# Push Notifications with Parse

**PART 1: Implementing Backend Server using Java Spring Framework**

**Author: Muhammad Shahab Hameed**

## Sources

Project homepage on Github - <https://github.com/shahabhameed/parse_push_notifications>

Download Zip – <https://github.com/shahabhameed/parse_push_notifications/archive/master.zip>

## Overview

This is a three-part tutorial series that will demonstrate how to send Parse’s Push Notifications from the backend server to the client devices (Android & IPhone). In the first part, a REST based backend server will be built with Java EE Spring4 and Jersey framework that will utilize Parse’s Rest API’s to send Push Notifications. In the second part, an Android client will be built using Android Studio IDE targeting Android Honeycomb (API SDK Level 11) and Marshmallow (API SDK Level 23) that will receive push notifications send by the backend server. Finally, in the third part, an IPhone client will be built using Xcode (7.0.1) for receiving push notifications.

## Requirements

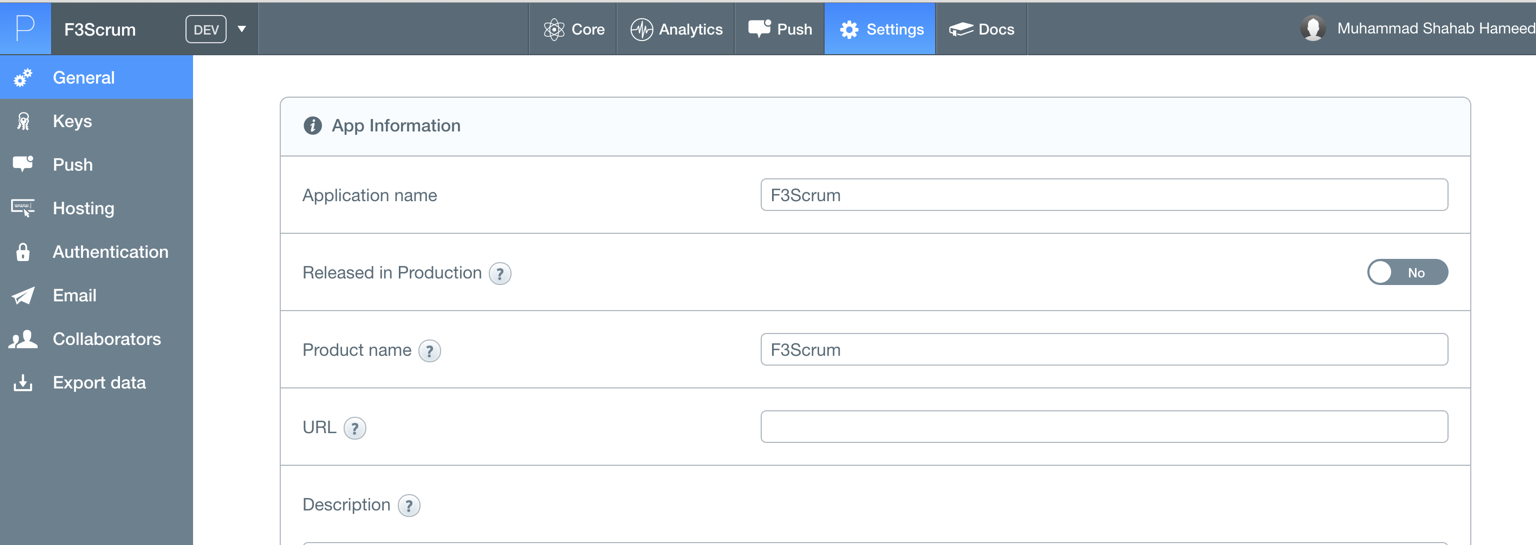
The following software have been used to implement this tutorial.

1. Java SDK 1.7+ (Version 8 update 65)
2. Eclipse Java EE IDE (Version Mars, Release 4.5.1)
3. Tomcat 8 Server (Version 8.028)
4. Postman REST API Client (Chrome Browser’s Extension)

## Implementing Push Notifications using Parse REST API

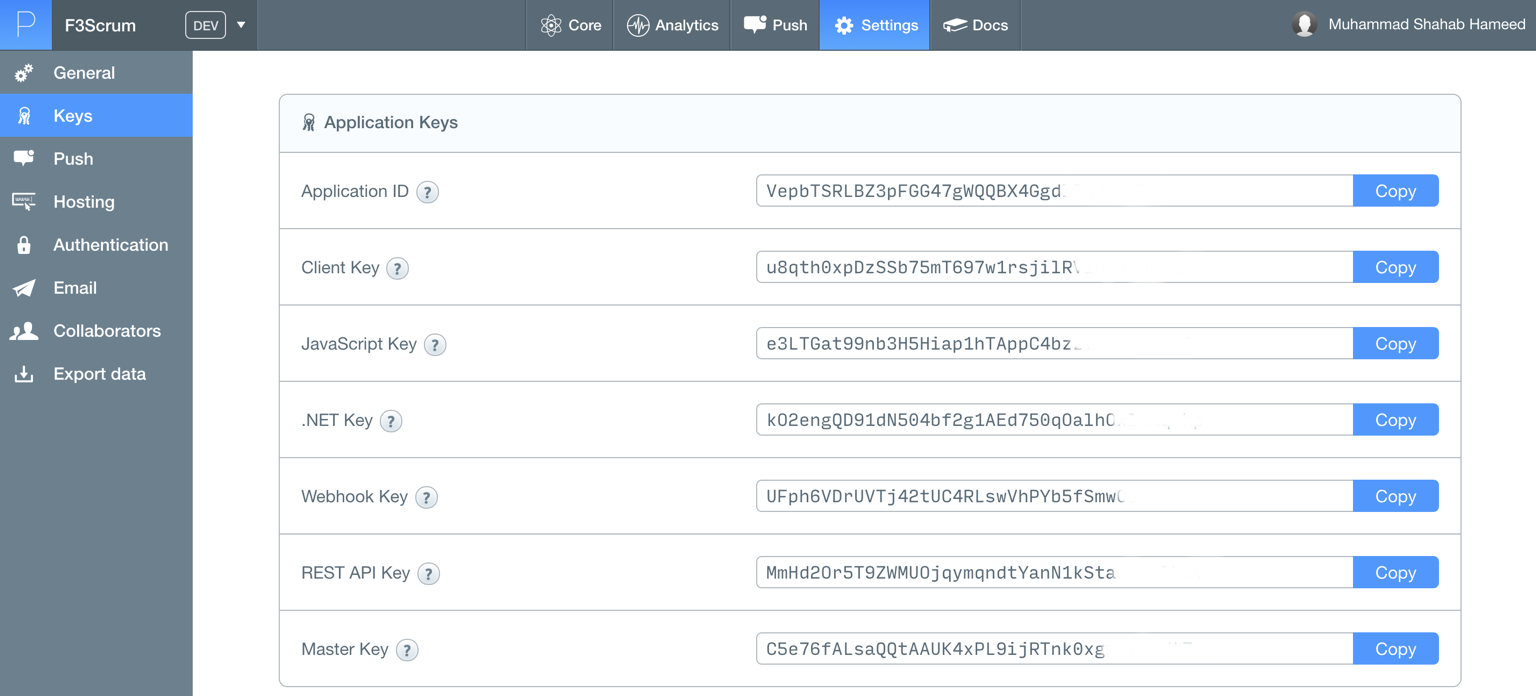
Step 1 Creating Application at Parse

First, we are going to register an application on Parse. For that navigate to [www.parse.com](http://www.parse.com) and create an account. I have signed up and created an application “F3Scrum” as shown below.



