МИНИСТЕРСТВО ВЫСШЕГО ОБРАЗОВАНИЯ И НАУКИ РФ

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Приамурский государственный университет имени Шолом-Алейхема»**

Факультет математики информационных технологий и техники

Направление Информационные системы и технологии на производстве

**РЕФЕРАТ**

**Тема: Принципы кроссплатформенной разработки**

**Обзор средств для разработки приложений сразу для нескольких мобильных операционных систем.**

Выполнил: студент 3 курса

группы 1701 Андриенко Иван

Преподаватель: Глаголев В.А.

Биробиджан

2022

Оглавление

[**Введение** 3](#_Toc124421244)

[**1. Анализ методов разработки кроссплатформенных мобильных приложений** 4](#_Toc124421245)

[**1.2 Сравнительный анализ нативного и кроссплатформенного подхода к созданию мобильных приложений** 4](#_Toc124421246)

[**1.2.1 Нативный подход** 4](#_Toc124421247)

[**1.2.2 Кроссплатформенный подход** 6](#_Toc124421248)

[**1.3 Обзор фреймворков для кроссплатформенной разработки** 10](#_Toc124421249)

[**1.3.1 Android Studio** 10](#_Toc124421250)

[**1.3.1 React Native** 11](#_Toc124421251)

[**1.3.2 Flutter** 12](#_Toc124421252)

[**1.3.3 Kotlin Multiplatform** 14](#_Toc124421253)

[**Заключение** 15](#_Toc124421254)

[**Список источников** 16](#_Toc124421255)

# **Введение**

Создание отдельных нативных приложений требует привлечения нескольких команд разработчиков, реализующих, по сути, одно и то же, и больше временных ресурсов. Cократить расходы и время на разработку сервиса помогают кроссплатформенные технологии. Однако необходимо четко понимать в каких проектах такие технологии принесут выгоду, а в каких наоборот убытки, и какую именно технологию выбрать для разный по назначению и сложности приложений.

В рамках работы изучен современный рынок мобильных приложений, выявлены причины появления кроссплатформенного подхода в разработке, проведено сравнение нативного и кроссплатформенного подходов к разработке мобильных приложений и определено для каких проектов выгоден каждый их них, проведен обзор наиболее популярных в настоящее время фреймворков для создания кроссплатформенных приложений, изучены направления развития кроссплатформенной разработки.

Данная статья ориентирована на архитекторов, разработчиков и специалистов по информационным технологиям, которые проектируют, создают или используют мобильные приложения и сервисы, для разных операционных систем.

# **1.** **Анализ методов разработки кроссплатформенных мобильных приложений**

В число задач данного отчета по НИР входит оценка тенденций рынка мобильных приложений, сравнительный анализ нативной и кроссплатформенной разработки, обзор наиболее востребованных кроссплатформенных фреймворков.

# 

# **1.2 Сравнительный анализ нативного и кроссплатформенного подхода к созданию мобильных приложений**

# **1.2.1 Нативный подход**

Под нативным подходом подразумевается использование оригинальных языков и инструментов разработки конкретной мобильной операционной системы. Так приложения для iOS создаются в среде разработки XCode на языке Swift, а приложения под Android в Android Studio на Kotlin (до 2018 года основным языком был Java).

Разработка мобильного приложения на родных технологиях и языках имеет следующие положительные моменты:

1. Гибкость в реализации

Можно реализовать все, на что способны технологии мобильной операционной системы. Кроме того, разработчик не ограничен в использовании сторонних библиотек.

1. Эффективность UI / UX

Поведенческие паттерны различны для, например, устройств iOS и Android. Пользователи зачастую находят реализации нативных версий более удобными.

1. Скорость работы

Полная аппаратная поддержка мобильной платформы позволяет нативным приложениям работать быстро и бесперебойно.

1. Доступность последних технологий

Новый программный и аппаратный функционал, предоставленный компаниями-производителями устройства и операционной системы, становится доступен для реализации сразу после выпуска соответствующих обновлений.

1. Легкость и качество тестирования

В процессе разработки не сложно отследить, в какой момент приложение начинает требовать большего объёма памяти или ресурсов процессора, а также обнаружить тормозящие процессы и оптимизировать их.

Более того, разработчик не ограничен в применение юнит-тестов и в интеграции систем удаленного мониторинга ошибок.

1. Полная поддержка со стороны магазинов приложений

Нативные приложения по умолчанию отвечают определенному числу требований магазинов, поэтому процесс публикации для них относительно прост.

