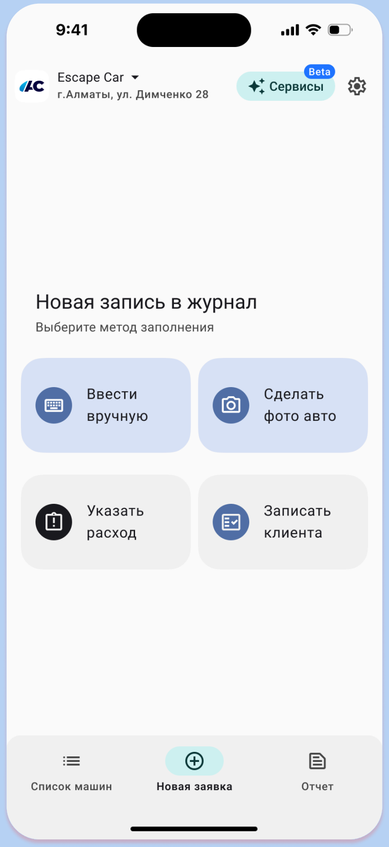
**Compose Multiplatform** - пет-проект
который приносит денюжки

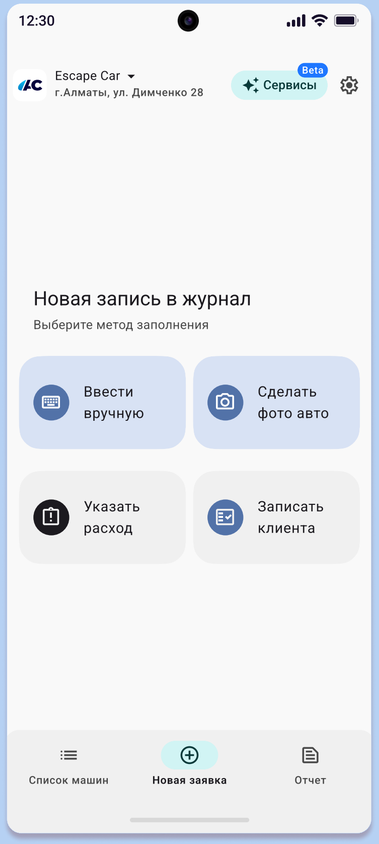
## ЧУТОК О СЕБЕ

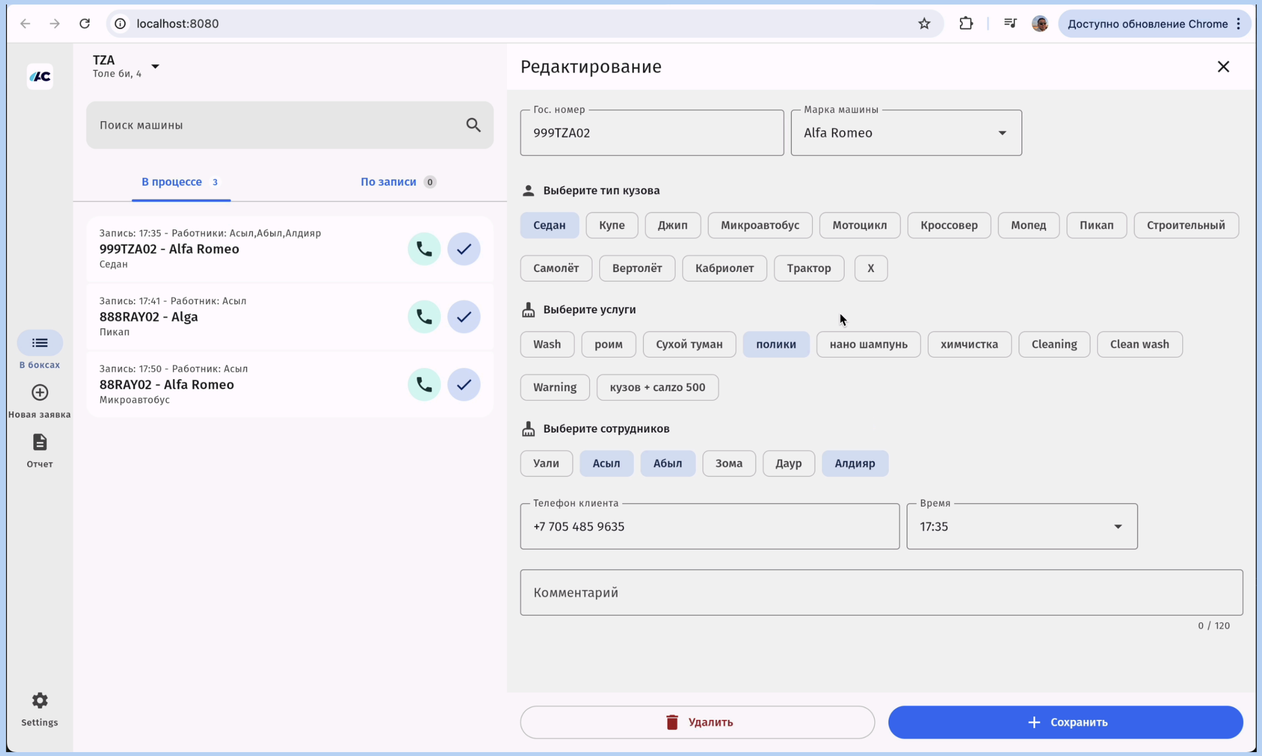
QIC DIGITAL HUB

- Сейчас Senior Android Developer в QIC
- До Android Teamlead в Globerce Capital (Freedom Business)
- Еще до ex-Choco, и разрабатывал разные проекты в Eventyr

# ЧУТОК ПРО ПРОЕКТ



## COMPOSE MULTIPLATFORM

- Платформенная логика изолирована, UI — работает на **SKIKO**
- Один UI-код для всех платформ (commonMain)
- Нативные API вызываются через expect/actual
- Компилируется в нативные бинари (не интерпретируется)

## ПРО TARGET-Ы

| Target | Что это? |
|---|---|
| **androidMain** | Android — JVM-байткод, стандартный Jetpack Compose |
| **iosX64** | iOS Simulator на Mac с Intel |
| **iosArm64** | iOS реальное устройство (физический iPhone/iPad) |
| **iosSimulatorArm64** | iOS Simulator на Mac с Apple Silicon (M1/M2/M3 и т.д.) |
| **nativeMain** | Общий код для нативных платформ (iOS + другие) |
| **wasmJsMain** | Kotlin/WASM — компилируется в WebAssembly |
| **jsMain (IR)** | Kotlin/JS — компилируется в JavaScript |
| **commonMain** | Общий код — работает на всех платформах |

# EXPECT / ACTUAL — КАК ЭТО РАБОТАЕТ

```
expect fun getPlatform(): Platform
expect fun isKeyboardOpenFlow(): StateFlow<Boolean>
expect fun Settings(): Settings
expect fun copyToClipboard(text: String)
expect fun rememberIntentOperations(): IntentOperations
@Composable expect fun CameraContent(...)
@Composable expect fun openUrl(urlString: String): Boolean
```

# IOS — КАК ЭТО РАБОТАЕТ ИЗНУТРИ

```
fun MainViewController(): UIViewController = ComposeUIViewController { App() }
```

```
