Spring 2011

FBchatters: A Facebook application for Gtalk

Ronak Shah
San Jose State University

FBchatters: A Facebook application for Gtalk

A Writing Project

Presented to

The Faculty of the department of Computer Science

San Jose State University

In Partial Fulfillment of the

Requirements for the

Degree Master of Computer Science

By
Ronak Shah
May 2011

SAN JOSE STATE UNIVERSITY

The Undersigned Writing Project Committee Approves the Writing Project Titled

FBchatters: A Facebook application for Gtalk

By

Ronak Shah

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Soon Tee Teoh, Department of Computer Science	05/ /2011
Dr. Robert Chun, Department of Computer Science	05/ /2011
Mr. Kartik Shah, Broadcom Corporation	05/ /2011

ACKNOWLEDGEMENTS

I am very pleased to my project advisor, Dr. Soon Tee Teoh, for his support and suggestions throughout my master's project and also I would like to thank my committee members, Dr. Robert Chun, and Mr. Kartik Shah for their time and effort.

I would like to thank my parents and my sister, for their encouragement and constant support. I would also like to thank my friends to give me a valuable feedback on my project.

ABSTRACT

FBchatters: A Facebook application for Gtalk

The motto of this project is to provide accessibility to Google talk (gtalk) service within a Facebook account by creating Facebook application called FBchatters and also by creating Firefox plug-in called FBchatters. Most of all internet users have account on Facebook and Google. Social network site, Facebook has more than 600 million users and half of them log on to Facebook in any given day [1]. Google also has 170 million users who are using gmail and gtalk services every day. There are lot of users who do chatting simultaneously on Facebook and Gtalk. Also, Google talk provides a facility to do chat with AOL users. It's really boring to chat on both at a same time and switch browser window to send and receive messages. It's good for both users to have access to both chatting application at one place. Using this application user can do chat with gtalk and AOL user within a Facebook. This application is hosted on Amazon elastic compute cloud (EC2). Reason behind using Amazon EC2 is it provides resizable compute capacity in cloud. There are lots of other features of Amazon EC2 like flexible, elastic, completely controlled, secure, reliable, etc that leads to use of Amazon EC2[7].

Table of contents

List of figures

List of tables

1. Introduction

1.1 Project Overview

Social networking websites connect people with their friends, family members and others. Facebook is the most popular social networking site and it has more than 600 million active users as of January 2011[2]. People do chatting, share photos, videos, links, news on Facebook. The most popular email (gmail) and chat (gtalk) service is provided by Google. The Google has 170 million unique users and they use gtalk service. Also, gtalk provides chatting facility to the AOL users within gtalk. The idea behind this project is to provide a user interface that allows user to do chatting with gtalk and AOL friends within a Facebook. In this project called FBchatters is a Facebook application and Firefox plug-in. They allow user to do chatting with three different clients in Facebbok. Without FBchatters user has to switch windows every time to look and send message to these different chatting clients.

This application is hosted on cloud computing service provided by Amazon Web service (AWS). Amazon Elastic Compute Cloud (Amazon EC2) is a cloud computing web service that allows resizable compute capacity in the cloud. Amazon EC2 allows developer to get and configure computing power within minimal time. It provides full control of computing resource. Also, the Amazon EC2 cuts down time to boot up new server instance to minutes. The main advantage of Amazon EC2 is, it charges by what type of computing service you used and for how long. You can find more details about Amazon EC2 in Topic 2.8.

1.2 Report Overview

The project report is divided in following chapters: Chapter 1 is about overview of project and report. Chapter 2 discusses the details of software, protocol and tools which are used to develop this application. Chapter 3 has details of what type of environment setup is required to develop this application and also it includes details of server environment setup to host this application. Chapter 4 discusses software architecture that used in project. Chapter 5 provides detail information about the directory structure of application and Firefox plug-in. Implementation details are included in Chapter 6. The comparison of Facebook application and Firefox plug-in is given in Chapter 7 and the comparison is done based on feedback that I received from a group of six students. In last, chapter 8 concludes the project. Appendix part has screenshots of application and plug-in. References are included in last part of report.

2. Protocol, Software and Tools

2.1 XMPP

Extensible Messaging and Presence Protocol (XMPP) is an open-standard communications protocol for send and receive messages. It is based on Extensible Markup Language (XML). XMPP protocol was initially known as Jabber. It was designed for real-time, instance messaging, contact list and presence information. This protocol also used in Voice over IP (VoIP) and to transfer files. In August 2005, Google launched gtalk application for VoIP and instance messaging using XMPP protocol. Also in February 2010 Facebook came with chat feature and it is running on XMPP protocol [3]. Anyone can run XMPP server on their domain. The standard TCP port for XMPP is 5222. For my project I have used gtalk domain gtalk.google.com on 5222 port number.

2.2 xmppy

xmppy is a python library that used to implement XMPP protocol. xmppy is available under GNU General Public License that enables freely redistribution and you can enhanced it according to your need. Using xmppy library I developed functions that authenticate user with gtalk server, send and receive messages, get friend lists and their presence.

2.3 Python 2.6.6

To develop gtalk application I have used python language. Python is an interpreted, powerful high-level programming language. It has a very simple syntax and dynamic typing, both features with interpreted environment makes python perfect language for scripting and rapid application development. Python has large library that helps in application development. It is freely available

and can be used on any platform. Python supports several programming paradigms like object-oriented programming, functional programming and automatic memory management.

2.4 Django

Django is an open source web framework for python. It's written in python and it follows model-view-template design pattern. This pattern is very similar to model-view-controller design pattern. Django's main aim is to allow developers to create complex and database-driven websites easily. It also provides reusability and pluggability of components. In Django development python is used all over, even for configuration and database.

2.5 XUL

The XUL is XML User Interface Language developed by Mozilla. XUL is used to create a cross-platform application for Mozilla such as Firefox and Flocl. XUL depends on several existing web technologies and standards. XULRunner is a tool that used to create XUL application and it is developed by Mozilla Foundation.

2.6 jQuery

jQuery is an open source JavaScript library. It distributed under MIT license and GNU General Public License. It is designed to implement cross-side client scripting of HTML. It runs on client's browser. It makes easier to navigate, select HTML document and DOM element and implement of AJAX application.

2.7 Apache/mod_python 3.3.1

mod_python is a module that integrates the python language into the Apache server. It is developed to substitute Common Gateway Interface (CGI) as a way of executing python script over a web server. The benefits of mod_python are: it executes fast then CGI; it maintains data for multiple sessions. In current state there is no development is going on for mod_python because the code and the project is very mature and it requires very less task to maintain it[6].

2.8 Amazon EC2

Amazon Elastic Compute Cloud is a part of Amazon Web Services (AWS). It provides cloud computing platform. User can create virtual computers according to their requirements. i.e user can select operating system, number of CPU, memory and disk storage. After user done with setting, he/she can start Amazon Machine Image (AMI) to create a virtual machine. AMI is a special type of virtual appliance which is used to instantiate (create) a virtual machine within the Amazon Elastic Compute Cloud. . It serves as the basic unit of deployment for services delivered using EC2 [4].The running virtual machine also knows as an instance. User can use this virtual machine to deploy their application. The term “elastic” is used because user can start, stop, terminate, reboot their virtual machine as needed and they need to pay by the hour of active machines.

2.9 Amazon EBS

Amazon Elastic Block Store (EBS) provides storage facility to use with Amazon EC2 Instance. EBS is independent from the life of an instance. Amazon EBS allows user to create storage from

1 GB to 1 TB and that can be attached to Amazon EC2 instance. Also, user can mount any number of EBS to same instance. User can use EBS as a hard drive.

3. Environment Setup

3.1 Development Environment

In this project, I have used Ubuntu linux as a platform. Python 2.6.6 is used as a developing language and django 1.3 used as a web framework. The mod_python is used to run application on server. It is an apache module that allows python interpreter to run within a server. With mod_python we can develop web based application that will run faster than usual CGI application. jQuery is used for making attractive user interface.

