Санкт-Петербургский Государственный Университет
Математико-механический факультет
Кафедра системного программирования

Тиражирование мобильных приложений для платформы iOS и Android

Курсовая работа студента 361 группы

Коршакова Степана Андреевича

Научный руководитель
Сабашный Вадим Евгеньевич,
Директор департамента мобильных технологий ЗАО «Ланит-Терком».

Санкт-Петербург - 2012

Введение

Рынок мобильных приложений за последний год вырос больше, чем в два раза [1], однако стоимость их разработки по-прежнему остается достаточно высокой.

В среднем разработка двух приложений для интернет-магазина под iOS и Android OS стоит от 100,000 рублей за простую версию, которая не отвечает современным требованиям пользователя по юзабилити, красоте и эргономике. Достаточно хорошее приложение стоит, которое по функционалу повторяет рядовой магазин и имеет продуманный интерфейс и приятный дизайн уже стоит от 500,000 рублей.

Данные цифры чрезвычайно высоки для любого магазина (не обязательно интернет-магазина) и очень редко кто согласится на такие траты.

Вторая сторона проблемы стоимости разработки – отсутствие достаточно удобного инструментария для разработки, отсутствие любых CMS, развитых платформ для разработки. Очень часто разработчики (в развитых организациях) создают для себя свои фреймворки для упрощения разработки и интеграции с разными сервисами.

Постановка задачи

Требуется следующее:

- Создать инфраструктуру для тиражирования типовых приложений с возможностью модификации приложений под нужды конкретного заказчика.
- Изучить методы оптимизации работы отдельных компонент системы – работы сетевой инфраструктуры и работы с изображениями

Обзор существующих решений

Решение для упрощения разработки

Одно из наиболее распространенных решений – это разработка с использованием HTML5 и JavaScript.

Как правило, приложения создаются с помощью библиотеки jQuery Mobile, которая, однако, до сих пор не работает достаточно стабильно, что бы приложения можно было считать качественным.

Если не использовать при разработке jQuery Mobile, то стоимость будет несколько уменьшена, но будет оставаться на уровне разработки одного приложения под iOS и Android. Однако, не смотря на универсальность, придется делать кастомизации приложения под разные платформы. Особенно это заметно будет при создании приложения под Windows Phone 7, где интерфейс сильно отличается от классических интерфейсов Android и iOS.

Другой подход – делать разработку под Android, iOS, Windows Phone на одном языке – на языке C#, это позволит объединить логику приложения между разными платформами, но иметь возможность разработки интерфейса отдельно под каждую платформу с помощью инструментов, специально созданных под эту платформу.

Решения для создания типовых мобильных приложений

Существует несколько сервисов, которые позволяют создать типовые приложения.

- http://www.biznessapps.com/ - позволяет создавать типовые приложения на основе идей вкладок. Пользователь может редактировать разные варианты вкладок в приложении.
- http://getshopapp.com/ - позволяет создать приложение для интернет-магазина на базе HTML. Живых магазинов в андроид-маркете не имеют.

Реализация

В рамках курсовой работы было создано средство для тиражирования типовых приложений для интернет-магазинов, проведено исследование по оптимизации разработки кастомных приложений на базе типовых, а так же подготовлена инфраструктура для разработки других типовых решений.

Серверная часть

Серверная часть состоит из нескольких компонент:

1. Multishop-сервер – с ним работают все мобильные приложения. К нему самые высокие требования из всех компонентов системы.
2. Build-сервера – заняты сборкой приложений по готовым данным
3. Amazon S3 – облако от Amazon используется в качестве хранилища всех сборок приложений и для хранения конфигурации, которая используется build-серверами для создания сборок
4. Конфигуратор – интерфейс пользователя для подключения магазинов, управления каталогами, для конфигурирования и создания приложений

Мною разработаны все, кроме первой компоненты.

Как база для сборки приложений используется продукт от JetBrains – TeamCity, очень удобный для конфигурирования билд-сервер.

Конфигуратор сверстан на базе Twitter Bootstrap.

Процесс создания приложения

Процесс создания приложения состоит из трех частей

1. Подключение интернет-магазина к системе
2. Создание конфигурации приложения
3. Непосредственно сборка приложения

Подключение

Пользователь заполняет настройки подключения к интернет-магазину и адрес электронной почты для отправки заказов пользователей в конфигураторе. После этого конфигуратор регистрирует в сервере Multishop магазин, который в свою очередь возвращает ссылку для API, которая используется в приложении.

Создание конфигурации и запрос сборки

Пользователь заполняет требуемые поля, загружает графику в конфигураторе, все это сохраняется в БД конфигуратора.

После этого пользователь выбирает «Запросить новую сборку», Конфигуратор загружает всю конфигурацию и графику на Amazon S3, а затем обращается к build-серверу и запрашивает новую сборку с этой конфигурацией.

