1. **Historial de cambios**

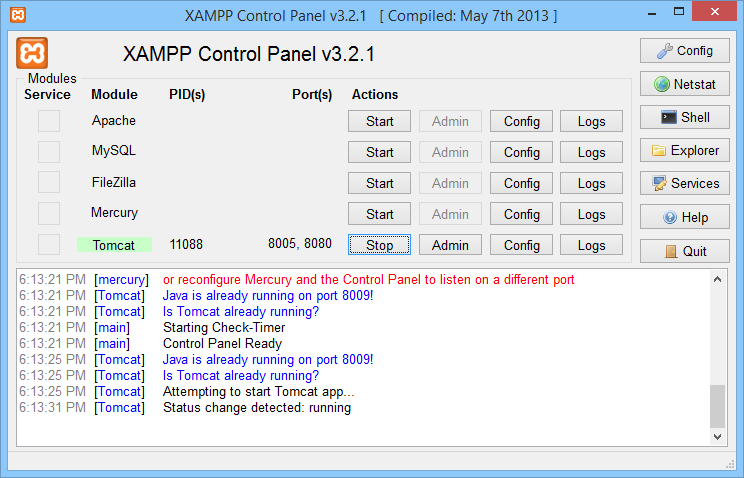
| **Fecha** | **Versión** | **Autor** | **Descripción del Cambio** |
| --- | --- | --- | --- |
| 13/05/2015 | 1.0 | Julián Andrés Henao Taborda | Creación del documento |

1. **Introducción**

El objetivo del tutoriales el desarrollo de una aplicación servidor en Java para hacer uso de Extensible Messaging and Presence Protocol*.*

Después de seguir este tutorial se obtendrá una aplicación que recibirá/enviará mensajes a dispositivos móviles previamente registrados. (Ver MANUAL DEMO XMPP APP ANDROID)

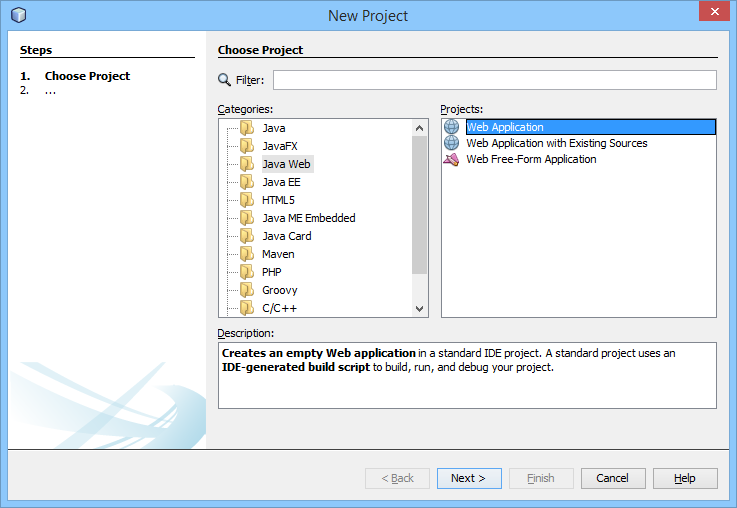
1. Descargar XAMPP
2. En el panel de control inicializar el servicio del servidor Tomcat.

