Что ваша команда получит от ревью кода

1

Команды, только начавшие работать с мультиплатформой, могут допускать неочевидные для новичков ошибки. Эти ошибки становятся заметны только в момент выхода на этап интеграции со второй платформой, потому что вынесенный код со стороны iOS выглядит не так, как разработчики этого ожидают.

2

Мы можем помочь избежать этих проблем. В ревью мы рассказываемо нетипичных способах решения задач и улучшения кода, о которых технические специалисты без опыта работы с мультиплатформой могут даже не догадываться. Мы расскажем, для каких задач лучше воспользоваться готовым решением и как лучше спроектировать публичное API, чтобы обе платформы удобно интегрировались.

3

В этом документе мы собрали общие ошибки начинающих команд. А на примере ревью, которое мы делали для компании Footballco, вы увидите частные ошибки и их решение.

Важно: ревью в примере содержит в себе техническую информацию. Если вы руководитель — покажите его своим специалистам, чтобы понять, нужен ли вашей команде такой же анализ.

Наш опыт в KMM

Мы занимаемся разработкой мобильных приложений на мультиплатформе более четырех лет. За это время мы реализовали более 20 проектов: от небольшого приложения для правильного питания до сервисов аренды мотоциклов.

Мы написали набор собственных KMM-библиотек — **MOKO**, которые активно развиваем и используем, и с начала 2022 года обучаем студентов работе с мультиплатформой с дальнейшим приемом на работу.

Распространенные ошибки новичков в KMM

Мало знакомы с экосистемой библиотек и дополнительных инструментов

Есть множество готовых библиотек и инструментов, которые помогают упростить код, увеличить его надежность и ускорить разработку приложения.

Не знают, что можно вынести в общий код

Некоторые функции непросто реализовать в общем коде, и сначала может показаться, что это невозможно. Понимание таких моментов приходит с опытом.

Делают ошибки при написании общего кода

В результате после компиляции со стороны iOS можно получить неудобный для использования API. Всплывают все эти ошибки в момент выхода на этап интеграции со второй платформой.

Делают ошибки при реализации многопоточности

Это свойство в iOS- и Android-частях реализуют по-разному, что может привести к ошибкам на этапе запуска приложения.

Как происходит ревью и какой результат получает клиент. Кейс Footballco

Footballco — это зарубежный сервис для просмотра онлайн-трансляций. Команда клиента уже сделала общую логику между Android и iOS и несколько готовых экранов. Прежде чем распространить технологию на весь проект, клиент пришел к нам как к экспертам по KMM, чтобы мы оценили код и подсказали, где есть ошибки.

Изучаем весь код. Основываясь на большом опыте мультиплатформенной разработки, наш специалист ищет в коде ошибки и неэффективные участки, которые можно улучшить.

Составляем ревью. В нем наш специалист описывает все, что удалось найти на предыдущем этапе, с объяснениями, почему это неправильно и как сделать лучше. Также мы даем ссылки на опенсорс-библиотеки, которые помогают быстро решить определенные задачи.

Помогаем клиенту внедрить наши рекомендации. Часть наших рекомендаций команда Footballco внедрила сама, а по непонятным моментам снова обратилась к нам. Мы созвонились и дали ребятам дополнительные пояснения.

Вот как выглядит текст, который наш специалист отправил клиенту

1. Gradle-конфигурацию проекта можно упростить, используя **Gradle convention plugin'ы**. Пример **объявления** и **использования** можно посмотёреть в нашей moko-template.

2. Чтобы корректно работали подсказки IDEA в iosMain sourceSet'е, вам нужно включить **Hierarchical Multiplatform**. Я вижу, что в проекте выключен кеш Kotlin/Native. Это плохое решение, потому что эти кеши значительно сокращают время сборки debug-варианта.

3. Чтобы улучшить время сборки, я рекомендую убрать флаг transitiveExport = true и заменить на указание export'а только тех зависимостей, которые нужны будут со стороны Swift.

4. Помимо влияния на время сборки, лишний экспорт также плохо влияет на размер бинарника, потому что происходит генерация proxy-объектов objc для Kotlin Objective C interop. **Вот видео** о размере бинарника.

5. Также для ускорения сборки вы можете свериться **с документацией**. Но я больше не вижу применимых для вашего проекта рекомендаций.

6 Чтобы избавиться от вручную написанных для iOS wrapper'ов над Flow, вы можете использовать KMP-NativeCoroutines — это решение автоматически генерирует аналогичные wrapper'ы и поддерживает вызов со стороны Swift как Publisher'ы Combine.

7 В наших проектах мы также переносим AndroidManifest.xml из main sourceSet'а в androidMain sourceSet. Это не влияет ни на что, но делает расположение Android-файлов более логичным. Чтобы сделать это, вы можете использовать плагин dev.icerock.mobile.multiplatform.android-manifest или написать свой convention-плагин с такой логикой.

8 Чтобы снизить вероятность ошибок работы с заморозкой при многопоточной работе, рекомендуем выполнять вызов freeze в init-блоке объекта сразу после полной его конфигурации. В таком случае, если есть лишние заморозки, то вы узнаете об этом сразу при запуске приложения, а не после какого-то времени работы приложения.

9 При ревью я обратил внимание на проблемы с запуском iOS-приложения, потому что Kotlin-фреймворк не был включен в само приложение. Я изучил проблемы и определил, что cocoapods gradle plugin настроен некорректно. Gradle-задача podspec настроена на создание static framework'а, но позже компиляция фреймворка перенастраивается в dynamic framework. Я поправил это, и теперь задача podspec генерирует корректную dynamic-framework-заглушку, после чего cocoapods корректно настраивает интеграцию с iOS-проектом и все компилируется без лишних вызовов pod install.

Также, возможно, будут полезны следующие советы:

1. Вы можете реализовать работу с хранением key-value, используя Settings и proxy-класс со строгими типами для свойств, как **тут**.

2. Для мест, как EditionManager#currentEdition, я бы использовал MutableStateFlow вместо StateFlow + setter.

3. Если вы используете OpenAPI-спецификацию для описания своего REST API, вы можете генерировать весь код сетевого слоя (запросы, модели, сериализацию), используя **moko-network**.

4. Я вижу, что вы используете ViewModel'и для Android. В своих проектах мы перенесли ViewModel'и в общий код, используя **moko-mvvm**. За счет использования общей ViewModel'и, мы получаем корректную работу с CoroutineScope для обеих платформ и единую логику для каждого экрана. Для интеграции ViewModel'ей со SwiftUI можно также использовать StateFlow + KMP-NativeCoroutines.

$ Стоимость и сроки

На анализ и составление плана обычно уходит от двух до пяти дней, а стоимость начинается от 100 000 рублей. Аудит для Footballco занял три дня и стоил ровно 100 000 рублей.

НАШИ КОНТАКТЫ

ICEROCK

Веб-сайт:

icerockdev.ru

Почта:

mobiledev@icerockdev.com

Адреса:

Новосибирск

ул. Советская, д. 23, офис 301
+7 (383) 373-29-19

Москва

Мира, 1
+7 (495) 109-73-29