Процесс сборки

Как только очередь дойдет до запрошенного приложения, то начинается сборка на build-агенте – компьютере, который подключен к build-серверу и имеет возможность собирать приложения для определенной платформы.

Build-агент сначала скачивает с Amazon S3 конфигурацию и ресурсы, затем простым копированием перезаписывает ресурсы приложения ресурсами из конфигурации и записывается конфигурационный файл.

Далее происходит исполнение специфичных для каждой платформы скриптов по интеграции конфигурации в приложение (например, поменять id приложения).

После этого при успешной сборке все загружается обратно на Amazon S3.

Для iOS есть некоторые особенности для сборки: сборка делается два раза – Debug и Release. Debug в последствии загружается на TestFlight, который позволяет просто устанавливать отладочные версии приложений для проверки их на айфоне тестировщиками. Release – это готовый пакет для отправки в App Store.

После сборки конфигуратор оповещается о результате сборки.

Реализация приложений

Особенности реализации приложения под Android

Приложение разделено на три отдельных проекта – два Library Project и обычный Android Project.

Первая часть – Framework

Она содержит в себе:

- Модуль для загрузки изображений. Позволяет в фоне в два потока загружать изображения – один скачивает, а другой загружает с диска, а потом уведомляет заинтересованные объекты в загрузке.
- Общая модель приложения – базовые классы с утилитными методами, которые включают в себя:
    - Определение параметров устройства
    - Создание и сохранение installationId (нужен из-за несовершенства deviceId в андроид)
    - Удобрая работа со статистикой –Google Analytics и Flurry
    - Методы для загрузки изображений
    - Конфигурационные переменные и удобная работа с ними
    - Ссылки на данные приложения
    - Базовое сетевое взаимодействие
        - Оптимизировано под работу в плохих условиях
        - Gzip
        - Выключен 100-continue, т.к. он в среднем в два раза удлинял время запроса
        - Реализация базового протокола взаимодействия с системами
- Модуль для работы с магазинами – реализация внутреннего протокола для работы с сервером Multishop

Framework сильно упрощает разработку на начальном этапе и можно практически сразу приступать к дизайну модели, интерфейса и методов протокола, что существенно повышает скорость любых разработок приложений, которые являются лишь интерфейсом к готовому бизнесу.

Вторая часть – Интерфейс приложения

Данная часть содержит в себе исключительно интерфейс приложения, запись данных в статистику и вызов методов из фреймворка.

Основная особенность проектирование интерфейса – это требование возможности отключать ту или иную функциональность и возможность менять «скины» приложения.

Третья часть – Приложение Android

В этой части содержится AndroidManifest.xml, который описывает все приложение для системы Android.

Необходим в виду того, что packageId очень сложно менять для одного определенного проекта – это заденет ВСЕ исходные файлы этого проекта. Потому было сделано решение добавить третий уровень, который не содержит в себе код.

Так же этот проект содержит конфигурационные и другие вспомогательные файлы (например, файлы справки) непосредственно, т.к. их невозможно положить в Android Library.

Особенности реализации приложения под iOS

Разработка под iOS совершается с помощью инструмента MonoTouch на языке C#. Данный инструмент был выбран ввиду того, что заплатить 400$ дешевле, чем осваивать новый язык Objective-C, к тому же C# можно использовать и на других платформах, например, Windows, Windows 8 RT, Windows Phone 7/8.

Приложение под iOS является портированной версией приложения Android, однако оно создано с учетом особенностей платформы.

В целом, приложение логически разделено на две части, но они, пока что находятся в одном проекте: Framework и Интерфейс.

Первая часть - Framework

С приходом Windows Phone будет выделено как минимум два проекта – кроссплатформенный и зависящий от платформы, но сейчас это все вместе находится в одном фреймворке.

По своей сути он является портированием фреймворка с платформы Android и по функционалу повторяет его.

Вторая часть – интерфейс

Имеет те же свойства, что и в версии под Android.

Оптимизация работы с сетью

При использовании Trial-версии Windows Azure, в которой выдаются два сервера в азии, были замечены серьезные проблемы с работой приложения из-за удаленности серверов.

В среднем на пустой запрос уходило больше секунды. Запрос каталога занимал больше 20 секунд, что было неприемлемо.

При работе в метро приложение было очень сложно заставить работать – постоянно выводилась ошибка связи.

Было выявлено несколько проблем:

1. Слишком большой пакет данных требовалось загрузить
2. Обрывы связи. В связи с п.1 повышает вероятность необходимости повторного запроса.
3. Большая удаленность сервера и большое время перемещения пакета из европы в азию и обратно
4. Использовался 100-continue – флаг в HTTP протоколе, при котором клиент ожидает «разрешение» на отправку данных запроса на сервер прежде чем отправить их
5. Отсутствие сжатия в запросах

Результаты оптимизаций

Сначала была предпринята попытка добавить сжатие в запросы.

