УДК 004.942

Чурсин А.Н.,
студент магистратуры
ФГБОУ ВО «РЭУ им. Г.В. Плеханова»
Россия, г. Москва

РАЗРАБОТКА КРОССПЛАТФОРМЕННЫХ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ. СРАВНЕНИЕ ПОДХОДОВ И ИНСТРУМЕНТОВ РАЗРАБОТКИ

Аннотация: *В статье рассматриваются и анализируются инструменты и подходы написания кроссплатформенных мобильных приложений. Предложен способ эффективного написания кроссплатформенного кода, отмечены преимущества использования данных подходов в проектах.*

Ключевые слова: *мобильные приложение, программирование, кроссплатформенные приложение, трансляция кода, кодогенераторы, нативные и сторонние языки программирования.*

Annotation: *The article discusses and analyzes the tools and approaches of writing cross-platform mobile applications. The method of effective writing of cross-platform code is offered, the advantages of using these approaches in projects are noted.*

Keywords: *mobile applications, programming, cross-platform application, code translation, code generators, native and third-party programming languages.*

Нативные и кроссплатформенные приложения

Рынку мобильных приложений уже больше десяти лет, однако он до сих пор бурно развивается. Спрос на создание мобильных приложений со стороны компаний постоянно растёт, и он всё ещё заметно превышает предложение, что приводит к постоянному удорожанию разработки. Одно из решений в удешевлении этого процесса — кроссплатформенная разработка, когда один

и тот же код используется на всех платформах, однако у родных (нативных) приложений, все еще есть ряд преимуществ.

Под нативным приложением подразумевается мобильное приложение, которое создается для определенной платформы и непосредственно устанавливается на устройство пользователя

Компании могут изготовить нативное приложение согласно индивидуальным запросам, чтобы затем пользователю было удобно им пользоваться, в дополнение к веб-сайту или другому каналу, которым он уже привык. Эта целостность и является существенным преимуществом нативных приложений.

Преимущества нативных приложений:

- скорость работы и отклика интерфейса. Приложение реагирует на нажатия мгновенно, практически отсутствуют задержки в анимации, скроллинга, получении и выводе данных;
- понятный и простой доступ к функциям и датчикам устройства. Для разработчика не представляет проблемы работа с геолокацией, пуш-уведомлениями, съёмкой фото и видео через камеру, звуком, акселерометром и другими датчиками;
- возможность углублённой работы с функциями смартфона. Как и в предыдущем пункте, такие вещи, как анимации, создание сложных интерфейсов и работа нейросетей прямо на устройствах реализуются, может быть, и не просто, но прогнозируемо;
- родной для платформы интерфейс. Нативные приложения обычно оперируют «платформенными» элементами интерфейса: меню, навигация, формы и все остальные элементы дизайна берутся от операционной системы и потому привычны и понятны пользователю.

Недостаток один — дороговизна разработки и поддержки. Для каждой платформы надо писать свой код. С ростом рынка мобильных приложений разработчики стали не просто дороги, а очень дороги.

Кроссплатформенные приложения пишутся сразу для нескольких платформ на одном языке, отличном от нативного. Тут тоже есть два подхода.

Первый заключается в том, что на этапе подготовки приложения к публикации он превращается в нативный для определённой платформы с помощью компилятора. Фактически один язык «переводится» на другой.

Второй — в том, что к получившемуся коду добавляется определённая обёртка, которая, работая уже на устройстве, на лету транслирует вызовы из неродного кода к родным функциям системы.

Как итог, несмотря на яркие преимущества нативных приложений, хорошо сделанное кроссплатформенное мобильное приложение, компенсирует неудобства тем, что пользователи на более чем одной платформе имеют доступ к вашему продукту или услуге.

Согласно данным аналитических агентств, к концу 2018 года кроссплатформенный рынок приложений достигнет $7,5 млрд. Количество кроссплатформенных инструментов разработки мобильных приложений растет, и у разработчиков появляется огромный выбор ресурсов и инструментов для написания программ для iOS и Android.

Подходы к написанию кроссплатформенных приложений

Стоит выделить четыре подхода к написанию кроссплатформенных приложений:

- Прогрессивное веб-приложение
- Гибридное мобильное приложение
- Нативное приложение на основе js
- Нативное приложение на основе трансляции языка

Каждый вид имеет, как ряд неоспоримых преймуществ, так и множество существенных минусов.

Основным минусом первых двух подходов, является постоянная зависимость от сети интернет. Как прогрессивное веб-приложение, так и грибное мобильное приложение по факту работают в окне браузера, откуда вытекают такие минусы как:

- Требование к подключению к сети интернет
- Невозможность выполнить рекомендации для каждой платформы
- Невозможность поддерживать все новейшие технологии разработчика платформы
- Низкая производительность
- Ограничение по интерфейсу и дизайну

К плюсам этих подходов стоит отнести:

- Высокую скорость разработки
- Несложную в реализации кроссплатформенность

Данные подходы можно реализовать такого инструмента как Adobe PhoneGap.

PhoneGap лучше всего подходит для мобильных приложений, которые не делают ставку на использование встроенных функций смартфона. PhoneGap упаковывает приложения в собственный контейнер, который позволяет JavaScript получать доступ к API устройства так же, как делают обычные приложения. Основным минусом, является низкая производительность приложений, однако стоит упомянуть и о положительных стонах PhoneGap.

К плюсам PhoneGap можно отнести следующие особенности:

- PhoneGap позволяет создавать гибридные приложения с использованием популярных веб-технологий (HTML5, CSS3 и JavaScript), которые легко доступны
- PhoneGap позволяет разворачивать единую кодовую базу на разных платформах, включая iOS, Android, Windows Phone, BlackBerry, Firefox OS и другие
- PhoneGap следует архитектуре плагинов, что означает, что доступ к собственным API устройств и многое другое может быть расширен модульным способом

- PhoneGap позволяет использовать встроенные интегрированные платежи через App Store для iOS, Google Play Store для Android и многое другое

Как итог, стоит отметить, что данные плюсы в состоянии компенсировать все минусы этих подходов, что делает их полезными в крайне редких ситуациях.

Немного более предпочтительным является вариант с Нативным приложением на основе js. За последнее время появилось несколько вариантов написания приложений на js, которые будут использовать собственные компоненты ОС. Наиболее популярным вариантом написания таких приложений является React Native. Он заложен в такие приложения как Instagram, Facebook, Skype. Однако, на мой взгляд, React Native не годится в текущем его состоянии для разработки мобильных приложений. Основной фактор — отсутствие качественных библиотек и детские болезни самого React Native. Качество, производительность и размер приложений оставляют желать лучшего. Для примера возьмем Facebook, использующей данную технологию и Вконтакте. Два приложения со схожим функционалом, однако facebook занимает более 300мб на устройстве, против 64мб, которые занимает Вконтакте. А ведь в вк встроен messenger, в то время как facebook messenger – это отдельное приложение. Что увеличивает разрыв еще сильнее.

Большую часть этих проблем, позволяет решить третий подход. Он заключается в том, что для приложений под разные платформы пишется общая логика, которая затем транслируется на нативные языки для каждой платформы. Это позволяет с одной стороны добиваться максимальной производительности и универсальности, а с другой пользоваться всеми возможностями «родных» для платформы языков. Существует ряд универсальных инструментов для реализации подобного подхода. Ярким представителем является Xamarin.

Xamarin помогает создавать собственные приложения для нескольких платформ с помощью общей кодовой базы C#. С помощью Xamarin можно реализовывать на C# все то же что и на Objective-C, Swift или Java, Это позволяет

везде использовать общую IDE, язык и API. Более того, интеграция Git встроена непосредственно в Xamarin Studio.

К плюсам Xamarin можно отнести:

- Около 75% общего разработанного кода можно использовать на основных мобильных платформах с помощью Xamarin, что значительно снижает затраты и время выхода на рынок, не жертвуя производительностью
- Он обеспечивает тестирование функциональности и обеспечение качества для многочисленных устройств
- Предлагает свой собственный эмулятор Android

Минусы: используя Xamarin, все равно не выйдет использовать многочисленные библиотеки с открытым исходным кодом для разработки под iOS и Android из-за проблем совместимости.

Так же данный подход используется в Monocross.

Monocross - это кросс-платформенный мобильный фреймворк с открытым исходным кодом, который позволяет создавать приложения для iPad и iPhone, смартфонов и планшетов Android, Windows Phone и устройств с поддержкой Webkit. Motocross использует C#, Microsoft .NET и Mono framework для разработки мультиплатформенных приложений.

