Московский Государственный Университет

Факультет вычислительной математики и кибернетики

Курсовая работа по теме

Сервис push-уведомлений

Студент магистратуры

Пустобаев А.И.

Научный руководитель

Намиот Д.Е.

Москва, 2014

Содержание

Введение

От традиционного обмена сообщениями между мобильными устройствами с помощью SMS технологии постепенно отказываются. На смену 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 [1] – WhatsApp. В мобильных приложениях широко используется механизм push-уведомлений. Например в работах, выполненных в лаборатории ОИТ [4][5]. Механизм 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.

Push уведомления хороши на мобильных устройствах тем, что они экономят заряд батареи. Это достигается а 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.

Так же 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.

Модели информационных систем

Push-уведомления классифицируют по двум типам native и rich. Native – чисто текстовые сообщения ограниченного размера. Они предоставляются на уровне операционной системы. Rich представляют собой Native сообщения, расширенные дополнительной информацией. Например: HTML страница, картинки, музыка. [6]

Регистрация подписчиков для native и для rich push-уведомлений

1. Подписчик (Subscriber) регистрируется на сервере собственной службы уведомлений (PSP).
2. В ответ подписчик получает регистрационный номер (для GCM – registrationID, для APNS – deviceToken).
3. Подписчик передает регистрационный номер серверу публикаций (Publication server). Также он может указать тему (topic) подписки или набор тегов (tags).
4. Сервер публикаций сохраняет регистрационный номер подписчика в базе данных.

A. Модель посылки native push-уведомлений

1. Публикатор размещает уведомление (оно может относиться или к какой-то теме, или к сегменту пользователей, или быть индивидуальным) либо с использованием API (удаленно), либо с использованием web – интерфейса (на стороне сервера публикаций).
2. Сервер публикаций выполняет задачу рассылки уведомлений соответствующим PSP с использованием регистрационных номеров подписчиков.
3. PSP отслеживает состояние подписчика (в сети или нет) и осуществляет дальнейшую доставку подписчику (мобильному устройству) с соответствующим регистрационным номером.

B. Модель посылки rich push-уведомлений

Rich-уведомления позволяют рассылать не только текст, но и мультимедийный контент [7]. Чаще всего контент хранится на сервере и предоставляется в виде некоторой HTML страницы по средствам REST API.

1. Публикатор создает rich push-уведомление, публикует его подписчикам какой-либо темы, сегменту (группе) пользователей, либо посылает индивидуально.
2. Сервер публикаций извлекает из базы данных необходимые регистрационные номера подписчиков.
3. Сервер публикаций выполняет рассылку уведомлений соответствующим PSP с использованием регистрационных номеров с учетом тем/тегов. В теле сообщения передается rich push ID – сформированный сервером публикаций.
4. PSP отслеживает состояние подписчиков (в сети или нет) и осуществляет дальнейшую доставку.
5. Подписчик передает rich push ID серверу публикаций.
6. Сервер публикаций возвращает HTML страничку.

Виды рассылок push-уведомлений

Push-уведомления так же разделяют по количеству адресатов, получающих одно и тоже уведомление, и по способу сегментации адресатов. То есть по способу выделения групп, сегментов пользователей, которые будут получать одинаковые уведомления.

A. индивидуальная рассылка (Unicast)

1. Публикатор размещает уведомление для подписчика с уникальным регистрационным номером.
2. Сервер публикаций осуществляет отправку уведомления с указанным regID соответствующему PSP.
3. PSP отслеживает состояние подписчиков (в сети или нет) и осуществляет дальнейшую доставку.

B. широковещательная рассылка (Broadcast)

1. Публикатор размещает уведомление в определенное приложение (application).
2. Сервер публикаций извлекает регистрационные номера подписчиков.
3. Сервер публикаций осуществляет рассылку уведомления с указанными regID соответствующим PSP.
4. PSP отслеживает состояние подписчиков (находится подписчик в сети или нет) и осуществляет дальнейшую доставку.

C. сегментная (таргетированная) рассылка (Multicast)

1. Публикатор размещает уведомление в определенный сегмент (набор тегов с логическими операциями, возможно указание диапазонов для значений тегов).
2. Сервер публикаций извлекает регистрационные номера подписчиков, удовлетворяющие определенным требованиям.
3. Сервер публикаций осуществляет рассылку уведомления с указанными regID соответствующим PSP.
4. PSP отслеживает состояние подписчиков (в сети или нет) и осуществляет дальнейшую доставку.

D. гео-рассылка push-уведомлений

1. Публикатор указывает серверу публикаций координаты точек с радиусами действия, координаты зон.
2. Подписчик после регистрации с определенной периодичностью посылает серверу публикаций свои координаты.
3. Сервер публикаций обновляет координаты подписчиков в базе данных.
4. Сервер публикаций извлекает регистрационные номера подписчиков, находящихся в определенном регионе и осуществляет рассылку уведомлений с указанными regID соответствующим PSP.
5. PSP отслеживает состояния подписчиков и осуществляют дальнейшую доставку.

