ORES
Научно-практический электронный журнал
Оригинальные исследования (ОРИС), *ISSN 2222-0402*
www.ores.su
original-research.ru

УДК 004.4

СРАВНИТЕЛЬНЫЙ АНАЛИЗ КРОССПЛАТФОРМЕННЫХ ТЕХНОЛОГИЙ ДЛЯ РАЗРАБОТКИ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ

Мишагин Дмитрий Владимирович
магистрант
Национальный исследовательский университет ИТМО, Санкт-Петербург, Российская Федерация
+79195759358
mi5ha6in@gmail.com

Аннотация

Использование кроссплатформенных инструментов позволяет реализовать только одно приложение, которое затем развертывается или экспортируется на все поддерживаемые целевые платформы. Такой подход экономит время и позволяет разработчикам использовать общую базу кода на едином языке программирования для нескольких мобильных или десктопных операционных систем. Несмотря на это преимущество, кроссплатформенные подходы также могут иметь свои недостатки, например, ограниченный доступ к аппаратным ресурсам устройства, высокое использование памяти, повышенная нагрузка на вычислительные устройства. Статья посвящена исследованию применения кроссплатформенных технологий, описанию их достоинств по сравнению с нативными приложениями. Проведен анализ исследований российских и зарубежных ученых по теме работы. Были рассмотрены архитектуры и принципы работы кроссплатформенных технологий, их отличие от нативных приложений. Разработаны критерии сравнения технологий и проведен сравнительный анализ.

Ключевые слова: кроссплатформенные приложения, веб-технологии, нативные приложения, React Native, Flutter, Ionic.

COMPARATIVE ANALYSIS OF CROSS-PLATFORM TECHNOLOGIES FOR MOBILE APPLICATION DEVELOPMENT

Dmitry V. Mishagin
undergraduate
National Research University ITMO, St. Petersburg, Russian Federation
+79195759358
mi5ha6in@gmail.com

ABSTRACT

Using cross-platform tools allows you to implement only one application, which is then deployed or exported to all supported target platforms. This approach saves time and allows developers to use a common code base in a single programming language for several mobile or desktop operating systems. Despite this advantage, cross-platform approaches can also have their drawbacks, for example, limited access to the device's hardware resources, high memory usage, and increased load on computing devices. The article is devoted to the study of the use of cross-platform technologies, a description of their advantages compared to native applications. The analysis of studies of Russian and foreign scientists on the topic of work. The architectures and principles of cross-platform technologies, their difference from native applications were considered. Criteria for comparing technologies are developed and a comparative analysis is carried out.

Keywords: cross-platform applications, web technologies, native applications, React Native, Flutter, Ionic.

Выбор технологий для проектирования кроссплатформенных мобильных приложений основан на их популярности среди разработчиков. На рисунке 1 представлена диаграмма используемых фреймворков за 2019 год по данным сайта Statista.

