**5. # ADR - Выбор фреймворка для разработки клиента**

Участники: Геворгян Арсен, Афанасьев Никита, Сахабутдинов Тимур

Дата: 25.09.2024

Статус: принят

**## Контекст**

Мы разрабатываем музыкальное стриминговое приложение для Android с возможностью расширения на другие платформы (iOS, Web) в будущем. Наша команда обладает опытом работы с Kotlin и хочет использовать его как для серверной, так и для клиентской части проекта. Мы стремимся к максимальной эффективности разработки, минимизации кода для разных платформ и поддержанию высокой производительности, особенно в обработке потокового аудио. Ограничения включают время на разработку и поддержку нескольких платформ, а также необходимость быстрого обучения новым инструментам и фреймворкам.

**## Рассматриваемые варианты**

**Native (Android/iOS/Web)**:

* **Плюсы**: Полная оптимизация под платформу, доступ ко всем нативным API, высокая производительность и отзывчивость.
* **Минусы**: Необходимость разработки и поддержки отдельных кодовых баз для каждой платформы (Android, iOS, Web), что увеличивает время и затраты на разработку.

**React Native**:

* **Плюсы**: Позволяет писать код один раз и использовать его на нескольких платформах. Поддержка большого количества библиотек и сообществ, что облегчает разработку.
* **Минусы**: Может возникнуть необходимость писать нативный код для специфичных функций, производительность может быть ниже, чем у нативного приложения.

**Compose Multiplatform**:

* **Плюсы**: Позволяет использовать один и тот же код для Android и iOS, использует Kotlin, что позволяет использовать ваши знания. Высокая производительность и возможность доступа к нативным API.
* **Минусы**: Это относительно новый подход, поэтому может быть меньше ресурсов и библиотек по сравнению с более зрелыми фреймворками.

**## Решение**

Compose Multiplatform(Jetpack compose)

**## Обоснование**

Мы выбрали Compose Multiplatform, поскольку он позволяет эффективно разрабатывать кроссплатформенные приложения, используя один код на Kotlin, что упрощает процесс разработки и снижает затраты на поддержку. Это решение также обеспечивает высокую производительность и доступ к нативным API, что критично для создания отзывчивого музыкального стриминга. Кроме того, наша команда уже обладает опытом работы с Kotlin, что делает переход на этот фреймворк более плавным и эффективным.

**## Последствия**

Плюсы:

* Кроссплатформенность
* Удобный синтаксис
* Полная совместимость с Kotlin, что позволяет использовать уже существующие навыки команды и легко интегрировать с серверной частью на Kotlin.

Минусы:

* Compose Multiplatform все еще является относительно новым фреймворком, что может привести к отсутствию зрелых библиотек и инструментов по сравнению с более устоявшимися решениями.
* Сравнительно меньше примеров, учебных материалов и документации по Compose Multiplatform, что может усложнить процесс обучения и решения возникающих проблем.

**## Риски**

Ограниченная поддержка библиотек и инструментов

Меры: Подбирать и адаптировать доступные библиотеки или игнорировать и писать кастомные решения.

Меньше документации и примеров

Меры: Уделять больше времени на обучение и исследования

**## Затронутые области**

Клиентская часть приложения, UI/UX дизайн

**## История**

25.09.24: Первоначальное обсуждение вариантов фреймворков для разработки клиента