E. рассылка на основании информации от тегов

1. Публикатор регистрирует на сервере публикаций Bluetooth–метки.
2. Подписчик обнаруживает Bluetooth–метку и отсылает серверу публикаций свой регистрационный номер и идентификатор метки (BeaconID).
3. Сервер публикаций определяет содержимое уведомления по присланному идентификатору метки. Сервер публикаций осуществляет посылку уведомления с указанным regID соответствующему PSP.
4. PSP отслеживает состояния подписчика и осуществляют дальнейшую доставку.

Разработка приложения с push-уведомлениями под OS Android

Рассмотрим, какие программные и физические средства требуются для разработки и тестирования приложения с push-уведомлениями.

Для операционной системы Android средой, в которой эта ОС установлена, может являться реальное, физическое мобильное устройство, например, планшет или телефон. Или виртуализуемое устройство. Рассмотрим подробнее второй случай. Виртуализа́ция - это предоставление набора вычислительных ресурсов или их логического объединения, абстрагированное от аппаратной реализации, и обеспечивающее при этом логическую изоляцию вычислительных процессов, выполняемых на одном физическом ресурсе.

Используя виртуализацию не важно на каком физическом устройстве в конечном итоге выполняется операционная система Android. Это может быть, как настольный персональный компьютер, так и любое другое устройство осуществляющее виртуализацию. В теории возможна вложенная виртуализация [8], когда моделируется одна виртуальная машина внутри другой виртуальной машины. В такой концепции может быть множество уровней вложенности. Для наших целей достаточного одного такого уровня.

Чтобы иметь возможность использовать push-уведомления приложение должно быть установлено на операционную систему Android с версией не ниже 2.2. Так же эта ОС должна иметь предустановленный Google Play Store.

Сравним два популярных способа запуска OS Android на персональном компьютере.

Использование Android Emulator в качестве среды для OS Android

Первый способ – использовать стандартный Android Emulator, поставляемый с SDK. Он эмулирует ARM-based архитектуру и из-за этого работает медленно. Также для этого эмулятора есть специальный образ от intel (x86 Emulator Accelerator Manager (HAXM)). Образ предназначен для процессоров серии x86. Это значительно увеличивает быстродействие, так как процессорные команды x86 совместимы практически со всеми персональными компьютерами и не требуется дополнительных преобразований, необходимых в случае ARM-based архитектуры.
Достоинства:

- бесплатный, поставляется с SDK

Недостатки:

- нет предустановленных Google Play Services, которые требуются для работы с уведомлениями.

Использование GENYMOTION в качестве среды для OS Android

Второй способ – воспользоваться сторонней программой для виртуализации. Например GENYMOTION. Она использует для своей работы VirtualBox и образ уже сделанный под процессор x86. Скорость работы мобильной операционной системы на некоторых тестах может быть даже выше, чем на самих мобильных устройствах. Она зависит от мощности компьютера и устройства. Эти возможности у GENYMOTION можно получить бесплатно, зарегистрировавшись.
Достоинства:

- есть поддержка OpenGL на уровне железа.
- есть возможность установить Google Play Services и сам магазин приложений.
- поддерживаются многие сенсоры: кнопки, видеокамеры, GPS.
- базовые возможности можно получить бесплатно, зарегистрировавшись.

Недостатки:

- не все возможности предоставляемые GENYMOTION бесплатны.

Требования к системе push-уведомлений

Для организации полноценной работы система должна включать в себя следующий минимальный набор компонент [9].

1) мобильное устройство или виртуальную машину в рассмотренными характеристиками;
2) сторонний (back-end) сервер приложений для отправки данных через службу GCM;
3) службу GCM – служба доставки сообщений до приложения клиента.

Служба уведомлений от Google - Google Cloud Messaging (GCM)

С помощью Google Cloud Messaging (GCM) можно отправлять сообщения (уведомления) на устройства с ОС Android. Сообщение может быть отправлено на конкретное устройство или же на множество устройств используя User Notifications.

Связь с сервером GCM со стороны отправителя может быть организована по протоколу HTTP или по двум протоколам HTTP и JSON. Во втором случае JSON сообщение помещается в тело HTTP сообщения.

Другой способ связи с сервером GCM - это через Cloud Connection Server (CCS) по протоколу XMPP. В этом случае помимо отправки сообщения от отправителя к получателю возможна обратная отправка сообщения от получателя к отправителю. Содержание сообщения может быть произвольным.

Отправителем в данной модели как правило является back-end сервер. Получателем – мобильное устройство.

Для работы со службой GCM используются следующие параметры доступа [10].

1) идентификатор отправителя (Sender ID)
2) идентификатор приложения (Application ID)
3) регистрационный идентификатор (Registration ID)
4) аккаунт пользователя в системе Google (Google User Account)
5) маркер авторизованного отправителя (Sender Auth Token)

Google предоставляет некоторые готовые примеры работы с GCM.

HTTP (or HTTP+JSON):

- Java пример “gcm-demo-server”, используя Google gcm-server java библиотеку. Выполняется на веб сервере поддерживающем Servlets API версии 2.5. Им может быть Glassfish 4.
- Java пример “gcm-demo-appengine”, используя Google gcm-server java библиотеку. Выполняется на Google App Engine for Java. Данный приложение является Cloud-Based.

