Санкт-Петербургский Государственный Университет

Программная инженерия
Кафедра системного программирования

Щербаков Александр Сергеевич

Разработка кроссплатформенной системы отправки и мониторинга доставки серверных уведомлений на мобильные устройства и браузеры

Дипломная работа

Научный руководитель:
ст. преп. Журавлев М. М.

Рецензент:
Ведущий Разработчик ООО "Невософт" Зубков И. В.

Санкт-Петербург
2017

SAINT-PETERSBURG STATE UNIVERSITY

Software engineering

Shcherbakov Aleksandr

Development of sending and delivery monitoring system of server-based push-notifications onto mobile devices and browsers

Graduation Thesis

Scientific supervisor:
sen. lector Maksim Zhuravlev

Reviewer:
Lead Developer "Nevosoft" Igor Zubkov

Saint-Petersburg
2017

Оглавление

Введение

После появлении технологии push-уведомлений на рынке, ей не сразу удалось создать высокий уровень доверия к себе со стороны разработчиков. Необдуманное использование этой технологии приводило к тому, что пользователи, получавшие большое количество не интересующих их рекламных сообщений, прекращали использование соответствующего приложения. Впервые технология появивилась на мобильных платформах в 2009 – 2010 году, после чего сильно видоизменилась, получила возможность модификации внешнего вида и распростанилась даже на браузеры.

Push-уведомления стали использоваться в приложениях для решения широкого круга задач. Так, например, новостные сайты могут оповещать пользователей о выходе новостей, приложения спортивных каналов – о начале спортивных трансляций, приложения авиакомпаний – о проведении скидочных акций. Получение пользователем с помощью push-уведомлений информации об изменениях в приложении или новых бонусах может способствовать удержанию пользователя в приложении. Также уведомления могут использоваться и для более серьезных задач, например, в спасательных службах и медицине.

Среди систем отправки и обработки push-уведомлений наибольшей популярностью пользуются веб-сервисы, однако существуют проблемы с их использованием, возникающие в результате необходимости переноса на них личных данных пользователей. С другой стороны, производители платформ не гарантируют доставку уведомлений. В то же время в готовых решениях, работающих на серверах разработчиков, нет модулей, которые могут быть встроены в клиентскую часть и способны осуществлять эту работу. Таким образом, в работе описывается создание технического решения, включающего в себя как клиентские, так и серверные модули. Результатом является разработанный прототип решения для работы с push-уведомлениями.

1. Постановка задачи

Целью данной дипломной работы является разработка прототипа системы отправки push-уведомлений с возможностями их персонализации. Для достижения поставленной цели были поставлены следующие задачи:

- Познакомиться с предметной областью, изучить принципы работы push-уведомлений, проанализировать продукты конкурентов.
- Разработать архитектуру сервиса push-уведомлений.
- Реализовать библиотеку для работы с сервисом для основных мобильных платформ.
- Реализовать серверную часть сервиса:
  - реализовать базовый API для отправки push-уведомлений;
  - реализовать способы выборки аудитории для отправки push-уведомлений;
  - реализовать возможность динамически-формируемых сообщений в push-уведомлениях.

2. Обзор предметной области

У разработчиков и маркетологов существует необходимость обращаться не ко всем пользователям с общим рекламным сообщением, а предоставлять уникальные предложения для каждого отдельного пользователя или для групп пользователей - персонализировать и сегментирвать уведомления. Так, например, пользователи приложений авиакомпаний могут получать скидочные предложения по перелетам в той местности, в которой они в данный момент находятся. Пользователи спортивного приложения заинтересованы в информации только по определенным спортивным мероприятиям или спортсменам.

Как отмечается в исследовании [7] компании Localytics, push-уведомления, сформированные на основе личных предпочтений пользователя, конвертируются в среднем в 3 раза лучше, чем несегментированные уведомления: 54% конверсии у первого типа уведомлений против 15% у второго типа. Под конверсией может подразумеваться какое угодно действие после перехода по уведомлению: от покупки товара в приложении или начала чтения предлагаемой статьи, до повышения уровня персонажа в мобильной игре после перехода по уведомлению.