For developing Firefox plug-in (Add-on), I have used XUL language and XULRunner as a developing tool. ZIP command is used to do packaging of plug-in files and that creates .xpi file that can be installed in Firefox.

Use putty software to connect Amazon EC2 instance securely. Also it is useful to transfer files to local machine to Amazon EC2 instance (remote machine).

3.2 Amazon EC2 (web hosting) Environment setup

To run gtalk application on Amazon EC2, there are several things that need to setup first. First we need to choose pre configured Amazon Machine Image (AMI) for linux operating system or create an AMI that containing your application and libraries. After successfully setup configuration, run AMI instance and when it gets up we need to install python and django on that instance. Other settings we need to do while creating security group.

1. Create key pair that will be used for to connect Amazon EC instance securely.
2. Enable 8000 port to run django-python web application on apache/mod_python.

3. Enable port number 21 for ftp and 22 for ssh. 22 port will be used for transferring data securely to the server.
4. We can assign elastic IP to our Amazon EC2 instance. After assigning IP address we can access the server using that Elastic IP.

4. Software Architecture

Django web framework is used to develop web user interface for FBchatters. Django framework follows Model-View-Template (MVT) architecture which is similar to Model-View-Controller (MVC) Architecture.

This application was designed using MVT software architecture. The main use of MVT architecture is to separate application data and business logic from the presentation data. The MVT gives reusability and using MVT application becomes more expressive. MVT architecture can describe using MVC.

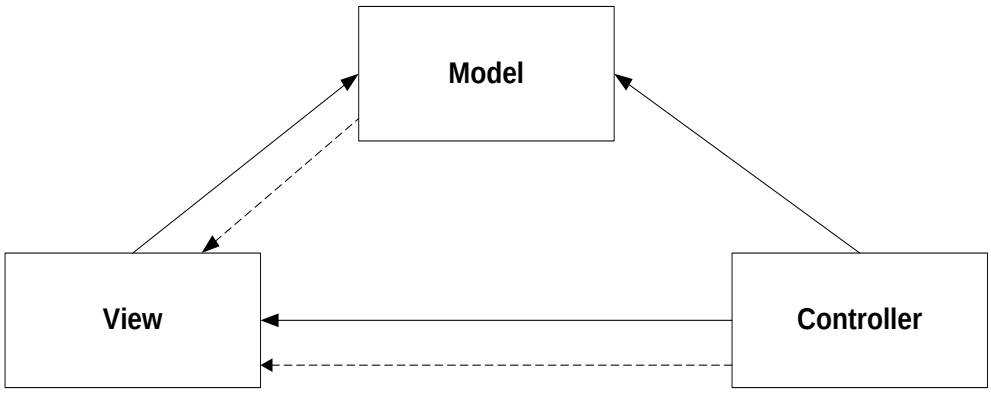


Figure 4.1 Model-View-Controller

1) Model:

The model object is responsible to show application data on view that is requested by user. It is also aware of all operation that needs to be applied to transform that object. The model represents enterprise data and business logic that control the access and updates of this data.

Model is not aware about the presentation data and how that data will be displayed to the browser [5]. The model object works same in both architectures.

2) View:

The view is used to show the presentation of an application. The view uses query methods to get data from model and renders it. The view is independent of the application logic. It wouldn't change if there is some updates in application logic and it is responsibility of the view to maintain consistency in its presentation when the model changes.

In django "view" is the Python callback function for a particular URL, because it callbacks function that describes which data should be presented. Moreover, it is sensible to separate content from presentation – which is where templates come in. In Django, a "view" describes which data is presented, but a view normally delegates to a template, which describes how the data is presented [5].

3) Controller:

The controller is responsible to handle every request from user and request always pass through the controller. The Controller calls model according to user's request and model takes appropriate action on data. After that it is a responsibility of controller to directing appropriate view to user. In web application, views and controllers work very closely.

In Django's case, the controller is probably the framework itself: the machinery that sends a request to the appropriate view, according to the Django URL configuration [5].

5. Web application and Mozilla plug-in Directory structure

The final outcome of this cs298 writing project is a web application and Mozilla plug-in (Add-on). Web application allows Facebook user to communicate with gtalk and AOL users within the Facebook. The web application is developed using Python, Djago, XHTML, jQuery, AJAX, CSS and hosted on Amazon EC2. Firefox plug-in is used for a same purpose but it divides Firefox window and create a new panel from which user can do chat with gtalk users.

5.1 Web application directory structure

Figure 5.1 demonstrates directory structure of website. The website follows model-view – template software architecture. It also has other module like xmpp. The htmls and site_media module is a template component of MVT pattern and mysite_final contains model file and view files. The url.py file allows only valid URL which is requested by user.

The web application follows MVT architecture to arrange code. Each subdirectory of mysite_final is represents MVT component. xmpp directory contains xmppy library files those use in authenticate user's credential with gtalk server. Directory htmls and site_media contains html template, css, images and JavaScript files. That use to describe the design of a webpage. View files implements business logic for the application. View files contain function list and each function defines business logic or actions.

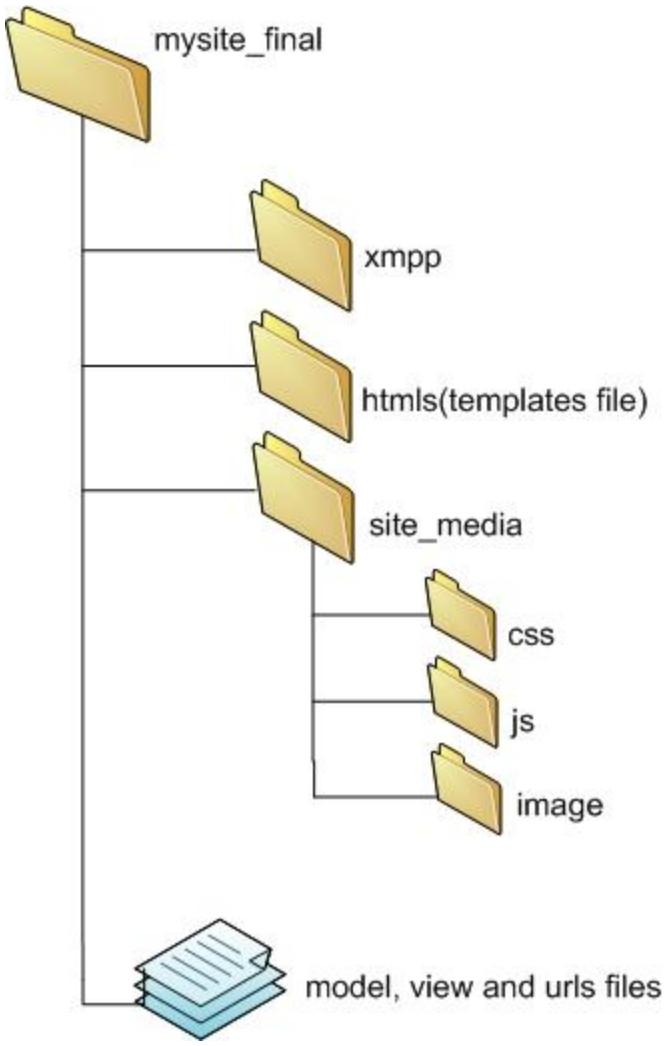


Figure 5.1: Directory structure for web application

Model file contains setting for a database connection. It represents python modal class and by using it we can create, insert, update and delete records of database. Benefit of implementing modal class is that we don't need to write SQL statements every time, we can use modal class method that executes SQL statements and it can be reused for same type of query. In this project I haven't use any database to store any information. For displaying users chat message I simply used python list. So, the chat history is deleted when user sign out from his account.

The url file specifies which view is called for requested URL pattern by user. According to requested URL, url file calls function of view file.