Пример отправки сообщения на устройство с регистрационным номером YourAppRegID от приложения разработчика с авторизационным ключем YourAuthKey по HTTP:

```
$ curl -X POST https://android.googleapis.com/gcm/send \
-H "Content-Type:application/json" \
-H "Authorization:key=YourAuthKey" \
-d '{"registration_ids" : ["YourAppRegID"], "data" : { "Hello" : "World" } }'
```

CCS (XMPP):

- Java пример, используя Smack library и json-simple library.
- Python пример, используя xmpp module (Ubuntu package “python-xmpp”).

Системы для поддержки push-уведомлений

Упоминалось ранее, что службы уведомлений (PSP) у производителей разных мобильных операционных систем разные. Каждая служба имеет свой API, свои протоколы взаимодействия. Перед программистом встает задача по изучению способов работы с этими системами. С каждой по отдельности.

Существуют сторонние библиотеки, программы или сервисы позволяющие абстрагироваться от API различных серверов уведомлений. Например от PSP серверов: GCM – Google, APNS – Apple, MPNS - Microsoft. Часть этих программ, библиотек или сервисов рассмотрена в [7]. Приведем некоторый обзор таких средств.

Uniqush (uniqush-push) : (Apache License 2.0) Программа. Поддерживает GCM for Android, APNS for iOS, ADM for Kindle tablets. Написана на языке Google Go. Для хранения пользователей и групп пользователей используется redis db. Взаимодействие с программой происходит посредством HTTP запросов.

Пример регистрации и отправки сообщения:

```
$ curl http://127.0.0.1:9898/addpsp \
-d service=myservice -d pushservicetype=gcm \
-d projectid=123 -d apikey=YourAuthKey

$ curl http://127.0.0.1:9898/subscribe \
-d service=myservice -d subscriber=uniqush.client \
-d pushservicetype=gcm -d regid=YourAppRegID

$ curl http://127.0.0.1:9898/push\
-d service=myservice -d subscriber=uniqush.client \
-d msg="Hello World"
```

В случае возникновения проблемы отправки нотификации после нескольких экспериментов, возможно, следует вручную очистить базу данных redis. Простой способ это сделать: “$ redis-cli flushdb”.

Zend_Mobile_Push : (New BSD License) Часть Zend Framework. Все компоненты написаны на полностью объектно-ориентированном коде PHP 5. Включает возможность работы с нотификационными серверами: APNS (iTouch/iPad/iPhone), GCM (Google Android) и MPNS (Windows Phone).

Windows Azure Notification Hub : сервис (Cloud-Based). Поддерживает платформы Windows/Windows Phone, iOS, Android. Является посредником между back-end серверами и PNS. 500 активных устройств на namespace и 100000 уведомлений в месяц бесплатно. Взаимодействие по средствам REST API.

Amazon Simple Notification Service (Amazon SNS) : сервис (Cloud-Based). Поддерживает iPhone, iPad, Android, Kindle Fire. Можно посылать до 1 миллиона уведомлений каждый месяц бесплатно. Взаимодействие через HTTP (API или браузер).

PubNub : сервис (Cloud-Based). Поддерживает iOS, Android, Blackberry, Windows 8, Symbian, Titanium, Corona, PhoneGap и другие. Более 50 SDKs для mobile, browser, desktop and server. API для Java, Ruby, Python, JavaScript и другие. Пропускная способность 3 миллиона сообщений в секунду. 20 активных устройств в день и 1 миллион уведомлений бесплатно.

App42 : сервис (Cloud-Based). Поддерживает iOS, Android, Windows Phone. Используется единый API. Рассылать сообщения можно с Windows Phone, Android, iOS, используя языки Java, .NET, Ruby, PhP. 1 миллион API вызовов и 1 миллион уведомлений в месяц бесплатно.

Перечисленные сервисы не предоставляют возможности получения приложения и серверной части для рассылки информации по средствам push-уведомлений без участия программиста. В данной работе рассматривается вопрос создания сервиса информационных систем, который позволит пользоваться технологией push без программирования.

Создание сервиса мобильных push уведомлений.

Требования к сервису

Сервис должен позволять не программисту получить мобильное приложение и доступ к ресурсу в интернете, с которого осуществляется рассылка уведомлений на данное приложение по средствам технологии Push.

Пользователи должны иметь возможность организовать рассылку уведомлений непосредственно после установки приложения и получения доступа к веб ресурсу.

Процесс получения доступа к ресурсу представляет из себя:

- Для незарегистрированных пользователей - регистрация и некоторые настройки.
- Для зарегистрированных пользователей - процесс аутентификации на веб странице ресурса.

Рассмотрим систему на примере организации и ее клиентов.

Организация регистрируется на ресурсе. Получает ссылку на приложение для мобильных устройств. Этой ссылкой организация делится со своими клиентами. Клиенты устанавливают себе это приложение.

Организация так же получает доступ к веб странице, с которой происходит рассылка уведомлений на мобильные устройства клиентов организации.