В то же время отправка большого количества уведомлений с целью наверняка покрыть потребность пользователя в нужной информации, приводит или к отключению уведомлений, или к окончательному удалению приложения.

Конечным пользователям уведомления доставляются через PNS - провайдеры - сервисы разработчиков мобильных ОС, например Firebase Notification Service[4] для Android и APNS[3] для iOS. Однако, они не обладают достаточной функциональностью, позволяющей покрыть все потребности разработчиков.

Одним из их недостатков является необходимость разработчику кроссплатформенных приложений реализовывать под все платформы, работая с каждым PNS-провайдером отдельно, клиентскую часть для работы с push-уведомлениями:

- обновление токенов push-уведомлений,

- их регистрацию на сервере приложений,
- отображение уведомлений в нужное время после их доставки на мобильное устройство,
- ведение статистики,
- другие функции.

Кроме того, PNS-провайдеры не обладают достаточными возможностями с точки зрения тонкой настройки процесса отправки уведомлений, а именно:

- отправки уведомлений группам пользователей,
- генерации текста сообщений, исходя их личных данных пользователя: имени, даты, геопозиционирования, языка пользователю,
- планирования отправки,
- других факторов.

В последнее время появляется все больше сторонних сервисов. Сервисы являются промежуточным звеном между разработчиками и PNS-провайдерами. С одной стороны они предоставляют большую функциональность при отправке:

- имеют более гибкую функциональность в контексте планирования групп уведомлений для отправки,
- настройки сегментов пользователей,
- сбора статистики и данных о доставленных, просмотренных, сброшенных и т.д. уыедомлениях,
- A/B тестирования для push-уведомлений.

С другой стороны они избавлют разработчика от необходимости разработки клиентской части под все платформы. Разработчик встраивает их SDK в свое приложение и перекладывает ответственность по

обработке уведомлений на него. Примерами таких сервисов являются Kumulos, Carnival, Push Woosh, Urban Airship.

У данных сервисов, однако, есть недостатки, из-за которых они могут подойти не всем разработчикам:

- относительно высокая месячная цена(доходящая до 3000$),
- достаточно большой размер клиентских SDK,
- необходимость перенесения логики группировки(сегментации) пользователей на чужой сервис.

Первой проблема заключается в том, что большое число таких сервисов предоставляется по модели Software as as Service(Saas). Одновременно с этим существует очень малое количество качественных сервисов, которые специализировались бы только на уведомлениях. Чаще данные сервисы представляют из себя мощные платформы, решающие задачи маркетинга и аналитики. Как резульат, пользователь должен платить не только функционал, который ему нужен, но и за дополнительный, которым он не будет пользоваться.

Большой размер клиентских SDK – вторая проблема, напрямую вытекающая из первой. В связи с достаточным количеством избыточной функциональности, размер библиотек, встраиваемых в систему, может достигать 15Мбайт. При размере приложения, не превышающем этого числа, встраивать такой модуль в приложение может быть неверным решением: несмотря на то, что объем памяти и скорость передчи данных на пользовательсих устройствах значительно вырос за последние годы, люди все также неохотно загружают приложения большого объема.

Третья проблема является самой серьезной из этого списка. Она связана с сегментацией и группировокой пользователей. Чаще всего разработчик хочет отправлять push-уведомления не каждому отдельнному человеку, а группам людей. Например, отправить уведомления только платящим пользователям, пользователям определенного возраста или

пола, или группе пользователей исходя из предметной области приложения(авиапассажирам, зарегистрированным на определенный рейс).

Когда разработчик работает со сторонними сервисами, ему необходимо перенести данные о том, каким группам принадлжеит пользователь, на сервис, который он будет использовать. Чаще всего осуществляется это с помощью механизма "тегов". Разработчик назначает "теги" пользователям, а потом выбирает пользователей с определенными тегами или группами тегов. Однако, этого механизма может быть недостаточно, и работать с ним не всегда бывает удобно:

- группы пользователей могут динамически меняться и этим нужно управлять,
- групп может быть очень много(до нескольких сотен), они могут пересекаться, содержать друг друга в себе и перенос логики на сторонний сервис - проблема,
- историчность - невозможность привязать последовательность действий пользователя к событию, чтобы отправить уведомелния после того, как оно случится.