5.2 Firefox plug-in directory structure

Figure 5.2 shows the directory structure of Firefox plug-in. Fbchatters is a main directory contains all required files to create a plug-in. It contains two main files:

1) install.rdf

   This file gives all information about the plug-in to Firefox. It has detail of plug-in name, developer name and id, compatible Firefox version (both minimum and maximum), etc. install.rdf file is written in Resource Description Framework (RDF) format. Its format is similar to XML format. It must be saved as install.rdf. This file must be located at a top level directory of plug-in.

2) chrome.manifest

chrome.manifest file tells Firefox about the contents of plug-in. It also gives information about which type of content it has and where they are located. This file is written in simple line-space delimited format. The file name must be called as a chrome.manifest and must be located at top level of plug-in directory.

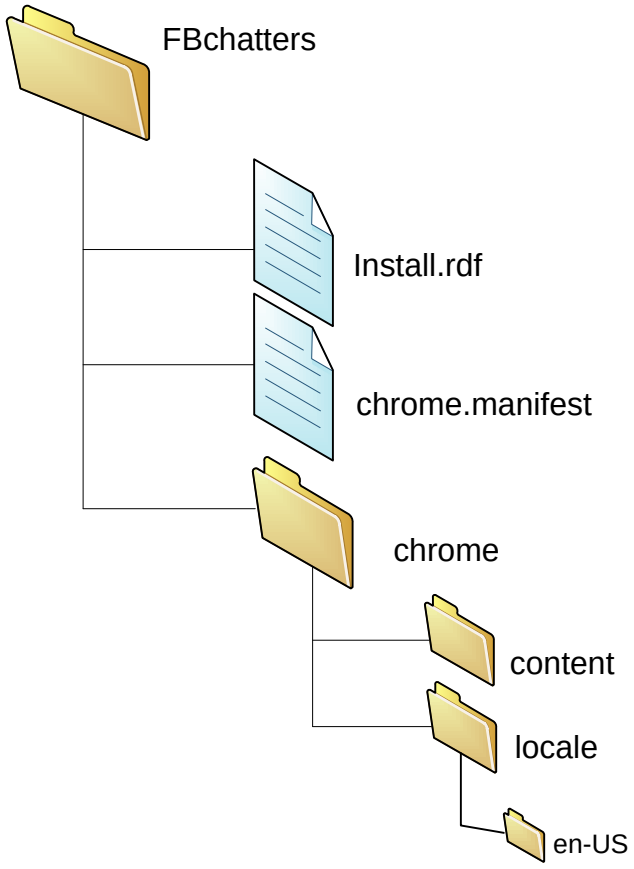


Figure 5.2: Directory structure for Firefox plug-in

Directory “content” contains xul, css and js files and locale contains files related to localization.

6. Implementation

6.1 Web Application (FBchatters)

FBchatters is a web application that integrates with Facebook. As we discussed earlier this application follows Model-View-Template architecture which is similar to Model-View-Controller. The application allows gtalk user to sign in with their gtalk username and password. So, the first requirement of this application is users should have gtalk account. This application validates username and password using xmppy library with gtalk server. In next section we will discuss the code for Template, View and Model.

First we will see the urls.py file. This file validates each and every urls which are requested by users.

```
1  from django.conf.urls.defaults import *
2  from mysite_final.login_view import *
3  from mysite_final.send_view import *
4  from django.conf import settings
5
6  urlpatterns = patterns('',
7      # Example:
8      # (r'^mysite_final/', include('mysite_final.foo.urls')),
9
10       ( r'^site_media/(?P<path>.*)$', 'django.views.static.serve', { 'document_root': settings.MEDIA_ROOT } ),
11       ( r'^check/$', login_check),
12       ( r'^$', login),
13       ( r'^home/$', ajaxsend),
14       url( r'^users/$', ajax_send,name ='ajsend'),
15
16 )
```

Figure 6.1urls.py

As shown in Figure 6.1, it uses python regular expression to validate the requested url. For example the line number 12 is (r'^$', login). In that the first argument is for validating url using Regex and second argument is function name. It means that when user enters site name in browser this line validates requested url and it calls login function.

This login function is imported from login_view.py file and this file is behaved as a controller object of MVC.

```
from django.template import Template, Context, RequestContext
import datetime, sys, xmpp, os, signal, time
from django.http import Http404, HttpResponse, HttpResponseRedirect
from django.shortcuts import render_to_response
from receive_list import *

def login( request ):
    template = 'login.html'
    return render_to_response( template,context_instance = RequestContext( request ) )

def login_check(request):
    uname=request.POST.get('uname', '')
    password=request.POST.get('pass', '')
    f=log_in(uname,password)
    print f
    if f==1:
        #template = '../home/'
        template = '../home'
        #main()
        #return render_to_response( template,context_instance = RequestContext( request ) )
        return HttpResponseRedirect(template)
    else:
        template = 'login.html'
        errmsg="Please check your username and password."
        return render_to_response( template,{"errmsg":errmsg},context_instance = RequestContext( request ) )

def show_home(request):
    template = 'ajaxsend.html'
    #return HttpResponseRedirect(template)
    return render_to_response( template,context_instance = RequestContext( request ) )
```

Figure 6.2 login_view.py

According to Figure 6.2 login_view.py file defines the login function which is called from urls.py file. The login function has one argument is “request” and it returns the rendered “login.html” page using django’s Template, Context and RequestContext module.

Figure 6.3 shows the code for login.html page. To see how login.html page looks like, go to Appendix and look at login.html page.

```
{% extends 'base.html' %}
{% block extra_js %}
<script type="text/javascript" src="{{ MEDIA_URL }}jquery.form.js"></script>
<script type="text/javascript">
        // wait for the DOM to be loaded
        $(document).ready(function() {
            // bind 'myForm' and provide a simple callback function
            $('#myloginForm').ajaxForm(function() {
                alert("Thank you for your comment!");
            });
        });
    </script>
{% endblock %}

{% block main %}

<div id="loginbox">
<form id="myloginForm" action="/check/" method="post"> {% csrf_token %}
<table width="200" border="0" align="center">
  <tr>
    <td height="31"><label for="uname">Username:</label></td>
    <td> </td>
    <td><input type="text" id="uname" value="Enter username" name="uname"/></td>
  </tr>
  <tr>
    <td height="31"><label for="pass">Password:</label></td>
    <td> </td>
    <td><input type="password" id="pass" name="pass" /></td>
  </tr>
  <tr align="center">
    <td colspan="3"><input type="submit" value="Sign in" /></td>
    </tr>
  <tr>
    <td colspan="3">{{ errmsg }}</td>
  </tr>
</table>
</form>
</div>
{% endblock %}
```

Figure 6.3 login.html

Using login.html page user can input their gtalk username and password and log in to the application. After successfully logged in to the application user can have access to their friend list and by clicking on friend name user can start chatting with their friends. For further information about user interface of the application look in to the appendix section. It contains screenshots of the application.

Now, we will see the implementation for validating user, getting friend list, receive message and send message.

- **Validating user**

```
def log_in(uname,password):
        cl = xmpp.Client('gmail.com')
        if cl.connect( server=('talk.google.com',5222)) == "":
                print "not connected"
                msg="Gtalk server is not responding..."
                f=0
        else:
                if cl.auth(uname,password,'') == None:
                        print "Authentication failed."
                        msg="user's credentials not matching."
                        f=0
                else:
                        f=1
                        main(uname,password)
        return f
```

Figure 6.4 validating user

Figure 6.4 shows the code for log_in function. This function validate user according to credential of gtalk account. It takes uname (username) and password as an argument and using xmppy library it connects to “talk.gtalk.com” server on port 5222. It returns 1 if username and password is correct otherwise 0.