struct ContentView: UIViewControllerRepresentable {
    func makeUIViewController(context: Context) -> UIViewController {
        return MainKt.MainViewController()  // вызов Kotlin
    }
}
```

# KOTLININTEROP — ВЫЗОВ НАТИВНЫХ IOS API

```
@OptIn(ExperimentalForeignApi::class)
import platform.UIKit.*
import platform.Foundation.*
import platform.CoreGraphics.*
import platform.Foundation.*
import platform.darwin.*
import kotlinx.cinterop.*

fun NSData.toByteArray(): ByteArray {
    val result = ByteArray(this.length.toInt())
    result.usePinned { pinned →
        val ptr = pinned.addressOf(0).reinterpret<ByteVar>()
        this.getBytes(ptr, this.length)
    }
    return result
}
```

# БАГ С ФОКУСОМ В CAMERACONTENT — ИСТОРИЯ И ПРИЧИНА

```
@Composable expect fun CameraContent(...)
```

```
private fun forceResetWindow() {
    dispatch_async(dispatch_get_main_queue()) {
        UIApplication.sharedApplication.keyWindow?.let { window →
            val root = window.rootViewController
            window.rootViewController = null  // сбрасываем
            window.setNeedsLayout()
            window.layoutIfNeeded()
            window.rootViewController = root  // восстанавливаем
            window.setNeedsLayout()
            window.layoutIfNeeded()
        }
    }
}
```

## WEB — ПОЧЕМУ НАЧАЛ С KOTLIN-WASM

- WebAssembly — низкоуровневый байткод, выполняется браузером напрямую
- Теоретически быстрее JS (нет парсинга, ближе к нативной скорости)
- JetBrains активно продвигала его как "будущее" CMP для Web
- Полная поддержка Skia через CanvasKit в WASM
- Типы данных из Kotlin напрямую → WASM без лишней конвертации

## KOTLIN JS — ПОЧЕМУ ПЕРЕСЕЛ И В ЧЁМ ОТЛИЧИЕ

СОВМЕСТИМОСТЬ:

JS работает везде — любой браузер, любая версия.

