MIND STORM SOFTWARE PVT LTD

Google App Engine Training

ExamResults – Step 4 – XMPP Service

In this session, we are going to build the XMPP Layer for the application. In the previous session, we build the Datastore layer for the ExamResults application that returned actual results from a datastore.

So what we plan to do here is the following:

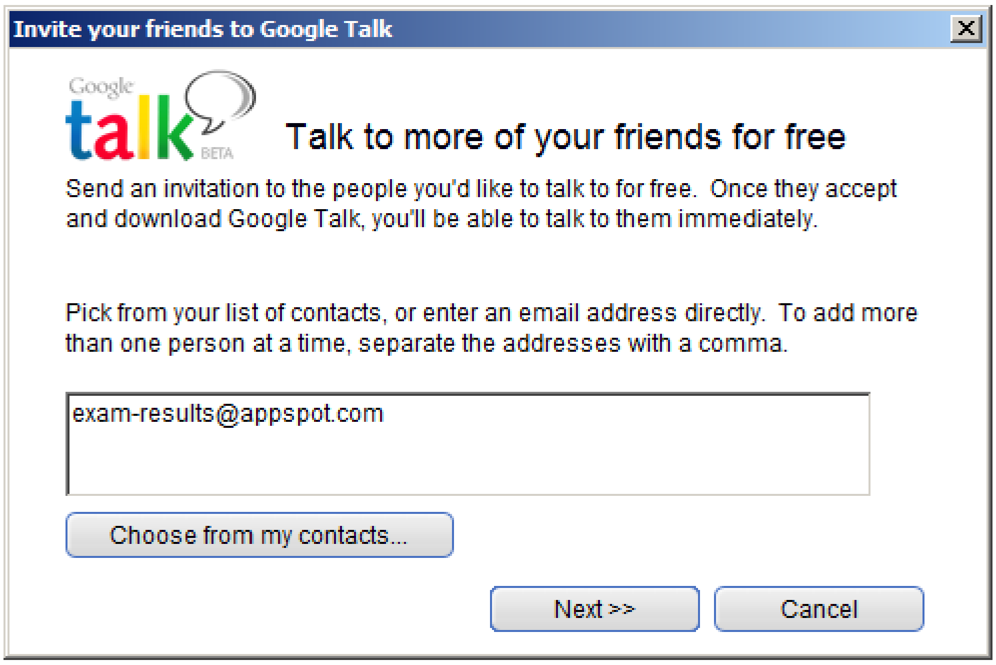
1) Allow a user to interact with the Exam Results app via XMPP client like Google Chat.
2) Using Google Chat, the user can add our application as a friend and send it some messages
3) Our application will interact with it and send back messages too.
4) We shall implement our Google Chat Bot to interpret basic commands sent to it like **about**, **help** and **<seatnumber>**. The last one will actually query the database for the result and give out the results in the Chat application itself.

ExamResults – Step 4 – See it in Action

Let us first see what we plan to achieve in this exercise. The screenshots below indicate the use of Google Chat to interact with the application.

The steps are given below:

1. The user adds our application as a friend in his/her Google Chat application as shown below:


Invite your friends to Google Talk
Google talk BETA

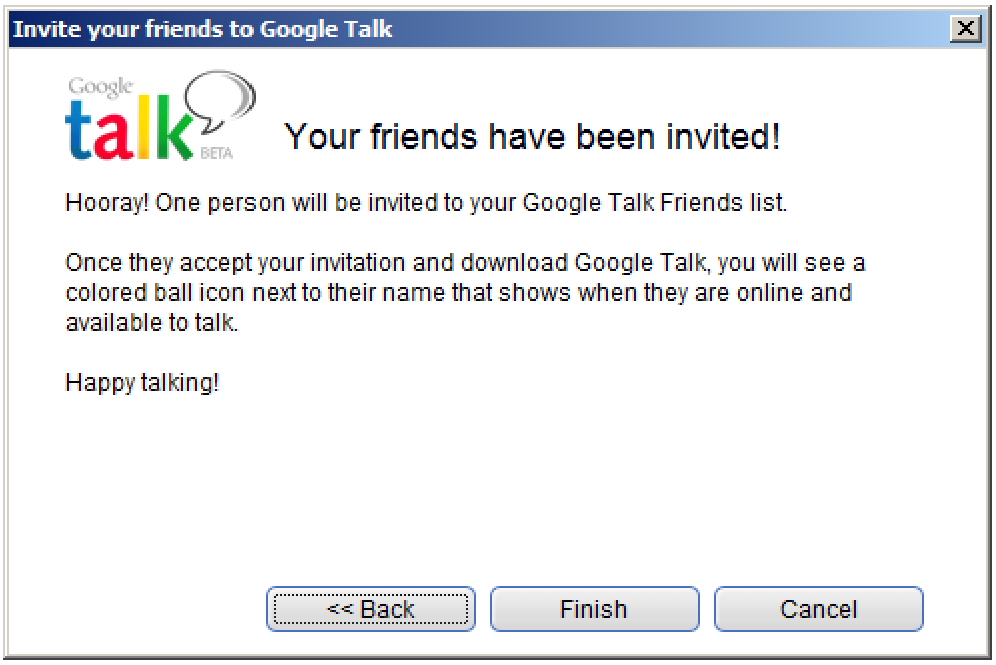
Talk to more of your friends for free

Send an invitation to the people you'd like to talk to for free. Once they accept and download Google Talk, you'll be able to talk to them immediately.

Pick from your list of contacts, or enter an email address directly. To add more than one person at a time, separate the addresses with a comma.

exam-results@appspot.com

Choose from my contacts...
Next >>
Cancel



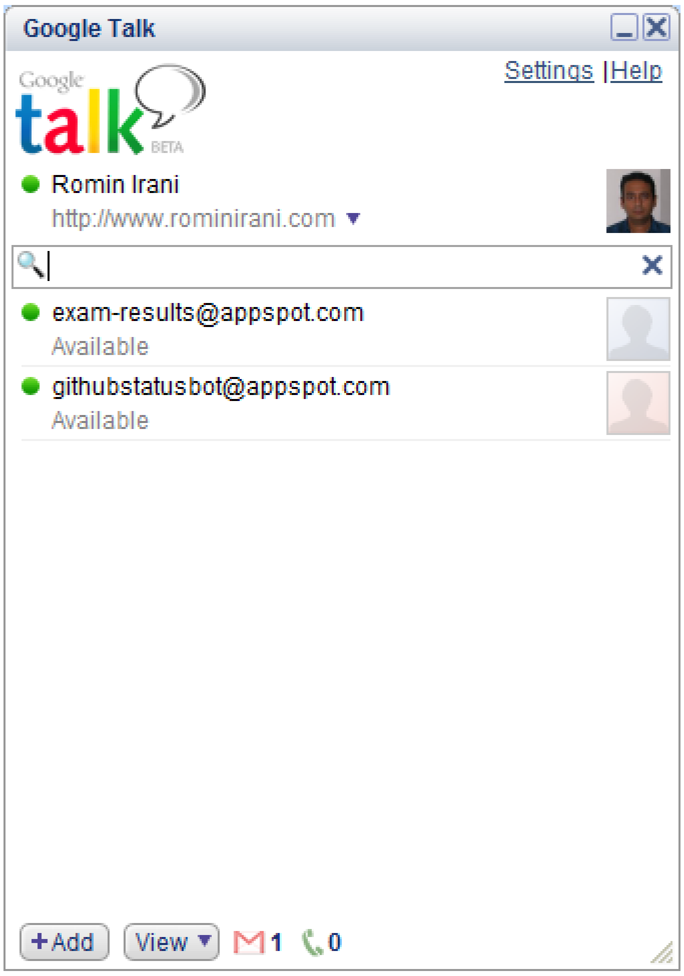
Your friends have been invited!

Hooray! One person will be invited to your Google Talk Friends list.