Вместе с тем, стоит учитывать, что нативная разработка под несколько платформ требует значительных ресурсных затрат и приводит к:

1. Увеличению стоимости разработки

Для разработки сервиса придется задействовать не одну, а несколько команд разработчиков, что приведет к увеличению издержек.

Кроме того, дорожает и поддержка, после выхода на рынок необходимо обновлять функционал и исправлять ошибки сразу на нескольких платформах, при этом на каждой из них возможно появление уникальных багов, и нет гарантии, что функционал всех версий приложения будет идентичен в любой срез времени, что ведет к ухудшению пользовательского опыта на одной или сразу на нескольких платформах.

1. Увеличению времени разработки

Реализация нескольких приложений затратна по времени, кроме того, даже при одновременной работе нескольких платформенных команд стоит учитывать, что каждая из них будет иметь собственный темп разработки.

# **1.2.2 Кроссплатформенный подход**

Существует два типа кроссплатформенных мобильных приложений:

* Гибридные гибридные приложения (Cordova 2013, Ionic 3013) опираются на веб-технологии (HTML, CSS и JavaScript/ECMAScript) и выполняются обычно в WebView. По сути, это веб-сайты, имитирующие работу приложения.
* Нативные кроссплатформенные приложения (Android studio, Xamarin 2011, React Native 2015, Flutter 2017, Kotlin Native 2017) пишутся на нативных или кроссплатформенных языках программирования и затем либо компилируются в ассемблерный код AOT, либо JIT, который использует нативные виджеты пользовательского интерфейса.

Выбор в пользу кроссплатформенного мобильного приложения обеспечит следующие преимущества:

1. Широкий охват рынка

Расширение пользовательской базы за счёт появления приложения одновременно в нескольких магазинах.

1. Дешевизна сборки за счет снижения рабочей нагрузки

Единая кодовая база позволяет инвестировать только один раз и только в одну команду.

1. Быстрая разработка сразу для нескольких платформ

Единая кодовая база легче тестируется и позволяет повторно использовать до 90% кода вместо написания полностью уникального под каждую платформу. В результате можно получить преимущество быстрого выхода на рынок (TTM – Time to Market).

1. Упрощенная поддержка продукта

Разработчикам необходимо поддерживать лишь одну версию приложения. А добавление функционала и исправление багов распространяется на все платформы разом.

1. Упрощенная процедура обновления

Кроссплатформенные приложения доступны через интернет, поэтому обновляются автоматически.

1. Простая интеграция с облаком
2. Единая логика приложения на всех платформах
3. Согласованность UI и UX

При разработке кроссплатформенных приложений различия в UI/UX между Android и iOS обрабатываются по умолчанию, что делает несогласованность дизайна и навигации менее вероятной.

Основные недостатки кроссплатформенного приложения:

1. Непредсказуемость пользовательского опыта и дизайна

На подсознательном уровне пользователь ждет от приложения отзывчивости. Кроссплатформенные решения обрабатывают больше данных, поскольку включают дополнительный уровень абстракции, поэтому работают медленнее нативных при наличии многоэлементного интерфейса.

Также для кроссплатформенной разработки нет гайдлайнов. У каждой из платформ свои стандарты, и если кроссплатформенное приложение разработано по гайдлайнам iOS, то пользователям Android это доставит неудобство, и наоборот.

1. Проблемы с внедрением новых функций

Кроссплатформенные решения – стороннее ПО, поэтому они не могут обеспечить немедленную поддержку последних обновлений iOS и Android.

1. Зависимость от мобильного устройства

Кроссплатформенные приложения обычно не работают оффлайн. Поэтому их возможности зависят от наличия стабильного подключения к интернету, версии операционной системы и модели устройства.

1. Проблемы с безопасностью

Для обеспечения надежной защиты требуется эффективное использование аппаратных ресурсов, с чем кроссплатформенные приложения справляются на порядок хуже нативных. А введение нового уровня поверх родной ОС приносит еще больше уязвимостей.

1. Низкие показатели производительности

Любое ответное действие со стороны приложения требует выполнения процессов сериализации и десериализации.

1. Большой вес приложения
2. Нет устойчивой поддержки сообщества

Сообщества разработчиков кроссплатформенных приложений в разы меньше, чем iOS- и Android-комьюнити. Поскольку экосистемы фреймворков постоянно развиваются, а обновления библиотек выходят едва ли не каждый месяц, разработчикам приходится вкладывать много времени и сил в изучение документации.

Более того, в некоторых случаях недостаток пользовательских компонентов приводит к тому, что приходится искать отдельные решения.