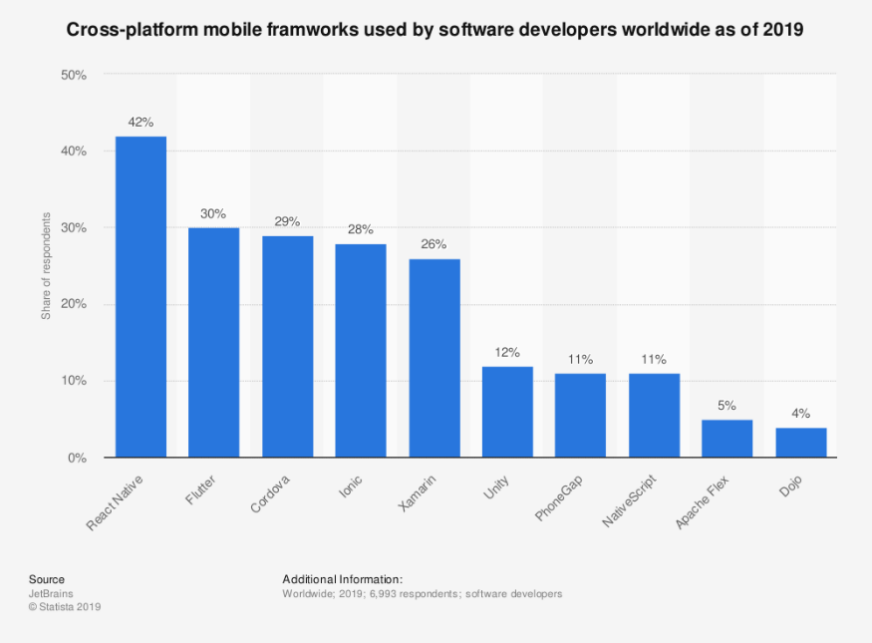

Рисунок 1. *Диаграмма используемых фреймворков за 2019 год [1]*

На рисунке 2 показана динамика популярности технологий на основе ресурса Google Trends за последние 12 месяцев по всему миру.

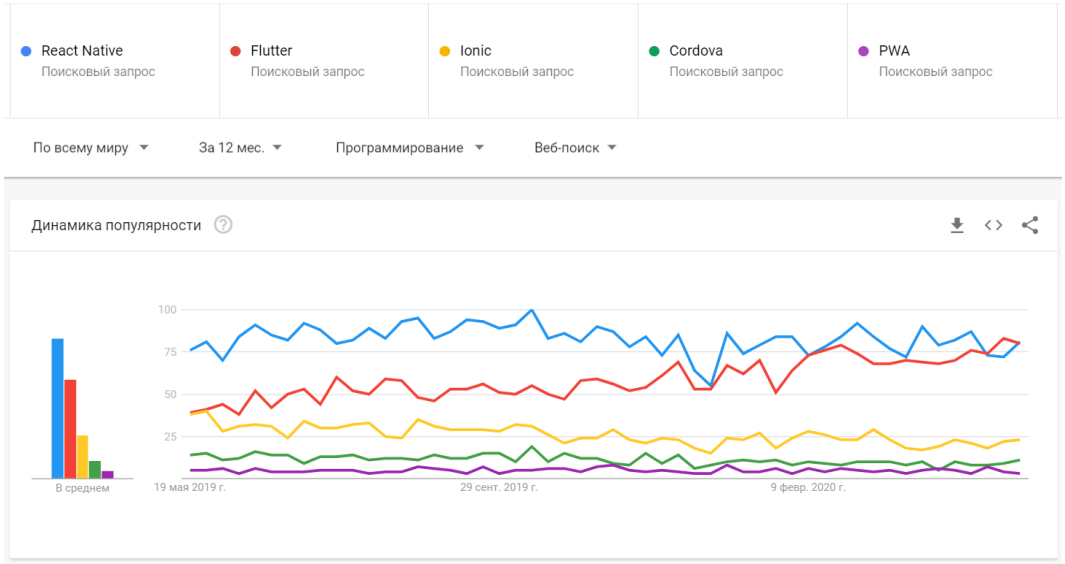

Рисунок 2. *Динамика популярности технологий за 12 месяцев [2]*

В результате были выбраны следующие технологии для обзора: Flutter, React Native, Ionic;

Flutter – это портативная среда пользовательского интерфейса Google для создания современных, нативных и реактивных приложений для iOS и Android, а также является основой пользовательского интерфейса для операционной системы Googles Fuchsia [3]. Google также работает над внедрением Flutter для встраиваемых устройств (Raspberry Pi, дома, автомобильной промышленности и т.д.). Объявлена поддержка для настольных и веб-платформ, выпущены инструменты для разработки приложений для Windows, macOS, Linux и веб.

Flutter имеет собственный движок рендеринга OpenGL – Skia 2D, который работает с различными типами аппаратных и программных платформ, а также используется в Google Chrome, Chrome OS, Android, Mozilla Firefox, Firefox OS и других [4]. Skia спонсируется и управляется Google и доступна по лицензии BSD Free Software.