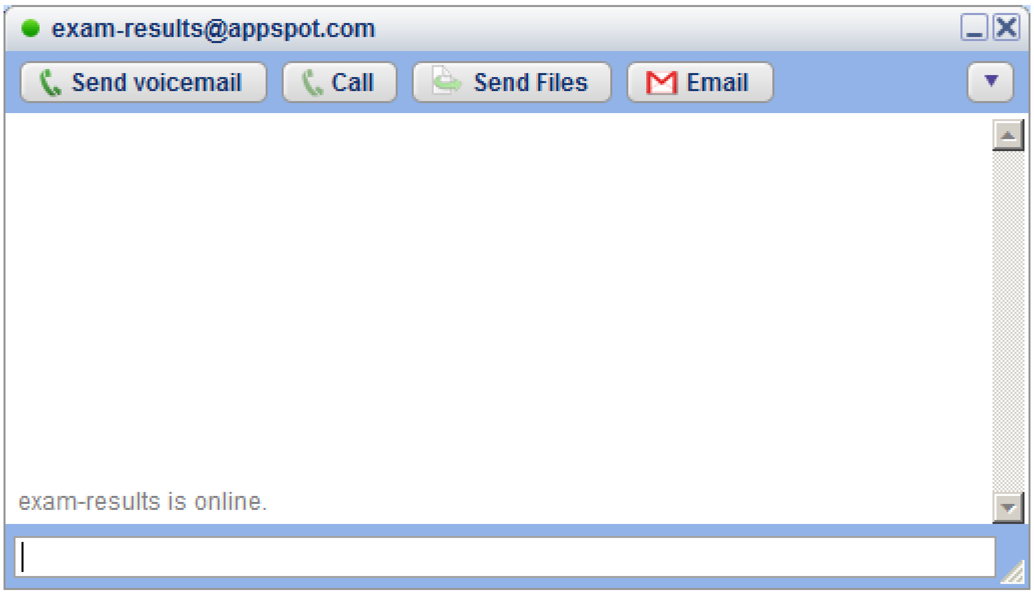
Once they accept your invitation and download Google Talk, you will see a colored ball icon next to their name that shows when they are online and available to talk.

Happy talking!

2. Our App Engine application on successful addition will appear in the friend list as given below:

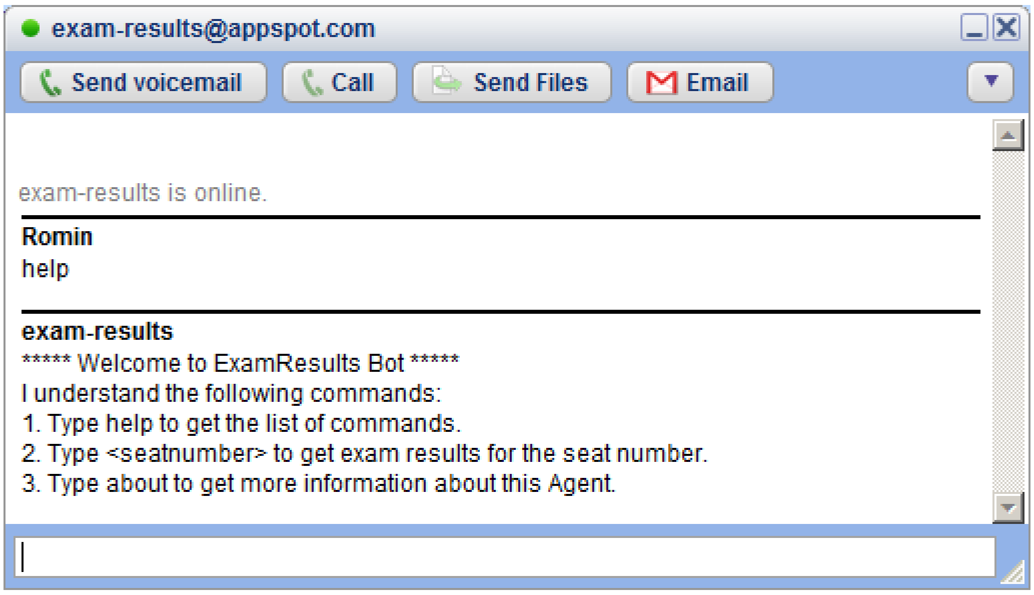


3. You can now open up a conversation with the Bot in the normal way:



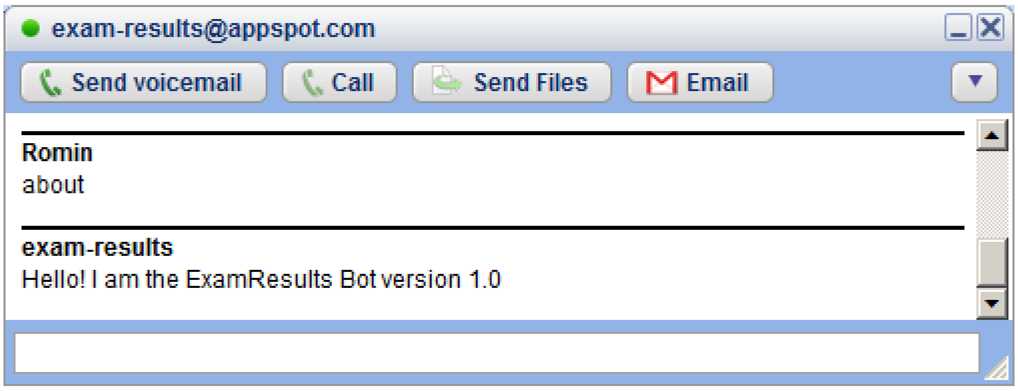
exam-results is online.