Адресатами подобного рода сервиса могут являться, например, учебные заведения, которые смогут организовать современное взаимодействие с учащимися. Учебная часть сможет рассылать уведомления об изменениях в расписании или же новостях касающихся конкретной группы студентов. Важно, что эти уведомления могут быть сегментированы, персонализированы. То есть информация касающаяся конкретных студентов доставляется только им. Другой пример адресатов – это предприятия торговли (retail). Они могут использовать такой сервис для уведомления покупателей. Количество заинтересованных в таких системах (в данном случае речь идет о системах рассылки уведомлений как таковых) больше количества разработчиков занимающихся этими технологиями, чтобы создавать, настраивать и поддерживать их всем заинтересованным. Поэтому можно сделать предположение о востребованности и явной коммерческой направленности данного сервиса.

Модель сервиса мобильных push уведомлений.

Создатель рассылки регистрируется на веб-сайте, указывая название канала рассылки и текстовую информацию о канале (описание). В качестве текстовой информации может быть, например, тематика уведомлений, предполагаемая аудитория (те для кого ведется рассылка), и кто осуществляет рассылку. Все пользователи мобильного приложения видят текущий список названий (рассылок), могут посмотреть описание (коментарий) и подписаться или отписаться.

После входа на веб-сайт под зарегистрированным аккаунтом, создания и отправки сообщения, оно пушится (отправляется по средствам технологии push-уведомлений) тем пользователям мобильного приложения, которые подписаны на канал рассылки. В данной простейшей модели одному аккаунту отправителя соответствует один канал рассылки.

Возможно расширение модели, когда у одного веб-аккаунта может быть несколько тем (каналов). При вводе сообщения публикатор указывает к какой из его тем оно относится. Другая дополнительная возможность - это просмотр статистики. Сколько всего скачали приложение (в данном случае имеется

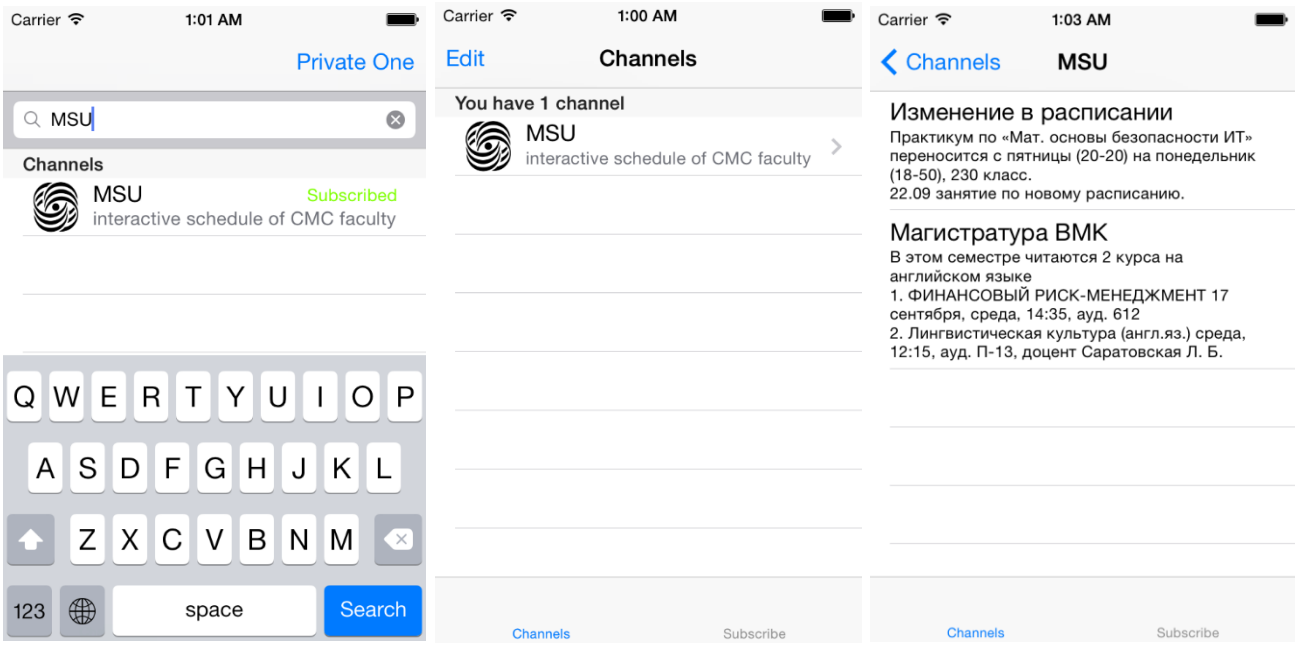
в виду количество приложений с разрешенным push). Какое количество из этих пользователей подписалось на темы веб-аккаунта и сколько на каждую тему. Может предоставляться информация об изменении этих показателей за последнее время.

Система должна иметь администратора, который может блокировать и разблокировать веб-аккаунты, смотреть статистику по всем темам всех пользователей.

Мобильное приложение

Мобильное приложение, являющееся частью сервиса, должно удовлетворять следующим минимальным требованиям:

1. Приложение должно иметь возможность получать уведомления от разных организаций, на которые подписан клиент.
2. Должна быть возможность подписаться и отписаться от организации.
3. После установки приложения пользователь может найти интересующую его организацию (публикатора) через поиск по организациям и подписаться на нее.
4. В контексте одной организации полученные уведомления отображаются списком.



a. Раздел поиска и подписки. b. Раздел работы с каналами. c. Уведомления.

Рисунок 1. Мобильное приложение.