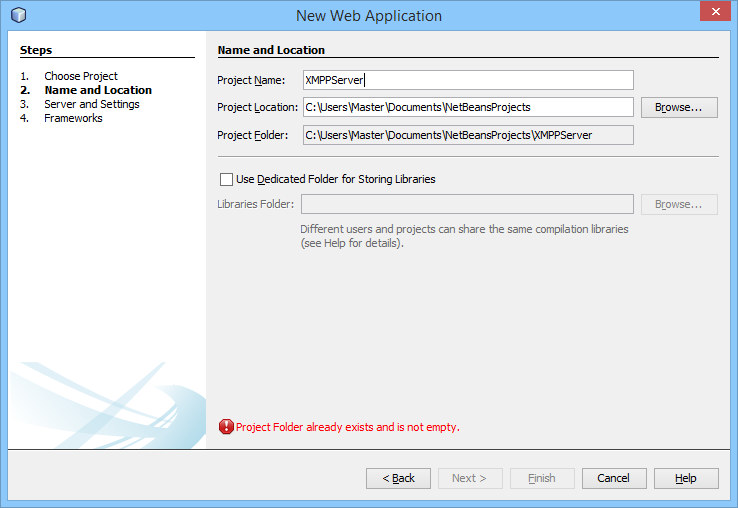


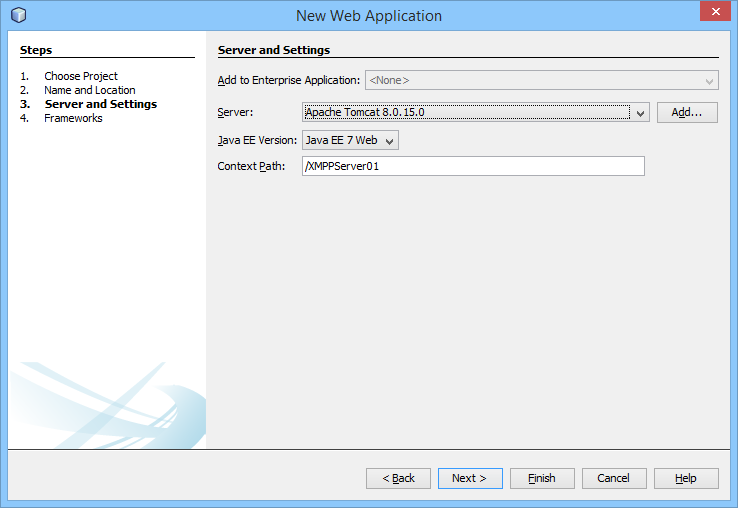
1. Descargar NetBeans



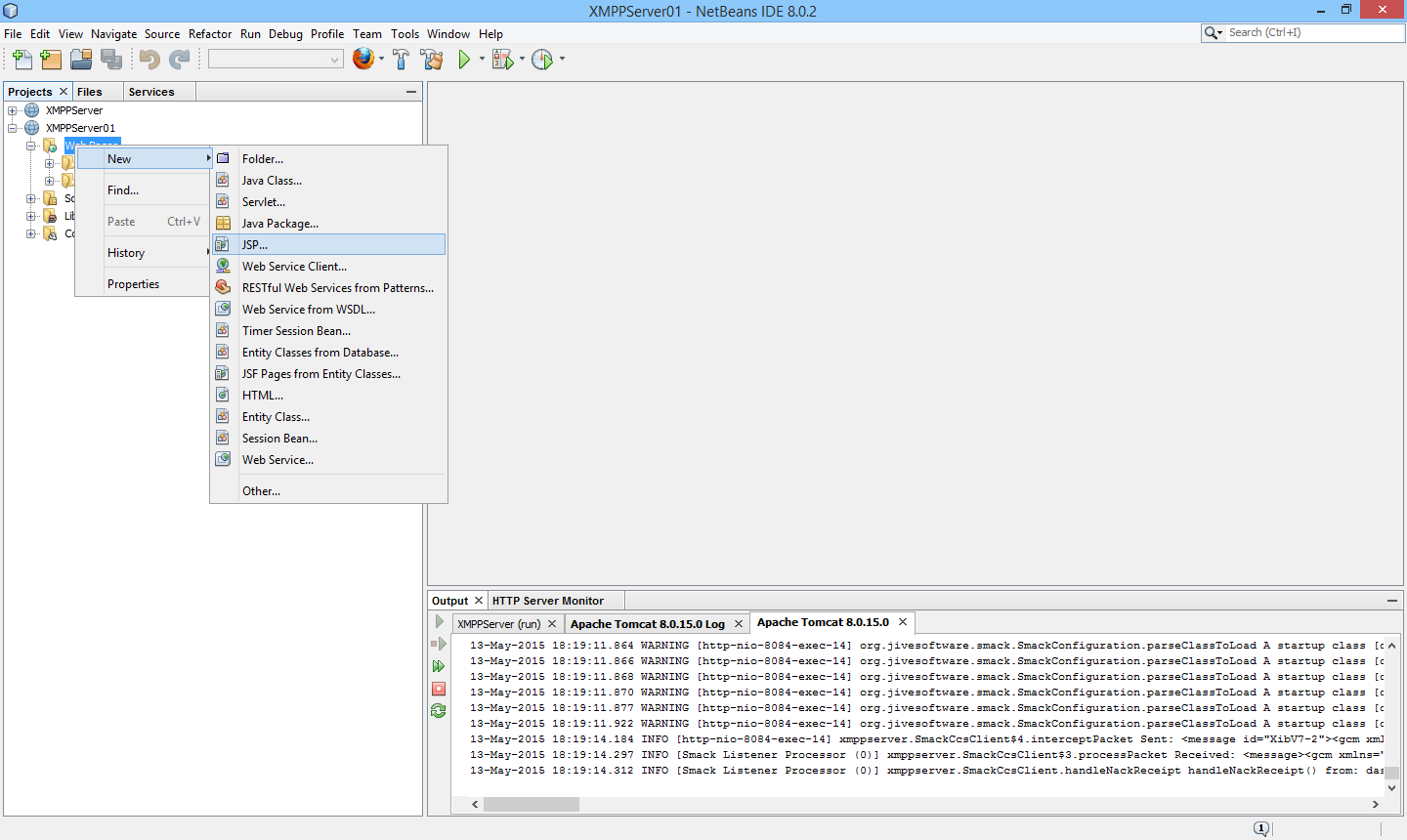
1. Crear un nuevo proyecto web

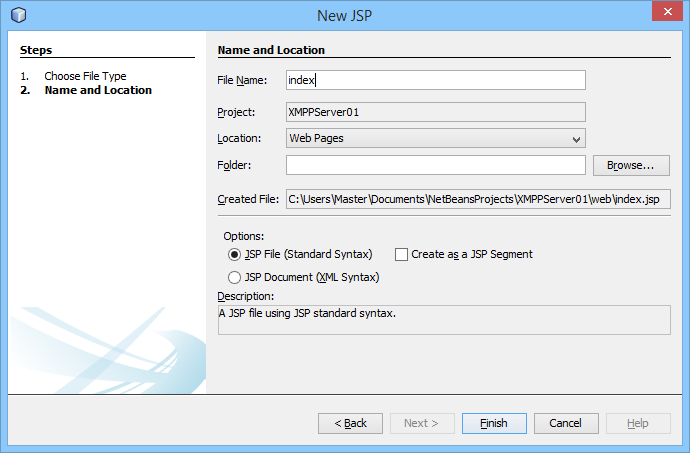






1. En el directorio “Web Pages” crear un nuevo JSP:





1. Reemplazar el código por el siguiente:

**En este archivo se define la interfaz gráfica de la aplicación.**

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<%

String pushStatus = "";

Object pushStatusObj = request.getAttribute("pushStatus");

if (pushStatusObj != null) {

pushStatus = pushStatusObj.toString();

}

%>

<head>

<title>XMPP Demo UAM</title>

</head>

<body>

<h1>XMPP Demo UAM</h1>

<form action="GCMNotification" method="post">

<div>

<textarea rows="2" name="message" cols="23"

placeholder="Mensaje a enviar al dispositivo"></textarea>

</div>

<br />

<div>

<textarea rows="2" name="regid" cols="23"

