## ADR - Выбор фреймворка для разработки клиента фотохостинга

Участники: Зыкин Алексей  
Дата: 5 декабря 2024  
Статус: -

**Контекст**

Разрабатывается мобильное и веб-приложение для фотохостинга, аналогичное Instagram. Основные требования включают высокую производительность, отзывчивый интерфейс, возможность кроссплатформенной разработки и легкость в поддержке. Также важно учитывать опыт команды с различными технологиями и наличие сообщества для поддержки.

**Рассматриваемые варианты**

**React (для веба)** - Высокая производительность, но может дополнительно потребоваться Redux.

**React Native (для мобильных приложений)** - Кросс-платформенность, хорошая производительность, ограничения в доступе к некоторым нативным API.

**Vue.js** - Хороший фреймворк, но у команды нет достаточного опыта, что может увеличить время на обучение.

**Flutter** - Перспективный, но требует изучения Dart и имеет меньшую популярность среди разработчиков.

**Compose Multiplatform** - Новая технология, но все еще находится в стадии активного развития.

**Native (Android/iOS)** - Высокие затраты на разработку и поддержку, так как требуется писать отдельный код для каждой платформы.

**Решение**

Выбрано решение: **React** для веба и **React Native** для мобильных приложений.

**Обоснование**

**React**Плюсы: Широкое сообщество и множество библиотек для интеграции; высокая производительность благодаря виртуальному DOM; легкость в создании компонентов и переиспользовании кода.  
Минусы: Может потребоваться дополнительная настройка для управления состоянием (например, Redux).

**React Native**Плюсы: Кросс-платформенность позволяет использовать один код для iOS и Android; хорошая производительность, близкая к нативным приложениям; возможность использования существующих библиотек React.  
Минусы: Ограничения в доступе к некоторым нативным API по сравнению с полностью нативными приложениями.

**Последствия**

* Упрощение разработки благодаря единому подходу к созданию интерфейсов
* Увеличение скорости разработки благодаря возможности переиспользования компонентов
* Возможные сложности с интеграцией нативных функций, которые могут потребовать дополнительных усилий

**Риски**

* Зависимость от экосистемы React и React Native, что может привести к проблемам, если сообщество не будет активно развиваться
* Возможные проблемы с производительностью на старых устройствах
* Необходимость в дополнительных тестах для обеспечения кросс-платформенной совместимости

Можно минимизировать риски через:

* Регулярные обновления библиотек и фреймворков
* Тестирование на различных устройствах и платформах
* Использование статического анализа кода для повышения качества

**Затронутые области**

* Интерфейс пользователя (UI) для веба и мобильных приложений
* Логика управления состоянием
* Интеграция с бэкендом

**История**

Обсуждение началось с анализа требований проекта и существующих технологий. Команда пришла к выводу, что **React** и **React Native** обеспечивают оптимальное сочетание производительности, гибкости и доступности ресурсов. После анализа альтернативных решений было принято окончательное решение о выборе **React** и **React Native**.