4. The user can send one of 3 messages that our Bot understands : about, help and <seatnumber>, where seat number is given to get the Exam Result of that student.
5. When the user sends a help message, the Bot responds as given below:



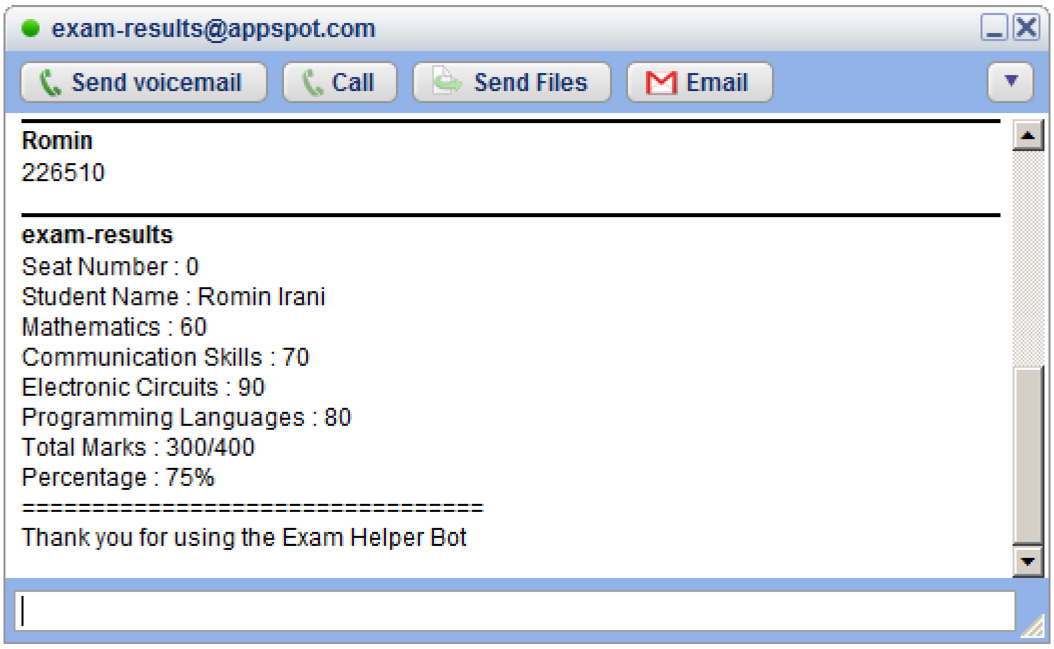
```
exam-results is online.
Romin
help
exam-results
***** Welcome to ExamResults Bot *****
I understand the following commands:
1. Type help to get the list of commands.
2. Type <seatnumber> to get exam results for the seat number.
3. Type about to get more information about this Agent.
```

6. When the user sends a about message, the Bot responds as given below:



```
Romin
about
exam-results
Hello! I am the ExamResults Bot version 1.0
```

7. When the user provides a seat number, the Bot retrieves the result from the datastore for the particular seat number and the result is shown as given below:



```
Romin
226510
exam-results
Seat Number : 0
Student Name : Romin Irani
Mathematics : 60
Communication Skills : 70
Electronic Circuits : 90
Programming Languages : 80
Total Marks : 300/400
Percentage : 75%
=================================
Thank you for using the Exam Helper Bot
```

Refer to **hands-on-exercises/Exam Results App/step 4** for full project source code. You can also import it directly into Eclipse and study the code.

Import the Project

We will import the Eclipse project at this stage so that you have all the working source code for this step.