Для данных трех пунктов можно привести пример: допустим, мы знаем, что определенный пользователь совершил покупку в приложении две недели назад. Предпоследнюю неделю он заходил каждый день, а за последнюю неделю он зашел не более двух раз. Последние два дня он совсем перестал появляться в приложении. Такого пользователя мы можем пометить, как потенциально уходящего.

Когда мы работаем со сторонним сервисом, нам постоянно придется помечать пользователей тегами, для того, чтобы как-то обозначить пользователя или пользователей, совершивших последовательность действий в течение некоторого промежутка времени. Часто может так быть, что определенным тэгом будет помечен только один или несколько пользователей из достаточно большого числа, что породит большое количество сегментов пользователей. При этом, некоторые сегменты

могут объединять группы других, и из-за этого на сторонний сервис придется переносить достаточно перусложненную логику сегментации.

Часть сервисов предлагают использовать модель подписки на события для своевременной смены сегментов пользователя после их наступления. Однако это порождает лишнее количество трафика, особенно в приложениях, в которых пользователь может часто переходить из одних групп в другие и сильно усложняет логику при большом количестве групп.

Отличным решением данных проблем будет использовать для отправки уведомлений инструмент другого типа: логика сегментации, которую необходимо переносить на сервера сторонних сервисов, уже есть на сервере приложения разработчика. Поэтому, гораздо удобнее было бы работать с этими данными прямо с него.

Из этих недостатко появились следующие требования к решению:

- поддержка основных мобильных платформ,
- поддержка webpush уведомлений,
- наличие открытой лицензии,
- наличие клиентского SDK,
- совмещение функциональности сервера и библиотеки.

В последнем пункте имеется ввиду возможность отправлять уведомления как с помощью вызовов процедур из собственного кода, так и Post-запросами на standalone-сервер, предзназначенный как для для отправки, так и для мониторинга доставки уведомлений, ведения статистики, управления информацией о пользователей и push-токенами, выдываемыми Pns-провайдерами на каждого их них.

Таким образом, разрабатываемый сервис будет имеет следующие преимущества:

- Автоматизация хранения и манипулирования профилями пользователей и их токенами, их обновления и т д.

- За счет этого - возможность работы с пользователем, используя его идентификатор в приложении. Это удобнее чем отправлять уведомления по push-токенам, где надо знать, какому Pns-провайдеру он принадлежит и в каком виде он хочет получить уведомление
- Наличие клиентского SDK для работы с уведомлениями, что убирает необходимость реализации отображения уведомлений, обновления и хранения токена, отправки уведомлений
- Возможность запускать сервис как отдельный сервер, так и использовать как библиотеку для отправки уведомлений
- Единый интерфейс для отправки Web и мобильных уведомлений.
- Возможность подключения приложений соц-сетей(Fb, Вк, Ок) для отправки уведомлений через соцсети.

3. Обзор существующих решений

Критерии обзора существующих решений были определены из требований к решению:

- поддержка основных мобильных платформ,
- поддержка webpush уведомлений,
- наличие открытой лицензии,
- наличие клиентского SDK,
- совмещение функциональности сервера и библиотеки.

По данным критериям было проведено сравнение существующих решений, составлена сравнительная таблица и описаны выводы.

3.1. Решения

3.1.1. Pushd

Описание: Pushd[10] - это подключаемый унифицированный сервер для отправки серверных push-уведомлений нам мобильные приложения, веб-приложения и т. Д. С помощью pushd есть возможность отправлять push-уведомления на любую поддерживаемую мобильную платформу, веб-приложение или HTTP-сервер из одной точки входа. Pushd заботится о том, какое устройство подписано на какие события и предназначено для поддержки неограниченного количества подписных событий.

Платформы: APN (iOS), C2DM / GCM[6] (Android), MPNS (Windows Phone), HTTP POST, EventSource, WNS[15] (служба уведомлений Windows).

Основные особенности:

- тихий режим подписки,
- шаблоны сообщений,
- массовая рассылка,

- отслеживание статистики,
- автоматическое отключение регистрации отказавшего абонента,
- Redis в качестве базы данных для хранения push-токенов.

