О сервисе рассылки push-уведомлений

Пустобаев А.И.

***Аннотация*—Работа посвящена исследованию касающегося созданию модели сервиса рассылки уведомлений. Рассматривается общая архитектура сервиса, применяемые подходы к разработке, а также технологии, используемые для уведомления клиента. Представлена разработанная клиент-серверная модель для сервиса рассылки уведомлений. Описаны отдельные компоненты мобильной экосистемы, а также то, как они связаны друг с другом.**

***Ключевые слова*— push, уведомления, мобильный клиент, CMS.**

I. Введение

В настоящее время тенденция отказа от традиционного обмена сообщениями между мобильными устройствами с помощью SMS технологии вполне очевидна. На смену SMS (MMS) приходят мобильные приложения обмена мгновенными сообщениями или MIM (mobile instant messaging applications) [1]. Сообщения этих приложений часто имеют больше социальный, неформальный и разговорный характер, в то время как SMS считаются более личными и, в целом, более надежными. Так же MIM приложения позволяют мобильным пользователям в реальном времени передавать текстовые сообщения одному пользователю или группе пользователей бесплатно.

Для приема или отправки SMS используется стандартный протокол SMPP [2] или некоторые надстройки над ним. Такого рода сервис по определению является операторским, он привязан к некоторому номеру, который может быть обычным телефонным или коротким сервисным. Современные тенденции развития телекоммуникационной отрасли показывают постоянную миграцию пользователей (абонентов) в сторону альтернативных программ обмена сообщениями, MIM. В работе [3] в качестве транспортного уровня для информационных систем рассматривается *Twitter*, в работе [1] – *WhatsApp*. В мобильных приложениях широко используется механизм push-уведомлений. Например, в работах, выполненных в лаборатории ОИТ [4][5]. Механизм рассылки таких уведомлений поддерживается практически всеми производителями мобильных операционных систем. Современные версии настольных операционных систем так же предоставляют возможность его использования.

В данной работе рассматриваются push-уведомления в качестве механизма доставки сообщений. Технология Push (server push) описывает один из способов доставки контента мобильным пользователям посредством сети Интернет. Данная технология использует две модели: клиент-сервер (client/server) и публикатор-подписчик (publisher/subscriber). Мобильный клиент (subscriber) подписывается на рассылку push-уведомлений, используя установленное приложение. Сервер публикаций (publication server) осуществляет рассылку подписавшимся клиентам. Рассылка может производиться или по инициативе публикатора (publisher), или по наступлению определенного события, например, по наступлению определенной даты и времени.

При создании мобильных приложений возникают новые, характерные для них проблемы. Существуют различные фреймворки, частично позволяющие решить их. Например, проблемы, касающиеся различия в самих мобильных устройствах и в их вычислительных мощностях. Важно понимать, какие инструменты имеются у разработчика и как они могут быть интегрированы для создания приложений высокого качества. Например, кросс-платформенная мобильная разработка с использованием Titanium позволяет сократить время, требуемое для появления продукта на рынке. Невысокая вычислительная мощность мобильных устройств может быть улучшена ресурсами облака. Синхронизация данных в распределенной системе требует сигнальных механизмов. Сторонние сервисы предоставляют плагины, которые могут использовать функциональные возможности и улучшить пользовательское впечатление от продукта, его восприятие. Создание успешного мобильного приложения требует понимания каждого компонента этой мобильной экосистемы и так же понимание того, как они могут быть соединены вместе.

В данной статье мы анализируем наиболее важные компоненты, составляющие инфраструктуру сервиса рассылки push-уведомлений, а также представляем инструменты для производительной разработки.

Мы так же покажем, как эти компоненты могут взаимодействовать друг с другом надежным и масштабируемым образом. В сервис рассылки так же входит мобильное приложение для подписки на информационные каналы и получения уведомлений со смартфонов.

Компоненты мобильной инфраструктуры вместе с инструментами, анализируемыми в этой статье, включают клиентское мобильное приложение (созданное с помощью Xcode), облачный сервис,

Статья получена 2 мая 2015.