1. In Eclipse, go to **File → Import**. This will bring up the Import dialog.
2. Select **General → Existing Projects into Workspace** and click on **Next**.
3. For Select root directory, click on browse and go to **C:\appengine-training\hands-on-exercises\Exam Results App\step 4**
4. Ensure that **Copy projects into workspace** is selected.
5. Click on **Finish**.

Add service entry to appengine-web.xml

Since are going to make use of the XMPP Service, we have to indicate to App Engine that we will be using that service by adding the service to the **WEB-INF/appengine-web.xml** file.

We have added the following entry to the file, anywhere inside of the root element.

```
<inbound-services>
      <service>xmpp_message</service>
</inbound-services>
```

Add the XMPP Servlet (ExamResultsBotServlet.java)

We will write a new Servlet **ExamResultsBotServlet.java** that will process the incoming XMPP Message, execute the required functionality and send back a response as an XMPP message itself.

When your App Engine application receives a XMPP Message, the App Engine infrastructure packages the XMPP Message into a HTTP Request so that it is easy for you to extract out the data and then do the processing depending on the text that has been sent.

Take a look at the source code for the **ExamResultsBotServlet..java**. The entire source code is present in **hands-on-exercises/Exam Results App/step 4/src/com/mindstormsoftware/examresults/ExamResultsBotServlet.java** file

Let us go through the code step by step:

1) The first thing that we do is get an instance to the XMPPService and we wrap the incoming HTTP Request object into a Message object.
2) Once we have the Message object, we can inspect it by calling methods like getFromJid() and getBody().
3) The getBody() method will return us the text message that was sent by the user. So if the user sent a text message named 'about', then the getBody() method will return the value 'about'.

4) Then you simply write basic Java code to determine what kind of text was invoked and form the response text.
5) To form the reply, we use the MessageBuilder class to build the reply message. We form the message by setting the recipient (which is the same value as the getFromJid() in step 2 above) and the new reply message that we formed in step 4.
6) We send the reply across by using the sendMessage() method of the XMPP Service.
7) Finally, we can optionally check if the delivery of message to the Google Infrastructure was successful or not. Note this is just about delivery to Google XMPP Server and not the final delivery of the message to the recipient, which could still fail if the other XMPP Server did not like something in the message like wrong recipient JID, etc.

Add Servlet entry to web.xml

We need to add the servlet entry to **web.xml**, so that our application is aware of the Servlet. Remember that all XMPP Messages to your application will be sent on the following web path **`/_ah/xmpp/message/chat/`**, so we will map our Servlet to that, so that whenever a message is sent, our servlet is invoked.

The following servlet entries are added to **web.xml**

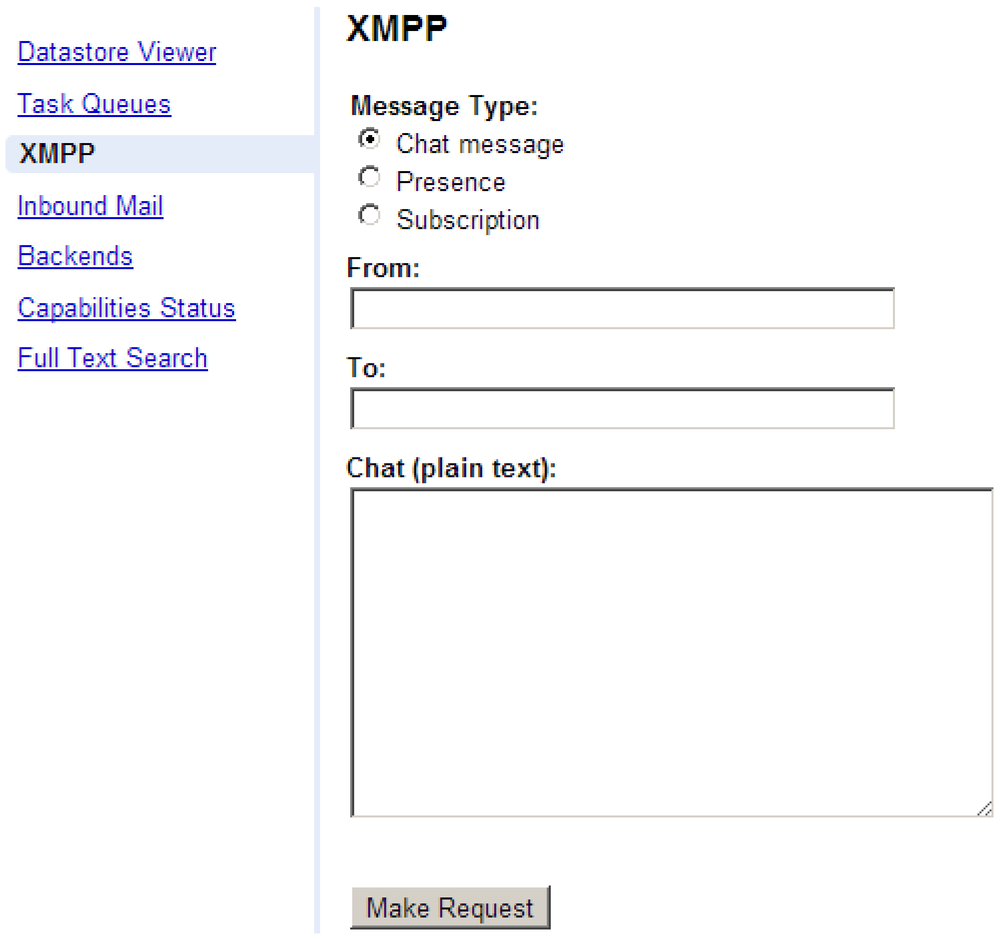
```
	<!-- ExamResults Bot -->
	<servlet>
		<servlet-name>ExamResultsBot</servlet-name>
		<servlet-
class>com.mindstormsoftware.examresults.ExamResultsBotServlet</servlet-class>
	</servlet>
	<servlet-mapping>
		<servlet-name>ExamResultsBot</servlet-name>
		<url-pattern>/_ah/xmpp/message/chat/</url-pattern>
	</servlet-mapping>
```