3.1.2. Rpush

Описание: Rpush[12] – мощная библиотека-сервер для отправки push-уведомлений в Ruby. Rpush достаточно простой в использовании, надежный и имеет богатый набор функций.

Платформы: Apple Push-notification Service[3], включая Safari Push-notification(iOS, Safari), Firebase Cloud Messaging(Android)[4], Windows Notification Service(Windows Phone, Windows)[15], Amazon Device Messaging[2].

Основные особенности:

- позволяет использовать ActiveRecord, Redis или MongoDB для хранения данных пользователе,
- позволяет использовать плагины для работы с попалярными сервисами уведомелиний, или даже написать собственные,
- может запускаться как отдельный процесс, внутри очереди заданий, осущствлять отправку из командной строки или встраиваться в другой процесс.
- масштабируется как вертикально(с увеличением ресурсов на одном узле), так и горизонтально(с увеличением числа узлов) ,
- имеет возможности для безотказной работы,
- работает с MRI, JRuby и Rubinius.

3.1.3. Aerogear

Описание: Aerogear[1] – достаточно мощный по функциональности бесплатный сервер для отправки и мониторинга доставки уведомлений, написанный на языке Java.

Платформы: Android(не поддерживается), iOS, Windows Phone(в разработке)

Основная особенность:

- наличие клиенсткого SDK для автоматизации регистрации пользователей на сервере и обработки статистики,
- поддержка отправки браузерных уведомлений на Mozilla Firefox.

3.1.4. Push Notify

Описание: Push Notify[9] – Библиотека для отправки Push-уведомлений, написанная на языке JavaScript и распространяемая под лицезнией MIT. Данную библиотеку нельзя запустить, как отдельный сервер и с помощью данной библиотеки нельзя отслеживать доставку отправленнх уведомлений, однако позволяет отправлять уведомления на поддерживаемые платформы. Сейчас уже почти не поддерживается, последние изменения были 3 года назад.

Платформы: iOS, Android, Windows Phone

Основная особенность:поддержка основных мобильных платформ, в т.ч. устаревшие сервисы: Android Cloud to Device Messaging (C2DM) и Microsoft Push Notification Service (MPNS).

3.1.5. Pushkin

Описание: Pushkin[11] – это бесплатный инструмент с открытым исходным кодом для отправки push-уведомлений. Разработчики говорят, что он был разработан с акцентом на скорость и возможность быстрого экспериментирования. В основном создан для поддержки онлайновых мобильных игр, но его можно легко распространить на любой тип приложений.

Платформы: Android и iOS.

Основные особенности:

- наличие режима работы с событиями, когда уведомления отправляются по событиям, произошедшим на сервере разработчика,

- легко масштабируется, можно запускать столько экземпляров сервиса, в скольких есть потребность,
- один экземпляр способен отправлять до 500 уведомлений в секунду.

3.1.6. Web-Push

Описание: Web-Push[14] – библиотека, позволяющая отправлять push-сообщения в веб-браузеры с серверов, написанных Ruby, используя протокол Web Push[13]. Он поддерживает шифрование сообщений протокола Web Push для надежной передачи сообщений от сервера пользователю.

Платформы: Общий вид push-уведомлений поддерживается Chrome50 +, Firefox48 +. Мобильные платформы не поддерживаются

Основные особенности:

- поддержка браузерных уведомлений,
- поддержка шифрования,

3.2. Сравнительная таблица

Как видим, ни в каком из представленных open-source сервисов нет поддержки Webpush уведомлений , за исключением webpush, который направлен исключительно на такие уведомления и ни на что больше, и Aerogear, который использует не Push Api, утвержденный организацией World Wide Web Concorcium, а проприетарную разработку компании Mozilla, работающую только с Firefox Os и браузером Mozilla[5].

Кроме того только в AeroGear есть наличие клиентской части SDK. Однако Aerogear уже не поддерживает Android и Winphone модуль находится в процессе разработки. Помимо этого предполагается, что модуль используется лишь для регистрации на их сервере, обработку и отображение уведомлений необходимо реализовывать самому по их гайдам.

Сервисы уведомлений	Платформы	Webpush	Клиентская часть	Варианты работы	Лицензия
Pushd	I, An, W	Нет	Нет	SaaS	MIT
Rpush	I, An, W, Am	Нет	Нет	SaaS, L	MIT
Aerogear	I, An, W	Mz	Есть	SaaS	Apache 2.0
Push Notify	I, An, W	Нет	Нет	L	MIT
PushJet	An	Нет	An	SaaS	BSD
Pushkin	An,I	Нет	Нет	SaaS	MIT
Web-Push	Нет	Есть	Нет	L	MPL, v2.0

Таблица 1: Сравнение популярных сервисов push-уведомлений

Таким образом мы приходим к выводу, что ни один из сервисов нам не подходит по требованиям, требуется их серьезная переработка. Можно подумать, что лучшим вариантом за основу будет взять имено Aerogear. Но из-за слабого коммьюнити, отсуствия поддержки выбор был остановлен на Rpush[12]. Серверная часть приложений Nevosoft разрабатывается на Ruby, как и Rpush, что облегчит поддержку и интеграцию, а из мобильных платформ у Rpush есть возможность отправки еще и на Amazon Fire OS.

4. Архитектура

4.1. Взаимодействие компонентов сервиса

Сервис состоит из четырех основных компонентов:

- сервера Push-уведомлений, предназначенного для их отправки на конечные устройства, обработки запросов на получение статистики, удаленной настройки уведомлений и планирования отдельных уведомлений или групп;
- сервера веб-приложения, предназначенного для отправки уведомлений пользователям или группам пользователей и отображения статистики по уведомлениям в формате графиков;
- базы данных, предназначенной для хранения информации о пользователях, их устройствах и push-токенах;
- клиентской части - библиотеки, обрабатывающей приходящие push-уведомления, отображающей их и отсылающей статистику просмотров уведомлений на сервер.

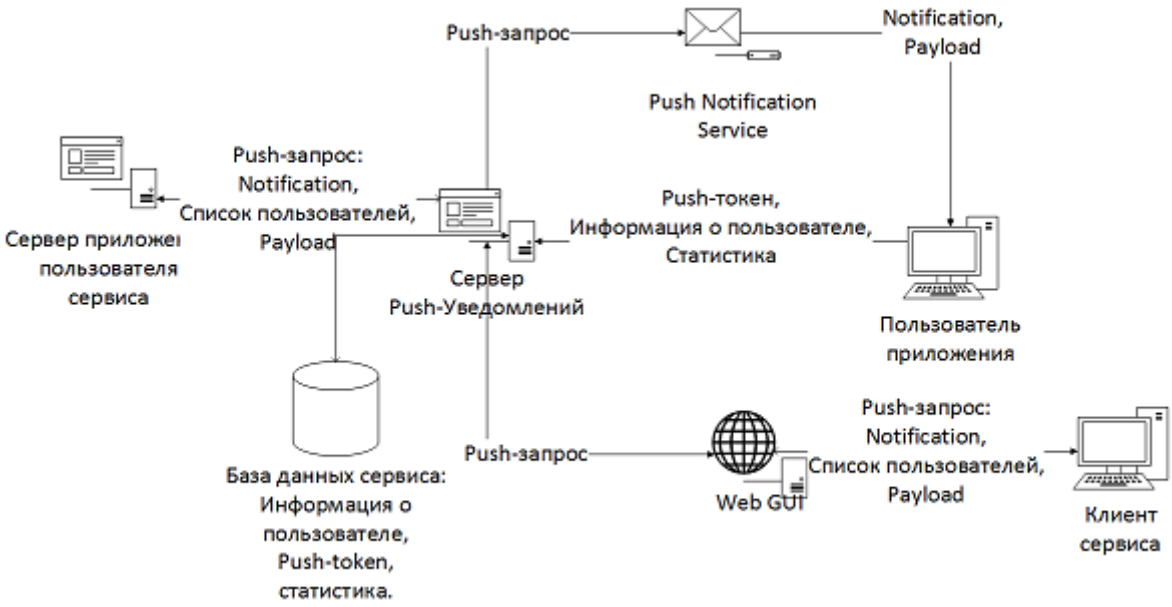

Рис. 1: Архитектура сервисая

Цикл работы сервиса происходит в два этапа:

На первом этапе, после запуска приложения, оно регистрируется у PNS-провайдера и получает push-токен - уникальное значение, идентифицирующее устройство для отправки уведомлений. Вместо с другой информацией о пользователе(данными о пользовательской платформе и другой вспомогательной информацией), токен отправляется на сервер сервиса push-уведомлений, где информация обрабатывается и сохраняется в базе данных.