Приложения Flutter написаны на Dart, объектно-ориентированном языке программирования, созданном Google. Первоначально он был предназначен для создания веб-приложений, но позже был адаптирован для Flutter, поскольку его функции, такие как поддержка интерпретированного и заранее скомпилированного исполнения, соответствуют требованиям Flutter [3; 4]. Dart используется и для создания пользовательского интерфейса, избавляя от необходимости использовать отдельные языки разметки или систему визуального дизайна.

Приложения построены из единой кодовой базы, скомпилированы в нативный код ARM, используют графический процессор (GPU) и могут получать доступ к определенным API-интерфейсам iOS и Android (таким как местоположение GPS, библиотека изображений), общаясь по каналам платформы.

Приложения Flutter обеспечивают высокую производительность для платформ iOS и Android. Во время разработки Flutter использует горячую перезагрузку, чтобы обновить работающее приложение за миллисекунды, когда меняется исходный код для добавления новых функций или изменения существующих. Горячая перезагрузка - это способ увидеть изменения, которые вносятся в код на симуляторе или устройстве, сохраняя на экране состояние приложения и данные.

Интерфейс Flutter реализован с использованием виджетов на основе современных реактивных фреймворках и использует собственный механизм рендеринга для их

отрисовки. Виджеты являются строительными блоками приложения Flutter, и каждый является неизменным объявлением пользовательского интерфейса.

Размещение виджетов вместе создает дерево виджетов. Flutter использует виджет в качестве конфигурации для создания каждого элемента, что означает, что этот элемент является виджетом, который монтируется на экране. Элементы, которые смонтированы на экране, создают дерево элементов. На рисунке 3 показаны дерево виджетов, дерево элементов и объект состояния.

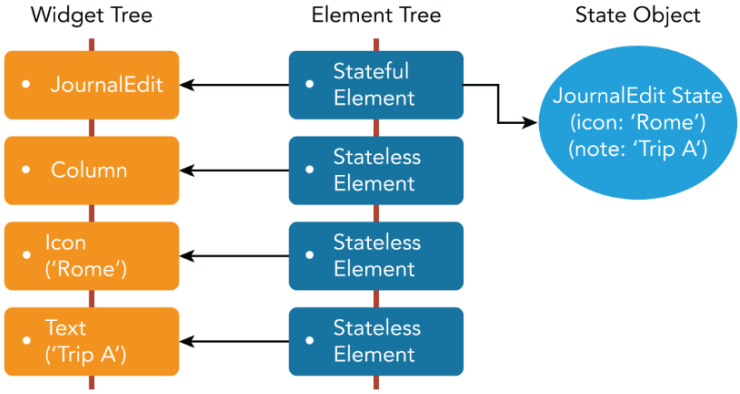

Рисунок 3. *Дерево виджетов, дерево элементов и объект состояния*

Для создания пользовательского интерфейса используется два основных типа виджетов, StatelessWidget и StatefulWidget. Виджет без сохранения состояния используется, когда значения не изменяются, а виджет с контролем состояния используется при изменении значения. Каждый виджет без сохранения состояния или с состоянием имеет метод сборки с BuildContext, который обрабатывает расположение виджета в дереве виджетов.

Объекты BuildContext на являются объектами Element, экземпляром виджета в определенном месте дерева. StatelessWidget создается на основе собственной конфигурации и не изменяется динамически. Например, на экране отображается изображение с описанием и не изменится.

Виджет без состояния объявляется с одним классом. Метод сборки виджета без состояния можно вызывать из трех разных сценариев. Его можно вызывать при первом создании виджета, при изменении родительского элемента виджета и при изменении InheritedWidget. На рисунке 4 показан жизненный цикл StatelessWidget.

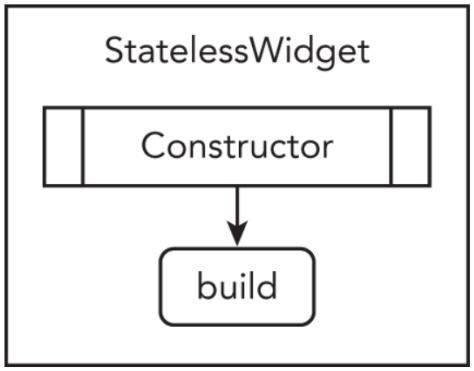

Рисунок 4. *Жизненный цикл StatelessWidget [4]*

StatefulWidget создается на основе собственной конфигурации, но может динамически меняться. Например, на экране отображается значок с описанием, но значения могут изменяться в зависимости от взаимодействия с пользователем, например, при выборе другого значка или описания. Этот тип виджета имеет изменчивое состояние,

которое может меняться со временем. Виджет с состоянием объявляется с двумя классами, классом StatefulWidget и классом State. Класс StatefulWidget перестраивается при изменении конфигурации виджета, но класс State может сохраняться, повышая производительность. Например, когда состояние изменяется, виджет перестраивается.

Если StatefulWidget удаляется из дерева, а затем вставляется обратно в дерево в будущем, создается новый объект State. При определенных обстоятельствах и ограничениях можно использовать GlobalKey (уникальный ключ для всего приложения) для повторного использования объекта State; однако глобальные ключи дороги. Вызывается метод setState (), чтобы уведомить платформу об изменениях этого объекта, и метод сборки виджета вызывается. Новые значения состояния устанавливаются внутри метода setState. На рисунке 5 отображен жизненный цикл StatefulWidget виджета.

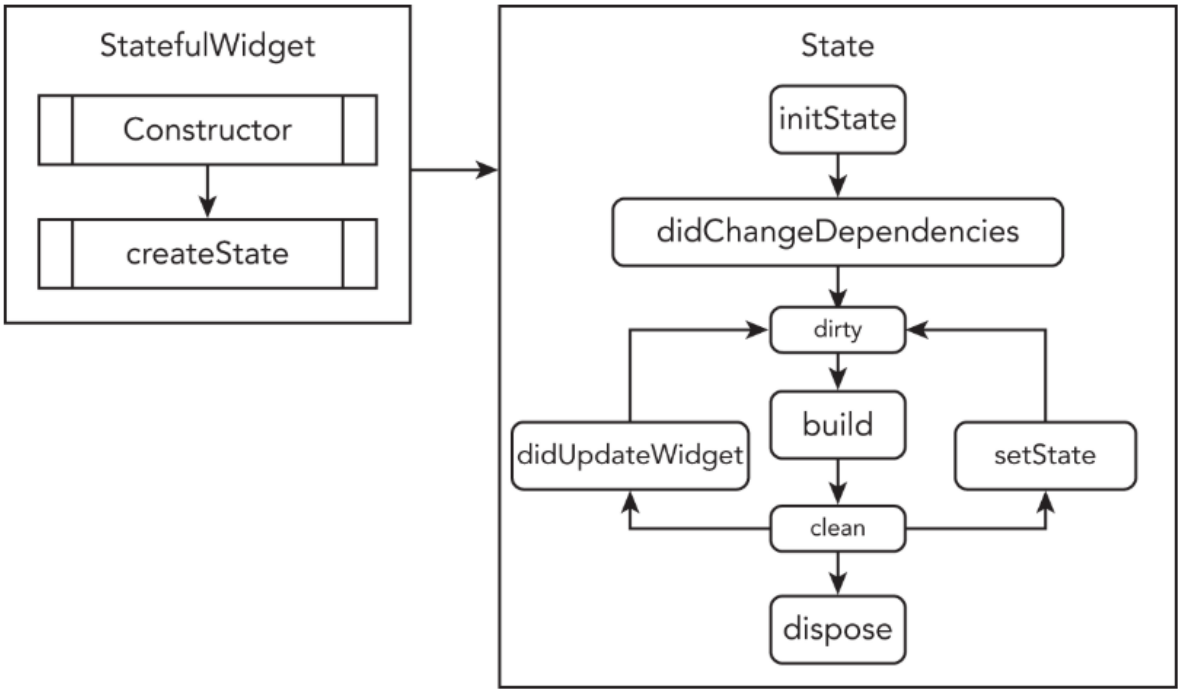

Рисунок 5. *Жизненный цикл StatefulWidget [4]*

React Native – это интерпретируемый кроссплатформенный инструмент для разработки приложений на основе JavaScript, созданный Facebook [5]. Он делится своими основными принципами и некоторым исходным кодом с ReactJS, который является библиотекой JavaScript для создания веб-приложений. Подобно React для веб-приложений, приложения React Native написаны с использованием смеси языков JavaScript и XML, известной как JSX.

С помощью React Native разработчики могут создавать нативные интерфейсы и получать доступ к компонентам, специфичным для платформы, используя JavaScript. Это отличает React Native от других гибридных сред приложений, таких как Cordova и Ionic, которые упаковывают веб-представления, созданные с использованием HTML и CSS, в нативное приложение. Вместо этого React Native компилирует JavaScript в настоящее нативное приложение, которое может использовать специфичные для платформы API и компоненты.

Альтернативные продукты, такие как Xamarin, используют тот же подход, но приложения Xamarin создаются с использованием C#, а не JavaScript. Благодаря использованию JavaScript, разработчики веб-приложений могут сделать переход к мобильным системам.

Архитектура приложения React Native состоит из виртуальной машины JavaScript, моста React Native и собственных модулей. Эта архитектура представлена на рисунке 6.

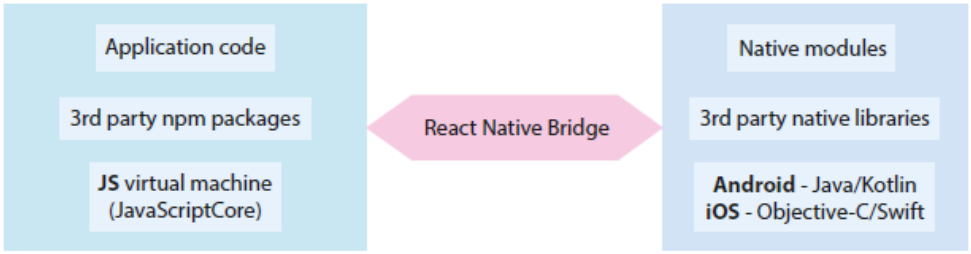

Рисунок 6. *Архитектура React Native [6]*

Код приложения работает на виртуальной машине JavaScript вместе с любыми сторонними библиотеками. Вызовы собственного модуля направляются через мост React Native к нативным API и сторонним библиотекам, и результаты передаются, а результаты при необходимости передаются обратно через мост. Это позволяет использовать JavaScript для разработки приложения, в то же время используя нативные компоненты пользовательского интерфейса и функции платформ.

Все операции JavaScript при взаимодействии с нативной платформой выполняются в отдельном потоке, что позволяет плавно выполнять пользовательский интерфейс и любые анимации. В этом потоке находится приложение React и обрабатываются все вызовы API, события касания и взаимодействия.

При изменении компонента с нативной поддержкой обновления отправляются на нативную сторону. Это происходит в конце каждой итерации цикла событий. Для большинства приложений React Native бизнес-логика выполняется в потоке JavaScript.

Весь код React состоит из компонент. Компоненты являются строительными блоками приложения React или React Native. Точка входа приложения – это компонент, который требует других компонентов и состоит из них. Эти компоненты могут также потребовать других компонентов и так далее.

Существует два основных типа компонент React Native: с сохранением состояния и без сохранения состояния. Основное отличие состоит в том, что компоненты без состояний не подключаются к каким-либо методам жизненного цикла и не хранят собственное состояние, поэтому любые данные, которые должны быть отображены, должны быть получены как свойства (props).

Одним из направлений в кроссплатформенном программировании является создание сайта, в который добавляется платформенный код с помощью фреймворка. Такой код транслирует вызовы от системы через API к приложению и обратно. Примером такого фреймворка является Ionic Framework.

Ionic Framework – это инструментарий пользовательского интерфейса с открытым исходным кодом, который фокусируется на внешнем интерфейсе пользователя и пользовательском интерфейсе приложения, таких как взаимодействия, жесты, анимация и элементы управления [7]. Позволяет разработчикам создавать приложения для всех основных платформ из единой кодовой базы. Благодаря возможности адаптивного стиля на основе текущей платформы, приложения Ionic также могут иметь нативный внешний вид.

Основным преимуществом Ionic является то, что есть возможность создавать гибридные мобильные приложения, которые обеспечивают бесперебойную работу на всех основных платформах из одной кодовой базы. Это избавляет от сложности наличия нескольких команд на разных платформах, что повышает производительность. Кроме

того, он дает разработчикам возможность развертывать свои приложения как iOS, Android, Electron, PWA.

Единственный компонент может быть создан и затем использован конечным пользователем, как ему угодно. Этот же компонент затем используется для Angular, React, Vue или любого другого фреймворка. Компоненты построены с использованием веб-стандартов с использованием HTML, CSS и JavaScript. Каждый Ionic компонент адаптирует свой внешний вид к платформе, на которой работает приложение.

Приложение Ionic, загруженное из Apple App Store, получит тему iOS, а приложение Ionic, загруженное из Play Store Android, получит тему Material Design. Для Progressive Web App, Ionic по умолчанию использует тему Material Design. Кроме того, решение о том, какой дизайн использовать в определенных сценариях, настраивается разработчиком.

Ionic поставляется вместе с новой средой выполнения гибридных приложений с открытым исходным кодом Capacitor. Capacitor позволяет запускать современные веб-приложения для iOS, Android, Electron и Web (с использованием технологии Progressive Web App), предоставляя мощный и простой в использовании интерфейс для доступа к нативным пакетам SDK и API на каждой платформе [7].

Используя Capacitor, разработчики могут создать одно приложение и настроить один набор API-интерфейсов независимо от платформы, на которой работает приложение, в отличие от управления несколькими API-интерфейсами для каждой целевой платформы. Это означает, что, например, для доступа к камере используется тот же код на iOS / Android, что и на Electron и веб версии.

Ionic упрощает создание одного веб-приложения, которое изначально работает на мобильных и десктоп устройствах, а также в веб версии, как прогрессивное веб-приложение. Capacitor является продолжением идеи такого инструмента как Apache Cordova и Adobe PhoneGap, и имеет обратно совместимую поддержку для многих существующих плагинов Cordova [8].

Ionic приложения создаются с использованием веб-технологий и визуализируются с использованием Web Views, которые представляют собой полноэкранный полнофункциональный веб-браузер.

Современные Web Views предлагают множество встроенных API-интерфейсов для доступа к аппаратным функциям, таким как камеры, датчики, GPS, динамики и Bluetooth, но иногда может потребоваться доступ к аппаратным API-интерфейсам для конкретной платформы, который можно получить через уровень моста, обычно с помощью нативных плагинов, которые предоставляют JavaScript API-интерфейсы. На рисунке 7 показана архитектура Ionic приложения.