Instructions to Test locally

Once you made the changes, test out the changes locally first. Stop the Web Application if it is already running by clicking on the Stop icon in the Console window. Restart the application i.e. Right-click on Project and select **Run As → Web Application** and navigate to http://localhost:8888

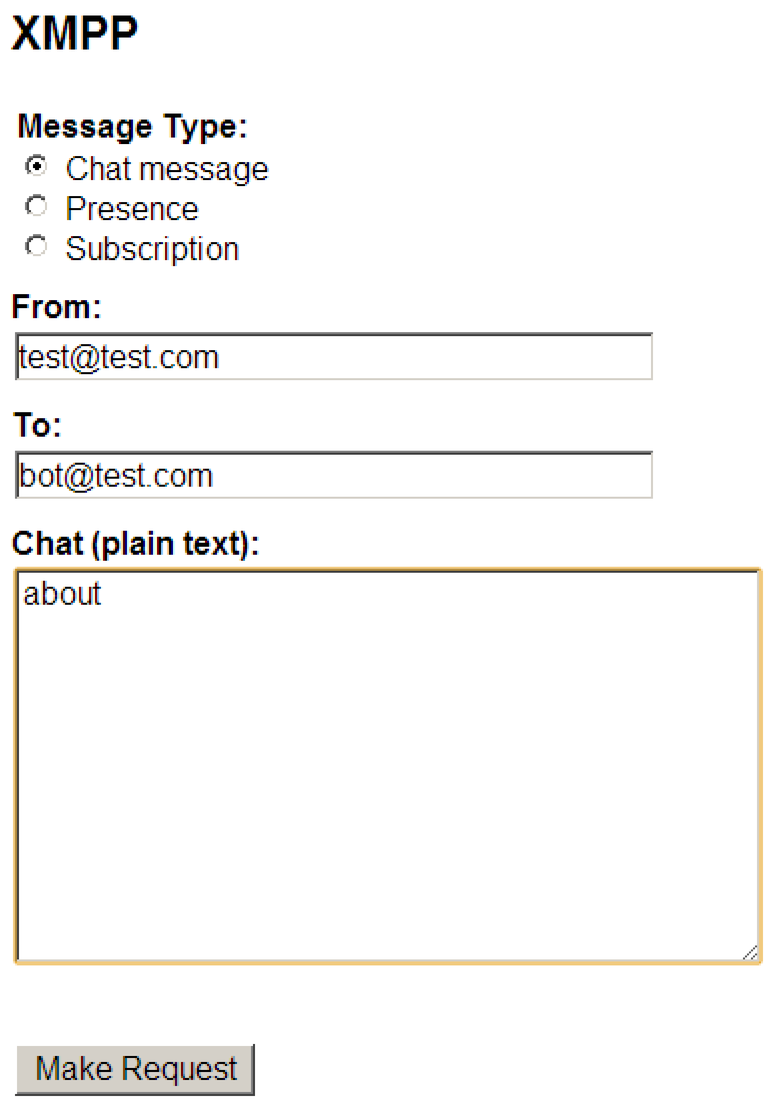
You can test out the XMPP interactions locally via the Admin console that is available on http://localhost:8888/_ah/admin url.

