УДК 004.738.52

М. Д. Андреев, Н. Е. Пособилов

ВЫБОР ПЛАТФОРМЫ ДЛЯ РАЗРАБОТКИ МОБИЛЬНОГО НАТИВНОГО КЛИЕНТСКОГО ПРИЛОЖЕНИЯ ДЛЯ ПОРТАЛЬНЫХ РЕШЕНИЙ ТЕЛЕКОМ-ОПЕРАТОРОВ

Нижегородский государственный технический университет им. Р. Е. Алексеева

В данной работе описана проблема выбора платформы для разработки мобильного нативного клиентского приложения для портальных решений телеком-операторов, исследованы показатели производительности типичных компонентов приложений ReactNativeApp и iOSApp и по результатам сравнения выбран наилучший вариант для разработки нативного приложения для портальных решений телеком-операторов.

Ключевые слова: нативное мобильное приложение, мобильное приложение для телеком-операторов, сравнение производительности ReactNativeApp и iOSApp.

Операторы, стремясь обеспечить высокую доходность, стараются играть на смежных рынках, и, бесспорно, рынок мобильных приложений и его инфраструктура представляют для них одно из приоритетных направлений развития. Мобильное приложение для телеком-оператора – это простой и доступный канал связи с потребителем услуг сотовой связи через удобный мобильный интерфейс, позволяющий клиенту удаленно взаимодействовать с сервисами сотового оператора.

С технической точки зрения, нативное мобильное приложение - это приложение, содержащее большое количество легконагруженных компонентов и экранов с бизнес-логикой и простой анимацией. Оно не требует высокой производительности от центрального процессора и графического процессора, поскольку исключает ресурсозатратное взаимодействие с информацией. Все данные приложение получает в обработанном виде с серверов телеком-оператора.

Для максимального охвата рынка сотовой связи и создания цельной инфраструктуры для различных мобильных операционных систем должны быть разработаны идентичные по интерфейсу и функционалу нативные приложения. Поскольку около 93% мобильных устройств находятся под управлением операционных систем Android и iOS, разработка нативных приложений целесообразна только под них.

Для исследования и дальнейшего сравнения технических результатов тестов взяли наиболее популярные платформы для разработки нативных мобильных приложений:

- Java / Swift;
- React Native.

Существуют и другие языки для разработки мобильных приложений, с которыми можно столкнуться в процессе выбора(Java / Kotlin под Android, Swift / Objective-C под iOS). Углубляться в их сравнение не стоит, поскольку все они напрямую взаимодействуют с API операционной системы и различия показателей производительности компонентов, написанных на смежных языках, стремятся к минимуму.

Более детального изучения и сравнения заслуживает ReactNative – это фреймворк, который позволяет создавать нативные мобильные приложения на языке JavaScript, путем компиляции JavaScript-кода в Java/Objective-C и дальнейшей сборки приложения под соответствующую платформу. Это позволяет сэкономить на одной группе разработчиков, существенно ослабляя финансовую нагрузку.

Для построения приложения, он использует собственные декларативные компоненты, удобные для стилизации и редактирования. Система стилей в ReactNative взята из web-

разработки – для редактирования и стилизации компонентов используются мощные и интуитивно понятные инструменты из CSS.

Содержание двух команд мобильных разработчиков под различные операционные системы (Java/Swift) довольно сильно нагружает финансовый бюджет, в отличие от одной команды JavaScript-программистов.

Казалось бы, выбор очевиден, но для полной достоверности стоит сравнить технические показатели. Приложение ReactNative строится из собственных неизменяемых компонентов, что не позволяет полностью взаимодействовать с памятью и другими физическими ресурсами мобильного устройства.

Для исследования характеристик было разработано два идентичных тестовых приложения, написанных на языке Swift и на ReactNative, содержащие экраны, используемые в клиентском приложении телеком-операторов:

- страница авторизации (рис. 1);
- основное меню;
- экран с виджетами;
- карта.