placeholder="Registration ID del dispositivo"></textarea>

</div>

<br />

<div>

<input type="submit" value="Enviar" />

</div>

</form>

<p>

<h3>

<%=pushStatus%>

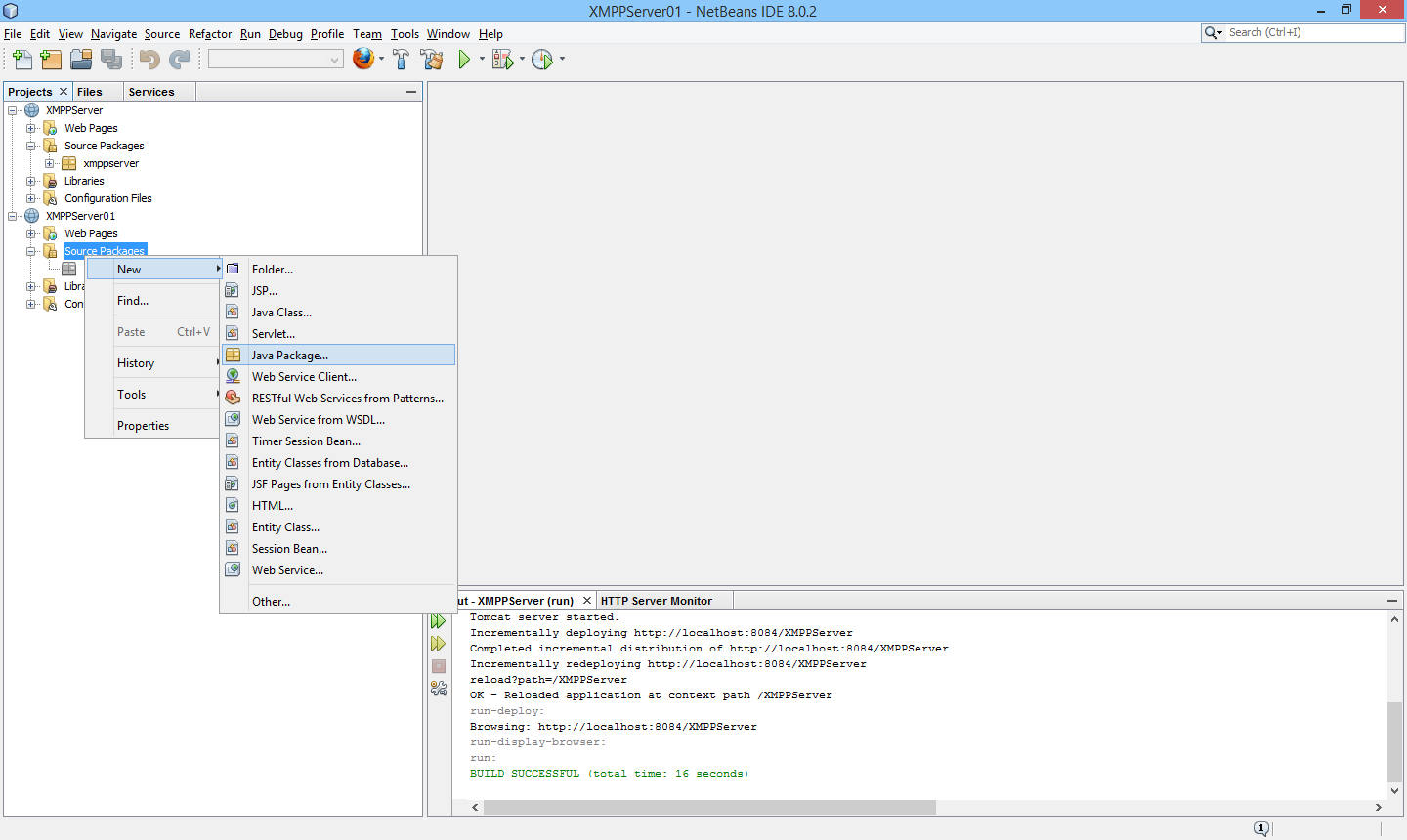
</h3>

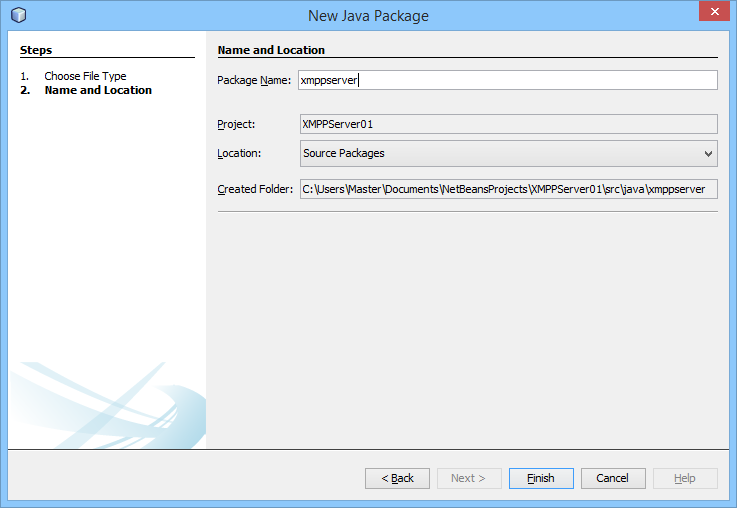
</p>

</body>

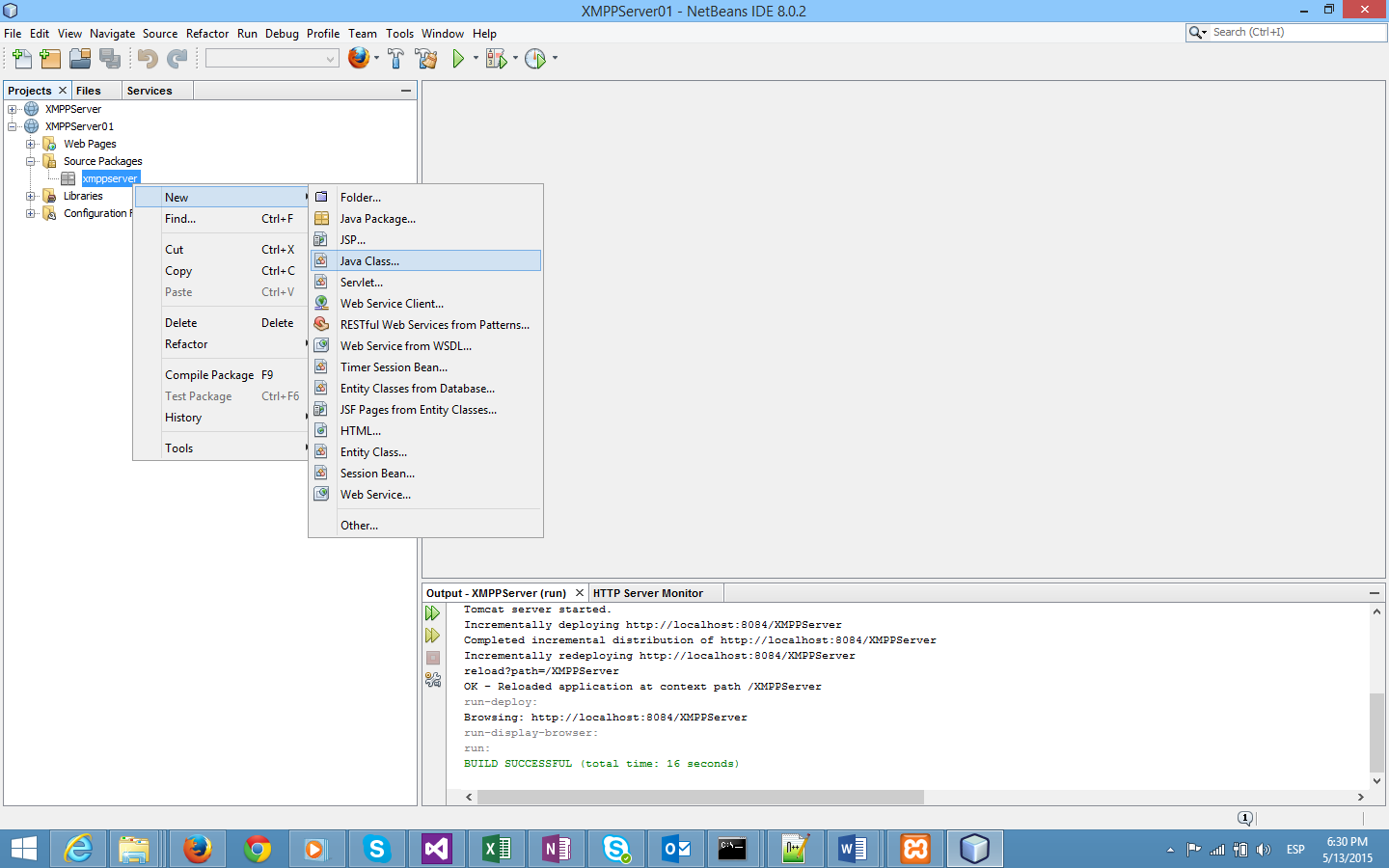
</html>

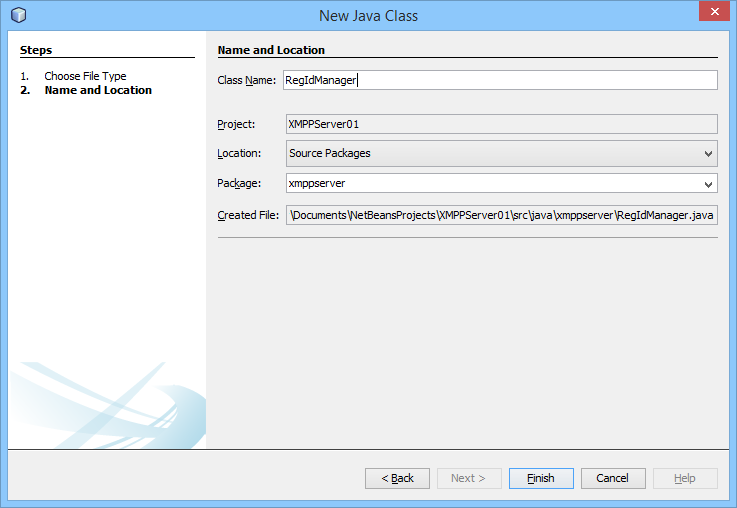
1. En el directorio “Source Packages” crear un nuevo package llamado “xmppserver”:





1. Adicionar una nueva clase en el package recientemente creado llamada “RegIdManager”:





1. Reemplazar el código por el siguiente:

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package xmppserver;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.io.PrintWriter;

import java.util.HashSet;

import java.util.Set;

public class RegIdManager {

static final String REG\_ID\_STORE = "GCMRegId.txt";

public static void writeToFile(String regId) throws IOException {

Set<String> regIdSet = readFromFile();

if (!regIdSet.contains(regId)) {

PrintWriter out = new PrintWriter(new BufferedWriter(

new FileWriter(REG\_ID\_STORE, true)));

out.println(regId);

out.close();