А.И. Пустобаев - выпускник магистратуры ВМК МГУ имени М.В.Ломоносова. (email: lesha.pus@gmail.com)

взаимодействующий с клиентом (созданный с помощью Ruby фреймворка Rails и размещенный на серверах Heroku), а также сторонние сервисы для push-уведомлений.

II. Общая архитектура системы

Архитектура, которую мы используем, представлена на рисунке 1.

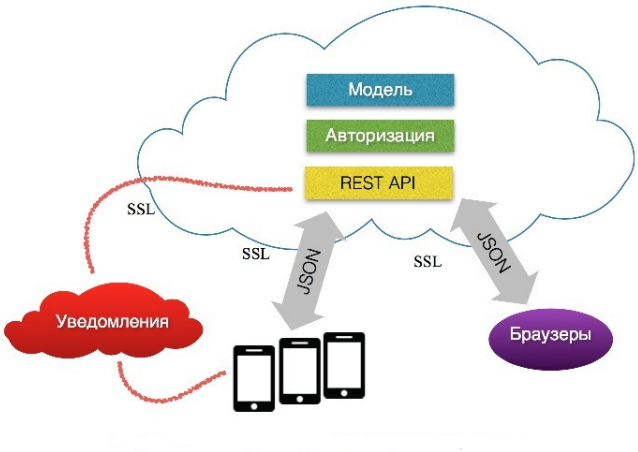

Рис 1. Архитектура системы уведомлений

Здесь могут быть идентифицированы два типа компонентов. Первый тип включает проприетарные компоненты. Базируется на специальных фреймворках и включает: компоненты в проприетарном облаке (обозначает услугу построенную разработчиками, которая размещается в Heroku в нашем примере), мобильное приложение, браузерный клиент.

Второй тип компонентов представлен сторонними сервисами, работающими через некоторый определенный API.

В данной работе мы используем первый тип внешних сервисов: Сервис уведомлений – используемый для рассылки push-уведомлений (способ оповещения мобильного устройства). Так же в эту категорию можно отнести, социальные услуги, которые предлагают интеграцию с Facebook и Twitter, сервисы аналитики по отслеживанию модели использования, сервис SMS шлюз для отправки SMS с кодом активации (требуемый по бизнес модели), и сервисы геокодирования и обратного геокодирования для преобразования гео-координат в географический адрес.

Для понимания того, как вся система работает вместе, рассмотрим наиболее важную функциональную возможность представленного приложения: отправка сообщения всем подписчикам канала рассылки.

Когда владелец канала через веб-форму создает в канале сообщение, оно посылается на сервер посредством серверного RESTful API. Сервер сохраняет сообщение в базе данных. Помечает его состояние как новое или неотправленное. Так же отмечается его принадлежность к конкретному каналу. По наступлению некоторого временного события (например, по таймеру) сервер для каждого нового сообщения определяет его канал и всех подписчиков этого канала с помощью опроса базы данных. Каждое сообщение преобразуется в в JSON-текст. После этого происходит рассылка push-уведомлений с новыми сообщениями подписчикам. Все запросы посылаемые проприетарным серверным компонентам должны использовать SSL для обеспечения безопасности в системе. В случае отправки запроса из мобильного приложения, запрос преобразуется в JSON, подписывается, используя ключ пользователя, и отправляется на сервер посредством серверного API [6].

III. Нативная кросс-платформенная мобильная разработка

Даже если на рынке доминирует два крупных игрока (Android и iOS), разработка для этих двух фреймворков является трудоемким занятием, так как очень мало сделанного для одного может быть использовано для другого. Языки программирования и среды разработки для iOS и Android сильно различаются. Кроме того, если еще одна ОС появится на рынке в будущем, потребуется переписывание приложения.

В целях повышения производительности может быть использован кросс-платформенный подход. В настоящее время существует широкий спектр доступных кросс-платформенных решений, включая Appcelerator Titanium, RhoMobile и PhoneGap. Appcelerator Titanium – open source проект, который имеет наиболее общие функциональности iOS и Android. Разработка с Titanium отличается от других кросс-платформенных решений в том, что позволяется нативный код. Например, используя PhoneGap вы, на самом деле, разрабатываете веб приложение, которое запускается в WebView UI компоненте. Используя Titanium, вы программируете на JavaScript, а фреймворк переводит код в нативный Java или Objective-C.