Для тестирования приложений мы будем использовать эмулятор мобильного устройства и профайлер в среде разработки xCode. Три параметра, которые мы будем исследовать: нагрузка на процессор, графический процессор и объем потребляемой памяти.

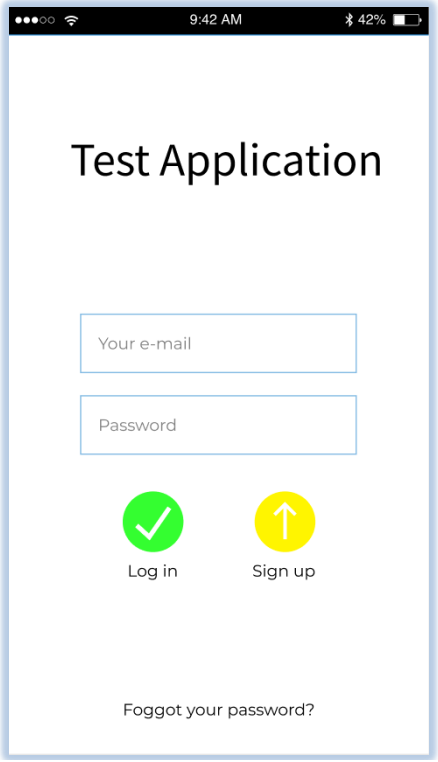

Рис.1. Страница авторизации

Замерим показатели нагрузки на центральный процессор виртуального устройства при использовании тестируемых экранов приложения (табл. 1)..

Рассмотрим каждый показатель отдельно:

1. Страница авторизации

С точки зрения использования ЦП, ReactNative оказался более эффективен (на 1,21%), чем Swift.

2. Основное меню

Во второй вкладке ReactNative снова стал победителем с эффективностью 0,66%.

3. Страница с виджетами

На этот раз Swift определенно стал победителем (на 5,55% эффективнее, чем ReactNative). Выполняя эту задачу, падение производительности и пик потребления ЦП были отмечены во время анимации виджетов.

4. Карты

Swift снова побеждает в этой категории, более рационально загружая CPU (на 9,7%). Следует отметить, что при выполнении этой задачи скачок в потреблении процессора происходит во время перехода на экран «Карты».

Таблица 1

Показатели нагрузки на процессор приложений (фрейм/с) на ReactNative и Swift

	React Native	Swift
Авторизация	16.44	17.65
Меню	26.28	26.94
Виджеты	15.15	9.6
Карты	34.27	24.57

Далее замерим показатели нагрузки на графический процессор виртуального устройства при использовании тестируемых экранов приложения (табл. 2).

Таблица 2

Показатели нагрузки на графический процессор приложений (фрейм/с) на ReactNative и Swift

	React Native	Swift
Авторизация	24.12	25.2
Меню	22.86	16.86
Виджеты	14.47	15.84
Карты	18.6	22.2

Рассмотрим каждую категорию и проанализируем результаты.

1. Страница авторизации

С разницей в 1,08 фрейм/сек, Swift едва ощутимо лидирует. Результаты были зафиксированы при нажатии на кнопку «LogIn».

2. Основное меню

Здесь ReactNative практически подтверждает свое превосходство, работая на 6 фреймов/сек выше, чем Swift.

3. Страница с виджетами

Swift превосходит ReactNative на 1,37 фрейм/сек на вкладке просмотра страницы. Наблюдая за цифрами, было обнаружено, что фрейм/сек увеличиваются до 50, если быстро переключаться между страницами.

4. Карты

Swift здесь явный лидер, так как работает на 3,6 фрейма/сек быстрее, чем ReactNative. Данные получены во время нажатия на кнопку «Карты».

И теперь замерим показатели нагрузки на оперативную память виртуального устройства при использовании тестируемых экранов приложения (табл.3).

Таблица 3
Показатели нагрузки на оперативную память приложений (фрейм/с) на ReactNative и Swift

	React Native	Swift
Авторизация	1.29	1.22
Меню	2.37	2.72
Виджеты	0.33	0.4
Карты	32.98	69

1. Страница авторизации

В этой категории Swift является победителем, потребляя памяти на 0,07% меньше. Хотя таким отклонением можно даже пренебречь. Скачок по памяти был записан в момент нажатия на кнопку «LogIn».

2. Основное меню

В этой категории ReactNative превосходит Swift, используя на 0,35% памяти меньше.

3. Страница с виджетами

Вновь ReactNative обошел Swift, затрачивая на 0,07% памяти меньше. При переключении между страницами резких нагрузок на оперативную память не наблюдалось.

4. Карты

ReactNative здесь твердый лидер, оставил Swift позади с массовым использованием памяти в 36,02% для этой категории. Пик нагрузки записан в момент нажатия на кнопку «Карты». Стоит отметить, что карта работает более плавно на Swift. Скорее всего, это связано с более жестким кэшированием, что мы и видим в результатах

Ссылаясь на исследование технических характеристик и описание проблемы, можно сделать вывод, что для разработки клиентского приложения для портальных решений телеком-операторов более подходит фреймфоркReactNative, поскольку разрабатываемое приложение не включает в себе высоко нагруженные функции, такие как сложные анимации, редактирование видео, работу с данными и в основном содержит бизнес-логику и интерфейсы, API которых полностью предоставлено в ReactNative. Кроме этого, разработка приложения на ReactNative позволит сэкономить на команде разработчиков, поскольку разрабатывается одновременно для двух операционных систем.

Результаты и выводы

В данной работе описана проблема выбора платформы для разработки нативного клиентского приложения для портальных решений телеком-операторов. Описаны требования для клиентского приложения телеком-операторов, выполнен анализ производительности основных компонентов приложения, разработанных на платформах ReactNative и Swift, и по полученным результатам сделан выбор в пользу фреймворка ReactNative.

Библиографическийсписок

1. Facebook Open Source, React Native. Build native mobile apps using JavaScript and React – [Электронный ресурс] / Техническая документация – USA, 2018 – Режим доступа: https://facebook.github.io/react-native/, свободный.

2. Apple Development Documentation, Framework Swift – [Электронный ресурс] / Техническая документация – USA, 2018 – Режим доступа: https://developer.apple.com/documentation/swift/, свободный.

3. Apple Development Documentation, Xcode Release Notes – [Электронный ресурс] / Техническая документация – USA, 2018 – Режим доступа: https://developer.apple.com/documentation/xcode_release_notes, свободный.

4. J'son & Partners Consulting, Стратегии телеком-операторов в сфере мобильных приложений – [Электронный ресурс]/ Электронная статья – Россия, 2014 – Режим доступа: http://json.tv/ict_telecom_analytics_view/strategii-telekom-operatorov-v-sfere-mobilnyh-prilojeniy, свободный.

5. Trnka D. Mobile App Development: ReactNativevs. Native (Android&iOS) – [Электронный ресурс] / Электронная статья – USA, 2018 – Режим доступа: https://medium.com/mop-developers/mobile-app-development-react-native-vs-native-ios-android-49c5c168045b, свободный.

M.D. Andreev, **N.E. Posobilov**

PLATFORM SELECTION FOR DEVELOPING A MOBILE NATIVE CLIENT APPLICATION FOR PORTAL SOLUTIONS TELECOM-OPERATORS

Nizhny Novgorod State Technical University n.a. R.E. Alekseev

This paper describes the problem of choosing a platform for developing a mobile native client application for portal solutions by telecom operators. The performance indicators of typical components of the React Native App and iOS App applications were investigated and, based on the comparison results, the best option was chosen for developing a native application for portal solutions of telecom operators.

Key words: native mobile application, mobile application for telecom operators, performance comparison of React Native App and iOS App.