На втором этапе, если пользователю сервиса необходимо отправить уведомления. Для этого он в Web-GUI или со своего сервера формирует сообщение, включаемые в уведолмнеие данные, и тем или иным образом выбирает пользователей или группы пользователей для отправки(т.е. push-токены, которым хотим отправить уведомление). Далее пользователь сервсиса отправляет запрос на сервер push-уведомлений, который формирует уведомление с токеном в том виде, который ждет от него PNS-провайдер, после чего уведомление попадает к нему. PNS-провайдер принимает уведомление и, если на телефоне не отключены уведомления и возможно установить сетевое соединение с сервером, пытается отправить уведомления на устройства. После доставки, показа, открытия или отклонения уведомлений, пользовательское приложение формирует и отправляет статистику на сервер-push уведомлений, который сохраняет ее в своей БД.

Необходимые действия для настройки сервиса следующие: разработчик добавляет библиотеку в мобильное приложение, прописывает в конфигурационном файле точку подключения к серверу, и идентификатор пользователя.

На сервере подключает серверную часть за основу которой взяты Rpush[12] и Webpush[14](в качестве библиотеки), в конфигурационном файле прописывает пути к базе данных и забывает про существование данной связи между клиентом и сервером. Теперь отправка уведомлений может производиться из кода с использованием не токенов push-уведомлений, а идентификатора пользователя приложения, например, логина в контексте сервера приложений разработчика.

Серверная часть сервиса, как было сказано ранее, объединяет в себе

библиотеку и серевер. Если разработчик использует сервис как библиотеку, то он имеет возможность направлять push-уведомления прямо из своего исходного кода прямо на сервера PNS-провайдеров. Если разработчик использует сервис как отдельный сервер, то он может работать с уведомлениями с помощью POST-запросов с информацией об уведомлениях и получателя, отправленных на данный сервис, который перешлет уведомления на PNS-провайдеров. В данном варианте подтверждение доставки, сохранение статистики, регистрация пользователей происходит на сервере, а в первом пользователь должен обрабатывать сам.

5. Реализация мобильной библиотеки

Мобильную библиотеку сервиса можно поделить на две части: платформонезависимую и и платформозависимую.

платформозависимая часть:

Analytics - сохранение и отправка на сервер данных о времени, ID и др. пришедших, показанных уведомлениях и т.д.

PushUser - хранение информации о пользователе приложения, его персональных данных, тэгов для группировки и сегментации, вторичных имен и идентификаторов.

PushManager - обработка настроек об отображении, звуке, вибрации, тихих часах, геопозиционировании и текущем пользователе.

Платформозависимая часть:

NotificationBuilder - формирование внешнего вида и сообщения на основе информации, пришедшей в уведомлении.

MessagingService - функции обратного вызова на все действия, связанные с приходом уведомления и регистрацией в сервисах.

Работа данного модуля происходит следующим образом: при запуске приложения PushManager регистрируется у Pns-провайдера и получает токен. Далее он создает экземпляр User и заполняет его данными пользователя, среди которых токен и находится. Далее, при необходимости он отправляет обновленную информацию на сервер. Клиентская часть готова к работе.

Далее, после прихода уведомления, информация о времени прихода сохраняется локально модулем *Analytics*, *PushManager* проверяется, можно ли показать уведомление, и в случае ночных часов отодлевает показ до их окончания. При необходимости показать уведомеление данные о нем(иконки, фоновые изображения, параметры текста) передаются в *NotificationBuilder*, который по этим параметрам строит уведомление и показывает его.

6. Реализация серверной части

Серверная часть сервиса разработана на языке Ruby с использованием фреймворка EventMachine для организации асинхронного сетевого взаимодействия.

В качестве базы данных использовалась MongoDB, а в качестве ODM-фреймворка для работы с базой данных – mongoid.