- **Get friend list**

As shown in Figure 6.5, the function RosterIqHandler and retFriend gives user friend list. The RosterIqHandler is a function of xmppy library. This function is override and it appends the friend in to the global friend variable. The global friend variable is return by the function retFriend.

```
def RosterIqHandler(dis,stanza):
    global friend
    for item in stanza.getTag('query').getTags('item'):
        jid=item.getAttr('jid')
        if item.getAttr('subscription')=='remove':
            if self._data.has_key(jid): del self._data[jid]
            raise NodeProcessed
        print "*"*100
        friend.append(jid)
        print jid
        print "*"*100
    raise NodeProcessed

def retFriend():
         global friend
         print "retFriend"
         print friend
         return friend
```

Figure 6.5 Get friend list

- **Send message**

The following Figure 6.6 shows code for send message to a friend. When user click on send button in message box, according to the recipient it send message. Here, you can see for each user it creates a new process. So, it wouldn't conflicts with other user send function.

```
def ajax_send( request ):
    print "function call"
    rec = request.GET.get('rec','')
    print rec
    if request.is_ajax():
        q = request.GET.get('q','')
        print q
        data=send_msg()
        if q is not None:
            data.append("me:" + q)
            p = Process(target=gtalk_send, args=(q,rec,))
            p.start()
            p.join()
            print data
            template = 'results.html'
            return render_to_response( template, {"sender":data},
                                       context_instance = RequestContext( request ) )
```

Figure 6.6 send message

- **Receive message**

```
def messageCB(conn,msg):
   global msgList
   print "Sender: " + str(msg.getFrom())
   print "Content: " + str(msg.getBody())
   sender=str(msg.getFrom()).split('@')[0]
   msgList.append(sender+":"+str(msg.getBody()))
   print msgList

def send_msg():
    global msgList
    return msgList

def StepOn(conn):
   try:
        conn.Process(1)
   except KeyboardInterrupt:
        return 0
   return 1

def GoOn(conn):
        while StepOn(conn):
                pass
def main(login,pwd):

        cl = xmpp.Client('gmail.com')

        if cl.connect( server=('talk.google.com',5222)) == "":
                print "not connected"
                sys.exit(0)

        if cl.auth(login,pwd,'ronak') == None:
                print "authentication failed"
                sys.exit(0)
        cl.RegisterHandler('message', messageCB)
        cl.RegisterHandler('presence',presenceHandler)
        cl.RegisterHandler('iq',RosterIqHandler)
        cl.sendInitPresence()
        thread.start_new_thread( GoOn, (cl,) )
        retFriend()
```

Figure 6.7 receive message

This function receive message whenever other user send message and it displays message in chat box. The receiving process will continuous run after user logged in and it runs until user sign out from his account. For every user it creates a new thread of process. So, there is a separate receiving process is running for each user.

6.1.1 Amazon EC2 Instance

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that allows you to launch and manage Linux/UNIX and Windows server instances. To create an Amazon Instance, a developer needs to sign up for Amazon EC2 services. After sign in to EC2 account, we can launch an instance as shown in figure.

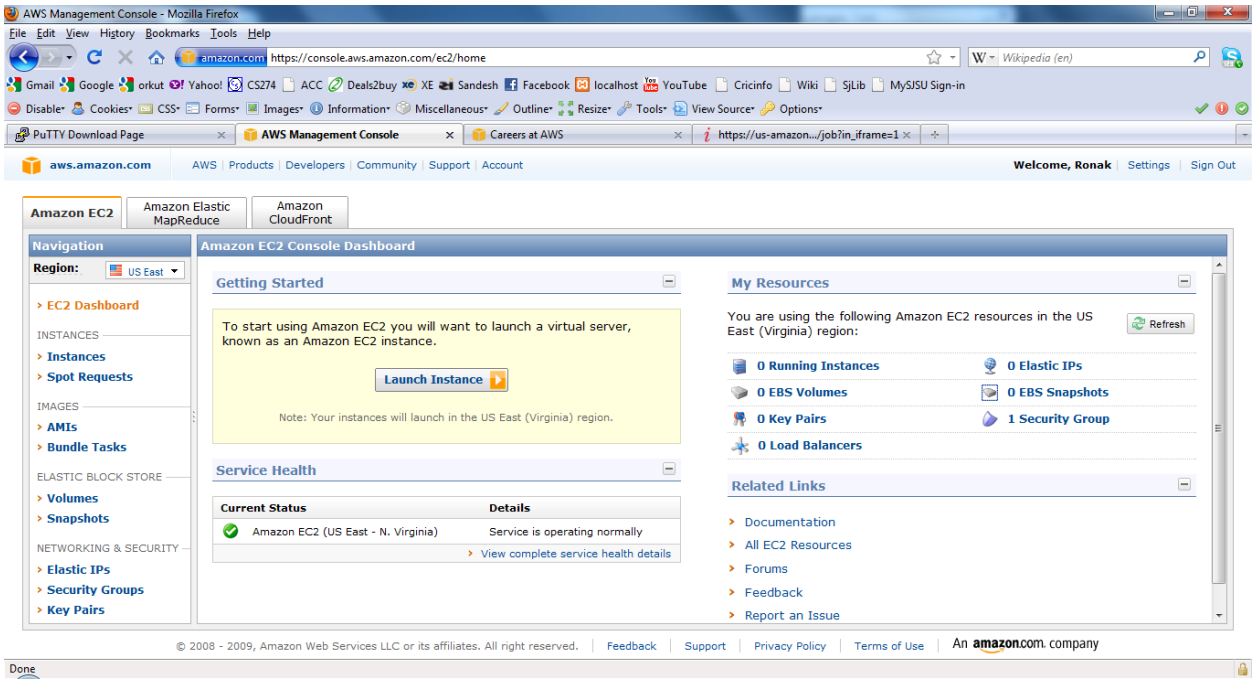


Figure 6.8 Amazon EC2 Instance

To explain whole process for launch and configure instance is out of scope for this report. You can refer “Amazon EC2 Getting Started Guide” [8].

Figure 6.9 shows the running Amazon EC2 Instance. After successfully started instance we can connect it through putty (Figure 6.10) and installed required software i.e python and django. Also, we can transfer application files to the server and run it on server. If it runs successfully, it gives a URL. Using that URL we can access application.

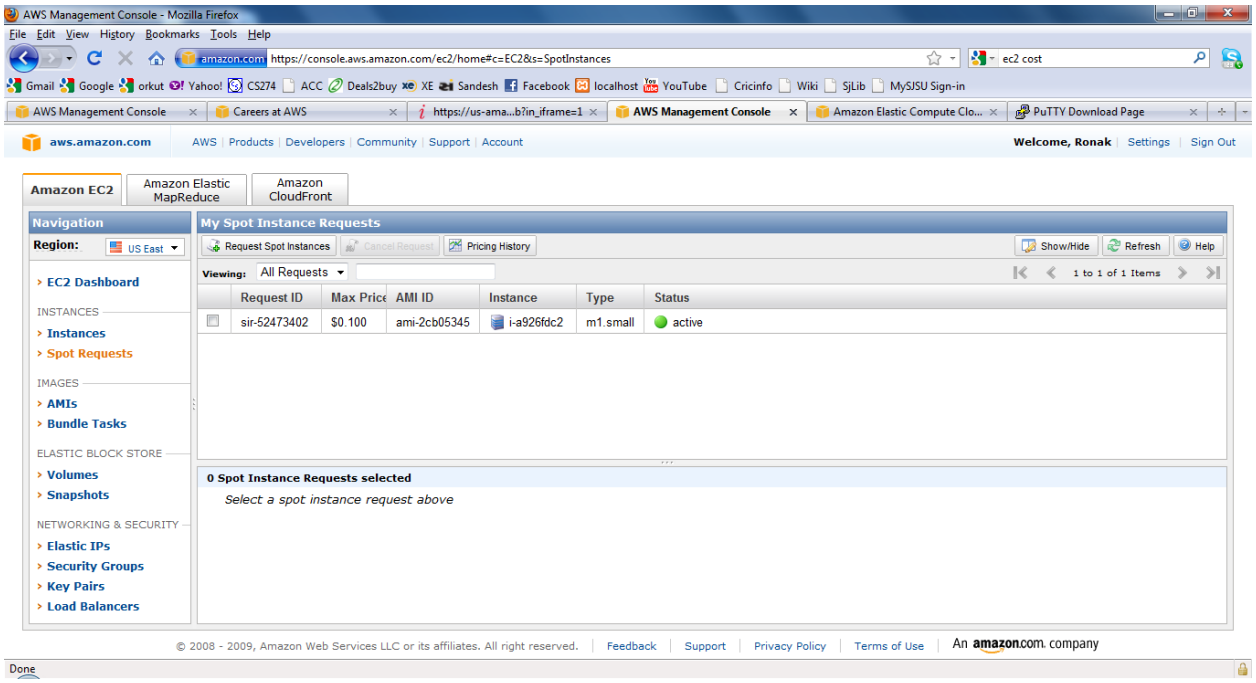


Figure 6.9 running Amazon EC2 Instance

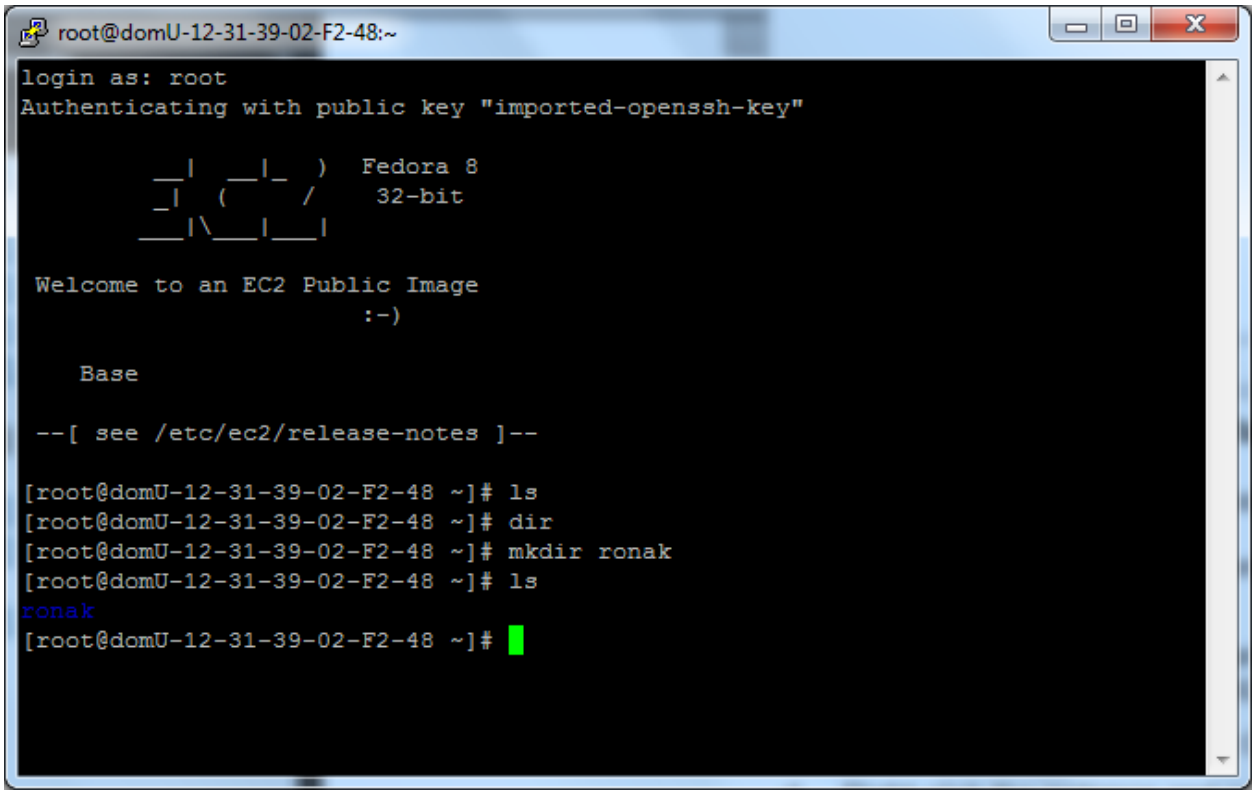


Figure 6.10 established connection to instance using putty.

6.1.2 Integrate Application with Facebook

To integrate Application with Facebook, first we need to make Facebook Application and then edit its configuration as shown in Figure 6.11.

In that we need to add site URL. This URL is an URL which we get from Amazon EC2 Instance.

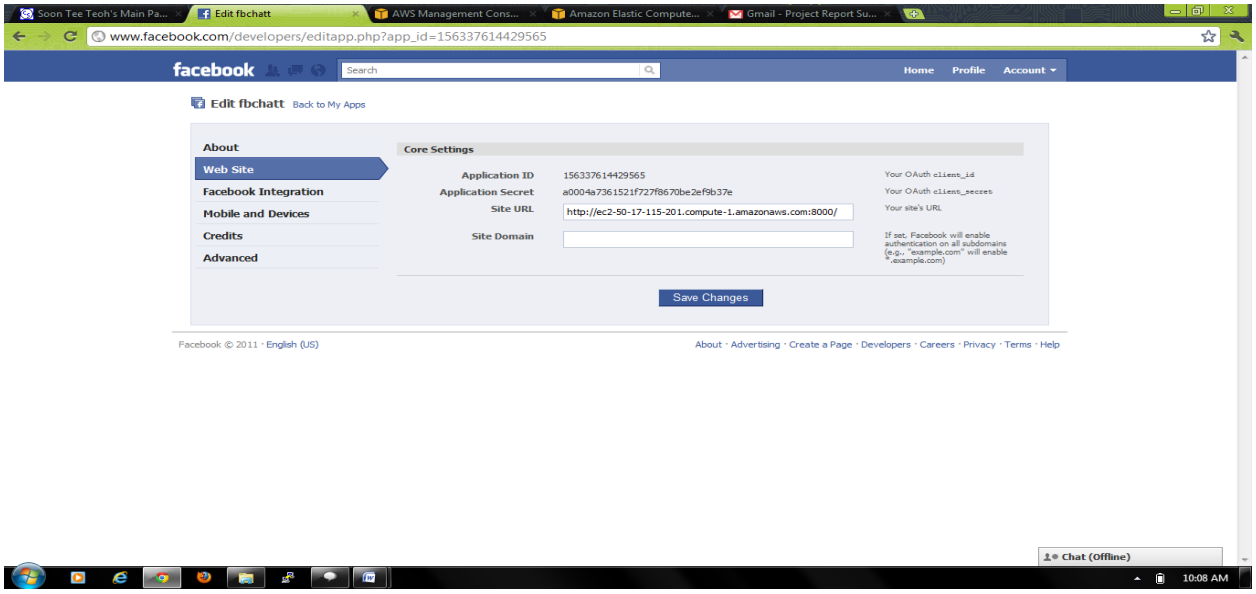


Figure 6.11 configure Facebook Application 1

Also, we need to add that URL in to Facebook Integration part as shown in Figure 6.12. In this part add canvas URL and tab URL.