Приложение (рис. 1) имеет два основных раздела.

Раздел Subscribe (рис. 1.a). В нем осуществляется поиск каналов по именам. Найденные названия сопровождаются краткой информацией о канале, картинкой и, если пользователь уже подписан на этот канал, пометкой (“Subscribed”). После выбора одного из найденных каналов, показывается экран с

расширенной информацией о рассылке и возможностью подписаться или отписаться, в зависимости от текущего состояния.

Раздел Channels (рис. 1.b). Данный раздел отображает все активные каналы пользователя, то есть каналы, на которые пользователь подписан. Перейдя в режим редактирования (кнопка “Edit” вверху экрана), пользователь может расположить каналы в удобном для него порядке, удалить не нужные (отписаться). В обычном режиме при выборе канала происходит переход на экран с его сообщениями (уведомлениями) (рис. 1.c). Уведомления упорядочены по времени отправления их публикатором. Новые сообщения и сообщения, которые не были ранее просмотрены дополнительно помечаются.

Возможные расширения функциональности.

1. Разделение по темам. Возможность подписаться на различные тематические рассылки внутри одной организации.
2. Открытые/скрытые организации. Пользователь не видит всех организаций через приложение. Часть организаций скрыта. В данной модели пользователь может подписаться только на те организации, которые для него открыты, или на те, на которые у него есть ключ-идентификатор. В приложении ввод ключа осуществляется в разделе Subscribe (рис. 1.a) после нажатия на кнопку “Private One”. Ключ-идентификатор подписчик получает у публикатора отдельно, за пределами приложения. Например, лично.
   Ключ-идентификатор может быть:
   - Разовым - после одной успешной подписки он становится не рабочим. Это можно использовать, например, если требуется производить индивидуальную рассылку.
   - Многоразовым - разным клиентам организации можно подписываться по одному и тому же ключу. Может использоваться для широковещательной и сегментной рассылки, когда этот ключ соответствует не организации в целом, а некоторой теме рассылки внутри организации. Недостаток: такой ключ кто-нибудь из подписчиков может выложить в открытый доступ. Тогда теряется смысл скрытой организации.

Ключ-идентификатор в простейшем случае состоит из уникального идентификатора организации. Но так же может быть расширен идентификатором темы внутри организации и токеном. Одноразовые ключи-идентификаторы получаются за счет разных значений этого токена, сгенерированного публикатором.

В рамках работы разработано iOS приложение (рис. 1.1) получающее локальные уведомления по расписанию. Присутствует возможность просмотра полученных сообщений, отправки текстовых уведомлений по расписанию.

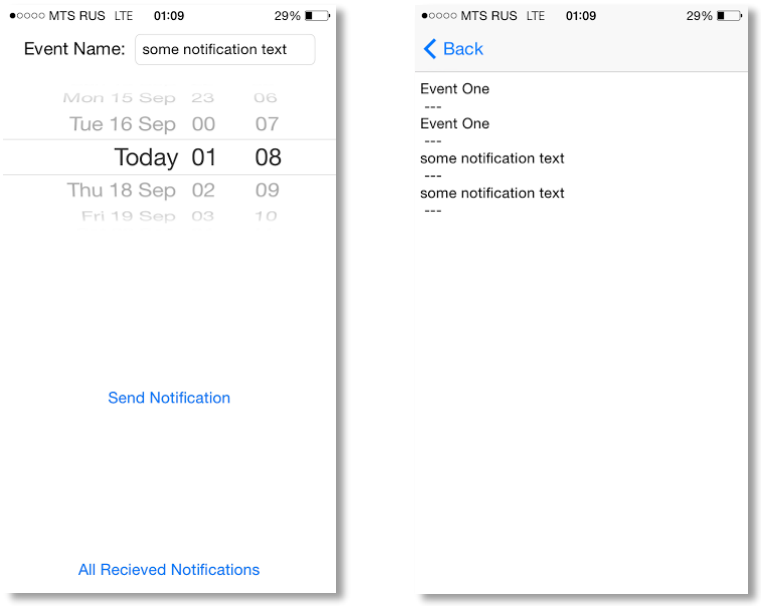


Рисунок 1.1. Мобильное приложение. Локальные уведомления.

Front end

В качестве браузерной клиентской части сервиса выбран HTML, CSS, JS фреймворк Bootstrap [11].

Вход в систему для зарегистрированных пользователей производится с помощью логина (адрес электронной почты) и пароля (рис. 2). Для новых пользователей система имеет страницу регистрации с подтверждением через электронную почту.

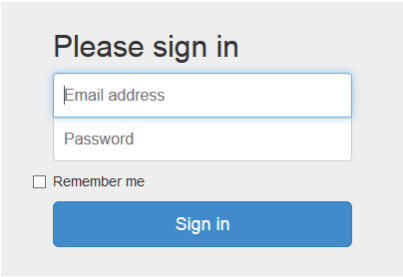


Рисунок 2. Страница входа в систему

После того как пользователь выполнил вход в систему, он попадает на страницу отправки уведомления всем подписчикам (рис. 3). Вверху страницы размещена панель управления. С помощью нее происходит навигация по сайту.