Отправка push-уведомлений происходит с помощью класса "PNS", который предоставляет общий интерфейс к двум другим классам – push_provider и webpush_provider. В основе первого находится библиотека rpush, в основе второго – webpush. Эти модули в совокупности абстрагируют разработчика от управления токенами, отдельными платформами, предоставляя общий интерфейс ко всем типам уведомлений.

На данный момент статистика отправленных и доставленных уведомлений хранится в базе данных сервиса По запросу можно получить статистику по:

- пришедшим,
- показанным,
- открытым,
- сброшенным уведомлениям,
- регистрациям и входам пользователя в приложение.

Планируется возможность подключения вместо внутреннего модуля, сторонние модули, например Google Analytics или Flurry.

6.1. Базовый API отправки push-уведомлений

Для отправки push уведомления, необходимо произвести post-запрос на адрес /push.

Пример отправки сообщения:

```
id = user.id
pns = Pns.instance
msg = {
  "message" => "Hi, people",
  "payload" => "SomeData",
...
  "url" => "http://../someUrl",
  "networks" => [ "android",  "ios", "wp", "fb", "vk",
}
pns.send(id, msg)
```

Каждая из платформ выберет для использования только ту только ту информацию, которую она поддерживает.

title – заголовок.

message – текст уведомления.

payload – текст, передаваемый в приложения после клика на уведомление.

image – иконка (на платформе Android – фон развернутого сообщения).

networks – строковый массив платформ, для которых предназначено уведомление.

Также отдельно для кастомизации внешнего вида Android-уведомления предоставляются следюущие поля: фоновая картинка, цвет фона, цвет свёрнутого фона, цвет текста, цвет текста свёрнутого сообщения, флаг отображения иконки.

Только для Web Push уведомлений устанавливается ссылка для перехода по клику на уведомление

6.2. Способы выборки аудитории для отправки уведомлений

Как уже говорилось ранее, одной из основных особенностей данного сервиса является возможность сохранить логику сегментации на ссво-

ем сервере приложений и не переносить ее на сервис увдомлений. Для этого в сервисе реализованы возможности автоматического хранения и манипулирования профилями пользователей, их токенами, обновлениями, за счет чего имеется возможность работы с пользователем, используя его идентификатор в приложении. Это удобнее чем отправлять уведомления по push-токенам, где надо знать, какому Pns-провайдеру он принадлежит и в каком виде он хочет получить уведомление.

Увдомления отправляются с прямо из сервера приложений вызовом специальных процедур, и так как вся информация о пользователях расположена на сервере приложений и в серверных базах данных, то пользователь может сам выбирать идентификаторы пользователей для отправки с помощью запросов к своей базе данных или путем назначения в качестве обработчика событий процедуры отправки уведомеления.

Однако, в сервисе была сохранена также и воможность назначения тегов пользователям. При необходимости, разработчик имеет возомможность зарегистрировать тэги для определенного пользователя прямо из приложения или сервера приложения, обновив массив тегов, и далее отправлять уведомления, передавая в качестве одного из параметров тэг или список тегов. Пользователь, у которого найдутся все пристуствующие в списке теги, будет выбран для отправки уведомления

6.3. Генерация динамически-формируемых сообщений в уведомлениях

Для генерации динамически-формируемых уведомлений в сервис интегрирован язык шаблонов Liquid[8]. Его использование в общем случае выглядит следующим образом:

```
@template = Liquid::Template.parse("hi {{name}}")
# Parses and compiles the template
@template.render('name' => '...')
# => "hi, Aleksandr"
```

Пользователь сервиса пишет сообщение push-уведомления с вставками названий полей в двойных фигурных скобках. Это могут быть

поля документа БД, содержащие информацию о пользователях его приложения.

Далее, используя язык шаблонов названия полей подменяются на вызовы запросов ODM-фреймворка к базе данных, возвращающих по названию поля соответствующую информацию для конкретного пользователя. Для тех пользователей, у которых значение нужного поля пусто, можно установить значение по-умолчанию. После всего этого push-уведомление направляется данному пользователю.

В общем случае, пользователь может задать в файле список соответствий между шаблонамии и полями базы данных, после чего модуль автоматически подставит нужное поле нужного пользователя.