}

}

public static Set<String> readFromFile() throws IOException {

BufferedReader br = new BufferedReader(new FileReader(REG\_ID\_STORE));

String regId = "";

Set<String> regIdSet = new HashSet<String>();

while ((regId = br.readLine()) != null) {

regIdSet.add(regId);

}

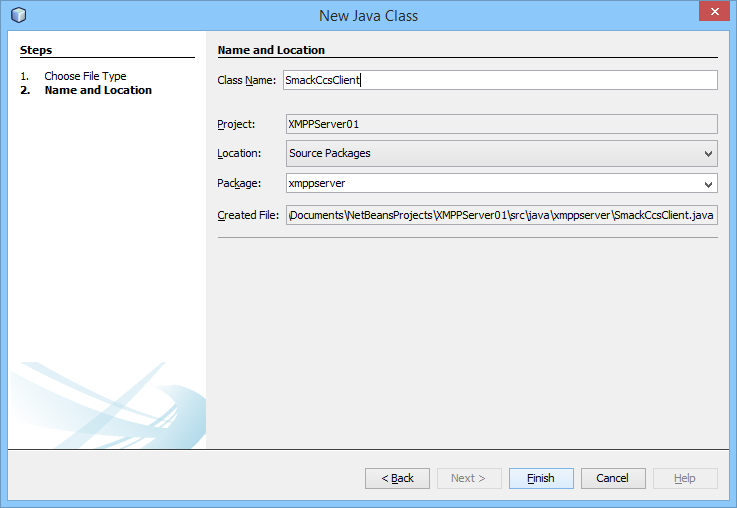
br.close();

return regIdSet;

}

}

1. Adicionar una nueva clase en el package recientemente creado llamada “SmackCcsClient”:



1. Reemplazar el código por el siguiente:

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package xmppserver;

import org.jivesoftware.smack.ConnectionConfiguration;

import org.jivesoftware.smack.ConnectionConfiguration.SecurityMode;

import org.jivesoftware.smack.ConnectionListener;

import org.jivesoftware.smack.PacketInterceptor;

import org.jivesoftware.smack.PacketListener;

import org.jivesoftware.smack.XMPPConnection;

import org.jivesoftware.smack.XMPPException;

import org.jivesoftware.smack.filter.PacketTypeFilter;

import org.jivesoftware.smack.packet.DefaultPacketExtension;

import org.jivesoftware.smack.packet.Message;

import org.jivesoftware.smack.packet.Packet;

import org.jivesoftware.smack.packet.PacketExtension;

import org.jivesoftware.smack.provider.PacketExtensionProvider;

import org.jivesoftware.smack.provider.ProviderManager;

import org.jivesoftware.smack.util.StringUtils;

import org.json.simple.JSONValue;

import org.json.simple.parser.ParseException;

import org.xmlpull.v1.XmlPullParser;

import java.io.IOException;

import java.util.HashMap;

import java.util.Map;

import java.util.Random;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.net.ssl.SSLSocketFactory;

/\*\*

\* Sample Smack implementation of a client for GCM Cloud Connection Server.

\*

\* For illustration purposes only.

\*/

public class SmackCcsClient {

static final String MESSAGE\_KEY = "SERVER\_MESSAGE";

Logger logger = Logger.getLogger("SmackCcsClient");

public static final String GCM\_SERVER = "gcm.googleapis.com";

public static final int GCM\_PORT = 5235;

public static final String GCM\_ELEMENT\_NAME = "gcm";

public static final String GCM\_NAMESPACE = "google:mobile:data";

static Random random = new Random();

XMPPConnection connection;

ConnectionConfiguration config;

/\*\*

\* XMPP Packet Extension for GCM Cloud Connection Server.

\*/

class GcmPacketExtension extends DefaultPacketExtension {

String json;

public GcmPacketExtension(String json) {

super(GCM\_ELEMENT\_NAME, GCM\_NAMESPACE);

this.json = json;

}

public String getJson() {

return json;

}

@Override

public String toXML() {

return String.format("<%s xmlns=\"%s\">%s</%s>", GCM\_ELEMENT\_NAME,

GCM\_NAMESPACE, json, GCM\_ELEMENT\_NAME);

}

@SuppressWarnings("unused")

public Packet toPacket() {

return new Message() {

// Must override toXML() because it includes a <body>

@Override

public String toXML() {

StringBuilder buf = new StringBuilder();

buf.append("<message");

if (getXmlns() != null) {

buf.append(" xmlns=\"").append(getXmlns()).append("\"");

}

if (getLanguage() != null) {

buf.append(" xml:lang=\"").append(getLanguage())

.append("\"");

}

if (getPacketID() != null) {

buf.append(" id=\"").append(getPacketID()).append("\"");

}

if (getTo() != null) {

buf.append(" to=\"")

.append(StringUtils.escapeForXML(getTo()))

.append("\"");

}

if (getFrom() != null) {

buf.append(" from=\"")

.append(StringUtils.escapeForXML(getFrom()))

.append("\"");

}

buf.append(">");

buf.append(GcmPacketExtension.this.toXML());

buf.append("</message>");

return buf.toString();

}

};

}

}

public SmackCcsClient() {

// Add GcmPacketExtension

ProviderManager.getInstance().addExtensionProvider(GCM\_ELEMENT\_NAME,

GCM\_NAMESPACE, new PacketExtensionProvider() {

@Override

public PacketExtension parseExtension(XmlPullParser parser)

throws Exception {

String json = parser.nextText();

GcmPacketExtension packet = new GcmPacketExtension(json);

return packet;

}

});

}

/\*\*

\* Returns a random message id to uniquely identify a message.

\*

\* <p>

\* Note: This is generated by a pseudo random number generator for

\* illustration purpose, and is not guaranteed to be unique.

\*

\*/

public String getRandomMessageId() {

return "m-" + Long.toString(random.nextLong());

}

/\*\*

\* Sends a downstream GCM message.

\*/

public void send(String jsonRequest) {

Packet request = new GcmPacketExtension(jsonRequest).toPacket();

connection.sendPacket(request);

}

/\*\*

\* Handles an upstream data message from a device application.

\*

\* <p>

\* This sample echo server sends an echo message back to the device.

\* Subclasses should override this method to process an upstream message.

\*/

public void handleIncomingDataMessage(Map<String, Object> jsonObject) {

String from = jsonObject.get("from").toString();

// PackageName of the application that sent this message.

String category = jsonObject.get("category").toString();

// Use the packageName as the collapseKey in the echo packet

String collapseKey = "echo:CollapseKey";

@SuppressWarnings("unchecked")

Map<String, String> payload = (Map<String, String>) jsonObject

.get("data");

String action = payload.get("ACTION");

if ("ECHO".equals(action)) {

String clientMessage = payload.get("CLIENT\_MESSAGE");

payload.put(MESSAGE\_KEY, "ECHO: " + clientMessage);

// Send an ECHO response back

String echo = createJsonMessage(from, getRandomMessageId(),

payload, collapseKey, null, false);

send(echo);

} else if ("REGISTER".equals(action)) {

try {

RegIdManager.writeToFile(from);

} catch (IOException e) {

e.printStackTrace();

}

}

}

/\*\*

\* Handles an ACK.