Go to the above url and click on the link **XMPP** as shown below:



You can now send a message to your application by populating the fields as given below:

- **Message Type**: Select as Chat message
- **From**: Provide a JID over here. It can be anything.
- **To**: Provide a JID here. It can be anything in the local environment.
- **Chat (plain text)**: This is the message that you want to send to your Bot. It is similar to chatting with someone. Remember that our bot understands 3 commands: **about**, **help** and ***<seatnumber>***
- Try giving the value: **about** and click on Make Request. Sample screenshot is shown below:


XMPP

Message Type:

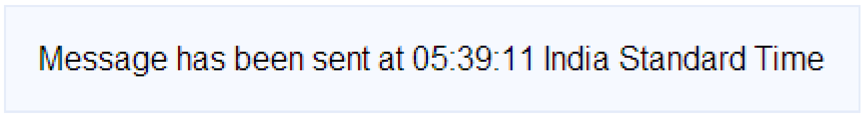
(●) Chat message
( ) Presence
( ) Subscription

From:
test@test.com

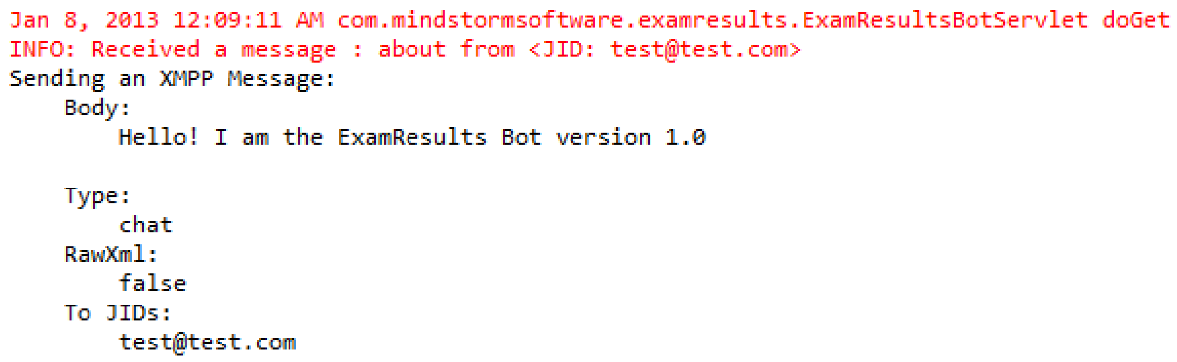
To:
bot@test.com

Chat (plain text):
about

- This will send the message to our running application. In the Admin console, you will just see a message sent successfully notification as shown below:


Message has been sent at 05:39:11 India Standard Time

- If you switch to your Eclipse and view the Console, you will see that the message was received successfully and that the message is sent out by our Application. You will just see the text but this validates that the Bot is working. A screenshot of the console is shown below:



```
Jan 8, 2013 12:09:11 AM com.mindstormsoftware.examresults.ExamResultsBotServlet doGet
INFO: Received a message : about from <JID: test@test.com>
Sending an XMPP Message:
    Body:
        Hello! I am the ExamResults Bot version 1.0

    Type:
        chat
    RawXml:
        false
    To JIDs:
        test@test.com
```

Instructions to Deploy and Test Live

To deploy to the cloud, right-click on project and then **Google → Deploy to App Engine.**

Once the application is deployed, you will need to now use a Chat application like Google Talk to interact with it.

Remember that your application is live at http://<appid>.appspot.com and your Jabber Id for your application will be **<appid>@appspot.com**. So when you add the Bot to Google Chat, add the Jabber Id which is in the form **<appid>@appspot.com.**

Once it is added, you should see it online and can then interact with it.

Next Session

In the next session, we shall see how to add Email Service support to the application. Users will then be able to get exam results directly by sending an Email to the application. The application will respond back via Email with the results.