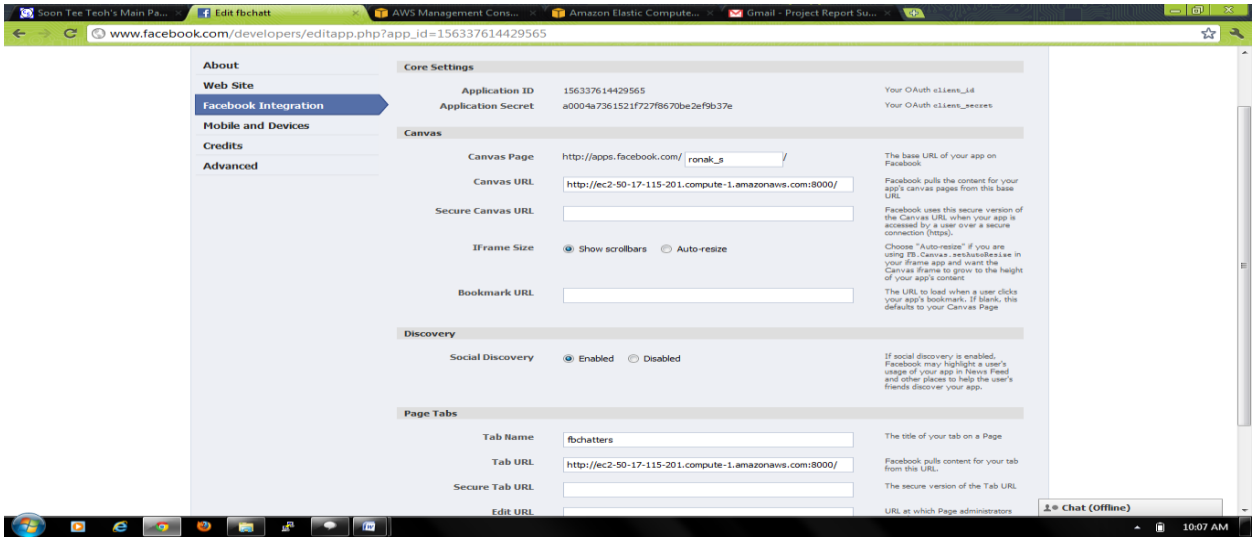


Figure 6.12 configure Facebook Application 2

6.2 Firefox plug-in

As discussed in chapter 5, there are two main files for Firefox plug-in:

1) install.rdf

2) chrome.manifest.

1) install.rdf

```
<?xml version="1.0"?>
<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:em="http://www.mozilla.org/2004/em-rdf#">
  <Description about="urn:mozilla:install-manifest">
    <em:id>ronak187@fbchatters.com</em:id>
    <em:name>FBchatters!</em:name>
    <em:version>1.00</em:version>
    <em:description>Open FBchatters! panel to left, bottom or right.</em:description>
    <em:creator>Ronak Shah</em:creator>
    <em:homepageURL>http://www.google.com</em:homepageURL>
    <!-- Firefox -->
    <em:targetApplication>
      <Description>
        <em:id>{ec8030f7-c20a-464f-9b0e-13a3a9e97384}</em:id>
        <em:minVersion>1.5</em:minVersion>
        <em:maxVersion>4.5</em:maxVersion>
      </Description>
    </em:targetApplication>
  </Description>
</RDF>
```

Figure 6.2.1 install.rdf

Install.rdf gives information about plug-in name, version, description, creator and which firefox version is compatible with this plug-in. It is written in RDF format which is similar to XML format.

2) chrome.manifest

```
overlay chrome://browser/content/browser.xul    chrome://fbchatters/content/overlay.xul
content fbchatters  chrome/
```

Figure 6.2.2 chrome.manifest

chrome.manifest gives information about the contents of plug-in. It also gives information about which type of content it has and where they are located.

As discussed in topic 5.2 the content directory contains actual implementation files for plug-in. It contains .xul files. These files define the structure of plug-in. The .js is a javascript file that gives interactivity to plug-in and the .css file defines the cascading style sheet for plug-in.

For example, the overlay.xul file defines the structure of plug-in.

```
<?xml version="1.0"?>
<!DOCTYPE window SYSTEM "chrome://browser/locale/browser.dtd" >
<!-- ************ Overlay ************ -->
<overlay id="splitpannelOverlay" xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
<script type="application/x-javascript" src="chrome://fbchatters/content/overlay.js"/>
<popup id="contentAreaContextMenu">
  <menuitem
        id="splitpannel-menuitem"
        label="Fbchatters!"
        tooltiptext="Open Fbchatters!"
        onclick="splitpannel.click(event);"/>
  <menu label="Fbchatters!" id="splitpannel-submenu">
    <menupopup>
        <menuitem label="Left"
              oncommand="splitpannel.toggle('http://ec2-50-17-115-201.compute-1.amazonaws.com:8000/', true, 'left');"/>
        <menuitem label="Bottom"
              oncommand="splitpannel.toggle('http://ec2-50-17-115-201.compute-1.amazonaws.com:8000/', true, 'bottom');"/>
        <menuitem label="Right"
              oncommand="splitpannel.toggle('http://ec2-50-17-115-201.compute-1.amazonaws.com:8000/', true, 'right');"/>
    </menupopup>
  </menu>
</popup>
<window id="main-window">
  <vbox id="splitpannel-box" hidden="true" persist="height width">
  <hbox  class="splitpannel-bar">
    <toolbar flex="1"
         id="splitpannel-nav-bar" class="chromeclass-toolbar splitpannel"
         mode="icons" iconsize="small" context="toolbar-context-menu">
    <toolbarbutton id="splitpannel-reload-button" disabled="false" hidden="true"
         class="toolbarbutton-splitpannel chromeclass-toolbar-additional splitpannel"
         oncommand="window.document.getElementById('splitpannel-browser').reload(); splitpannel.title();"
         label="&reloadCmd.label;"
         tooltiptext="&reloadButton.tooltip;"/>
    <toolbarbutton id="splitpannel-stop-button" disabled="false" hidden="false"
         oncommand="window.document.getElementById('splitpannel-browser').stop(); splitpannel.title();"
         class="toolbarbutton-splitpannel chromeclass-toolbar-additional splitpannel"
         label="&stopCmd.label;"
         tooltiptext="&stopButton.tooltip;"/>
</window>
</overlay>
```

Figure 6.2.3 overlay.xul

7. Feedback of users

The outcome of this project is a Facebook application and a Mozilla Firefox plug-in. Both provide same functionality. To access application FBchatters users need to log in their Facebook account. While using plug-in you can chat with gtalk user without sign in to the Facebook. Both have their advantage and drawback. To find which solution is best I decided to give both solutions to the group of six students and take their feedback. In next section we will see the feedback of students.

The following Table discusses comparison of each solution.

	Facebook Application FBchatters.	**Mozilla Firefox plug-in FBchatters.**
1	It can be accessible from Facebook Account only.	It can be accessible without sign in to the Facebook account.
2	It works in any browser and on any platform.	It can work only in Firefox because it is a firefox plug-in. But it is a platform independent.
3	While using this application; user can't use other part of Facebook. If they want to use they have to leave this application page.	User can access whole Facebook features because this plug-in divides browser window in two parts. One part is Facebook and other part contains FBchatters website.
4	Sometimes it receives message late because it runs under Facebook.	This is not a problem because it is separate from the Facebook.
5	The browser version doesn't affect this application.	If Firefox version doesn't compatible to plug-in then it won't work.

Table 7.1 Comparison of Facebook application and Firefox plug-in

Feedback from six students:

There are two outcomes of this project. To decide which one is better in use I have selected group of six students and gave them both outcome to test. According to their review:

- Two students like Facebook application. Because it is accessible within Facebook without installing any plug-in for it and it also run on any browser.
- One student was neutral. He liked Facebook Application and also plug-in.
- Three students were like Firefox plug-in. Because after installing plug-in they can chat to gtalk user while browsing with any website.

8. Conclusion

Facebook is the most used social networking web site. It changed the way of sharing information over the internet. Facebook users can add their friends, family members and they can share messages, photos, events, etc. Also Google talk is a major messaging service provider over the internet. Internet users like to do chat on Facebook and Gtalk with their friends and there are a large number of users who use both services at a time. Those people feel boring to switch window for chatting. FBchatters addressed to that problem and provides a solution by offering one place to chat with both users.