Среда разработки Titanium использует свой собственный IDE на основе Eclipse, Titanium Studio, с языком программирования – JavaScript. Использование JavaScript в качестве языка программирования дает преимущество низкого порога вхождения по сравнению с Objective-C и Java.

Одна из отличительных особенностей разработки на JavaScript состоит в том, что требуется понимание организации кода при решении крупных проектов. Это связано с тем, что язык ранее использовался только для написания маленьких блоков кода в браузерах. Рекомендуемая практика разработки в Titanium – использование MVC шаблона. Так же, несмотря на разногласия в том, является ли JavaScript объектно-ориентированным языком или нет, рекомендуется применение лучших практик ООП, в частности инкапсуляции. Использование шаблона MVC помогает создавать код чище и способствует переиспользованию.

Альтернативой кросс-платформенной разработке является разработка под конкретную платформу. Для некоторых платформ, плюсом такой разработки может являться то, что доступны все новые возможности последней версии операционной системы. Сюда, помимо функциональных нововведений, так же относится и стиль интерфейса. Это может быть важно для привлечения интереса к новому продукту, особенно

среди Apple пользователей. А если разработка идет в кросс-платформенном фреймворке, то про доступность новинок ОС говорить можно не всегда. Обычно разработчикам фреймворка требуется время для адаптации к нововведениям в ОС. А у кросплатформенных фреймворков с разрешенным нативным кодом так же есть ограничения.

Для данного проекта была выбрана нативная разработка мобильного приложения в Xcode.

IV. Использование шаблона MVC

Рассмотрим три компоненты данного приложения: модель, представление и контроллер.

Представление отвечает за создание окна и добавление нужных UI компонент. Представления могут дополнительно определять стилевые свойства (цвет, шрифт и позиционирование). Рекомендуемая практика – хранить все свойства стиля во внешнем файле. Это позволяет легко модифицировать и настраивать эти свойства. Рисунок 2 показывает представление для приложения получения уведомлений. Оно состоит из окна, на которое добавлены текстовые поля и кнопка. Нажатие кнопки создает событие, которое обрабатывается контроллером.

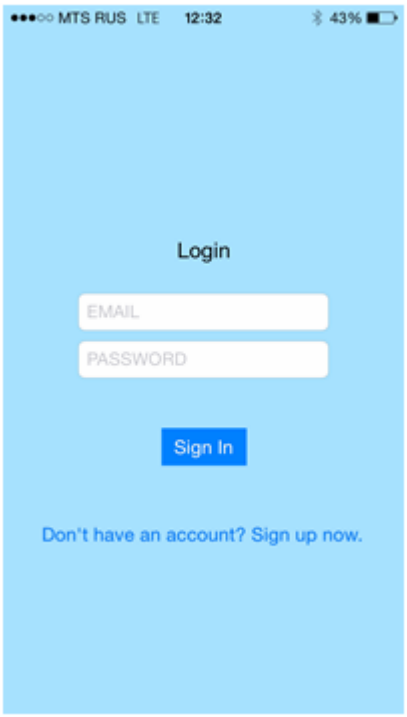

Рис. 2 Представление

Контроллер отвечает за управление событиями (нажатия кнопок). Например, когда зарегистрировано событие нажатия на кнопку *Sign In*, уведомление вместе с данными, введенными пользователем в текстовые поля, отправляется в модель. Некоторые разработчики для связи с сервером используют обработки событий внутри контроллера. Хотя эта практика и удобна для простых представлений, это приводит к неорганизованному коду в случае, если есть несколько доступных сервисов. В таком случае следует переместить задачи серверных взаимодействий на модель. Там они могут быть инкапсулированы и могут обеспечить лучшее переиспользование.

Модель, как правило, оперирует с данными в формате JSON, извлеченными из RESTful сервиса. Например, метод *onResponse* возвратит информацию, полученную от сервера в качестве ответа.