\*

\* <p>

\* By default, it only logs a INFO message, but subclasses could override it

\* to properly handle ACKS.

\*/

public void handleAckReceipt(Map<String, Object> jsonObject) {

String messageId = jsonObject.get("message\_id").toString();

String from = jsonObject.get("from").toString();

logger.log(Level.INFO, "handleAckReceipt() from: " + from

+ ", messageId: " + messageId);

}

/\*\*

\* Handles a NACK.

\*

\* <p>

\* By default, it only logs a INFO message, but subclasses could override it

\* to properly handle NACKS.

\*/

public void handleNackReceipt(Map<String, Object> jsonObject) {

String messageId = jsonObject.get("message\_id").toString();

String from = jsonObject.get("from").toString();

logger.log(Level.INFO, "handleNackReceipt() from: " + from

+ ", messageId: " + messageId);

}

/\*\*

\* Creates a JSON encoded GCM message.

\*

\* @param to

\* RegistrationId of the target device (Required).

\* @param messageId

\* Unique messageId for which CCS will send an "ack/nack"

\* (Required).

\* @param payload

\* Message content intended for the application. (Optional).

\* @param collapseKey

\* GCM collapse\_key parameter (Optional).

\* @param timeToLive

\* GCM time\_to\_live parameter (Optional).

\* @param delayWhileIdle

\* GCM delay\_while\_idle parameter (Optional).

\* @return JSON encoded GCM message.

\*/

public static String createJsonMessage(String to, String messageId,

Map<String, String> payload, String collapseKey, Long timeToLive,

Boolean delayWhileIdle) {

Map<String, Object> message = new HashMap<String, Object>();

message.put("to", to);

if (collapseKey != null) {

message.put("collapse\_key", collapseKey);

}

if (timeToLive != null) {

message.put("time\_to\_live", timeToLive);

}

if (delayWhileIdle != null && delayWhileIdle) {

message.put("delay\_while\_idle", true);

}

message.put("message\_id", messageId);

message.put("data", payload);

return JSONValue.toJSONString(message);

}

/\*\*

\* Creates a JSON encoded ACK message for an upstream message received from

\* an application.

\*

\* @param to

\* RegistrationId of the device who sent the upstream message.

\* @param messageId

\* messageId of the upstream message to be acknowledged to CCS.

\* @return JSON encoded ack.

\*/

public static String createJsonAck(String to, String messageId) {

Map<String, Object> message = new HashMap<String, Object>();

message.put("message\_type", "ack");

message.put("to", to);

message.put("message\_id", messageId);

return JSONValue.toJSONString(message);

}

/\*\*

\* Connects to GCM Cloud Connection Server using the supplied credentials.

\*

\* @param username

\* GCM\_SENDER\_ID@gcm.googleapis.com

\* @param password

\* API Key

\* @throws XMPPException

\*/

public void connect(String username, String password) throws XMPPException {

config = new ConnectionConfiguration(GCM\_SERVER, GCM\_PORT);

config.setSecurityMode(SecurityMode.enabled);

config.setReconnectionAllowed(true);

config.setRosterLoadedAtLogin(false);

config.setSendPresence(false);

config.setSocketFactory(SSLSocketFactory.getDefault());

// NOTE: Set to true to launch a window with information about packets

// sent and received

config.setDebuggerEnabled(true);

// -Dsmack.debugEnabled=true

XMPPConnection.DEBUG\_ENABLED = true;

connection = new XMPPConnection(config);

connection.connect();

connection.addConnectionListener(new ConnectionListener() {

@Override

public void reconnectionSuccessful() {

logger.info("Reconnecting..");

}

@Override

public void reconnectionFailed(Exception e) {

logger.log(Level.INFO, "Reconnection failed.. ", e);

}

@Override

public void reconnectingIn(int seconds) {

logger.log(Level.INFO, "Reconnecting in %d secs", seconds);

}

@Override

public void connectionClosedOnError(Exception e) {

logger.log(Level.INFO, "Connection closed on error.");

}

@Override

public void connectionClosed() {

logger.info("Connection closed.");

}

});

// Handle incoming packets

connection.addPacketListener(new PacketListener() {

@Override

public void processPacket(Packet packet) {

logger.log(Level.INFO, "Received: " + packet.toXML());

Message incomingMessage = (Message) packet;

GcmPacketExtension gcmPacket = (GcmPacketExtension) incomingMessage

.getExtension(GCM\_NAMESPACE);

String json = gcmPacket.getJson();

try {

@SuppressWarnings("unchecked")

Map<String, Object> jsonObject = (Map<String, Object>) JSONValue

.parseWithException(json);

// present for "ack"/"nack", null otherwise

Object messageType = jsonObject.get("message\_type");

if (messageType == null) {

// Normal upstream data message

handleIncomingDataMessage(jsonObject);

// Send ACK to CCS

String messageId = jsonObject.get("message\_id")

.toString();

String from = jsonObject.get("from").toString();

String ack = createJsonAck(from, messageId);

send(ack);

} else if ("ack".equals(messageType.toString())) {

// Process Ack

handleAckReceipt(jsonObject);

} else if ("nack".equals(messageType.toString())) {

// Process Nack

handleNackReceipt(jsonObject);

} else {

logger.log(Level.WARNING,

"Unrecognized message type (%s)",

messageType.toString());

}

} catch (ParseException e) {

logger.log(Level.SEVERE, "Error parsing JSON " + json, e);

} catch (Exception e) {

logger.log(Level.SEVERE, "Couldn't send echo.", e);

}

}

}, new PacketTypeFilter(Message.class));

// Log all outgoing packets

connection.addPacketInterceptor(new PacketInterceptor() {

@Override

public void interceptPacket(Packet packet) {

logger.log(Level.INFO, "Sent: {0}", packet.toXML());

}

}, new PacketTypeFilter(Message.class));

connection.login(username, password);

}

public static void sendMessage(String userName,

final String GOOGLE\_SERVER\_KEY, String toDeviceRegId, String message) {

SmackCcsClient ccsClient = new SmackCcsClient();

try {

ccsClient.connect(userName, GOOGLE\_SERVER\_KEY);

} catch (XMPPException e) {

e.printStackTrace();

}

String messageId = ccsClient.getRandomMessageId();

Map<String, String> payload = new HashMap<String, String>();

payload.put(MESSAGE\_KEY, message);

payload.put("EmbeddedMessageId", messageId);

String collapseKey = "sample";

Long timeToLive = 10000L;

Boolean delayWhileIdle = true;