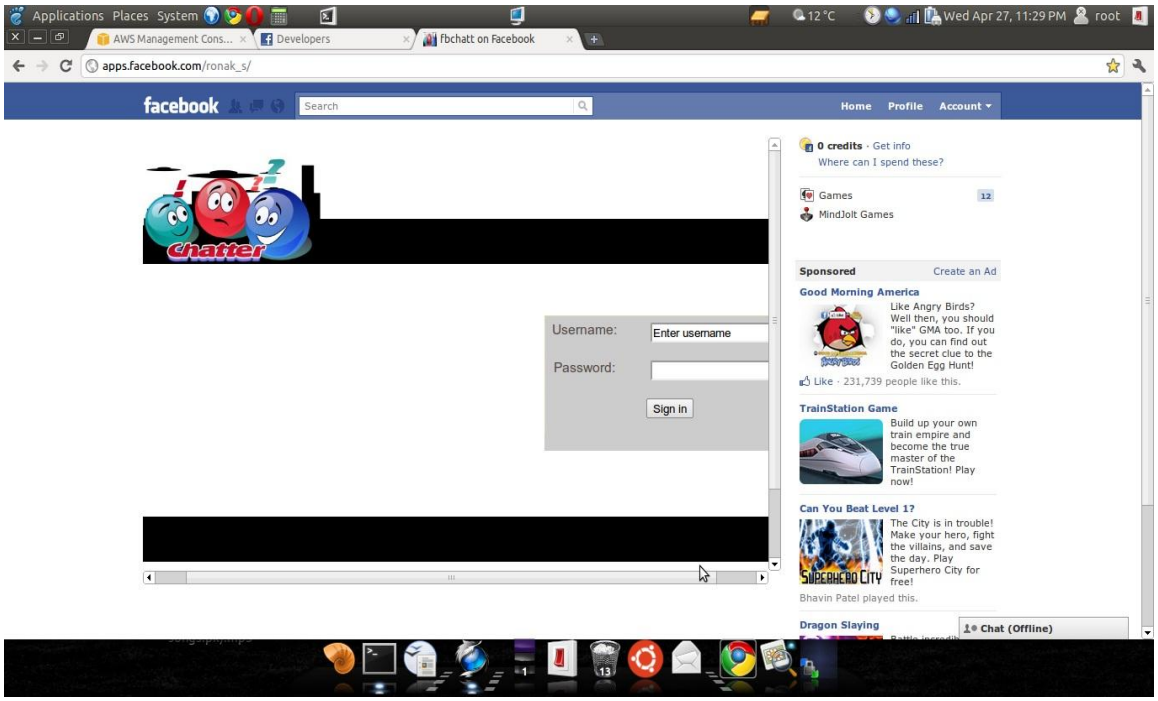
Moreover, in this project two solutions are provided. Either you can use Facebook application or you can use Firefox plug-in. Both solutions are tested on Windows, Linux, Mac OS and also on different browsers. It works perfectly in any situation. But the problem with plug-in is, it can work with Firefox only because it is not supported on other browser.

A group of 6 students was selected for to test both solutions. Based on their feedback both solutions worked as expected. From them 3 students preferred plug-in solution was best for them. One student was neutral; he liked both options and 2 students like Facebook application. According to their usage patterns, it was found that they were able to communicate with Facebook and Gtalk using FBchatters.

