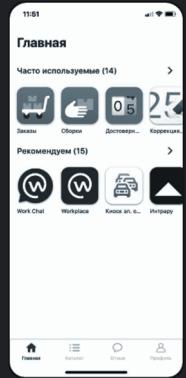
Результат



Desktop



Android



iOS

JS

In Progress

Заключение

Заключение . Минусы

- > Очень мало готового

Заключение . Минусы

- > Очень мало готового
- > Много нюансов каждой платформы (Desktop WebView)

Заключение . Минусы

- > Очень мало готового
- > Много нюансов каждой платформы (Desktop WebView)
- > Высокий уровень входа. Необходим широкий кругозор

Заключение . Плюсы

- > Ускорение разработки. Доступность всех платформ

Заключение . Плюсы

- > Ускорение разработки. Доступность всех платформ
- > Легкая интеграция с платформенными фичами

Заключение . Плюсы

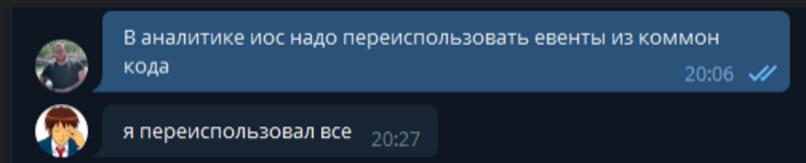
- > Ускорение разработки. Доступность всех платформ
- > Легкая интеграция с платформенными фичами
- > Нативная скорость, надежность

Заключение . Плюсы

- > Ускорение разработки. Доступность всех платформ
- > Легкая интеграция с платформенными фичами
- > Нативная скорость, надежность
- > Kotlin :)

Спасибо за внимание

Вопросы?



Контакты для связи

Email: bob298@yandex.ru

Youtube: MobileDeveloper

Telegram: @mobiledevnews