Once the application is created go to “Settings” -> “Keys” and note down the following keys that will be used later as configuration in our backend server implementation.

1. Application ID
2. Client Key
3. Rest API Key
4. Master Key

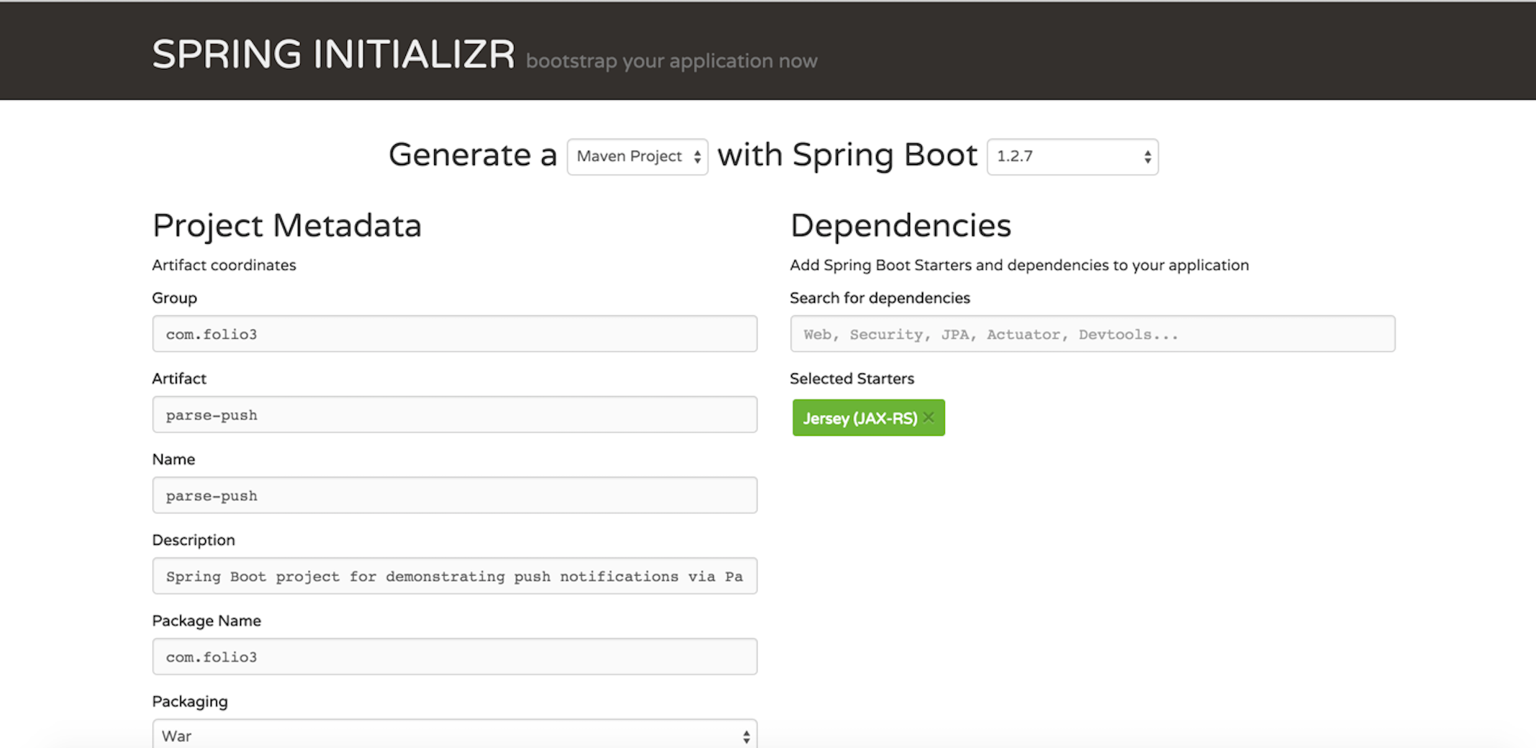
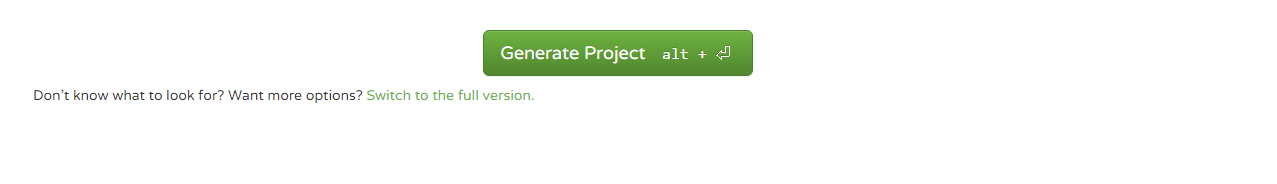


Step 2 Bootstrapping Spring Project

Next, we are going to bootstrap our JavaEE Spring application using Spring Boot. Spring Boot facilitates in creating a skeleton for our application using a build dependency system.

