**Sendify**

# **Feasibility Report**

## Index

1. Introduction
2. Scope
3. Market Analysis
   1. Sophisticated Analysis
   2. User Control
   3. An Integrated Strategy
4. Technical Analysis
   1. Push notification - An Overview
   2. How Sendify works
   3. Steps in the process
   4. How the network interfaces with Sendify

## 1. Introduction

Developers working on Mobile or web applications need to set up suitable databases (create custom tables and schema) to create custom notification systems as one of the features of an end product. The end product, hence, will actually be a set of APIs and SDKs for certain platforms (Javascript and Android) which they can integrate with ease into their systems and create an entire notification system with a few lines of code.

## 2. Scope

Essentially, we will create schemas and write custom logic for handling this notification system. Originally, of course, it is by no means easy to build seamless real-time functionality and notification system, when developers take this task upon themselves, it requires them to learn a whole new framework, and worry over the configuration of the already existing infrastructure. Thus, this framework will be provided by our APIs.

The basic system functions like a channel for the publisher (our client) to publish information on, and which will be subscribed to by the subscriber (the client of the publisher), and thus with the aid of web socket programming, this end will be achieved.

## 3. Market Analysis

Marketers need to integrate real-time notifications as part of their overall direct marketing and CRM (Customer Relationship management) strategy to take full advantage of this important new opportunity.

Real-time notifications are an important new channel for marketers because these alerts can help increase traffic, optimize app use and drive conversions. As the time spent on apps continues to grow, the importance of real-time notifications is also growing.

RTN are not necessarily a replacement for email, SMS, and other direct messaging tactics. They offer key benefits over this first generation of mobile messaging tools, they are mostly complementary. Moving forward, we expect to see real-time notifications integrated as a core component of any direct marketing and relationship program.

## 3.1 Sophisticated analytics

RTN offer some key benefits over other messaging strategies, such as being less interruptive than SMS, providing the ability to view now or later and not as likely to be lost as an email.

RTN also offers more sophisticated analytics with in-depth data about delivery receipt, open rate, time and engagement. The data provided from RTN can be used to understand how customers react to messages. Real-time is also an inexpensive way to reach out to a brand's best customers.

One of the ways that real-time is expected to complement other messaging and CRM tactics is its extension to different operating systems, smartphones and tablets. Additionally, a wider variety of marketing vendors is expected to add real-time notification capabilities.

## 3.2 User control

One of the key takeaways factor is that users want to be able to set their preferences regarding the type and timing of the content they receive.

The ESPN Score Center is an example of a marketer that is doing a good job letting users personalize their notifications. Users can choose from several different points during a game that they want to receive alerts as well as indicate a quiet time when they do not want to receive any notifications.

What marketers need to keep in mind is that RTN can be used for many different purposes and that marketers should start using them to benefit their marketing goals.

However, it is important to ensure that notifications are not viewed as intrusive or uninteresting. To avoid this, marketers should deliver highly relevant alerts that make the most of mobiles or any device unique benefits of intimacy, immediacy and context.

## 3.3 An integrated strategy

Marketers should also consider ways to reward their customers at specific moments of engagement to increase satisfaction.

An integrated strategy means making sure that all the messages you send, via any messaging channel, serve common goals and that taken altogether, those messages aren’t overwhelming.

This implies that you should invest internal resources to define how best to engage customers through different channels — and avoid outsourcing this strategic task to an agency or a vendor.

## 4. Technical Analysis

Sending simultaneous push notifications in a low-latency way to millions of mobile users, and handling real world requirements such as localization, multiple platform devices, and user personalization is a hard task.

The main reason this is challenging is that push notifications are delivered to the devices by platform-specific services. For instance, you have to use the Apple Push Notification service (APNs) for iOS, Google Cloud Messaging (GCM) for Android and WebSockets (Browser).

All these platform services work by having the device app request a user-, device-, and app-specific ChannelURI and then store it somewhere in the app back-end. Then, when sending a notification, the app back-end posts the notification payload to the ChannelURI in order to reach that particular device. To complicate matters, the ChannelURIs expire (and thus have to be continuously re-uploaded to the back-end) and multiple ChannelURIs can be active at the same time for the same device. When coding multi-platform apps, this problem is intensified because each platform handles push notifications in a slightly different way.

Assuming we have to send notifications about breaking news, and each user can subscribe to different categories, you will inevitably end up managing various information and details as a database table.

It’s clear that if you want to allow users to have multiple devices, and if you want to start handling localization and user preferences, this approach can quickly become problematic. If you have thousands of devices to notify, having a process go through a for loop won’t give your users a great experience. This means, for example, the latency of the notification will quickly become unacceptably long, and you won’t be able to recover in case something happens to your back-end in the middle of a push. Clearly, while you can solve these problems by storing your device information in multiple databases (i.e. sharding), creating multiple virtual machines that send notifications in parallel, and periodically saving your progress to persistent storage while sending notifications to large numbers of devices, it’s a hard task to implement these patterns in yet another part of your back-end!

Summarizing, we just saw how implementing a real world push notification solution in your app can quickly become large and complex. The biggest challenge is managing device information (especially for cross-platform and/or localized apps) and scaling up of the push infrastructure to handle up to millions of devices

We will be solving all these problems. On the back-end stack, we will be using

ejabbered and MySQL for database. Our push notification will be based on sockets/XMPP.

## 4.1 Push Notifications – Overview

Once you're provisioned, your application can submit a notification to Sendify's service. Sendify pushes your notification to the appropriate platform for routing to the end user's device. You can submit notifications individually, to a list of instances or to groups, and to either to an individual network or to both.

## 4.2 How Sendify Push Notifications works

By integrating developer’s application with Sendify Push Notifications, you can send a notification to a target application on an end user's device. The process of sending a notification involves these functions:

1. Registration of the application instance with Sendify Push Notifications.

2. A POST call is submitted to Sendify to send the push notification to one or more

destinations, or to a group of destinations.

3. Sendify returns a ticket ID and sends the notification to the specified destination(s).

Registering the application instance: Registration of an application instance, and management of that information, involves supplying Sendify with the destination ID, and optionally group membership information. You can design your application to self-register by supplying the registration information directly to Sendify, or your application can provide the information to your servers which then pass it on to Sendify. Regardless of how you design your application, you use the same APIs.

Specifying the destinations: You have a couple options for how to specify the target destination of a push notification. You can submit a request to send a notification to one destination—one application instance. You can also define one or more groups of destinations, which gives you the ability to send a notification to multiple destinations. This capability is useful if you want to segment your customer base. You can also send a notification to a group defined as "all", which results in the notification being sent to all registered application instances that are enabled.

## 4.3 Steps in the process

1. The end user launches an application that he previously downloaded to his device. (When he installed the application, he configured it to allow push notifications.) The application obtains a destinationId from the network and returns it to your servers.

2. Your system calls PUT New Application Instance to register the destination ID with Sendify Push Notifications.

3. Sendify responds with a success or fail, and if successful, returns the instance ID.

4. Upon successful device registration, your system submits a POST Notification to

Sendify.

5. Sendify returns a ticket ID.

6. Your service can, as an option, call GET Notification Status to retrieve the state of the notification. Sendify will return the status, appId, instanceId, ticketId, createdDate, and processedDate.

If the notification's state is "failed", Sendify returns a failure reason.

## 4.4 How your application interfaces with Sendify and external networks

The server side of your application is the sender of a push notification. To route a notification to an application instance through Sendify, you first get provisioned to use Sendify Push Notifications. For Apple devices, this involves obtaining security certificates that Sendify will use to connect to APNS (Apple Push Notifications Service) environments. GCM requires a Google account, which Sendify uses to connect to the GCM interface.