К плюсам Xamarin можно отнести:

- Мощности API устройств, программируя на C#
- Потребуется знание только языка C#

Минусы: документация, ресурсы и поддержка Monocross, доступные в Интернете, недостаточны, что затрудняет работу по разработке приложений с использованием этой платформы. Так же продукты разработанные Microsoft, в том числе Microsoft .NET, оставляют желать лучшего, как и я зык C# в целом.

Данные инструменты, куда лучше располагают к написанию кроссплатформенных приложений, однако и они имеют ряд минусов, которые можно предотвратить, действуя в рамках подхода трансляции кода.

С помощью утилиты dropbox djinni, есть возможность вынести на сторонний язык только общую логику, оставляя пользовательский интерфейс и особенности систем на нативном языке.

Djinni-это инструмент для создания объявлений типов на разных языках и привязок интерфейса. Он предназначен для соединения C++ с Java или Objective-C. Поддержка Python доступна в экспериментальной версии в ветке python.

Особенности Djinni:

- Создаются параллельные определения типов C++, Java и Objective-C из одного файла описания интерфейса.
- Поддерживаются пересечение примитивных типов трех основных языков и пользовательских перечислений, записей и интерфейсов.
- Создается код интерфейса, разрешающий двунаправленные вызовы между C++ и Java (с JNI) или Objective-C.
- Позволяет автоматически генерировать функции внутри компилятора.

Для того, чтобы понять все преимущества такого подхода, стоит обратиться к реальному проекту. Для примера возьмем приложение для изучения слов английского языка. Приложение должно генерировать карточки со словами, основываясь на ряде параметров. Сложность реализации заключалась в таких как пунктах как:

- Масштабируемость. В приложении должны были регулярно появляться новые упражнения
- Сложность тестирования. Отследить насколько четко алгоритм подбирает слова для каждого пользователя, трудно из-за объемов информации и индивидуальности каждого ученика.
- Высокая вычислительная сложность. Приложение должно иметь одну эффективную логику построения обучения, основываясь на различных данных пользователя.
- Портируемость. В будущем приложение должно заработать на настольных операционных системах и в интернете.

- Возможность работать с приложением без постоянного подключения к сети Интернет

Язык C++ так же выбран не случайно так как:

- Большинство разработчиков сталкивались с C++
- C++ нативно поддерживается на iOS и Android
- Легко отслеживать ошибки

Как итог мы получили универсальный язык, способный генерироваться на самые разные платформы и утилиту кодогенерации Djinni. Данная утилита по заданному IDL (interface description language) файлу генерирует код на Objective-C и Java, а так же прослойку. Данный подход позволил решить все необходимые задачи без существенных недостатков. Данный путь реализации не самый простой, однако наиболее надежный и гибкий.

Вывод

Каждый из описанных подходов имеет право на существование. Веб-зависимые подходы не подойдут громоздким приложениям, работающим с большими объемами данных, требующими быстрый отклик или использующих много графики, однако хороши как небольшие дополнения к сайтам. Способы с трансляцией кода более сложны в исполнении, но позволят добиться хорошей производительности. Некоторые из инструментов, реализующих данный подход еще молоды, но активно развиваются и пополняются новыми возможностями.

В итоге, так же хочется отметить, что разработка приложений с расчётом на мультиплатформенность положительно влияет как на репутацию, так и на продажи разработчика или компании. Это дает возможность утверждать, что количество подходов к разработке будет неуклонно расти и модифицироваться, и рано или поздно появятся более простые, но не менее эффективные способы и инструменты написания кроссплатформенного кода.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Журнал practical ios development [Электронный ресурс] / Режим доступа: https://medium.com/practical-ios-development/introduction-to-ios-unit-tests-6ea1ff4e961ctutorial (дата обращения 12.07.2018)

2. Сайт компании raywenderlich [Электронный ресурс] / Режим доступа: https://www.raywenderlich.com/150073/ios-unit-testing-and-ui-testing-tutorial (дата обращения 12.07.2018)

3. GitHub [Электронный ресурс] /. — [Электронный ресурс]— Режим доступа: https://github.com/dropbox/djinni (дата обращения 12.07.2018)

4. Hongkiat [Электронный ресурс] /. — [Электронный ресурс]— Режим доступа: https://www.hongkiat.com/blog/cross-mobile-platform-framework-wora/ (дата обращения 12.07.2018)

5. Habr [Электронный ресурс] /. — [Электронный ресурс] — Режим доступа: https://habr.com/post/319348/ (дата обращения 12.07.2018)