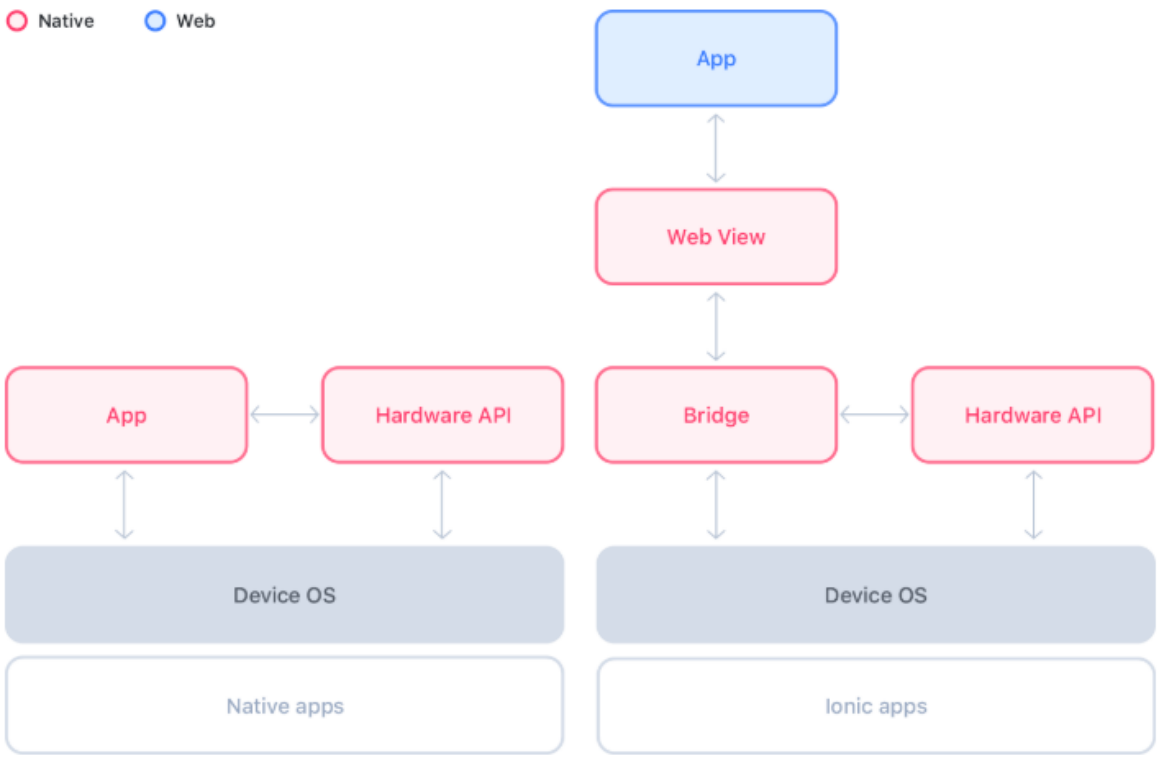

Рисунок 7. *Архитектура Ionic приложения [8]*

Обзор технологий, проведенный в предыдущей главе, показал на сколько сильно отличаются кроссплатформенные технологии друг от друга. Каждая технология имеет уникальный набор свойств и имеет свою сферу применения.

Выбор подходящей технологии становится не тривиальной задачей для разработчиков. Чтобы показать преимущества и недостатки определенной технологии, были определены критерии сравнения, после чего проведен сравнительный анализ.

К сравнению кроссплатформенных технологий были также включены нативные для полноты исследования. Результаты сравнительного анализа отражены в таблице 1.

Таблица 1. *Сравнительный анализ Native, Ionic, Flutter и React Native*

Критерий	Native	Flutter	Ionic	React Native
Кроссплатформенность	Нет	Мобильные платформы. Планируется расширение.	Да	Мобильные платформы
Доступ к аппаратным возможностям платформы	Да	Да	Ограниченный, через плагины	Да
Производительность	Максимальная	Близкая к максимальной	Средняя	Близкая к максимальной
Сообщество	Большое	Среднее	Среднее	Большое
Наличие CLI	Нет	Нет	Да	Да
Языки разработки	Java/Kotlin, Swift/Objective-C .Net/C/C++	Dart	HTML, CSS, JavaScript, TypeScript	JavaScript, TypeScript JSX

Ui / Ux	Нативный	Хороший	Удовлетворительный	Хороший
Стоимость разработки	Максимальная	Высокая	Средняя	Высокая
Публикация в магазине приложений	Да	Да	Да	Да
Работа без установки	Нет	Нет	Нет	Нет
Документация	Да	Да	Да	Да
Простота обновления	Низкая	Средняя	Высокая	Средняя
Порог вхождения	Высокий	Высокий	Средний	Высокий
Потребление памяти	Минимальное	Минимальное	Большое	Близко к минимальному
Фоновая синхронизация	Да	Да	Да	Да
Push-уведомления	Да	Да	Да	Да

По результатам сравнительного анализа технологий можно сделать следующие выводы. Нативные приложения создаются для конкретной платформы и поддерживаемых устройств. Поскольку нативные приложения созданы для поддержки конкретной операционной системы, существует полный доступ к новейшим аппаратным и программным технологиям для конкретных устройств.