Преимущества использования MVC включают улучшение организации и читабельности кода, а также способствуют повторному использованию и облегчают сопровождение системы.

V. Уведомления для веб-клиентов

В то время как отправка данных на сервер может быть реализована достаточно просто, получение и обновление входящей информации требует новых подходов из-за того, что с мобильного пользователя может дополнительно взиматься плата за трафик, а также для того, чтобы избежать увеличения расхода ресурса батареи [7].

Традиционно веб приложения используют polling технологию для получения обновлений. Например, биржевой робот знает, что обновления приходят каждые 10 секунд. Поэтому разумно сделать таймер, по которому получать обновления с сервера каждые 10 секунд. Для приложения, которое связано с получением уведомлений по подпискам эта схема не подходит, так как polling интервал неизвестен. Например, сообщения в канал могут не посылаться в течении нескольких часов, и при этом клиент сделает poll несколько тысяч раз чтобы получить ожидаемый результат, а с другой стороны сразу может быть отправлено много сообщений в течении нескольких минут. Например, с интервалом меньше 2 секунд, если клиент подписан на несколько каналов, что будет означать высокую нагрузку на сервер и низкую масштабируемость.

Long polling - это разновидность традиционной технологии polling, которая позволяет эмулировать доставку информации по инициативе сервера [8]. Используя long polling, клиент запрашивает информацию с сервера таким же способом, как и при polling. Однако, если сервер не имеет новой информации доступной клиенту, вместо того, чтобы отправить пустой ответ, сервер удерживает запрос и ждет поступления доступной клиенту информации. Рисунок 3 иллюстрирует эту функциональность, сравнивая polling основанный на времени и long polling. Как только информация становиться доступной (или после достаточного тайм-аута), ответ отправляется клиенту. Клиент моментально снова запрашивает информацию с сервера. То есть, сервер всегда имеет доступный ожидающий запрос, который он использует для отправки данных в ответ на событие.

Long polling сам по себе не является push технологией, но может быть использован в условиях, когда реальный push не представляется возможным.

Используя long polling в мобильном приложении, клиент получает уведомление о необходимости обновить свои данные, необходимости синхронизации с сервером, как только новые данные стали доступны.

При использовании подхода long polling мы имеем низкую нагрузку на сервер и высокую масштабируемость.

Рис

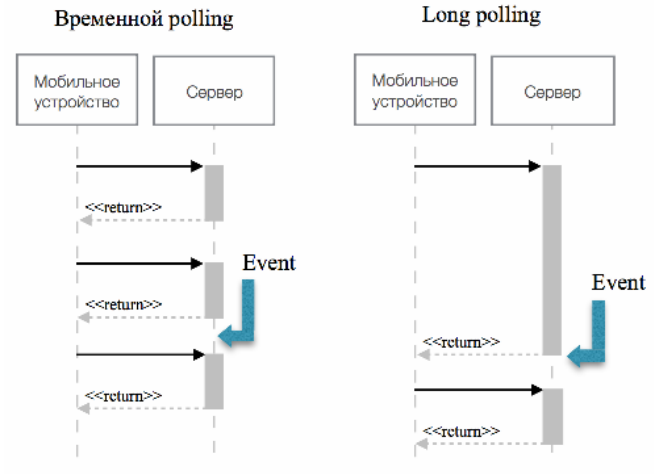

Рис. 3 Long polling

VI. PUSH-УВЕДОМЛЕНИЯ

Мобильным приложениям разрешается осуществлять только определенную активность в фоновом режиме. Это связано с необходимостью сохранять заряд батареи.

Естественно, во многих случаях требуется какой-то способ, чтобы предупредить пользователя об интересных событиях, которые случаются, когда пользователь не находится в соответствующем приложении. Мобильная Операционная Система предоставляет такое решение. Вместо того, чтобы приложение постоянно проверяло наличие новых событий и выполняло бы работу в фоновом режиме, эту задачу можно переложить на сервер.

Когда интересуемое событие появляется, серверный компонент может послать приложению push-уведомление, которое рассматривается приложением, как сигнал, того, что на сервере появилась информация о новом событии. Push-уведомления изначально не рассматривались как способ передачи данных клиенту. Они использовались как способ сигнализации, когда приложение не запущено.