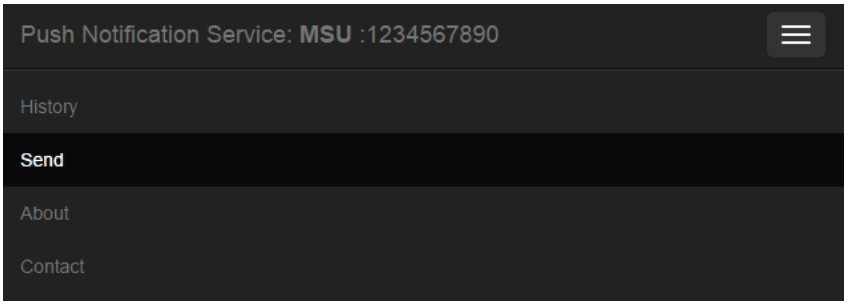


Рисунок 3. Панель управления

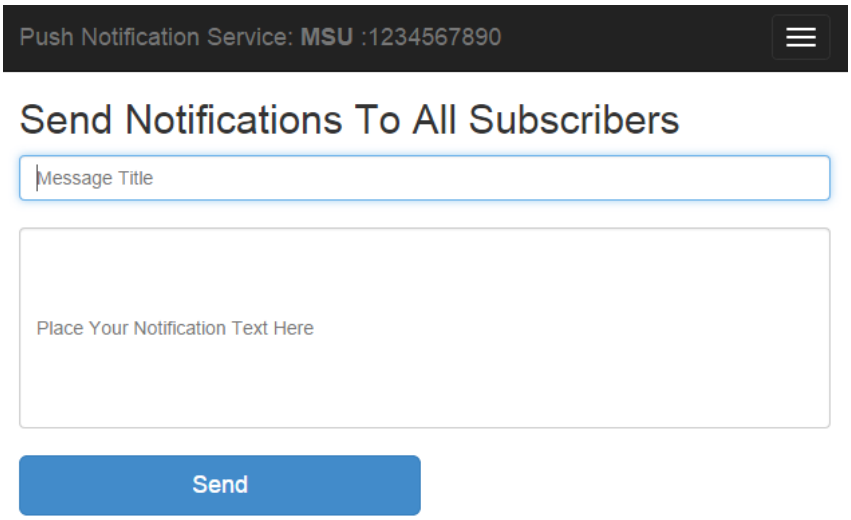


Рисунок 4. Страница создания и отправки уведомления

Так же эта панель содержит информацию об организации: название и уникальный идентификационный номер. Номер генерируется системой при регистрации нового пользователя. По номеру организации подписчики могут найти публикатора (организацию) в своем мобильном приложении и подписаться на него. Поиск осуществляется так же и по названию организации, но название может быть не уникальным.

Страница создания и рассылки уведомлений (рис. 4) в простейшем случае содержит поле для ввода заголовка сообщения, поле для ввода текста уведомления и кнопку разослать. Рассылка сообщения производится всем клиентам одновременно (широковещательная).

После отправки уведомления публикатор на странице с историей сообщений имеет возможность посмотреть статус уведомления (“отправлено”, “отправляется” или “не отправлено”).

Возможное расширение данной модели:

- рассылка уведомлений некоторой группе клиентов.
- рассылка по расписанию.
- рассылка rich push-уведомлений

Back end

В качестве языка и фреймворка серверной части выбраны Ruby и Ruby on Rails [12].

Ruby on Rails – веб фреймфорк, является открытым программным обеспечением и распространяется под лицензией MIT. Использует Model-View-Controller модель и REST-стиль построения веб-приложений.

Rails использует множество соглашений по конфигурации. Что позволяет быстро разрабатывать большинство приложений. Специальная конфигурация требуется редко.

Так же быстрой разработке способствует минимизация дублирования кода (Механизмы повторного использования).

В серверной части сервиса push-уведомлений можно выделить такие сущности: Пользователь (User), Устройство (Device), Приложение (Permitted App).

Приведем код этих сущностей:

app/models/user.rb

```ruby
class User < ActiveRecord::Base
  has_one :device, dependent: :destroy

  validates :email, :sex, presence: true

  accepts_nested_attributes_for :device
end
```

Рисунок 5. Код сущности Пользователь (User)

app/models/device.rb

```ruby
class Device < ActiveRecord::Base
  belongs_to :user

  validates :token, presence: true, uniqueness: { scope: :user_id }
  validates :platform, presence: true, inclusion: { in: %w(ios android), case_sensitive: false }

  # Allows us to perform scoped queries a la Device.platform_is(:iOS).load
  scope :platform_is, -> (device_platform) { where(platform: device_platform.to_s.downcase) }

  before_validation :format_attributes_for_persist!

  def platform_is?(device_platform)
    !!(device_platform.to_s =~ /#{platform}/i)
  end

  def format_attributes_for_persist!
    platform.downcase! if attribute_present?("platform")
  end
end
```

app/models/permitted_app.rb

```ruby
class PermittedApp < ActiveRecord::Base
  # Permitted applications get tokens
  # They'll never be super secret, since this is stored on the client (mobile app).

  before_save :ensure_authentication_token!

  def ensure_authentication_token!
    if authentication_token.blank?
      self.authentication_token = generate_secure_token_string
    end
  end

  def generate_secure_token_string
    SecureRandom.urlsafe_base64(25).tr('lIO0', 'sxyz')
  end
end
```

Рисунок 6. Код сущностей Устройство (Device), Приложение (Permitted App).