ЭКОСИСТЕМА:

Все нужные библиотеки имеют JS-артефакты

ОТЛАДКА:

Source maps, нормальные стектрейсы в DevTools, hot reload.

ЗРЕЛОСТЬ:

Kotlin/JS IR — стабильный компилятор с 1.7.x.

## РАЗЛИЧИЕ KOTLIN JS И WASM В ПРОЕКТЕ

| Параметры | Kotlin/JS (IR) | Kotlin/WASM |
|---|---|---|
| **Вывод при сборке** | .js файл | .wasm + .js glue |
| **Runtime** | V8 / SpiderMonkey / JavaScriptCore | WasmGC-совместимый браузер |
| **Размер** | Меньше | Больше |
| **Скорость выполнения** | Просто хороший (JIT) | Потенциально выше (почти нативка) |
| **Браузерная поддержка** | Практически везде | Chrome 119+, FF 120+, Safari 18+ |
| **Interop с JS** | Прямой | Через extern |
| **Зрелость** | Относительно стабильный | Хоть и beta (все равно экспериментальны) |
| **Доступность либ** | Практически все +- | Ограниченная |

## ПОЧЕМУ WEB-СБОРКА ТАКАЯ ДОЛГАЯ

```
./gradlew jsBrowserDistribution
```

- KOTLIN → JS КОМПИЛЯЦИЯ
- WEBPACK BUNDLING
- KOTLIN DAEMON
- ВСЕ ЗАВИСИМОСТИ

## ПОЧЕМУ WEB-СБОРКА ТАКАЯ ДОЛГАЯ

```
./gradlew jsBrowserDistribution
```

- KOTLIN → JS КОМПИЛЯЦИЯ
- WEBPACK BUNDLING
- KOTLIN DAEMON
- ВСЕ ЗАВИСИМОСТИ

```
./gradlew jsBrowserDistribution --no-daemon --max-workers=4 --parallel
```