1. Сложности при построении команды

Кроссплатформенная разработка требует от разработчиков глубокого понимания целевых ОС, ведь не смотря на одну кодовую базу, им придется разворачивать приложения и в Google Play и в App Store, а также писать дополнительные куски кода под каждую платформу и думать о том, как наиболее эффективно обработать различия операционных систем.

1. Ограничения в поддержке магазинов

Кроссплатформенные приложения зачастую не соответствуют всем требованиям магазинов или же имеют по некоторым пунктам низкие показатели. В результате чего, они редко попадают в Featured-разделы App Store и Google Play.

Резюмируя вышеописанное, можно сделать вывод, что нативная разработка выгодна в том случае, если заказчику достаточно охватить аудиторию лишь одной платформы или необходимо добиться безупречного пользовательского опыта от конкретной платформы. Также в сторону нативного приложения склоняют требования к беспрепятственному использованию аппаратного обеспечения мобильного устройства или специфичных для платформы функций и высокой производительности. Кроме того, если планируется создать сложное и долгосрочное приложение с широким функционалом, многоэлементным интерфейсом или сложной анимацией, нативное приложение – единственный выбор. Об этом говорит опыт таких крупных компаний как Facebook, Airbnb которые разработали кроссплатформенные приложения для своих целей, но в итоге вернулись к нативному способу разработки, так как кроссплатформенное приложение не удовлетворяло их требованиям, с трудом масштабировалось и не оправдывало ожидания пользователей.

Однако мир программного обеспечения сильно изменился с тех пор, как Цукерберг сделал свое заявление о частичном уходе от кроссплатформенных технологий в 2012 году, когда использовался, кстати, в основном гибридный подход. Современные фреймворки уходят от гибридного подхода и предлагают собственные инструменты разработки и компиляцию в Swift и Kotlin, что позволяет повысить производительность. Каждый фреймворк имеет свои проблемные точки, но многие копании все равно продолжают инвестировать в них, за счет того, что общая прибыль с приложения превышает затраты на обход препятствий к лучшей производительности, а некоторые даже вносят свой вклад в кодовую базу фреймворков на основе опыта самостоятельных исправлений критических моментов.

Есть случаи, когда лишь некоторый функционал приложения разрабатывается на кроссплатформенных технологиях, таким образом подход используется максимально эффективно там, где он действительно нужен.

Разработаны бренд-ориентированный, платформа-ориентированный и смешанный подходы к адаптации с UI/UX в кроссплатформенных приложениях , а Material Design даже выпустили best practices по этой теме. Таким образом, кроссплатформенное приложение подойдет и тем, кто считает, что приложение должно выглядеть одинаково везде, и тем, кто считает, что приложение должно соответствовать платформе.

В последние годы и Google, и Apple позаимствовали декларативную систему пользовательского интерфейса из Интернета, используя ее во фреймворках Flutter, Jetpack Compose и SwiftUI. Это означает, что в будущем разработчику Swift больше не нужно понимать структуру Android XML, а разработчику Android больше не нужно знать, как работать с файлами XIB или раскадровками.

Кроссплатформенный подход позволяет сэкономить 10-30% бюджета, многие известные приложения такие как Walmart, Tesla, UberEats, Google Ads Alibaba, Wix, Discord, воспользовались этим и сумели быстро выйти на рынок и получить широкий охват пользователей. Рынок, сообщество и популярность кроссплатформенных фреймворков растет, о чем говорят статистические данные Markets And Markets research, Google Trends и Stack Overflow Trends. Все вышеперечисленное говорит о том, что кроссплатформенный метод разработки – это хорошая практика, которая значительно упростит реализацию приложения сразу под несколько платформ, но подходит она не под каждую задачу, нужно обязательно анализировать тип приложения, стек компании и опыт команды разработчиков. Выгодно использовать данный подход при построении проектов автоматизации бизнес-процессов, B2B-приложений с массивной бизнес-логикой, приложений для распространения контента, игровых и сравнительно простых в своем функционале, интерфейсе и анимациях приложений.

# **1.3 Обзор фреймворков для кроссплатформенной разработки**

В рамках данного раздела будут рассмотрены нативные кроссплатформенные фреймворки: Android studio, React Native, Flutter и Kotlin Multiplatform, так как они занимают большую часть кроссплатформенного рынка и их популярность все еще растет, в отличии от Xamarin, вместе с тем, как прогрессивные (PWA) и гибридные фреймворки (Cordova, Sencha, Ionic) уже реже используются в разработке из-за низкой производительности, неточного интерфейса и наличия JavaScript моста.