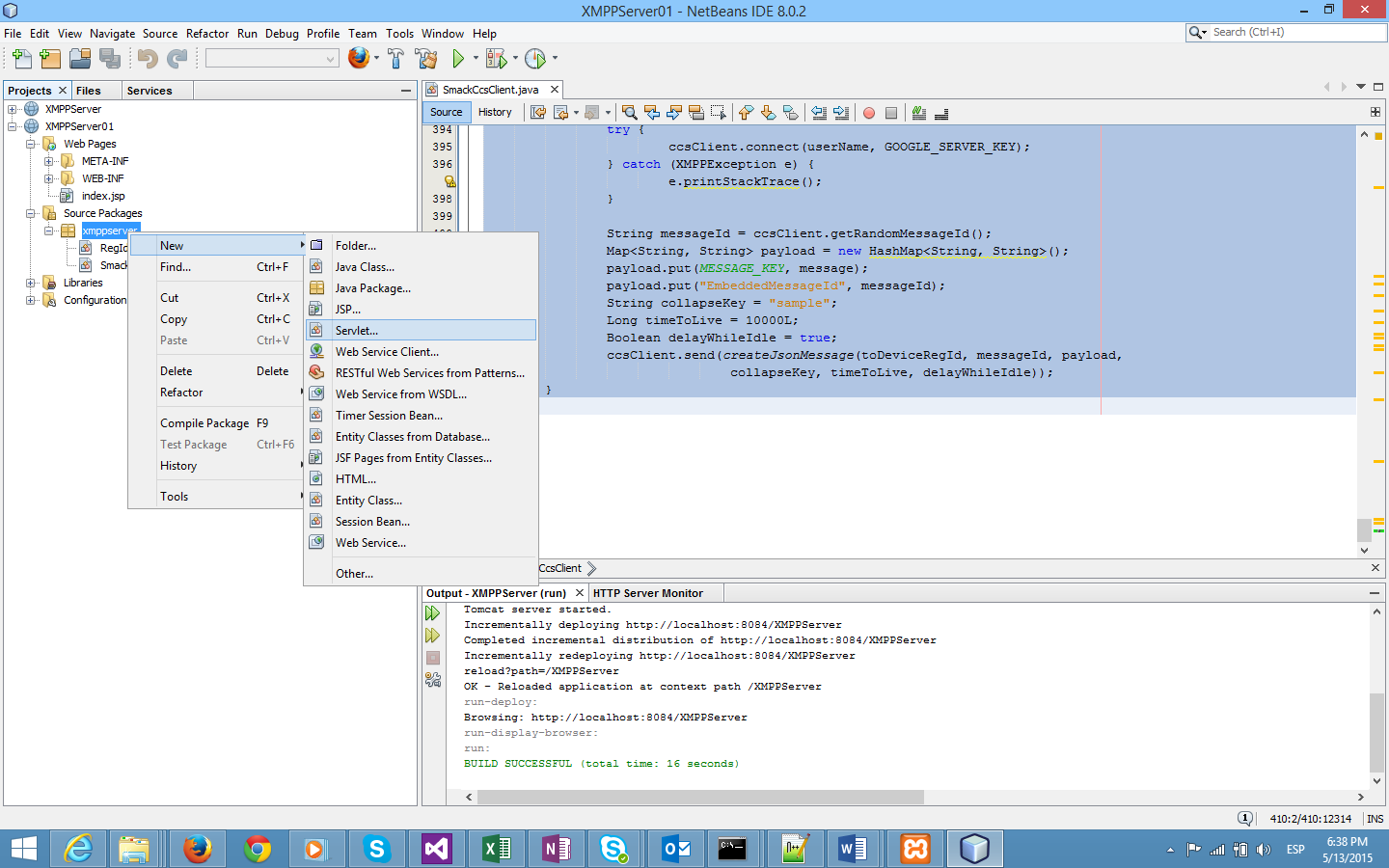
ccsClient.send(createJsonMessage(toDeviceRegId, messageId, payload,

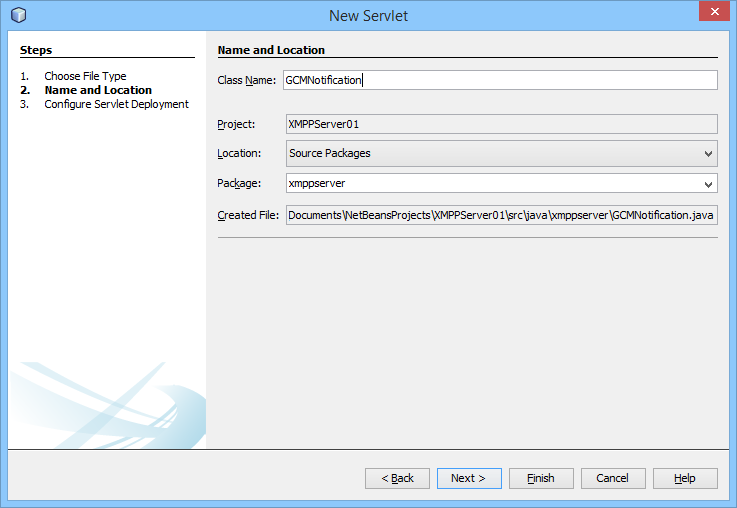
collapseKey, timeToLive, delayWhileIdle));