Нативные приложения предлагают лучшую производительность и минимальную задержку. Недостатком является то, что приложению также потребуются нативные реализации для других платформ с отдельными базами кода, что значительно увеличивает время и стоимость разработки, требует несколько команд разработчиков.

React Native и Flutter являются лучшей заменой нативных приложений для мобильных платформ. Производительность этих инструментов незначительно хуже нативных за счет применяемых технологий. Единая кодовая база позволяет существенно снизить затраты на разработку и поддержку приложения. Недостатком является сложность реализации пользовательского интерфейса, неотличимого от нативного.

Ionic за счет работы с WebView значительно уступает в производительности как нативным приложениям, так и технологиям React Native и Flutter. Кроме того, доступ к нативным функциям платформы осуществляется посредством плагинов, которые часто оказываются устаревшими. Но за счет низкого порога входа позволяют в краткие сроки создавать мобильные приложения веб-разработчикам. Ionic имеет дополнительное преимущество в возможности создавать приложение не только под мобильные, но и под веб и десктоп платформы, за счет использования Capacitor. Два эти инструмента подходят больше для создания прототипов, с помощью которых можно оценить реальную потребность в приложениях на мобильных платформах.

Список литературы

1. Cross-platform mobile framworks used by software developers worldwide as of 2019 [Электронный ресурс] – Режим доступа: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/ (дата обращения 05.02.2020).
2. Google Trands [Электронный ресурс] – Режим доступа: https://trends.google.ru/trends/explore?geo=RU&q=React%20Native,flutter,Ionic,Corodova,PWA (дата обращения 07.02.2020).
3. Windmill, E. Flutter in Action / E. Windmill – Manning Publications, 2020 – P.310.
4. Napoli, M. L. Beginning Flutter / M. L. Napoli – Wrox, 2019 – P.528.
5. Nalwaya, A., Paul, A. React Native for Mobile Development: Harness the Power of React Native to Create Stunning iOS and Android Applications / A. Nalwaya, A. Paul – Apress, 2019 – P.119.
6. Dabit, N. React Native in Action / N. Dabit – Manning Publications, 2019 – P.320.
7. Cheng, F. Build Mobile Apps with Ionic 4 and Firebase: Hybrid Mobile App Development / F. Cheng – Apress, 2018 – P.238.
8. Singh, I., Phan, H. Ionic Cookbook - Third Edition / I. Singh, H. Phan – Packt Publishing, 2018 – P.390.

References

1. Cross-platform mobile framworks used by software developers worldwide as of 2019 [Electronic resource] - Access mode: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/ (accessed date 02/05/2020).
2. Google Trands [Electronic resource] - Access mode: https://trends.google.com/trends/explore?geo=RU&q=React%20Native,flutter,Ionic,Corodova,PWA (accessed 07.02.2020).
3. Windmill, E. Flutter in Action / E. Windmill - Manning Publications, 2020 - P.310.
4. Napoli, M. L. Beginning Flutter / M. L. Napoli - Wrox, 2019 - P.528.
5. Nalwaya, A., Paul, A. React Native for Mobile Development: Harness the Power of React Native to Create Stunning iOS and Android Applications / A. Nalwaya, A. Paul - Apress, 2019 - P.119.
6. Dabit, N. React Native in Action / N. Dabit - Manning Publications, 2019 - P.320.
7. Cheng, F. Build Mobile Apps with Ionic 4 and Firebase: Hybrid Mobile App Development / F. Cheng - Apress, 2018 - P.238.
8. Singh, I., Phan, H. Ionic Cookbook - Third Edition / I. Singh, H. Phan - Packt Publishing, 2018 - P.390.