Push-уведомления не гарантируют доставки сообщения (как и SMS). Поэтому, например, вы можете использовать эту технологию для предупреждения приложения, что пользователь имеет новые сообщения, но, скорей всего, не будете использовать её для передачи самого сообщения.

Рисунок 4 показывает, как работают push-уведомления. Приложение должно зарегистрироваться в APNS (Apple Push Notification Service) для Apple устройств и GCM для Android [9][10]. В результате регистрации приложение получает ID, называемое токеном устройства, которое приложение может отправить на сервер. Если серверу нужно отправить push-уведомление, он использует этот токен устройства, чтобы указать приложение, которому, он хочет отправить уведомление. Токены устройств могут меняться со временем. Поэтому, приложению следует обновлять информацию о токене устройства каждый раз после запуска приложения. Пользователь может так же не разрешить push-уведомления. Они опциональны. Или же вовсе удалить приложение. В таком случае уведомление не будет получено пользователем. Сервисы уведомлений предоставляют API, используя который сервер может периодически запрашивать какие токены устройств всё еще доступны. Из-за того, что взаимодействия между сервером и сервисами уведомлений не тривиальные, существуют разные служебные сервисы для организации push-уведомлений: Urban Airship, Parse и др [11]. Другие операционные системы, например Windows Phone или Blackberry, так же предоставляют push-уведомления.

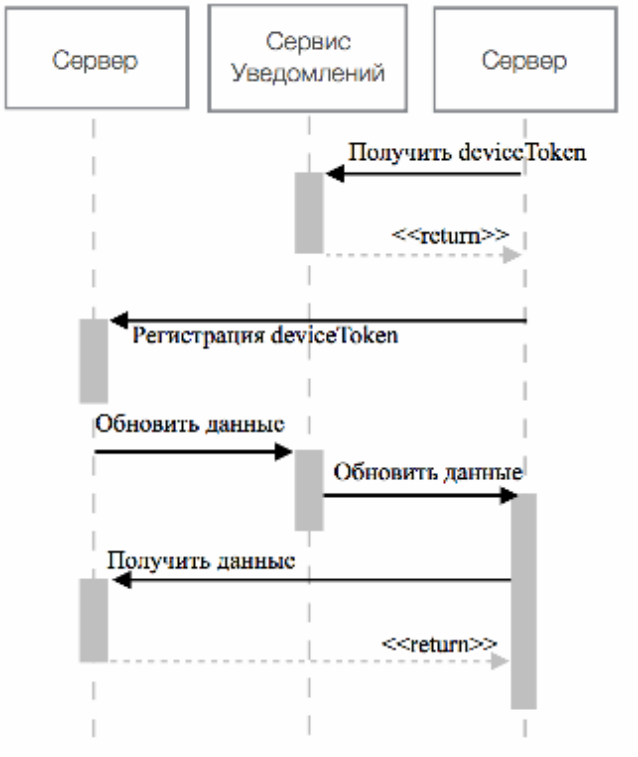

Рис. 4 Push-уведомления

Push уведомления удобны на мобильных устройствах тем, что они экономят заряд батареи. Это достигается за счет двух основных факторов. Во-первых, приложению, принимающему уведомления, не нужно периодически опрашивать сервер публикаций, или какой-либо другой сервер. За него это делает операционная система. Подобных приложений на одном устройстве может быть много, а такое централизованное решение уменьшает количество запросов к серверу, что положительно сказывается на заряде. Во-вторых, клиентское приложение, вообще, может не выполняться даже в фоне. Уведомление всё равно будет получено операционной системой. В некоторых случаях закрытое приложение может получить на некоторое время управление и сигнал о том, что пришло новое уведомление.

Также, экономичность push-уведомлений достигается за счет введения дополнительного звена, посредника между сервером-отправителем и приложением-получателем. Это звено у разных производителей

операционных систем разное. Общее название - служба уведомлений PSP (Push Notification Provider). Наиболее известные PSP:

- GCM (Google Cloud Messaging) для мобильных устройств под Android;
- APNS (Apple Push Notification Service) для устройств под iOS, OSX;
- MPNS (Microsoft Push Notification Service) для устройств под Windows [11].