7. Проверка работоспроообности системы

Для проверки работоспособности истемы были проведены следующие мероприятия:

- С помощью инструмента Postman был разаботан ряд тестов для тестирования API сервера: отправки, регистрации пользователей, обработки статистики по дошедшим уведомлениям.
- Для проверки корректности работы связки "клиентский SDK - сервер" было проведено ручное тестирование по методу описанному ниже.

Для каждой платформы было запущено приложение. При запуске приложения была проверена работоспособность получения push-токена, регистрации пользователей на сервере и сохранения дополнительной информации уже зарегистрированного пользователя. Корректность работы была проверена как в идеальных условиях, так и в условиях нестабильной связи и вообще при ее отсуствии.

Аналогично, с помощью инструмента Postman на каждое из устройств были отправлены уведомления через сервис. Была проверена регистрация и сохранение информации о времени отправки уведомлений на сервере, а также корректность информации о времени доставки и отображении уведомления при каждом из данных событий.

На клиентской части было проверено правильное отображение уведомелений на мобильных устройствах и браузерах, с учетом всех данных об оформлении отправленных с уведомлением и "тихий режим", в котором пришедшее ночью уведомление не отображается. Также были провены очистка уведомлений и передача определенных данных приложению при открытии уведомлений.

Заключение

В ходе данной работы были достигнуты следующие результаты:

- Составлена сравнительная таблица конкурентных продуктов, выделены сильные и слабые стороны каждого из них.
- Разработана архитектура сервиса push-уведомлений.
- Реализована библиотека для работы с сервисом под платформы Android, FireOS, WinPhone, IOs.
- Разработана серверная часть сервиса:
  - Реализован API для отправки, валидации, планирования и тестирования push-уведомлений, а также генерации отчетов по их доставке.
  - На основе идентификаторов устройств, тэгов и динамически изменяемых сегментов пользователей реализована и внедрена возможность выбора необходимой аудитории для отправки уведомлений.
  - На основе метаданных в тексте и языка шаблонов Liquid[8] реализована возможность отправки push-уведомлений группам пользователей с генерацией сообщений на основе индивидуальной для каждого из них информации.

Список литературы

[1] Aerogear. — 2017. — URL: https://aerogear.org (online; accessed: 10.03.2017).

[2] Amazon Device Messaging. — 2017. — URL: https://developer.amazon.com/device-messaging (online; accessed: 15.05.2017).

[3] Apple Push Notification Service. — 2017. — URL: https://en.wikipedia.org/wiki/Apple_Push_Notification_Service (online; accessed: 15.05.2017).

[4] Firebase Cloud Messaging. — 2017. — URL: https://firebase.google.com/docs/cloud-messaging/ (online; accessed: 15.05.2017).

[5] FirefoxOS Push Api. — 2017. — URL: https://developer.mozilla.org/en-US/docs/Archive/Firefox_OS/API/Simple_Push_API (online; accessed: 15.05.2017).

[6] Google Cloud Messaging. — 2017. — URL: https://developers.google.com/cloud-messaging/ (online; accessed: 15.05.2017).

[7] The Inside View: How Consumers Really Feel About Push Notifications. — 2017. — URL: https://goo.gl/7By95s (online; accessed: 11.05.2017).

[8] Liquid MarkUp Language. — 2017. — URL: https://github.com/Shopify/liquid (online; accessed: 15.05.2017).

[9] Push-notify. — 2017. — URL: https://github.com/neoziro/push-notify (online; accessed: 10.03.2017).

[10] Pushd. — 2017. — URL: https://github.com/rs/pushd (online; accessed: 10.03.2017).

[11] Pushkin. — 2017. — URL: http://pushkin.io (online; accessed: 10.03.2017).

[12] Rpush. — 2017. — URL: https://github.com/rpush/rpush (online; accessed: 10.03.2017).

[13] Web Push Api. — 2017. — URL: https://www.w3.org/TR/push-api/ (online; accessed: 15.05.2017).

[14] Webpush. — 2017. — URL: https://github.com/zaru/webpush (online; accessed: 15.05.2017).

[15] Windows Notification Service. — 2017. — URL: https://en.wikipedia.org/wiki/Windows_Push_Notification_Service (online; accessed: 15.05.2017).