И их миграций.

User and Device Migrations

```ruby
class CreateDevices < ActiveRecord::Migration
  def change
    create_table :devices do |t|
      t.text :platform
      t.text :token

      t.integer :user_id

      t.timestamps
    end
  end
end
```

```ruby
class CreateDevices < ActiveRecord::Migration
  def change
    create_table :devices do |t|
      t.text :platform
      t.text :token

      t.integer :user_id

      t.timestamps
    end
  end
end
```

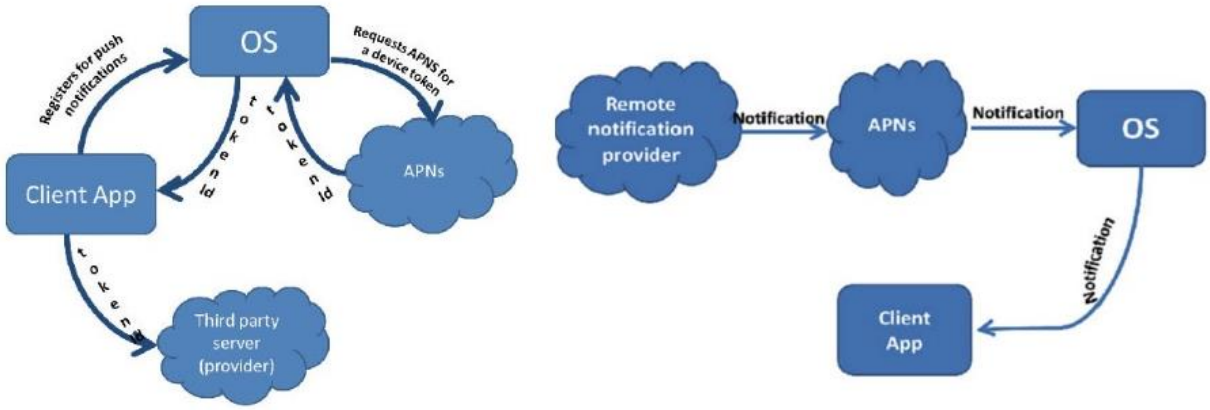
Permitted App Migration

```ruby
class CreatePermittedApps < ActiveRecord::Migration
  def change
    create_table :permitted_apps do |t|
      t.text :authentication_token

      t.timestamps
    end
  end
end
```

Рисунок 7. Код миграций сущностей Пользователь (User), Устройство (Device), Приложение (Permitted App).

Как работают push уведомления в Apple.



Первая Фаза Вторая Фаза

Рисунок 8. Схема работы push-уведомлений через APNS

Первая Фаза:

Приложение запрашивает у операционной системы токен. Операционная система запрашивает его у APNS (Apple Push Notification Service) [13]. Клиентское приложение получает токен, который отправляет на back-end сервер, и с помощью которого сервер будет рассылать клиентскому приложению уведомления во второй фазе.

Вторая Фаза:

Сервер отправляет уведомления с указанием токенов на APNS. APNS рассылает уведомления устройствам и клиентское приложение получает уведомление от операционной системы.

Background processing в Ruby.

Для увеличения производительности при работе с асинхронными задачами в частности для отправки уведомлений на APNS или GCM выбран фреймворк Sidekiq [14].

Sidekiq использует базу данных Redis и в ней хранит свои промежуточные данные.

Redis – NoSQL база данных. Данные в ней хранится в виде словаря, в котором ключи связаны со своими значениями. Одно из ключевых отличий Redis от других хранилищ данных заключается в том, что значения этих ключей не ограничиваются строками. Поддерживаются следующие абстрактные типы данных:

- Списки строк
- Множества строк (коллекции неповторяющихся несортированных элементов)
- Сортированные множества строк (коллекции неповторяющихся элементов, упорядоченных по некоторому вещественному числу)
- Словари, в которых и ключи и их значения – строки.

Высокая производительность Redis обуславливается тем, что все данные хранятся в оперативной памяти.

Реализация отправки уведомлений в PNS.

Для отправки уведомлений в APNS выбран gem Grocer [15].

iOS worker for Sidekiq

```ruby
class IosAlertWorker
  include Sidekiq::Worker
  include Sidekiq::Schedulable

  # Sidetiq should schedule this job every half hour between 08:30 - 16:30 every Monday-Friday.
  # Note: The job will be queued at this time, not executed.
  recurrence do
    weekly(1).
      day(:monday, :tuesday, :wednesday, :thursday, :friday).
      hour_of_day((8..16).to_a).
      minute_of_hour(0, 30)
  end

  # Set the Sidekiq Queue to 'ios_alert_worker_queue', enable backtrace logging, and retry if the job fails
  sidekiq_options :queue => :ios_alert_worker_queue, backtrace: true, retry: true

  def perform
    # Keep the connection to gateway.push.apple.com open throughout the duration of this job
    pusher = Grocer.pusher(
      certificate: "/path/to/cert.pem",      # required
      passphrase:  "",                       # optional
      gateway:     "gateway.push.apple.com", # optional
      port:        2195,                     # optional
      retries:     3                         # optional
    )

    User.joins(:device).where('devices.platform = ?', 'ios').pluck('devices.token').each do |token|
      notification = Grocer::Notification.new(
        device_token:      token
        alert:             "Hello from Grocer!",
        badge:             42,
        sound:             "siren.aiff",         # optional
        expiry:            Time.now + 60*60,     # optional; 0 is default, meaning the message is not stored
        identifier:        1234,                 # optional
        content_available: true                  # optional; any truthy value will set 'content-available' to 1
      )
      pusher.push(notification)
    end
  end
end
```

Рисунок 9. Код выполняющий рассылку уведомлений с помощью гема Grocer.