VII. Модель сервиса уведомлений

Для сервиса рассылки уведомлений была разработана модель, в базе данных сервера которой имеются такие сущности: пользователь (*User*), устройство (*Device*), канал (*Channel*), подписка (*Subscription*), сообщение (Message) и разрешенное приложение (*PermittedApp*). На рисунке 5 показана ER диаграмма базы данных сервера.

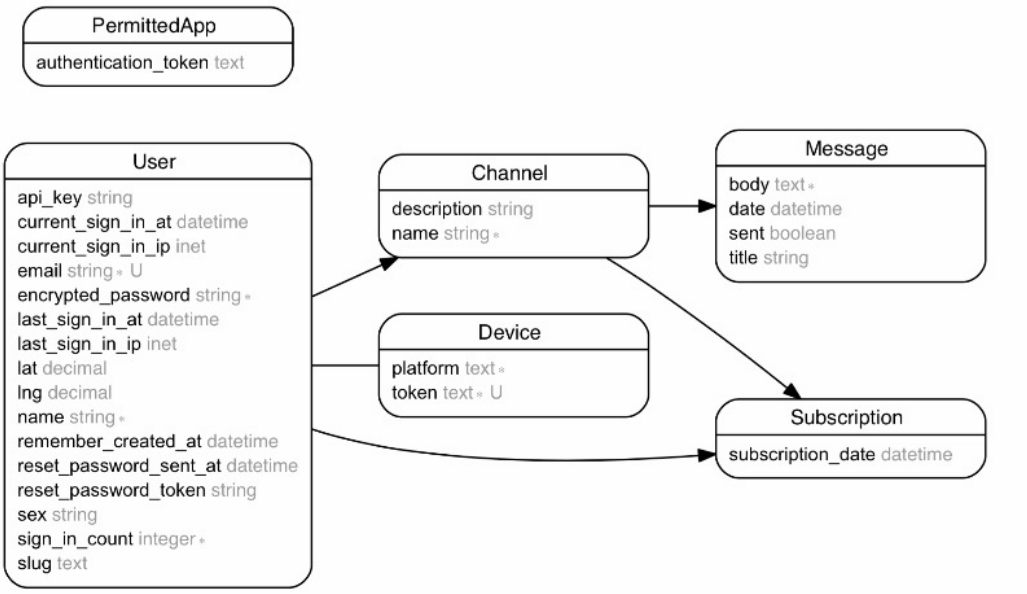

Рис. 5 ER-модель

В данной модели каждый пользователь имеет одно устройство. Устройство состоит из платформы/типа и токена устройства. Значение типа может быть, например, *ios* или *androi*d. Токен получается мобильным приложением и отправляется на сервер. Он нужен для рассылки push-уведомлений.

Каждому пользователю соответствует множество собственных каналов. Пользователь является их создателем и владельцем.

Помимо личных каналов, есть возможность подписаться на чужие каналы. Каждому пользователю соответствует множество каналов через подписки. Одна подписка - один канал. Сохраняется дата создания подписки.

Канал состоит из названия (name), описания/расшифровки (description) и имеет множество сообщений. Каждое сообщение соответствует только одному каналу.

Сообщение состоит из заголовка (title), тела (body), даты создания (date) и статуса отправки (sent).

Пользователь состоит из имени (name), электронной почты (email), пола (sex), его gps-координат (lat, lng), зашифрованного пароля (encrypted_password), API-ключа (api_key) и некоторой дополнительной административной информации нужной для смены/восстановления пароля, учета количества авторизаций одного и того же пользователя и другого.

Сущность - разрешенное приложение хранит http_token, требуемый для отправки запроса на регистрацию или авторизацию пользователя из мобильного приложения. В данной модели этот токен (токены, если разные приложения) известен заранее. Они также хранятся в архиве мобильного приложения.

После авторизации по электронной почте и паролю приложение получает и сохраняет уникальный ключ. Этот ключ - некоторый хеш код, с помощью которого сервер аутентифицирует пользователя в каждом api-запросе отличном от регистрации и авторизации. В остальных api запросах аутентификация происходит с помощью HTTP токена, который представляет из себя строку “*email:api_key*”, где email - почта пользователя, api_key - полученный уникальный ключ. Хеш код нужен для того, чтобы пароль не хранился на устройстве. После выхода пользователя из системы (logout) приложение удаляет из памяти выданный хеш код.

Таким образом, после авторизации мобильное приложение может получать информацию разрешенную пользователю по API сервера.

В начале работы с системой пользователь регистрируется либо через веб форму, либо через мобильное приложение. После регистрации нужно сделать вход в систему на веб ресурсе и в приложении. На веб ресурсе можно создать свой канал и/или подписаться на существующий. После этих действий мобильное приложение начнет принимать push уведомления от интересующих пользователя каналов рассылки.

В приложении отображаются каналы и принятые сообщения с датой отправки каждого. Также через приложение отображаются некоторые данные о

пользователе. Каналы представляются списком с возможностью перейти к сообщениям. Сообщения представлены сплошным текстом (логом).

На рисунке 6 показана ER диаграмма базы данных клиента (мобильного приложения). Получаемые данные, push-уведомления сохраняются в SQLite базе данных. Имеется две связанные сущности: канал (*channel*) и сообщение (*message*).

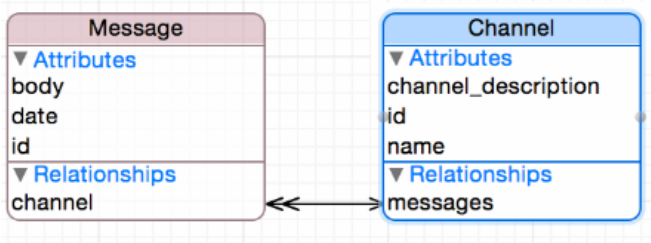

Рис 6. База данных клиента

VIII Интеграция клиентского приложения

Не всем мобильным приложениям нужен серверный компонент. Например, приложению, показывающему заряд батареи не нужна серверная компонента. Но многие приложения имеют серверную компоненту и, если логика приложения довольно простая, то разработчик может воспользоваться проприетарным сервером, например, Parse.com. Для сложных задач требуется создание собственной модели и API для получения требуемых услуг [12].

Mobile Cloud Computing - инфраструктура, где и хранение данных и обработка данных происходят не на мобильном устройстве. Мобильные приложения переносят часть своих задач на ресурсы облака [13].

Одна из основных проблем с мобильными устройствами связана с их батареей. Чем больше вычислений производиться, тем быстрее расходуется заряд. Можно использовать более дорогие процессоры, которые потребляют меньше энергии, но это не оправдано. Поэтому и предложена техника разгрузки устройства, которая переносит задачи на внешние ресурсы, тем самым уменьшая время выполнения локальных задач и уменьшая потребление заряда.

Хранение данных и выполнение приложений в облаке повышает надежность, так как в облаке для данных происходит репликация на разные устройства хранения, а в случае вычислительных ресурсов при неисправности может произойти быстрая замена на другой работающий вычислительный узел. Для сокращения времени доступа может также использоваться Content Delivery Network (CDN).

Несколько слов о возможных форматах передачи данных. Сравним несколько основных способов:

A. JSON
- может быть прочитан/изменен людьми
- может быть разобран без знания схемы
- отличная поддержка формата со стороны браузеров
- менее многословный в сравнении с XML

B. XML
- может быть прочитан/изменен людьми
- может быть разобран без знания схемы
- стандарт для SOAP
- хорошая инструментальная поддержка (xsd, xslt, sax, dom)
- довольно многословный

C. ProtoBuf
- очень уплотненные данные (маленький размер обработанных данных)
- трудно декодировать/разбирать не зная схемы (формат данных не однозначный и требует схемы для разъяснений)
- очень быстрая обработка
- не предназначен для прямых модификаций людьми (плотный двоичный формат)

Так как применительно к задаче сервиса рассылки уведомлений один пользователь в среднем не будет подписываться на огромное количество каналов. И уведомлений приходящих к нему будет относительно немного, то в данном случае скорость обработки не так критична. Для этой задачи лучше производить передачу данных в формате JSON. С другой стороны, если у самого сервиса большое количество клиентов, то, чтобы снизить нагрузку на него, можно использовать протокол ProtoBuf.

Если сервер имеет открытый API, для общения с сервером лучше использовать JSON протокол. Он быстрее, чем XML, и читаемый человеком. Если API приватное, ProtoBuf - лучшее решение. ProtoBuf обрабатывается быстрее и тратит в 5-10 раз меньше памяти, чем JSON.

IX Примечание редактора

Это уже не первая работа, выполненная в магистратуре в Лаборатории Открытых Информационных технологий, которая посвящена push-уведомлениям. Здесь можно сослаться, например, на работы [11, 14, 15]. Цель - создание открытой Content Management System, которую можно было бы использовать для организации рассылок, с использованием push-уведомлений. Каждая из перечисленных работ имеет свои достоинства, в данной статье излагаются результаты магистерской диссертации, которая наиболее ощутимо продвинулась в плане практической реализации. Хотя полностью задача так и не решена. Результаты данной работы доступны на Github [16].

Библиография

[1] Church, K., & de Oliveira, R. (2013, August). What's up with whatsapp?: comparing mobile instant messaging behaviors with traditional SMS. In Proceedings of the 15th international conference on Human-computer interaction with mobile devices and services (pp. 352-361). ACM.

[2] Brown, Jeff, Bill Shipman, and Ron Vetter. "SMS: The short message service." Computer 40.12 (2007): 106-110.

[3] Намиот Д. Е. Twitter как транспорт в информационных системах //International Journal of Open Information Technologies. – 2014. – Т. 2. – №. 1. – С. 42-46.

[4] Namiot, D., & Sneps-Sneppe, M. (2013, May). Local messages for smartphones. In Future Internet Communications (CFIC), 2013 Conference on (pp. 1-6). IEEE.
[5] Sneps-Sneppe, M., & Namiot, D. (2013). Spotique: A New Approach to Local Messaging. In Wired/Wireless Internet Communication (pp. 192-203). Springer Berlin Heidelberg.
[6] Cesare Pautasso, Olaf Zimmermann, Frank Leymann. RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision, 2010.
[7] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. Web Services: Concepts, Architectures, Applications. Springer, 2004.
[8] Crane, Dave; McCarthy, Phil (October 13, 2008). Comet and Reverse Ajax: The Next-Generation Ajax 2.0. (pp. 72-75)
[9] Apple Push Notification Service – https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ApplePushService.html
[10] Google Cloud Messaging for Android – http://developer.android.com/google/gcm/gs.html.
[11] Павлов А. Д., Намиот Д. Е. Системы для поддержки push-уведомлений //International Journal of Open Information Technologies. – 2014. – Т. 2. – №. 7. – С. 37-44.
[12] L. Bass, P. Clements, and R. Kazman,Software Architecture in Practice. Addison Wesley, 2003.
[13] A. Smailagicand M. Ettus, "System Designand Power Optimization for Mobile Computers, " in Proceedings of IEEE Computer Society Asia-Pacific on Web Conference (APWEB), June 2010
[14] Павлов В., Намиот Д. Анализ и Разработка Системы Push-уведомлений с Использованием Технологий Google //International Journal of Open Information Technologies. – 2013. – Т. 1. – №. 3. – С. 20-24.
[15] Павлов А. Д., Намиот Д. Е. Информационные системы на основе push-уведомлений //International Journal of Open Information Technologies. – 2014. – Т. 2. – №. 8. – С. 11-19.
[16] WhisperLab https://github.com/whisper-lab Retrieved: May, 2015

On service for push-notifications management

Alexey Pustobaev

***Abstract*—This paper is devoted to a mobile notification service. We consider the general architecture of the service, the approaches to the development, as well as the technology used to notify the client. The final goal for the project described in this paper is content management system for push-notifications management. We describe the individual components of the proposed system, as well as the way they relate to each other.**

***Keywords*— push, notifications, mobile client, CMS.**