# **1.3.1 Android Studio**

Среда разработки Android Studio является весьма молодой средой разработки. Но, безусловно, данная IDE весьма перспективна и востребована. К тому же, она бесплатна. Данный продукт создан компанией Google специально для осуществления разработки мобильных приложений на языке Java для операционной системы Android, также являющейся их разработкой. Среда разработки Android Studio является кроссплатформенной и разработана на основе IntelliJ IDEA.

IDE Android Studio, как уже было сказано, предназначена именно для разработки приложений на ОС Android, ориентирована исключительно на данную систему, а потому достаточно удобна, понятна и функциональна. Некоторые специалисты считают, что данная среда разработки имеет ряд недоработок, которые необходимо исправить. Но даже они признают, что IDE Android Studio весьма конкурентоспособна.

Среда разработки Android Studio имеет ряд преимуществ, таких как, например:

* Превью отображение изменений при редактировании xml;
* Возможность создания шаблонных активностей;
* Улучшенный интеллектуальный анализ кода;
* Полноценный режим отладки и улучшенное добавление точек остановки. Принимая во внимание перечисленные преимущества, а также тот факт, что разрабатываемое приложение предназначено для мобильных устройств, использующих операционную систему Android, логичным будет выбрать данную среду разработки, поскольку она ориентирована на создание программ именно для системы Android и, к тому же, создана компанией производителем данной операционной системы.

# **1.3.1 React Native**

Данный фреймворк - продукт внутреннего хакатона, проведенного в 2013 году Facebook. Компания хотела перенести на мобильные устройства преимущества веб-разработки, такие как быстрые итерации и наличие единой команды для создания всего продукта.

**Достоинства:**

1. Регулярные обновления и реакция на потребности разработчиков. Растущее признание сообщества, простота поиска разработчиков.
2. Потенциал RN выходит за рамки разработки мобильных приложений . Фреймворк поддерживают Microsoft и Shopify.
3. Поддерживается большинством IDE.
4. Использует самый популярный язык JavaScript
5. Много готовых компонентов и сторонних библиотек.
6. Хорошая совместимость со сторонними плагинами (Google Maps).
7. Почти нативный пользовательский опыт благодаря использованию контроллеров, встроенных как в Android, так и в iOS. Использование библиотеки ReactJS с элементами пользовательского интерфейса, что позволяет ускорить процесс проектирования.
8. Доступ к акселерометру и камере, однако для использования этих функций необходимо писать нативный код.

**Недостатки:**

1. Готовые компоненты зачастую требуют дополнительных корректировок в коде.
2. Продолжительный процесс тестирования и отладки.
3. Не является полностью кроссплатформенным, для реализации многих функций необходимо писать нативный код.
4. Наличие JavaScript моста

Используется в приложениях: Tesla, Walmart, Pulse SoundCloud, Bloomberg, Pinterest, Instagram.

# **1.3.2 Flutter**

Данный фреймворк - продукт Google, его принцип – создание приложений с единой кодовой базой для мобильных платформ, веба и десктопа.

**Достоинства:**

1. Использует оптимизированный язык Dart, который имеет современные подходы, многопарадигмальность и ООП для создания кроссплатформенных приложений. Dart считается легким в освоении по причине того, что заимствован из C/C++ и Java и имеет обширную и понятную документацию.
2. Приложения запускаются с помощью изначально скомпилированного кода.
3. Высокая производительность, так как собственные компиляторы, набор виджетов и графический движок Skia устраняют необходимость в мосте JavaScript. Рассчитан на достижение постоянной скорости 60 кадров в секунду.
4. Возможность создавать сложные дизайны пользовательского интерфейса с богатыми переходами и анимацией
5. Создан владельцами Android, поддерживается Google. Удобен для Android-разработчиков.
6. Поддерживается Android Studio / IntelliJ и Visual Studio Code.
7. Поддерживает API Firebase и другие сервисы Google.
8. Flutter rendering engine предлагает большой набор готовых настраиваемых виджетов. Следует подходу «UI-as-a-code». Позволяет создать приятный, продуманный, пиксельный интерфейс с использованием как собственных, не привязанных к платформе, так и имитирующих привычные iOS/Android элементов. Поддерживает material design.
9. Позволяет выпускать приложения для разных версий Android и iOS без дополнительных корректировок в коде.
10. Идеально подходит для создания MVP приложений.
11. Flutter – это реактивный фреймворк.
12. Облегченное и быстрое по времени тестирование при отсутствии специфичных для платформ функций.