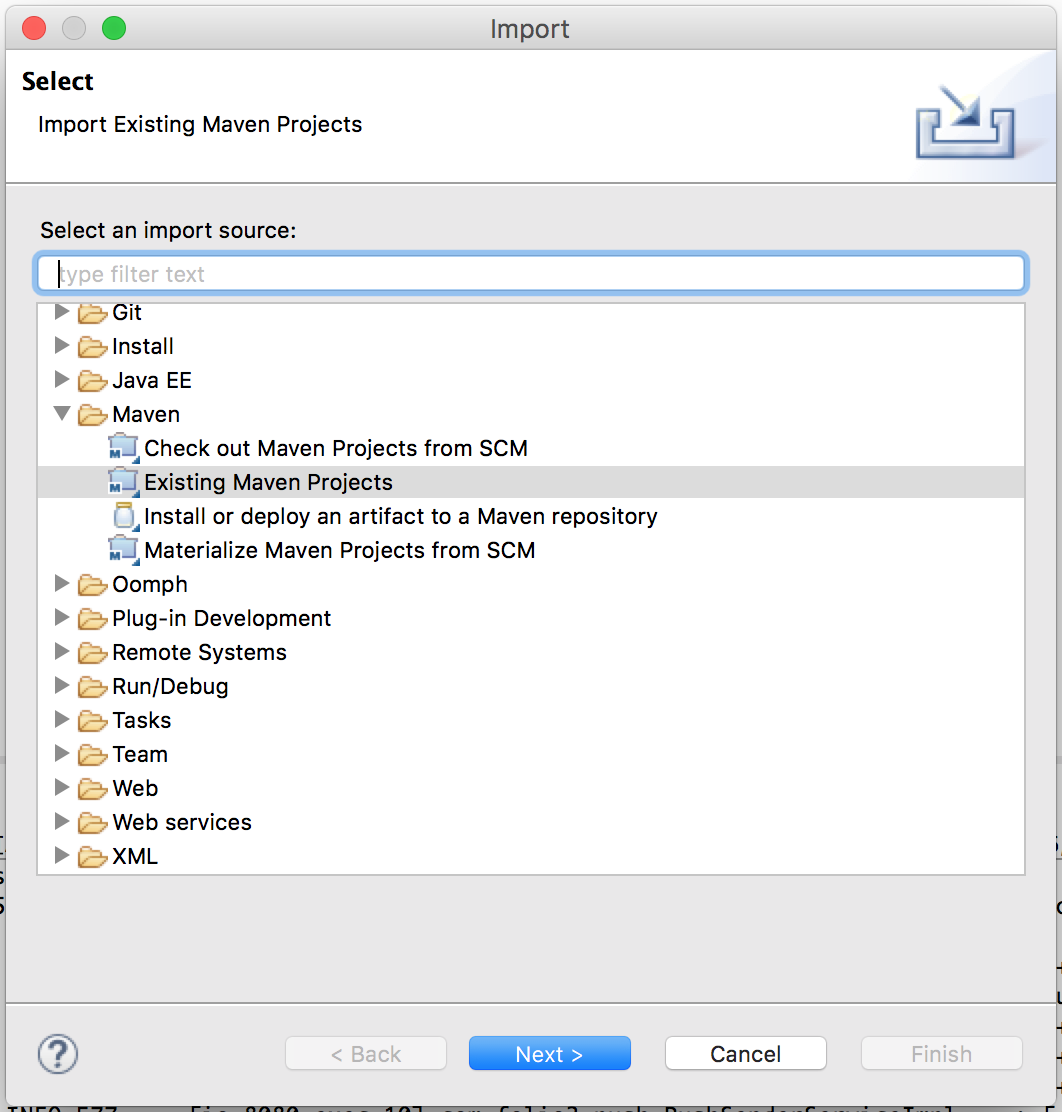
For this, we go to Spring Initializr website ([www.start.spring.io)](http://www.start.spring.io)) and configure our application. Since we are using Jersey REST framework for developing APIs we will select Jersey (JAX-RS) from the options provided. The packaging is done in web-archive (war) format so that it could be deployed on our Tomcat application server. Also we are going to select Maven as our build dependency system.

Once all the fields have been filled, we will click “Generate Project” and download it on our computer.

Step 3 Configuring and Downloading Maven dependencies

Once the project has been downloaded, we will extract and import it in Eclipse IDE. Click “File” -> “Import…” -> “Existing Maven Projects” and browse the extracted folder and download it. The maven will auto download all the dependencies required for building this project defined in the “pom.xml” file.



Step 4 Adding project specific dependencies

Next, we need to modify “pom.xml” file to include Google’s Gson library that maps Java class to JSON and IOUtilities package as one of the “**dependencies”** in our project.

<dependency>

<groupId>commons-io</groupId>

<artifactId>commons-io</artifactId>

<version>2.4</version>

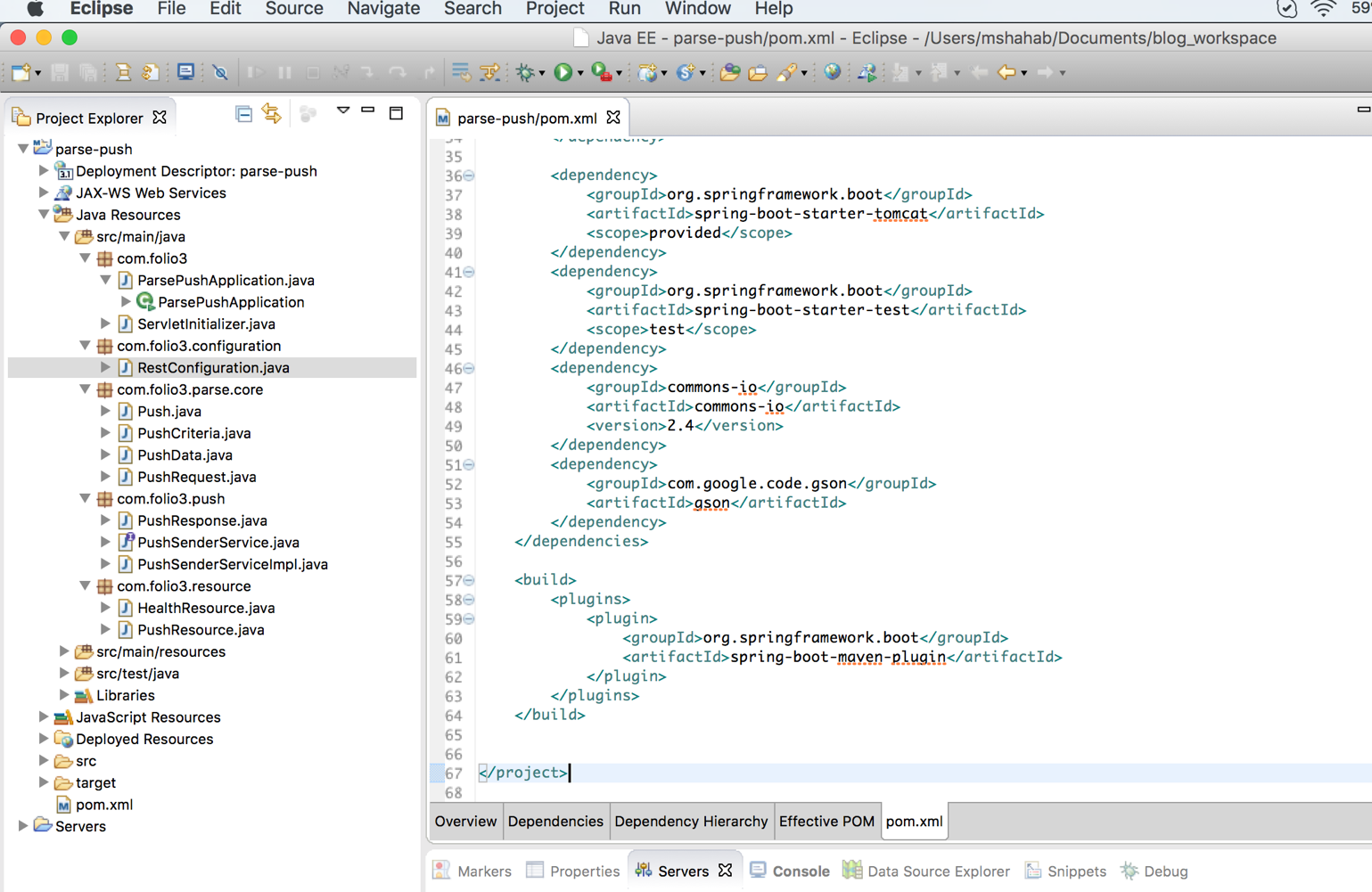
</dependency>

<dependency>

<groupId>com.google.code.gson</groupId>

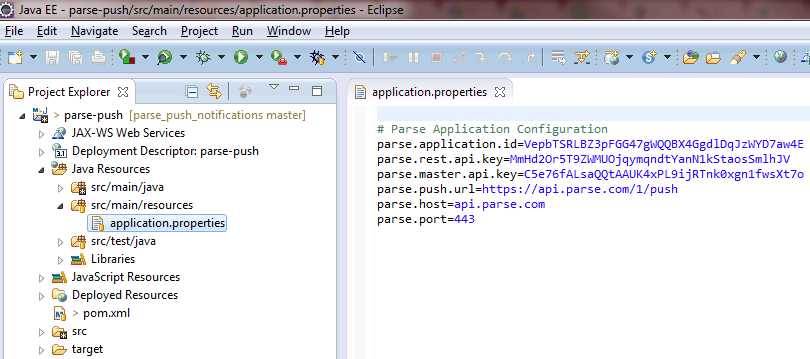
<artifactId>gson</artifactId>

</dependency>



Step 5 Creating properties file

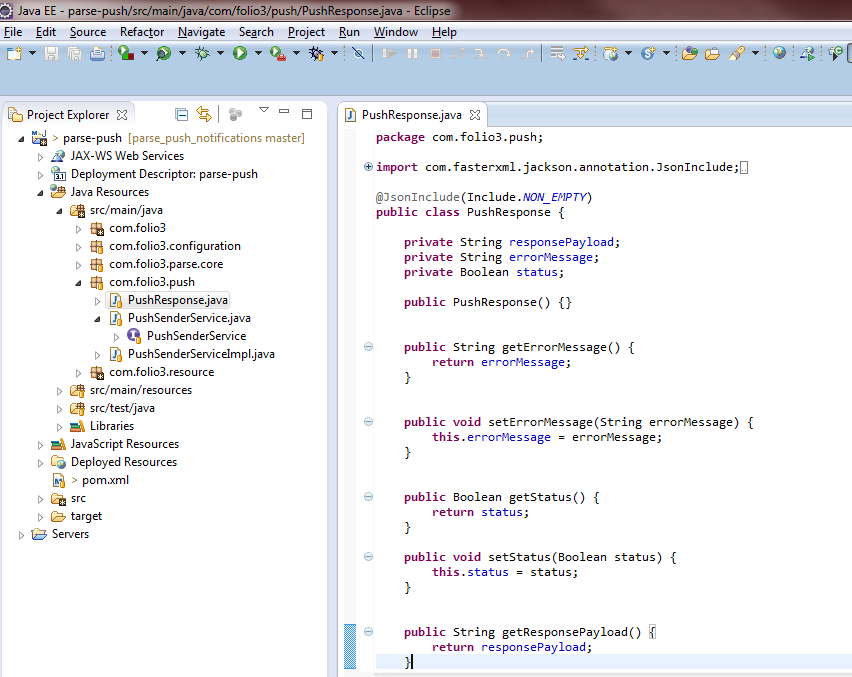
Next, we will define the credential keys of Parse Application in “application.properties” file in the resources folder so that it could be read in the service class.

**File: “application.properties”** 

Step 6 Creating Push Response Class

Next, we will create a “PushResponse” class that will hold the response error code, payload and status of the rest call to the Parse Server. The response will include only non-empty / non-null attributes.

**File: “PushResponse.java”**



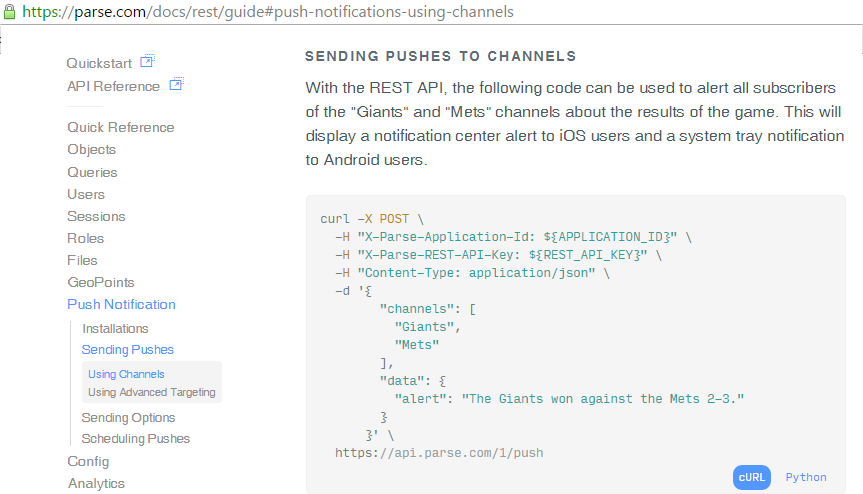
Step 7 Creating the Services

Next, we will create a **service layer** interface and implementation classes that will actually post REST API requests to the Parse Server for push notifications.

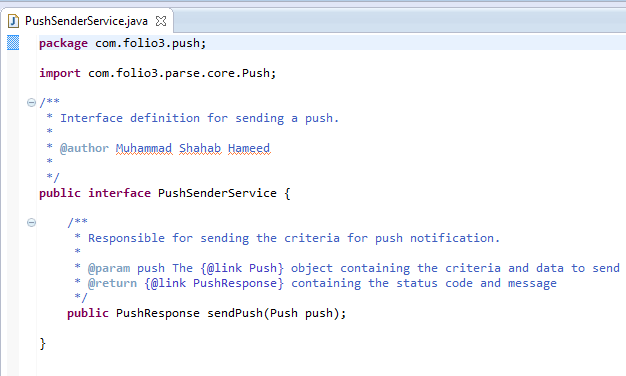
There are two ways to send push notifications to Parse.

1. Channels
2. Advanced Target Pushing

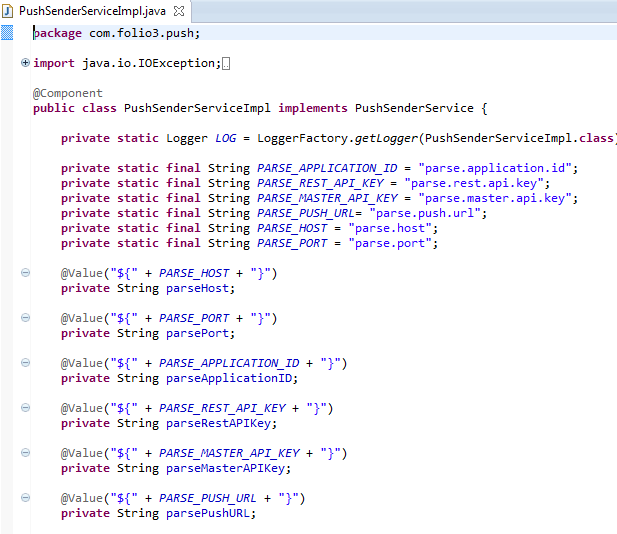
We will be sending Push Notifications via channels in this tutorial. The format for it defined in the Parse Rest API docs is as follows.



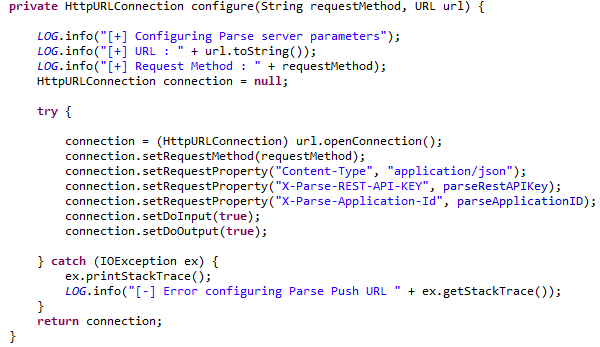
**File: “PushSenderService.java” (Interface)**



We will use annotation-based approach provided in the Spring4 framework to read the properties from application.properties file.

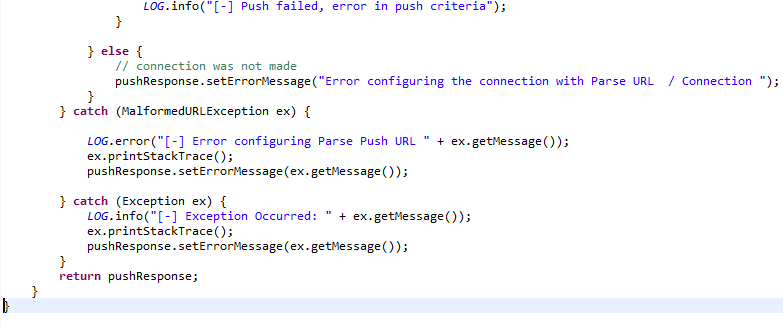


**Code Snippet for configuring connection to the server.**

****

The “sendPush” function will send the push notification criteria to Parse, based on which the Push Notifications will be sent by the Parse Server.

****

****

Step 8 Creating Push Request Classes

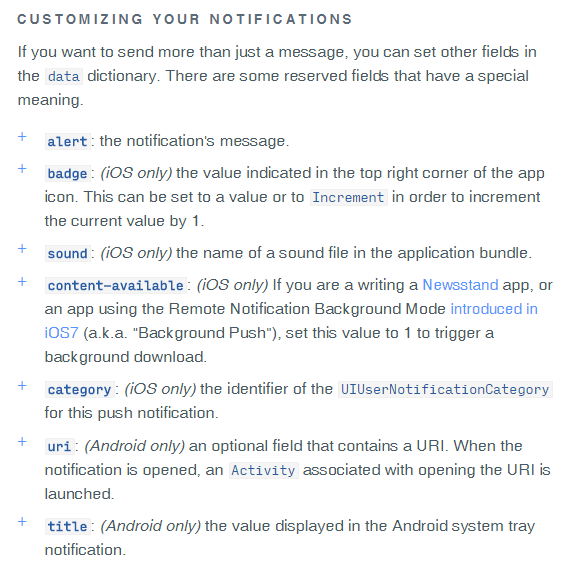
Next, we will create bean classes that will be converted to JSON by the Gson library to send requests to Parse Server.

1. **PushData.java**

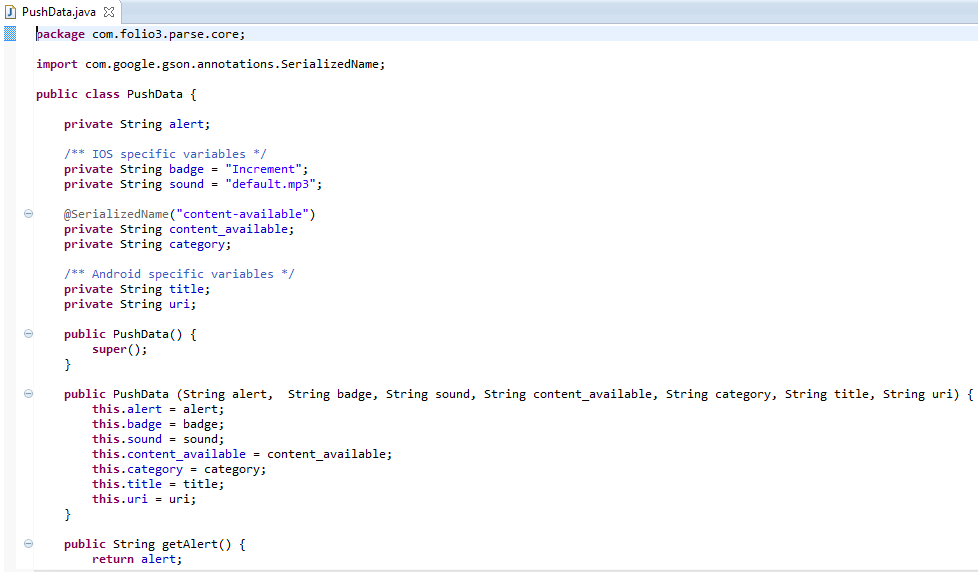
The PushData class contains attributes of the Parse “data” dictionary.

Parse Rest Documentation

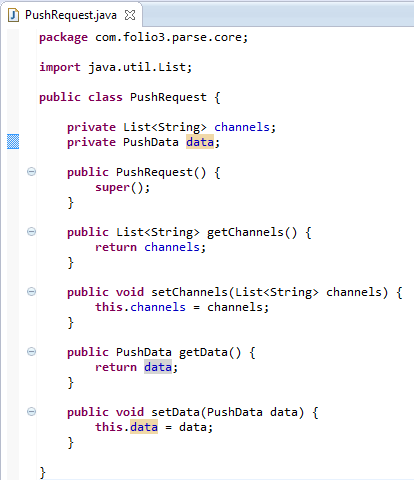
[(https://parse.com/docs/rest/guide#push-notifications-customizing-your-notifications)]((https:/parse.com/docs/rest/guide%23push-notifications-customizing-your-notifications))



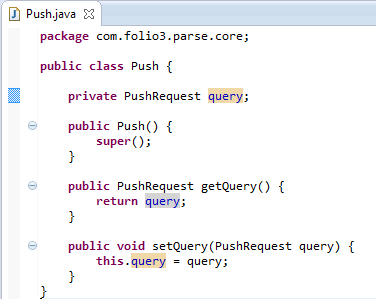
PushData bean class containing these attributes and their getter and setter methods that will be used in the PushRequest class.

****

1. **PushRequest.java**

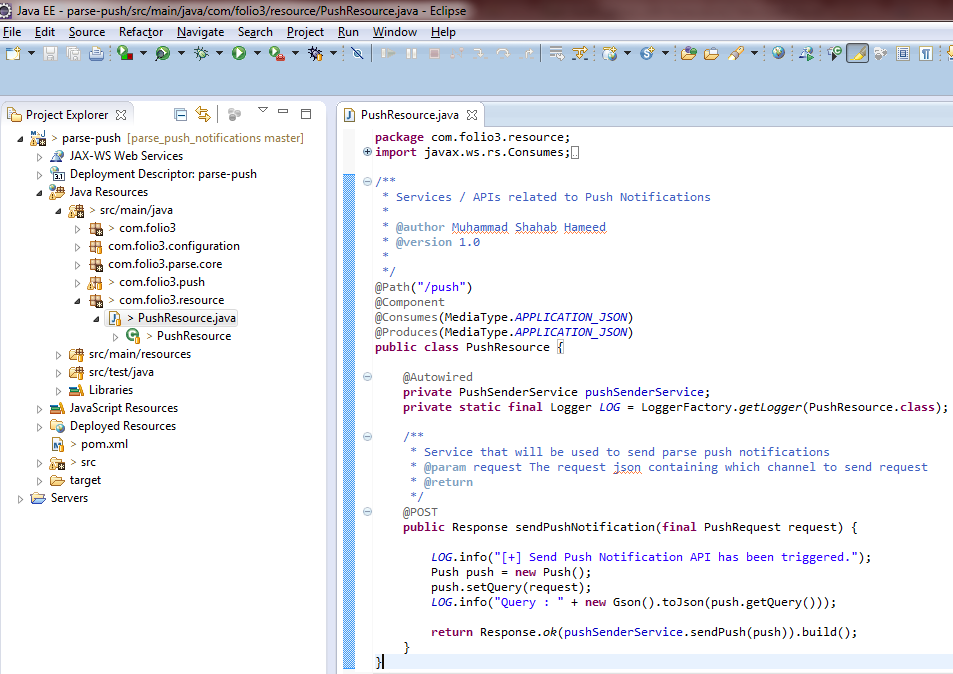
****

1. **Push.java**



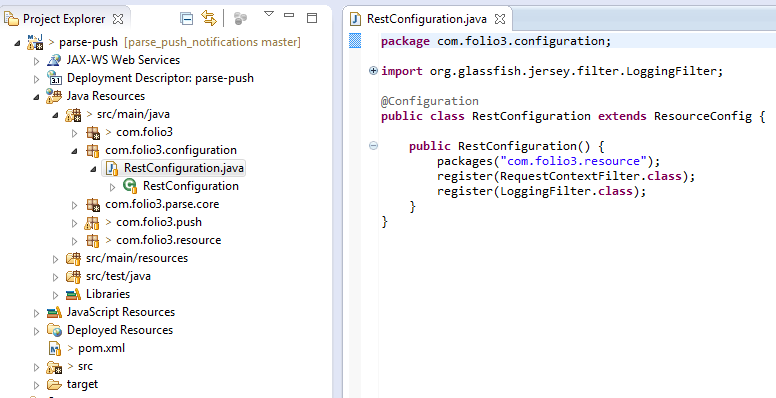
Step 9 Creating the Resource Endpoint

Now, we need to expose an endpoint that will be used to define a criteria based on which push notifications will be sent by the client. For this, we will create a PushResource class and create a service that will respond to HTTP POST requests using Jersey framework. It will accept JSON payload and will respond back in JSON format. The PushService will be autowired (i.e. a suitable implementation of PushService will be initialized and injected) by the Spring4 framework.

****

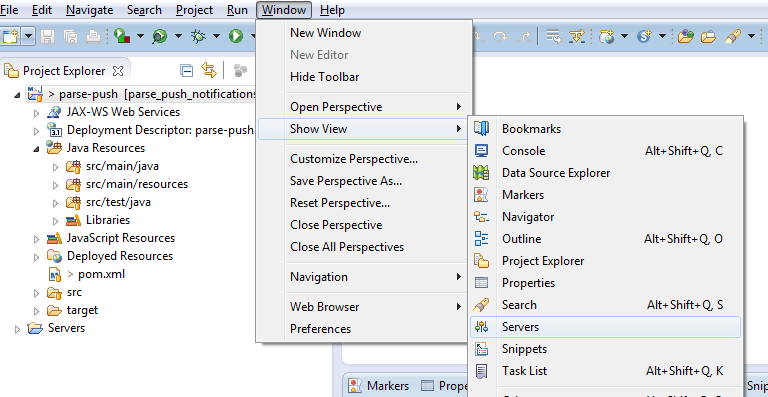
Step 10 Resource Configuration

Finally, we need to tell Spring-framework the package containing the resource layer to scan so that the appropriate Resource components are initialized. In addition, we will register LoggingFilter that will log request response payloads in the server logs and context filter to know the context of the requesting user.

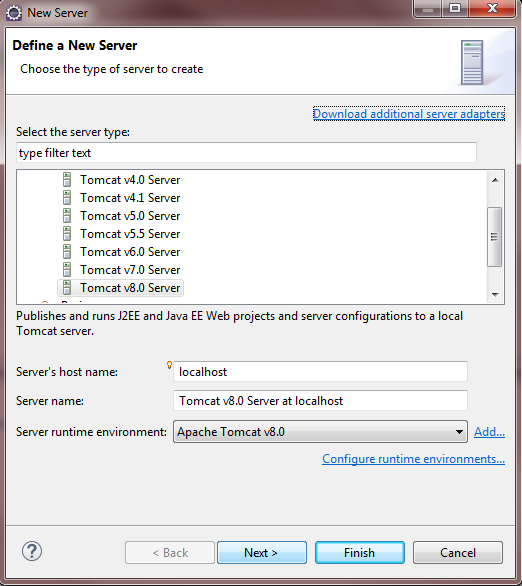


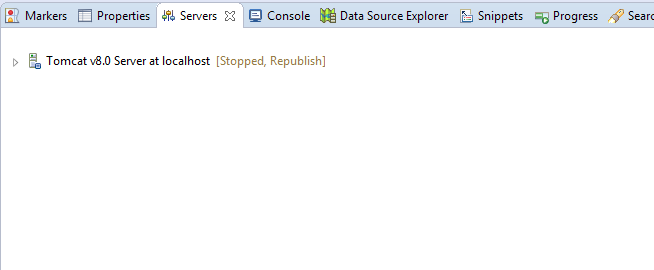
Step 11 Configuring Tomcat Server

To run this project, we need to install tomcat server and deploy our application on it. For that, In Eclipse, open “Servers” Tab by going to “Windows”->”Show View” -> “Servers”. The default port of tomcat server is 8080.



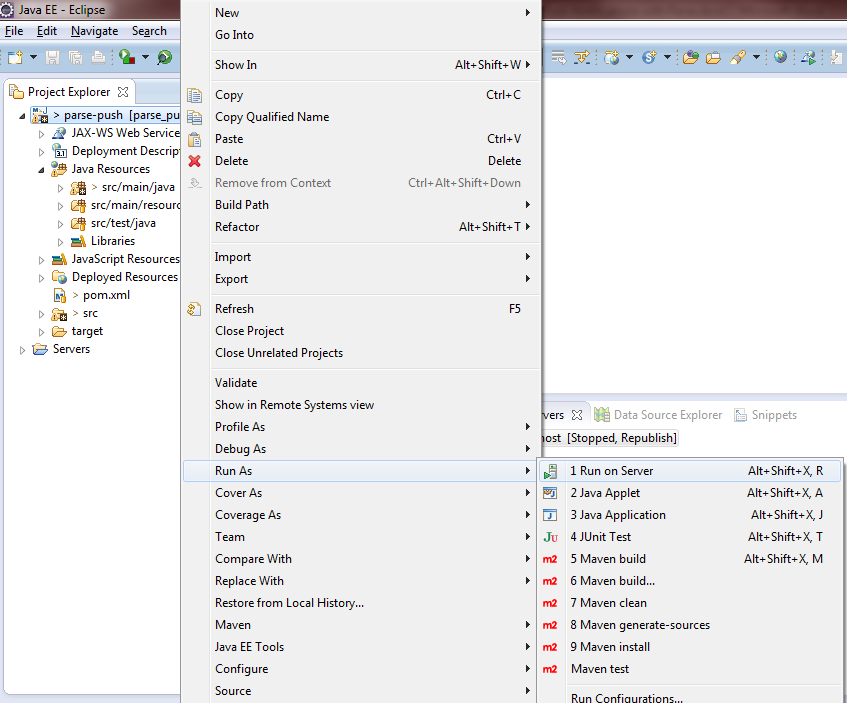
In the “Servers tab”, click to create a new “Tomcat 8” server and browse to the extracted Tomcat directory to integrate the tomcat server with Eclipse IDE. The Tomcat8 server can be downloaded from <https://tomcat.apache.org/download-80.cgi> . The zip file containing the core binary distribution is located at “<http://www.us.apache.org/dist/tomcat/tomcat-8/v8.0.28/bin/apache-tomcat-8.0.28.zip>”.

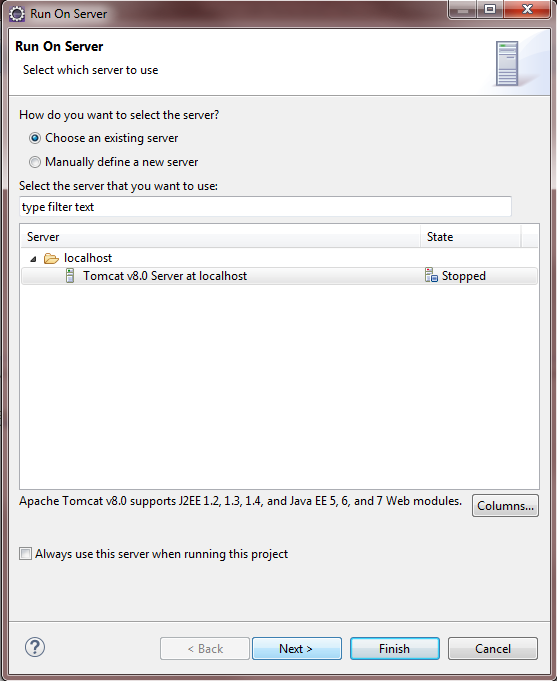


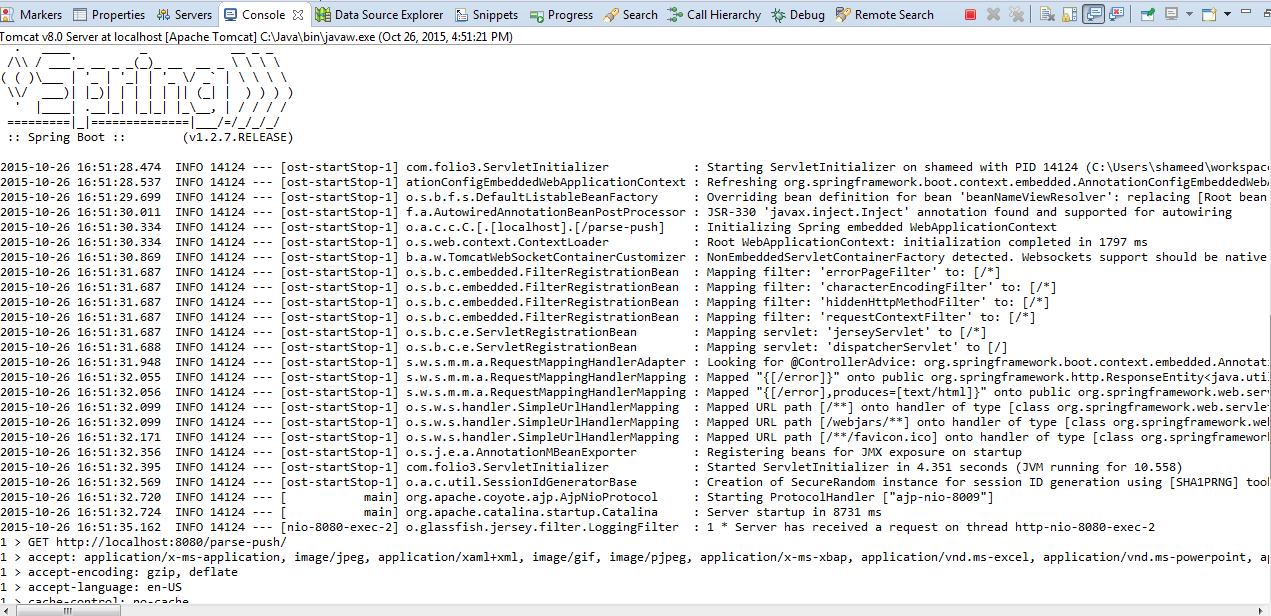


Step 12 Deploying the Application

Right click on the project -> Run As -> Run On Server, and deploy this project on the server. The following are the screenshots to deploy it on our server.

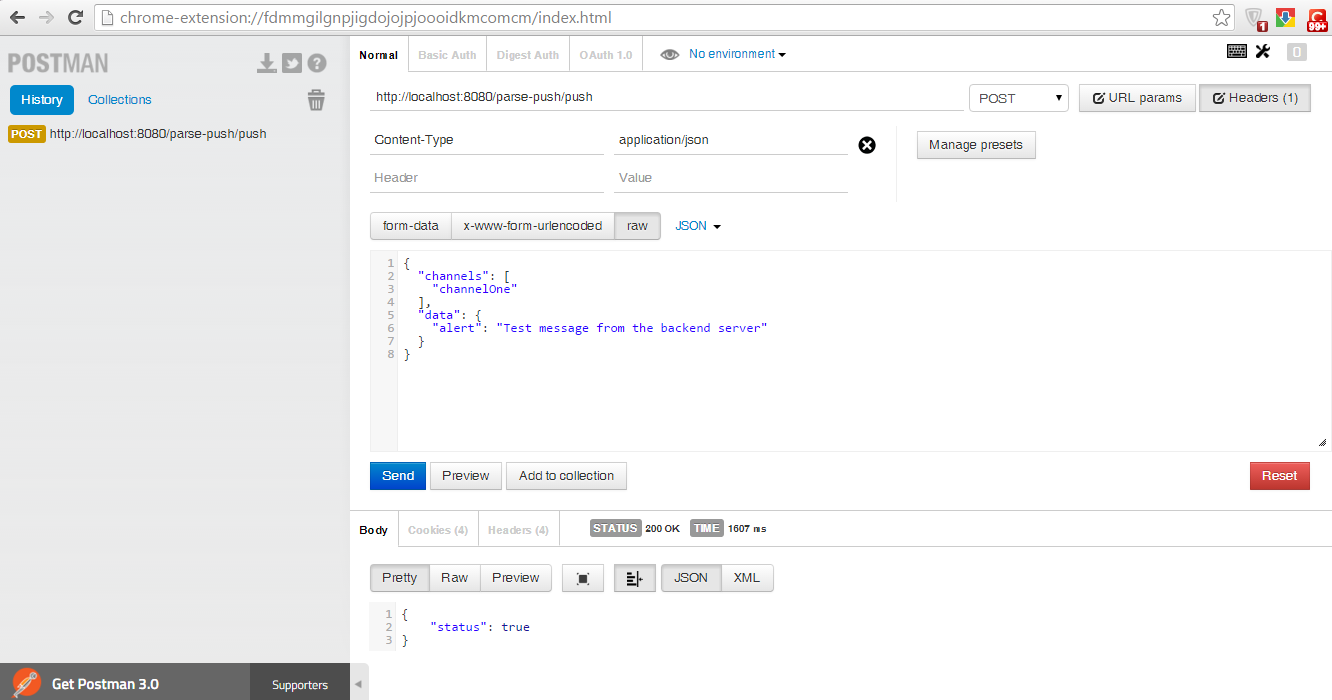




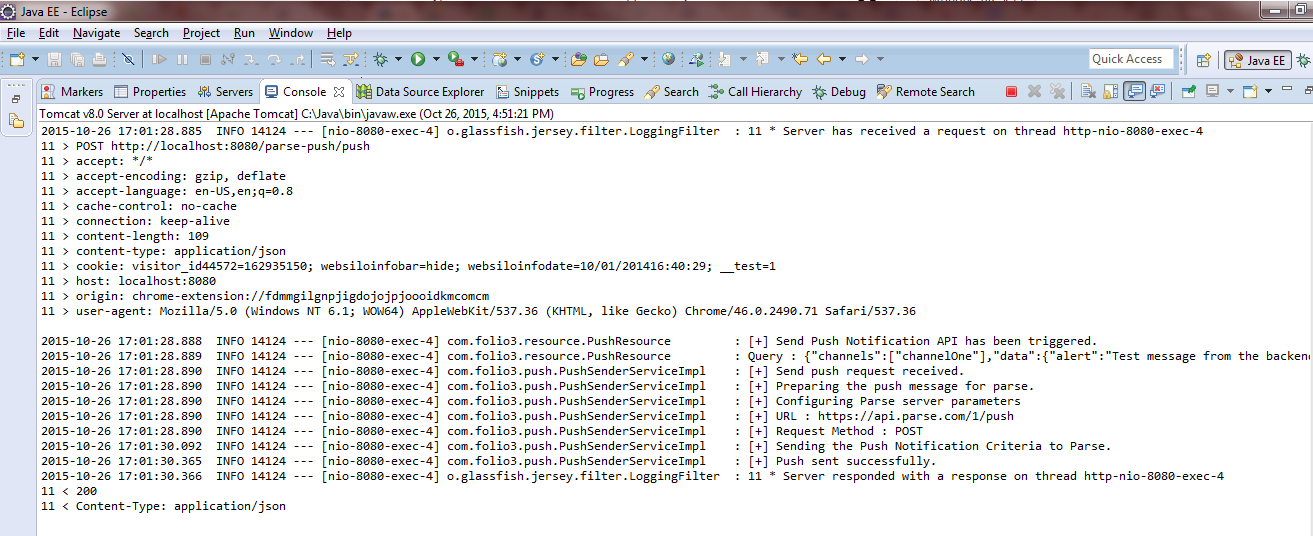


Step 13 Testing API with Postman

Postman is a chrome browser’s extension that can be used to test REST API Services. Open it up and create a push request. The path is : [**http://localhost:8080/parse-push/push**](http://localhost:8080/parse-push/push)



We can see the logs generated by our server in the Console window as well in Eclipse.



Summary

In this tutorial, we created implemented a backend server using Java Spring and Jersey framework that sends Push notifications to our clients. We configured and deployed our application on the Tomcat servlet container and tested our service using the Postman client. In the second and third parts, we will create an Android and IPhone application to receive push notifications.