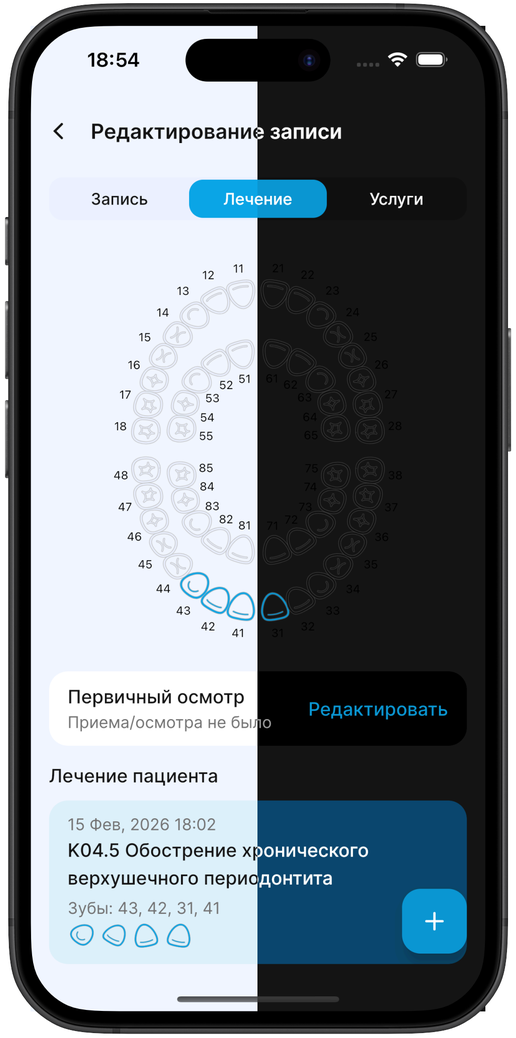
## СЛОЖНОСТИ МЕТОДА СБОРКИ

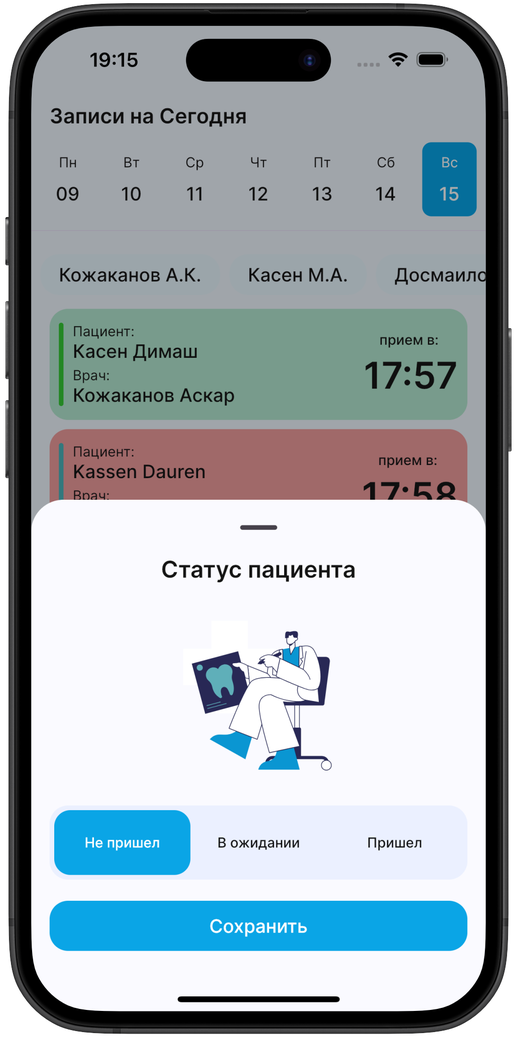
- Build types/build flavors с build-logic = сложно и не получиться
- Toggle pick by cmd в .properties - вери хард
- BuildKonfig как решение

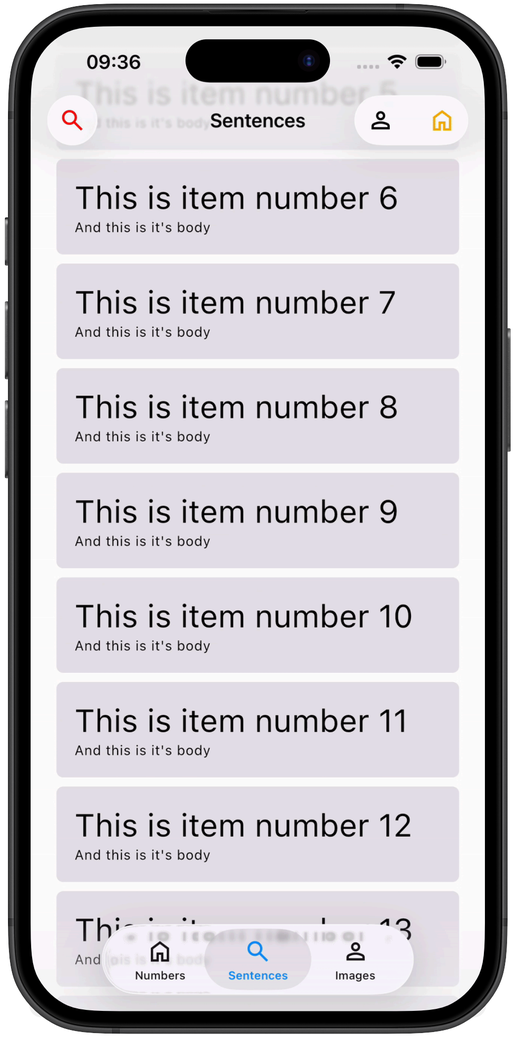
## КАК ИТОГ

- Очень многое можно портировать из Jetpack Compose
- Kotlin/Native + cinterop, хороший иструмент
- Есть проблемы с window
- Есть баги и недоробатоки с нативкой и их много

# БУДУ ЛИ Я ПРОДОЛЖАТЬ?



## РЕСУРСЫ В AQUACRM

- ktor - для работы с сетью
- navigation3 - для работы с навигацией
- com.russhwolf:multiplatform-settings - для хранение ключ значений
- org.jetbrains.kotlinx:kotlinx-datetime - для работы с датами