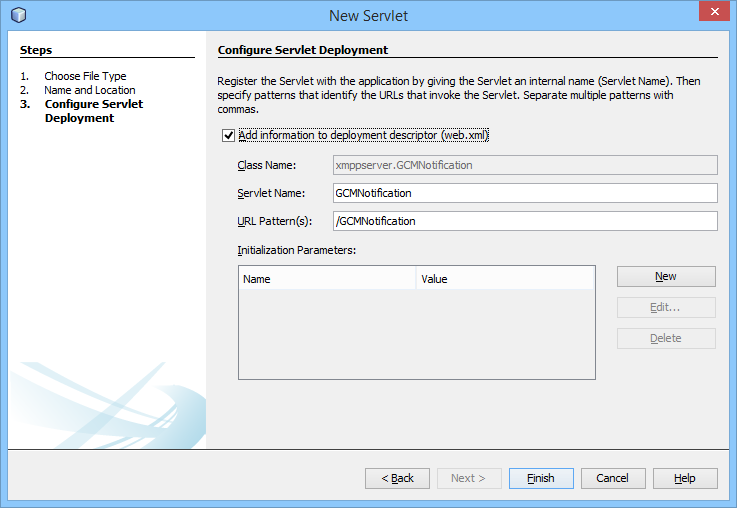
}

}

1. Adicionar un nuevo Servlet en el package recientemente creado llamada “GCMNotification”:

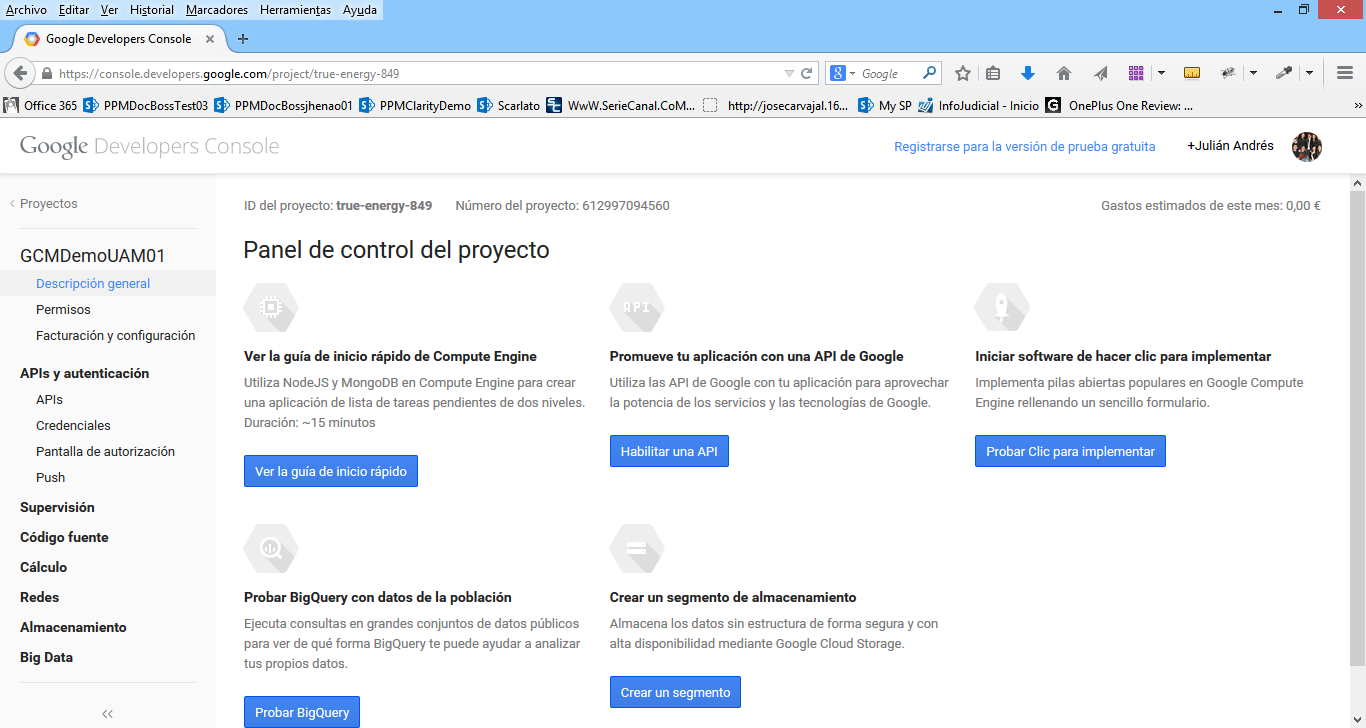






1. Reemplazar el código por el siguiente:

Reemplazar el valor antes de la “@” de la variable *GOOGLE\_PROJECT\_ID* por el ID del proyecto creado en la consola de Google:



/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package xmppserver;

import java.io.IOException;

import java.util.Set;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

@WebServlet("/GCMNotification")

public class GCMNotification extends HttpServlet {

private static final long serialVersionUID = 1L;

// Put your Google API Server Key here

private static final String GOOGLE\_SERVER\_KEY = "AIzaSyBrdsbpPVX8N6cv0jZ0UOaGw0VM4jTUG68";

// Put your Google Project number here

final String GOOGLE\_USERNAME = "156429511339" + "@gcm.googleapis.com";

public GCMNotification() {

super();

}

protected void doGet(HttpServletRequest request,

HttpServletResponse response) throws ServletException, IOException {

doPost(request, response);

}

protected void doPost(HttpServletRequest request,

HttpServletResponse response) throws ServletException, IOException {

try {

String userMessage = request.getParameter("message");

/\*

Set<String> regIdSet = RegIdManager.readFromFile();

String toDeviceRegId = (String) (regIdSet.toArray())[0];

\*/

String toDeviceRegId = request.getParameter("regid");

SmackCcsClient.sendMessage(GOOGLE\_USERNAME, GOOGLE\_SERVER\_KEY,

toDeviceRegId, userMessage);

request.setAttribute("pushStatus", "Mensaje envíado");

}

/\*

catch (IOException ioe) {

ioe.printStackTrace();

request.setAttribute("pushStatus",

"RegId required: " + ioe.toString());

}

\*/

catch (Exception e) {

e.printStackTrace();

request.setAttribute("pushStatus", e.toString());

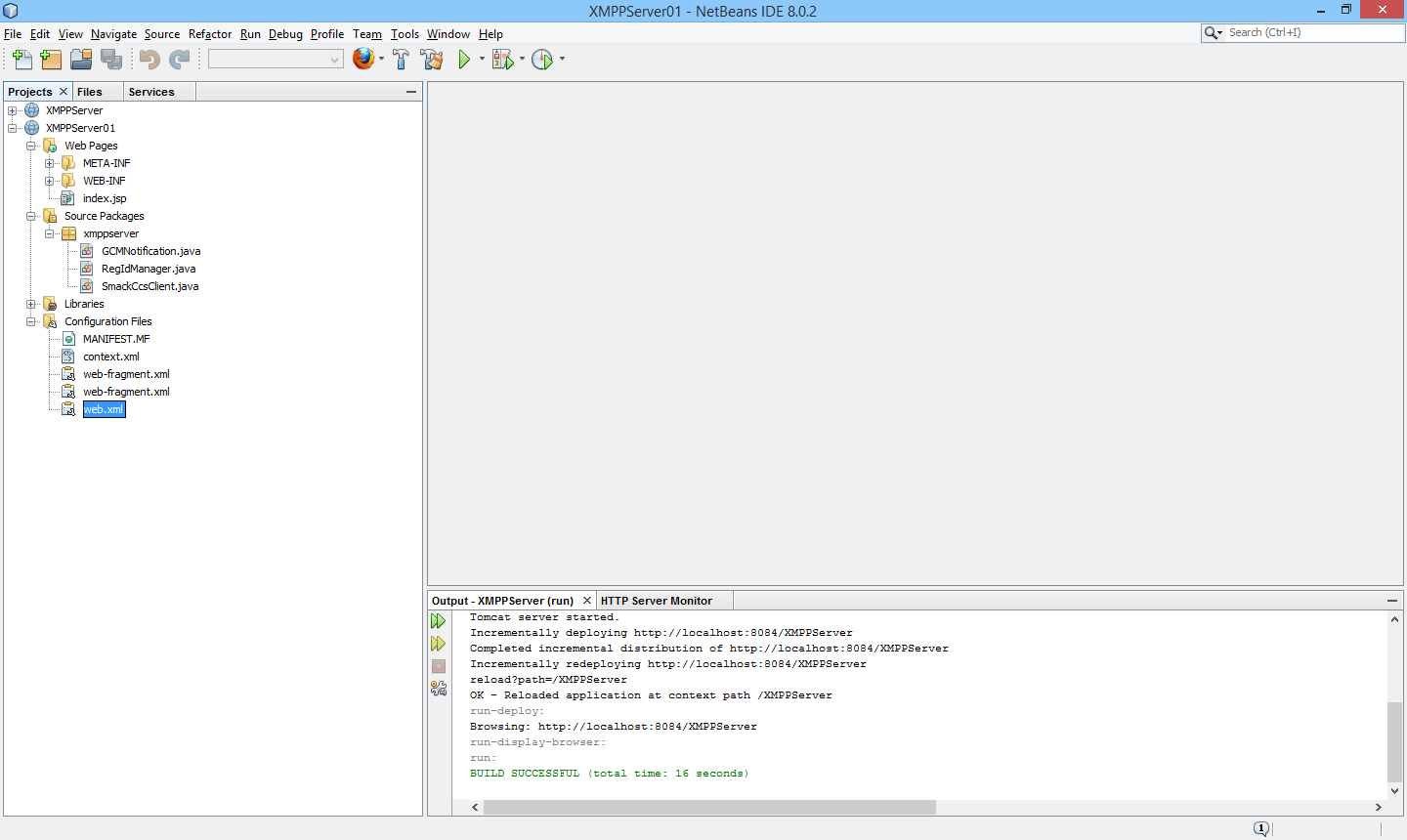
}

request.getRequestDispatcher("index.jsp").forward(request, response);