**Недостатки:**

1. Необходимо изучать новый язык.
2. Нет автоматической адаптации стилей для разных платформ.
3. По причине незрелости инструмента может возникнуть необходимость в написании собственных функциональных элементов. Мало поддерживаемых библиотек и плагинов. Не поддерживает сервисы Amazon Web Services.
4. Не позволяет полностью абстрагироваться от платформы, требуются нативные знания и неизбежна концепция трех кодовых баз при использовании специфичных для платформ функций.
5. Недостаточно развитое сообщество.
6. Возможны проблемы с публикацией в магазинах приложений.
7. Приложения не поддерживаются веб-браузерами.
8. Большой размер приложения.
9. Не предлагает поддержки Android TV и Apple TV.

Используется в приложениях: Alibaba, Hamilton Musical, Greentea, Google Ads, Philips Hue.

# **1.3.3 Kotlin Multiplatform**

У JetBrains есть важное отличие от многих других сторонников кроссплатформенности: там изначально не ставят целью «полностью универсальную кодовую базу». Радикальный подход «всё общее» в большинстве случаев работает плохо или накладывает массу ограничений, поэтому компания и не пытается его воплотить.

Вместо этого предлагают переиспользовать только по-настоящему общую часть кода. А всё различающееся писать нативно. UI здесь никак не объединяют.

Всё общее можно выделить в библиотеку на Kotlin, подходящую сразу для iOS и Android, а дальше обращаться к ней с обеих сторон.

Выбрав подход «проект из трёх частей», JetBrains направили усилия на максимально бесшовное взаимодействие этих частей на уровне языка.

**Достоинства:**

1. Благодаря популярности Kotlin в Android-разработке «три кодовых базы» могут писаться на одном языке. Многие мобильные разработчики уже пишут на Kotlin. iOS-разработчикам возможно придётся изучить новый язык, но этот процесс упростит сходство Kotlin со Swift.
2. Kotlin позволяет создавать максимально чистый код.
3. Облегченное и быстрое тестирование
4. Команды нативных программистов могут распределять задачи между собой и использовать код друг друга в дальнейшем.
5. UI/UX разрабатываются отдельно для каждой платформы, хороший пользовательский опыт.

**Недостатки:**

1. Границы платформ размываются и появляется «общая» часть, но она не одинаково близка обеим сторонам.
2. Очень маленькое сообщество. Сложно найти разработчиков, экспертов в технологии.
3. Возможно, придется изучать новые язык программирования и среду разработки.
4. В команде обязательно наличие нативных разработчиков.

Используется в приложениях: Яндекс.Карты и Яндекс.Диск (несколько модулей), Netflix, Quizlet, Autodesk.

# **Заключение**

Перспектива кроссплатформенных разработок бесспорно положительная: некоторые приложения можно будет писать под несколько платформ сразу без заметного ухудшения их характеристик, что довольно удобно для разработчиков.Кроссплатформенные приложения пишутся на независимом языке, затем компилируются для каждого устройства отдельно.

Нет единственно верного решения при выборе подхода к мобильной разработке. Однако, за кроссплатформенными приложениями будущее, они предлагают более продвинутые функции, чем нативный инструментарий и помогают экономить.

# **Список источников**

1. Smartphone users worldwide 2020 / Statista [Электронный ресурс]. — URL: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/
2. DIGITAL 2020: GLOBAL DIGITAL OVERVIEW [Электронный ресурс]. — URL: https://datareportal.com/reports/digital-2020-global-digital-overview
3. Mobile Operating System Market Share Worldwide / Statcounter [Электронный ресурс]. — URL: https://gs.statcounter.com/os-market-share/mobile/worldwide
4. How much does it cost to make an app? [Электронный ресурс]. — URL: https://thinkmobiles.com/blog/how-much-cost-make-app/
5. iOS 14 Apple [Электронный ресурс]. — URL: https://www.apple.com/ru/ios/ios-14/
6. Android 11 [Электронный ресурс]. — URL: https://www.android.com/
7. Xcode 12 [Электронный ресурс]. — URL: https://developer.apple.com/xcode/
8. Swift – Apple [Электронный ресурс]. — URL: https://www.apple.com/ru/swift/
9. Android Studio [Электронный ресурс]. — URL: https://developer.android.com/studio?hl=ru (Дата обращения 01.12.2020).
10. Kotlin [Электронный ресурс]. — URL: https://kotlinlang.org/
11. Develop Android apps with Kotlin [Электронный ресурс]. — URL: https://developer.android.com/kotlin?hl=ru
12. Oracle Java [Электронный ресурс]. — URL: https://www.oracle.com/ru/java/
13. Нативная или кроссплатформенная разработка? Наглядное сравнение. [Электронный ресурс]. — URL: https://appcraft.pro/blog/pochemu-my-vybiraem-nativnuyu-razrabotku/