Для отправки уведомлений в GCM выбран gem GCM [16].

Android worker for Sidekiq

```ruby
require 'gcm'

class AndroidAlertWorker
  include Sidekiq::Worker
  include Sidekiq::Schedulable

  # Sidetiq should schedule this job every day, Monday-Friday, at 08:30
  # Note: The job will be queued at this time, not executed.
  recurrence do
    weekly(1).
      day(:monday, :tuesday, :wednesday, :thursday, :friday).
      hour_of_day((8..16).to_a).
      minute_of_hour(0, 30)
  end

  # Set the Sidekiq Queue to 'ios_alert_worker_queue', enable backtrace logging, and retry if the job fails
  sidekiq_options :queue => :android_alert_worker_queue, backtrace: true, retry: true

  def perform
    registration_ids = User.joins(:device).where('devices.platform = ?', 'android').
                        pluck('devices.token')
      # Android's GCM implementation will also accept an array of registration_ids, reducing the number of
      # requests we need to send (as long as our notification is common among all the devices)
      gcm = GCM.new(ENV['gcm_key'])
      options = {
        'data' => {
          'message' => "Hello from GCM!"
        },
          'collapse_key' => 'this_messages_key'
      }
      response = gcm.send_notification(registration_ids, options)
    end
  end
end
```

Рисунок 10. Код выполняющий рассылку уведомлений с помощью гема GCM.

Размещение backend приложения.

В качестве облачной платформы для размещения Rails приложения выбран heroku [17].

Некоторые возможности heroku:

- языки: Ruby, Java, Node.js, Python, PHP,
- масштабирование:
    - по характеристикам железа. На heroku есть возможность использовать разные размеры виртуальных машин, то есть виртуальные машины с разными вычислительными характеристиками. Под масштабированием понимается изменение этих характеристик по мере надобности.
    - по количеству вычислительных единиц. Под масштабированием понимается добавление и удаление вычислительных единиц по мере надобности.
- мониторинг, логирование.
- база данных: Postgres (предоставляется SQL database-as-a-service).

В базе данных бесплатно можно размещать до 10 тысяч записей. Одна виртуальная машина с характеристиками: 512MB RAM, 1x CPU Share предоставляется бесплатно.

heroku имеет большое количество расширений, дополнений (Addons), которые распространяются как бесплатно, так и за деньги. Например RedisToGo [18]. Это дополнение бесплатно предоставляет доступ к базе данных Redis.

К недостаткам heroku можно отнести то, что в нем используется файловая система с доступом только на чтение (read-only filesystem). Выделено два временных каталога, в которые возможна запись. В контексте heroku это означает, что если виртуальная машина будет остановлена или перезагружена, то все файлы, записанные в эти директории будут уничтожены. Когда случится следующая остановка или перезагрузка заранее не известно.

Заключение

В данной работе были приведены информационные модели регистрации подписчиков и посылки native push-уведомлений, посылки rich push-уведомлений, приведена классификация видов рассылок.

Были рассмотрены программные и физические средства требующиеся для разработки и тестирования приложения с push-уведомлениями под операционной системой Android. Так же были рассмотрены некоторые существующие системы для поддержки push-уведомлений.

В настоящей работе была предложена модель сервиса рассылки уведомлений с помощью технологии Push, в котором не требуется участие программиста. Были рассмотрены средства для его реализации, использующие технологии компаний Google Inc, Apple Inc. Архитектура серверной части на языке Ruby. Частично была выполнена реализация модели сервиса для посылки native push-уведомлений для мобильных устройств под iOS. В дальнейшем планируется создать полноценный сервис.

Список используемых источников

[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] 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.

[7] Павлов А. Д., Намиот Д. Е. Системы для поддержки push-уведомлений //International Journal of Open Information Technologies. – 2014. – Т. 2. – №. 7. – С. 37-44.

[8] Orit Wasserman, Red Hat (2013). "Nested virtualization: Shadow turtles". KVM forum. Retrieved 2014-04-07.

[9] 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.

[10] Google Cloud Messaging for Android http://developer.android.com/google/gcm/gs.html

[11] Bootstrap – http://getbootstrap.com

[12] Ruby on Rails – http://rubyonrails.org

[13] iOS API – https://developer.apple.com

[14] Sidekiq – http://sidekiq.org

[15] Grocer – https://github.com/grocer/grocer

[16] GCM gem – https://github.com/spacialdb/gcm

[17] heroku – https://www.heroku.com

[18] RedisToGo – https://devcenter.heroku.com/articles/redistogo