}

}

1. Reemplazar el contenido del archivo “Configuration Files/web.xml por el siguiente:



<?xml version="1.0" encoding="UTF-8"?>

<web-app

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns="http://java.sun.com/xml/ns/javaee"

xmlns:web="http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"

id="WebApp\_ID"

version="3.0">

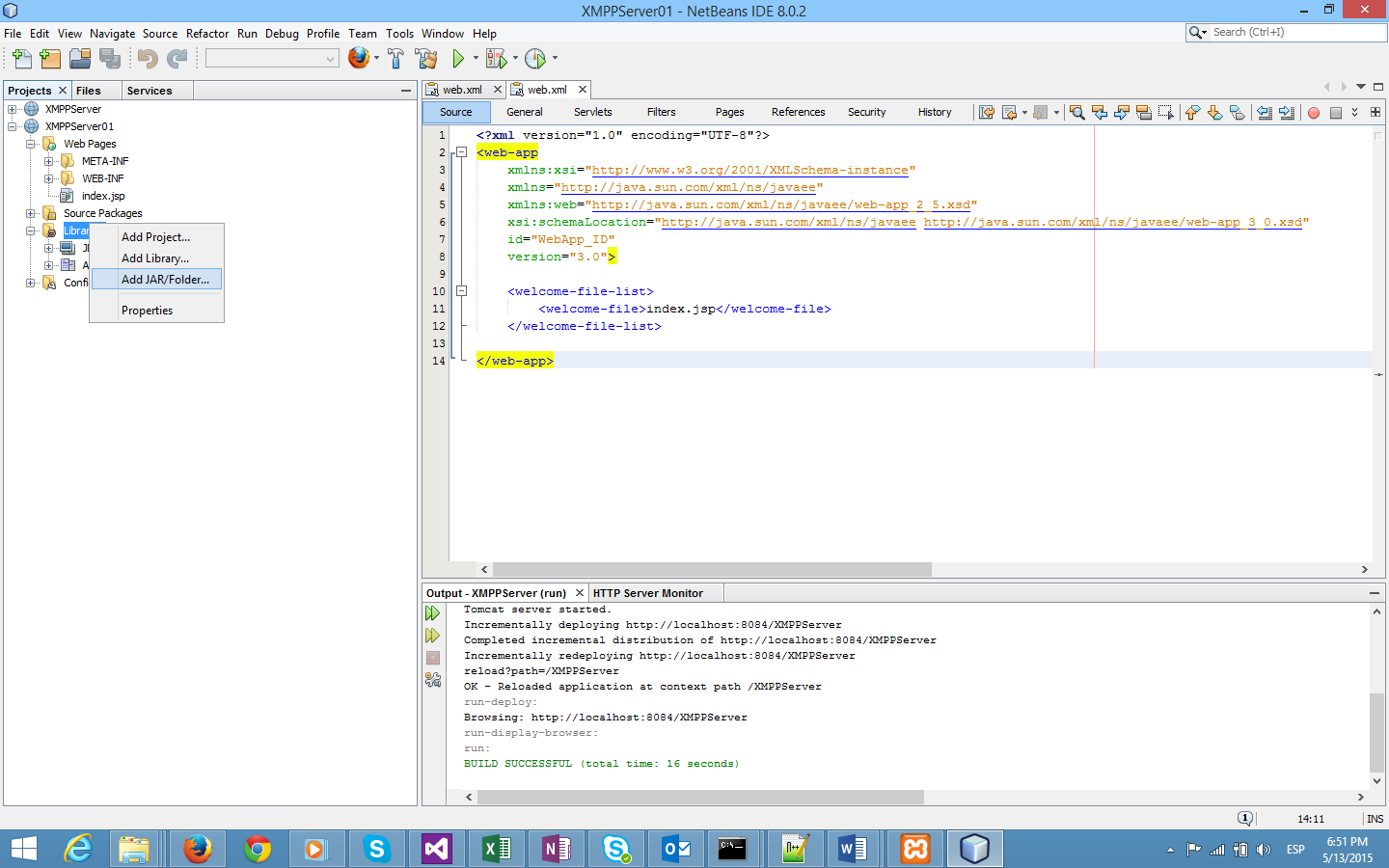
<welcome-file-list>

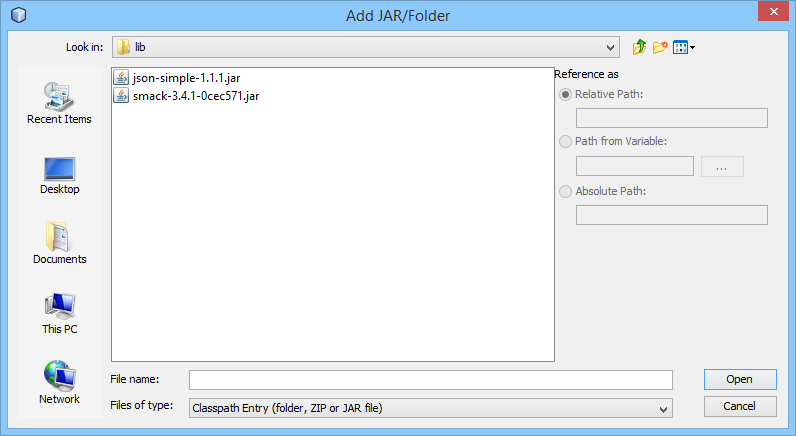
<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

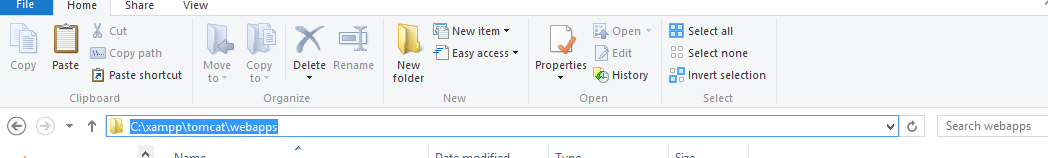
</web-app>

1. En el directorio “Libraries” adicionar las referencias a las librerías: “smack-3.4.1-0cec571.jar” y “json-simple-1.1.1.jar”.

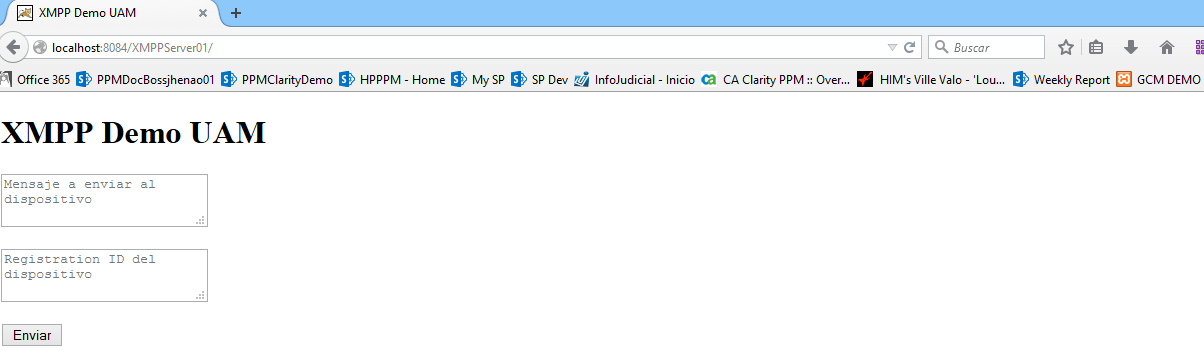




1. En el directorio “xampp/Tomcat/webapps” pegar la carpeta del proyecto creado.



1. En el navegador abrir la url del proyecto:



1. Diligenciar los campos:

**Mensaje:** Mensaje a enviar al dispositivo.

**Registration ID:** Pegar el id de registro generado por el dispositivo



1. Click en el botón “Enviar”:

