2014

# Software Requirements Specification for Notification System

VERSION 1.0

VAMSHI CHENNA

PREPARED BY CHINAR MEHTA, ANUSHREE GUPTA, VIJAY KHOTOLIYA | [Company address]

## Table of Contents

## Revision History

| Date | Reason For Change | Version |
|---|---|---|
| 18/2/14 | Creation | 1.0 |

# 1. Introduction

## 1.1 Purpose

The purpose of this document is to present a detailed description of the functionalities of Sendify - a custom notification system which can be integrated as one of the features of an end product. The document explains the purpose and features of the system, the interfaces of the system, what the system will do, the constraints under which it must operate and how the system will react to external stimuli. This document is intended for the developers who intend to incorporate a notification system within their application.

## 1.2 Document Conventions

The overall idea behind the document is to outline the system features, the functional and the non-functional requirements of the product Sendify. This has been done to the best of the ability of the developers of the product.

Formatting conventions of the document are:

| Sr. No | Conventions | Metrics | Inference |
|---|---|---|---|
| 1 | <Title> | Font size: 21<br>Font type: Times new Roman<br>Font style: Regular | Title |
| 2 | <Heading-1> | Font size: 18<br>Font type: Times new Roman<br>Font style: Bold | **Heading-1** |
| 3 | <Heading-2> | Font size: 13<br>Font type: Times new Roman<br>Font Style: Bold | **Heading-2** |
| 4 | <Main Body> | Font size: 11<br>Font type: Times new Roman<br>Font style: Regular | Main Body |

## 1.3 Intended Audience and Reading Suggestions

This document is intended for all individuals participating in and/or supervising the Sendify project.

- Readers interested in a brief overview of the product should focus on the rest of Section 1 (Introduction), as well as Section 2 of the document (Overall Description), which provides a brief overview of each aspect of the project as a whole.
- Section 3 (External Interface Requirements) offers further technical details, including information on software platforms on which the application will run. Readers who wish to explore the features of Sendify in more detail should read on to Section 4 (System Features), which expands upon the information laid out in the main overview.
- Readers interested in the non-technical aspects of the project should read Section 5 (Other Nonfunctional Requirements), which covers performance, safety, security, and various other attributes that will be important to users.

## 1.4 Product Scope

Sendify uses the “publish-subscribe” (pub-sub) messaging paradigm, with notifications being delivered to clients using a “push” mechanism that eliminates the need to periodically check or ‘poll’ for new information and updates. With simple APIs requiring minimal up-front development effort and no maintenance or management overhead, the software gives developers an easy mechanism to incorporate a powerful notification system within their applications with only a few lines of code.

### 1.5 References

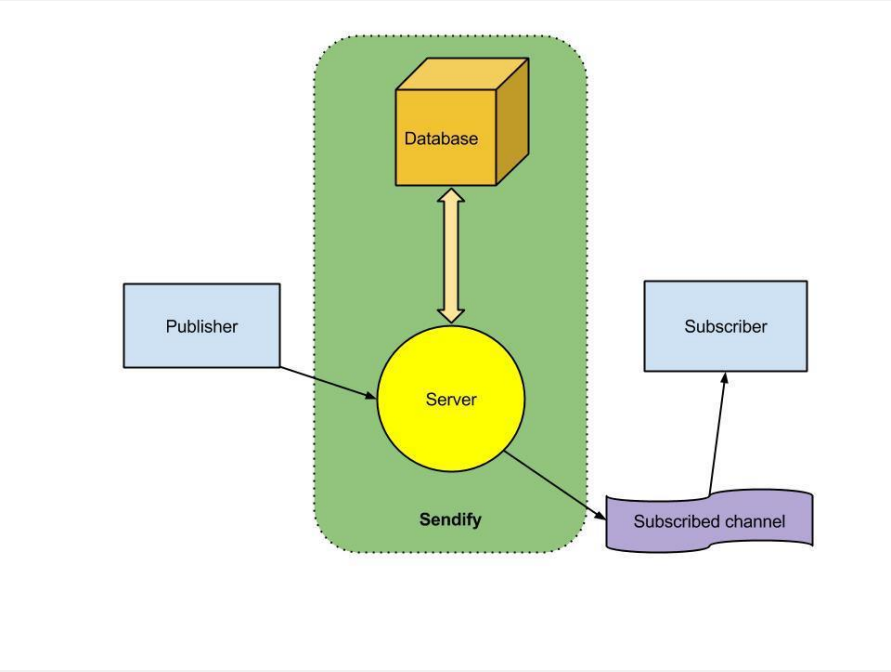
# 2. Overall Description

## 2.1 Product Perspective

Sendify is a new, self-contained product intended for use on the 'Web' as well as the 'Android' platforms.

The system has two types of clients - publishers and subscribers - also referred to as producers and consumers respectively. Publishers communicate asynchronously with subscribers by producing and sending a message to a topic, which is a logical access point and communication channel. Subscribers (that is, web servers, email addresses, et al.) consume or receive the message or notification over one of the supported protocols (that is, HTTP/S, XMPP, et al.) when they are subscribed to the topic.

The following diagram illustrates the above mentioned interactions between the system and its clients, that is, the publishers and subscribers:




When using Sendify, you (as the owner) create a topic and control access to it by defining policies that determine which publishers and subscribers can communicate with the topic. A publisher sends messages to topics that they have created or to topics they have permission to publish to. Instead of including a specific destination address in each message, a publisher sends a message to the topic. We will match the topic to a list of subscribers who have subscribed to that topic, and deliver the message to each of those subscribers. Each topic has a unique name that identifies endpoint for publishers to post messages and subscribers to register for notifications. Subscribers receive all messages published to the topics to which they subscribe, and all subscribers to a topic receive the same messages.

Our software allows applications to send time-critical messages to multiple subscribers through a ‘push’ mechanism, eliminating the need to periodically check or ‘poll’ for updates.

## 2.2 Product Functions

Following are the functions that the product should be able to do:

(i) Allow publishers to push notifications to the subscriber.

(ii) Allow publishers to publish contents and make use of this service in multiple devices i.e., cross-platform service is available.

Section 4 provides a more detailed description pertaining to the functions mentioned above.

## 2.3 User Classes and Characteristics

(i) Publisher:

- Publisher will be recognized in the system by a unique ID.
- Publisher can use any of the platforms - web, or mobile for publishing.
- Publisher will have the option to change the notification even though it has been delivered to the subscribers. In such a case a separate notification will be sent to those who have seen the notification and the notification will be updated in those cases where the subscribers haven’t seen the notification.
- Publisher can only add a subscriber. He can neither see the list of all subscribers subscribed to the topic, nor terminate the subscription of a particular subscriber.

(ii) Topic Owner:

- Any user in the system (either the developer or the customer) can create a topic. He will be called as the topic owner.
- There can be many publishers for a topic but there will only be a single topic owner.
- Topic owner has an option to add/delete the users to the topic himself or let the users to subscribe/unsubscribe to the topics on their own. In the earlier case, the user has to confirm his subscription/unsubscription.
- Only the Topic owner can delete a topic.
- Topic owner can see who all are subscribed to a topic.
- Topic owner can add publishers to a topic.

## 2.4 Operating Environment

The end-product of the Sendify project will be Software Development Kits and REST APIs for integrating the custom notification system within applications developed for the ‘Web’ and ‘Android’ platforms.

The Sendify server will be built on top of ‘ejabberd’ (an XMPP application server written in

Erlang Programming Language) and could therefore be run under several UNIX-like operating systems (such as Mac OS X, GNU/Linux, et al) as well as Microsoft Windows.

The database will be stored on the server using a Relational Database Management System such as MySQL or PostGreSQL.

## 2.5 Design and Implementation Constraints

- The Internet connection is a constraint for the application. Since the application fetches data from the database over the Internet, it is crucial that there be an Internet connection for the application to function.

- To send a single push notification message to one mobile user is relatively straight forward. However, 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 much harder.

- Sendify requires platform specific services to deliver push notifications to multiple devices.

- Assuming one wants to send notifications about breaking news, and each user can subscribe to different categories, one will inevitably end up managing a huge database table. If we want to allow users to have multiple devices, and also start handling localization and user preferences, this approach can quickly become problematic. The database design needs to be made in a way so that it scales well under heavy load conditions. Also the applications needs to handle most of the complexity so that it works with minimum administration overhead and is not only easy to maintain, but also easy to scale.

- The number of notifications which can be published will be constrained by the capacity of the database.

- **Scale:** Scaling the infrastructure of this project has two aspects:

    1. Per PNS (Push Notification Service) guidelines, device tokens must be refreshed every time the app is launched. This leads to a large amount of traffic (and consequent database access) just to keep the device tokens up to date. When the number of devices grows (possibly to millions), the cost of creating and maintaining this infrastructure is nontrivial.

    2. Most PNSs do not support broadcasting to multiple devices. As such, a broadcast to millions of devices results in millions of calls to the PNSs. Being able to scale these requests is nontrivial, because app developers usually want to keep the total latency low (for example, the last device to receive the message should not receive the notification 30 minutes after the notifications has been sent, because for many cases it would defeat the purpose of having push notifications).

- **Routing:** PNSs provide a way to send a message to a device. In most apps, however, notifications are targeted at users and/or interest groups (for example, all employees assigned to a certain customer account). As such, the app backend must maintain a registry that associates interest groups with device tokens in order to route the notifications to the correct devices. This overhead adds to the total time to market and maintenance costs of an application.

- **Monitoring and Telemetry:** Tracking and aggregating the outcomes of millions of notifications is not trivial, and it is usually an important component of any solution that uses push notifications.

### 2.6 Assumption and Dependencies

- **Platform dependency:** In order to send notifications to devices on different platforms, one must code multiple interfaces in the backend. Not only are the low-level details different, but also the presentation of the notification (tile, toast, or badge) is platform-dependent. These differences lead to complex and hard-to-maintain back-end code.

- The message that the publisher publishes has to be in the correct format, and the correction of such is not the responsibility of the application Sendify.

# 3. External Interface Requirements

### 3.1 Software Interfaces:

The Sendify project is supposed to provide SDKs and REST APIs for the ‘Web’ and ‘Android’ platforms using the Ruby On Rails framework, the Java JDK (Java Development Kit) and Android SDK tools respectively.

The notifications will be pushed to the Sendify server (built on top of ‘ejabberd’) via either XMPP (in case of mobile platform) or Web-sockets (in case of web platform).

The database (for queueing notifications) will be stored on the server using a Relational Database Management System such as MySQL or PostGreSQL.

### 3.2 Communication Interfaces

Communication between different parts of the system is of essence since they depend on each other. The way in which this communication will be achieved is specified as follows:

The notifications will be pushed to the Sendify server (built on top of the ‘Ejabberd’ server) via XMPP - a communication protocol for message-oriented middleware based on XML - in the case of mobile (Android) platform, and via web-sockets (based on HTTP) in the case

of Web platform.

Also, a specific message formatting, as defined by the developer, would have to be conformed to since the system behaves like a dumb-pipe. Moreover, the receipt and delivery of notifications would be synchronized based on the first-in first-out queueing mechanism.

# 4. System Features

## 4.1 Real-time Notification API (Plug and use)

### 4.1.1 Description and Priority:

The publisher will be able to send real time notifications to its subscribers with a very short latency period. This is a High Priority feature, and the basis of Sendify. All communication technologies used is to make sure that latency is as low as possible and notifications reach the subscriber almost as soon as they are published. This quick delivery is possible only if the subscriber is connected to the Internet at that point in time.

### 4.1.2 Functional Requirements:

A short latency period can be ensured only if the communication is fast, and because of this Web sockets and XMPP are used (which are fast transport mechanisms).

## 4.2 Cross-Platform Functionality

### 4.2.1 Description and Priority:

The APIs could be used by any application developed for a platform capable of making HTTP requests. The web browsers and Android APIs are high priority tasks.

### 4.2.2 Functional Requirements:

The possibility to communicate with the server without web-sockets (due to inefficiency of web sockets on Android device performance). But, websocket communication should also be possible since web browsers work best with sockets.

## 4.3 Queueing Capability

### 4.3.1 Description and Priority:

This feature enables the subscriber to receive notifications even if they are offline i.e., they receive the notification when they next have internet connectivity. This is a High Priority feature.

### 4.3.1 Functional Requirements:

This feature requires the notifications that are not delivered to be stored in the database temporarily, so that they can be fetched again when the subscriber is listening to the channel once again. All the notifications missed why they were offline will then be received. TBD.

## 4.4 Analytics (Notifications usage graph)

#### 4.4.1. Description and Priority:

This feature shows the client (the publisher) information regarding the activities of the subscribers (when, and how many notifications have been received) and the analytics related to that. This is a Medium priority feature.

**4.4.2 Functional Requirements:** TBD.

# 5. Other Non-functional Requirements

## 5.1 Performance Requirements

- The application is a multi-user interface. There is a possibility of an unexpected increase in the number of users. The system will be designed such that it is able to handle the traffic efficiently.
- The application will be available to the user 24x7, i.e. the server will be on all the time.
- The application provides real-time notification, and is therefore very fast and reliable so (as long as the subscriber is online).

## 5.2 Safety and Security Requirements

- Sendify does not manipulate the data provided by the publisher. In a way it acts as a dumb pipe which takes input from the publisher and gives output to the subscriber.
- Sendify uses well established, tested and reliable platforms to do its operation. So, safety and security issues about third party sneaking will not be a major problem.
- Sendify employs SSL encryption (provided by 'Ejabberd') for all data transfer.

## 5.3 Software Quality Attributes

- Flexibility: Sendify allows applications and end-users on different devices to receive notifications via Mobile Push notification (Apple, Google and Kindle Fire Devices), HTTP/HTTPS, et al.
- Latency: The application provides real-time notification, and is therefore very fast and reliably so.
- Scalability: Sendify will be designed to meet the needs of the largest and most demanding applications, allowing applications to publish an unlimited number of messages at any time.
- Security: Sendify provides access control mechanisms to ensure that topics and messages are secured against unauthorized access. Topic owners can set policies for a topic that restrict who can publish or subscribe to a topic. Additionally, topic owners can ensure that notifications are encrypted by specifying that the delivery mechanism must be HTTPS.
- Real time: The publisher will be able to send real time notifications to its subscribers.

- Simplicity: Developers can get started with sendify by simply including its API and using its abstract functions.

- Inexpensive: Customers of Sendify benefit from pay-as-you-go pricing with no up-front fees or commitments. The only costs of sending messages through Sendify are costs for storing the information in the data center.

### 5.4 Business Rules

The Sendify APIs will be free initially, but will cost according to the usage of the publisher. We only charge once application exceeds number of daily active devices. A win-win situation for all, since testing becomes easy for the client. This is a capital efficient business. The only costs for us are storage and bandwidth, which are cheap.