В среднем это уменьшило 50кБ категорий до 10кБ, что уже значительно ускорило загрузку.

Однако, для больших магазинов (вроде sotmarket.ru) не сжатые категорий занимали до 150кб и сжимались до около 30кб. Это порождало на Edge/gprs задержки до 20 секунд.

Как результат была произведена оптимизация протокола и постепенная подргузка каталога – по уровням вложенности. Это уменьшило объемы данных каталога меньше 5кб в худшем случае, что, как правило, уже приходило за один запрос и быстро.

Однако, при работе с сервером из азии мы столкнулись с проблемой 100-continue и при отключении ее мы в два раза ускорили работу приложения.

Как результат – загрузка любой категории каталога на EDGE (и даже GPRS) происходит гарантированно меньше 10 секунд, если, конечно, нет никаких серьезных проблем со связью. Стало возможным использовать приложение при остановках в метро или находясь далеко за городом.

На статистике количество отказов пользователей упало с 50% до 5%-9%. До этих оптимизаций пользователи после первого экрана в 50% случаев не дожидались загрузки каталога.

Оптимизация работы с изображениями

Работа с изображениями – очень важная часть системы – их надо загружать из сети, сохранять, кешировать и отображать. Все это должно быть очень быстро даже на слабых смартфонах, т.к. от этого серьезно зависит User Experience. Быстро означает не только скорость отклика системы, но и экономию батареи.

Для оптимизаций работы с изображениями сделаны две вещи:

1. Система оптимизаций изображений на стороне сервера
2. Оптимизация работы в приложениях

На данный момент сервис на стороне сервера отключен по причине того, что мы не знаем сколько это будет стоить в плане серверных мощностей и траффика + нет уверенности в стабильности решения.

Оптимизация изображений на серверной стороне

Был создан сервис для оптимизации размера и объема графики. Сделан на базе продукта ImageMagik, как наиболее распространенный среди консольных приложений для конвертирования графики.

Основной сервер посылает запрос на этот сервис и регистрирует изображение с нужными размерами и типом масштабирования, а в ответе сервер выдает новую ссылку на изображение. При первом запросе изображения сервис производит загрузку исходного, его конвертирование и сохраняет ее в кеше.

Оптимизация работы с изображениями в приложениях

Загрузка изображений происходит в два потока. Приоритеты потоков были настроены для обеспечения наиболее хорошего User Experience – не замедляют работу интерфейса и других систем телефона, но не работают слишком медленно, что бы пользователю надо было ждать загрузки слишком долго.

Процесс загрузки изображения

Два потока – Loader и Downloader, которые загружают в память и из интернета на диск соответственно.

Так же есть простой кеш в памяти – MemoryCache.

Имена файлов на диске представлены в виде base64-закодированной строки ссылки. Как простой и универсальный способ сделать escape.

Процесс выглядит следующим образом:

1. Приходит запрос в поток Loader
2. Если есть в MemoryCache тут же возвращаем значение, иначе добавляем в очередь в Loader и ничего не возвращаем.
3. Подходит очередь загрузки в Loader’е.
4. Если файл есть на диске и его можно открыть, то оповещаем об успешной загрузке изображения

5. Если файл отсутствует, он старый или не открывается, то отправляется запрос на загрузку в Downloader
6. Подходит очередь загрузки в Downloader’е
7. Если файл уже есть на диске – пытаемся его открыть, если открывается – оповещаем об успешной загрузке изображения
8. Иначе пытаемся загрузить файл из сети
9. Если загрузили – сохраняем и делаем запрос в Loader – переходим к п.1
10. Иначе ждем секунду и повторяем попытку
11. После нескольких попыток – уведомляем о невозможности загрузить изображение

Такой процесс позволяет быстро загружать то, что уже и так находится на диске, перезагружать файлы, которые побились (были такие случаи), а так же это не мешает интерфейсу и пользователь не испытывает дискомфорта.

Заключение

Итогом работы является система для автоматического тиражирования приложений для интернет-магазинов для платформ Android и iOS.

Изучены способы оптимизации работы с изображениями и оптимизация работы с сетью для обеспечения высокого качества работы сервиса со стороны пользователя.

Дальнейшая работа заключается в унифицировании разработки типовых приложений и создание более общего фреймворка.

Вывод фреймворка и инструментария в OpenSource и дальнейшая его поддержка.

Источники

[1] http://techcrunch.com/2011/12/20/distimos-year-end-report-shows-why-developers-love-ios-iphone-4x-android-revenue-ipad-2x/