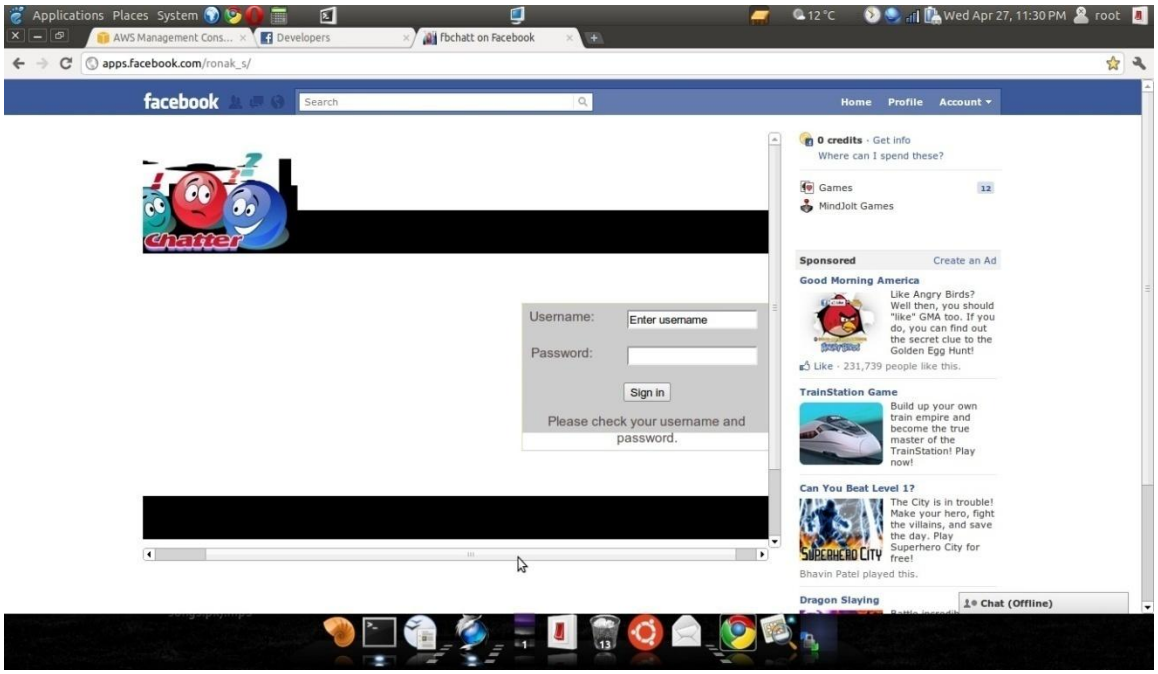
Appendix

Web application UI

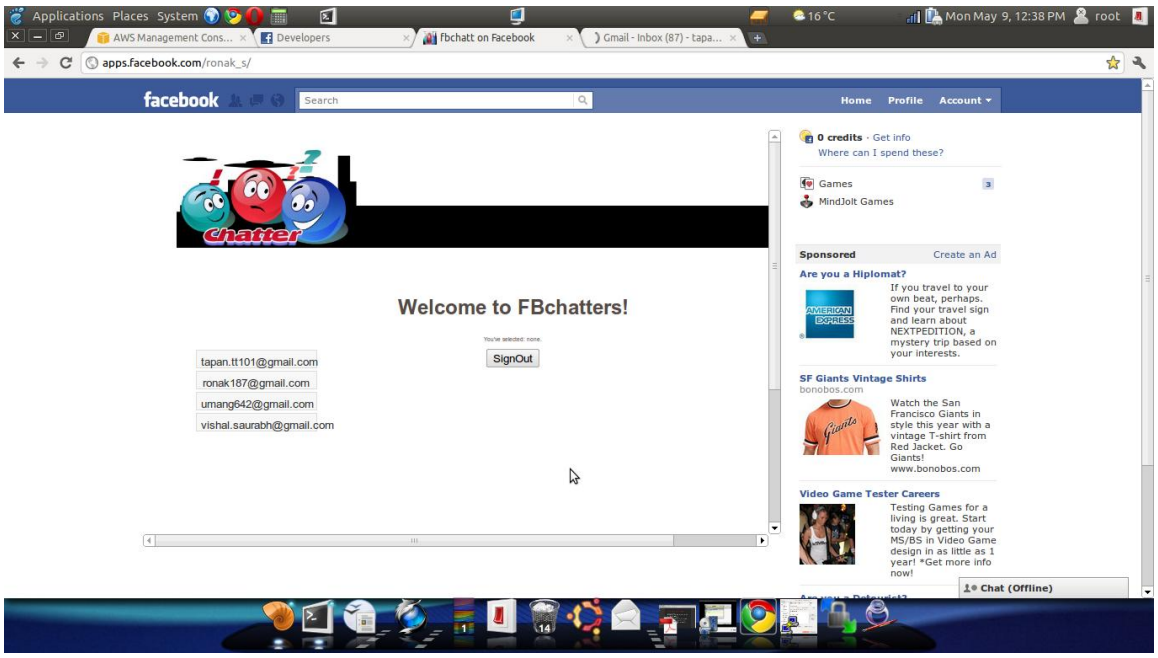
1) Login page



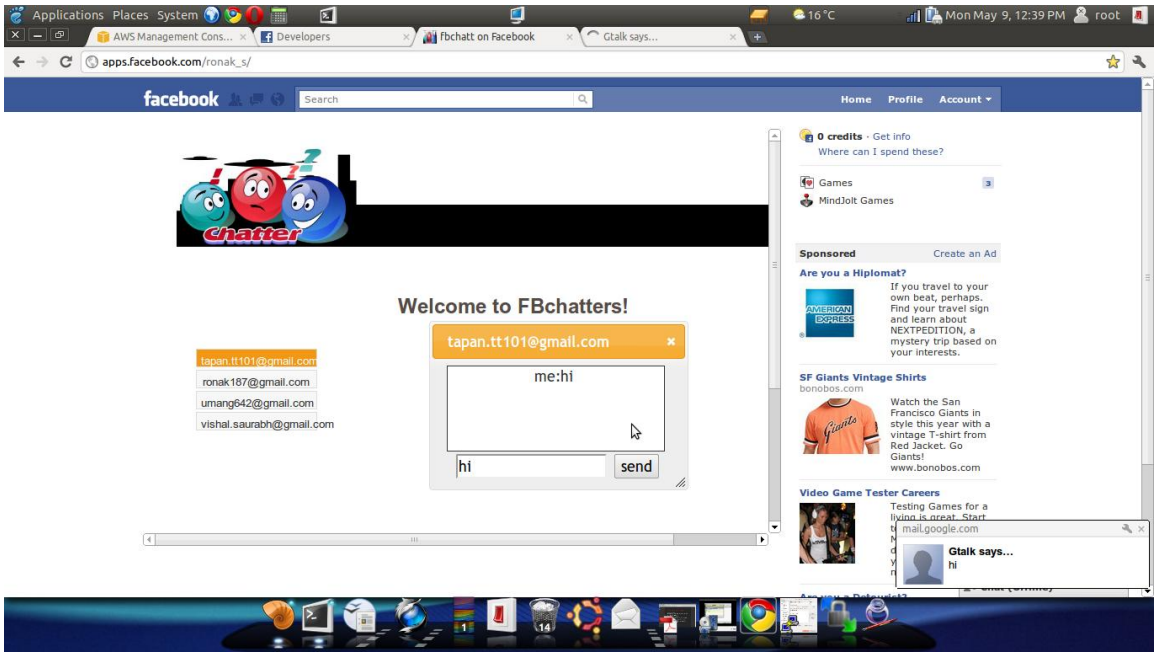
2) Validation error message



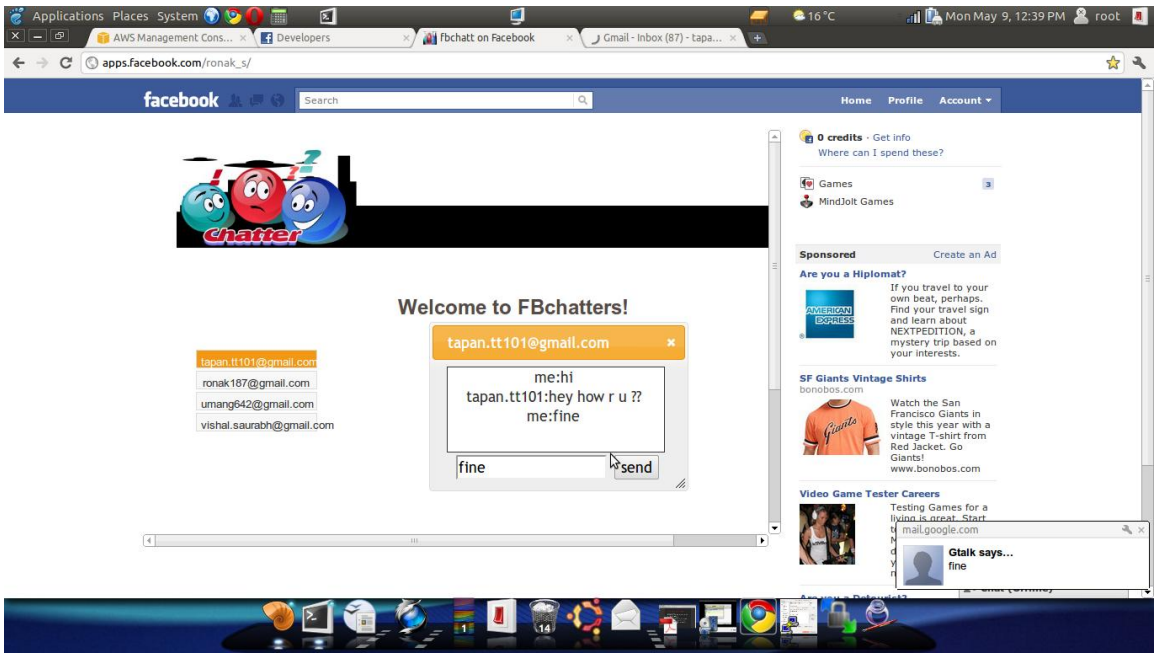
3) Home page (displays friend list and sign out button)



4) Chatbox

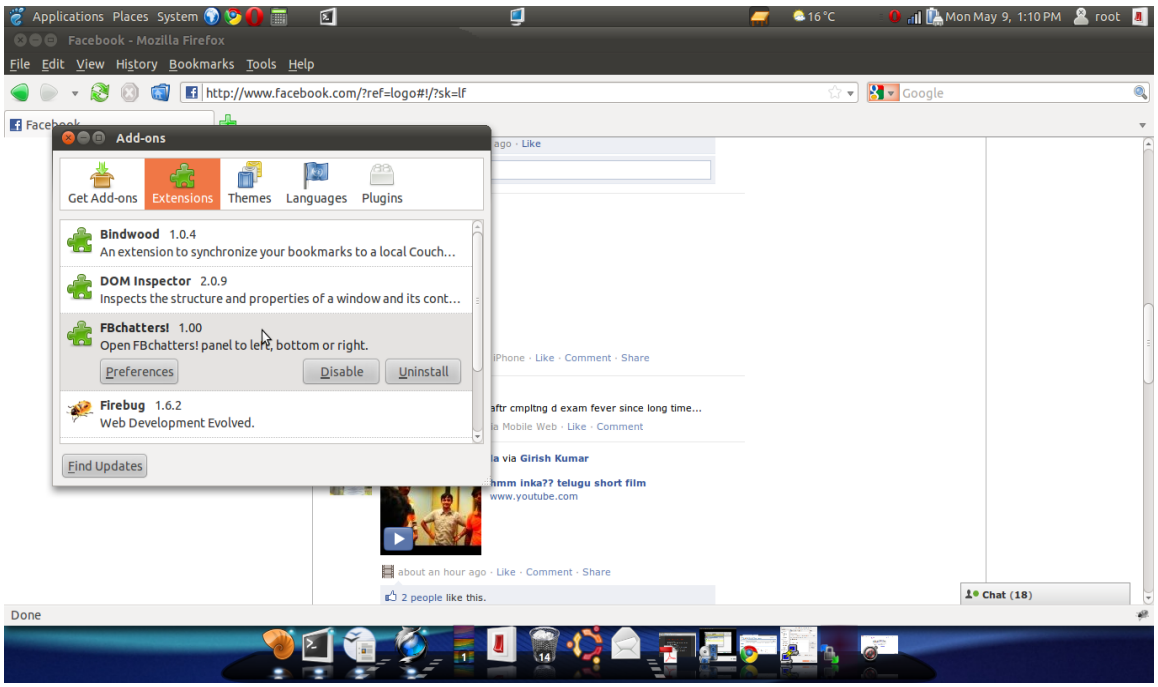


5) Send – Receive

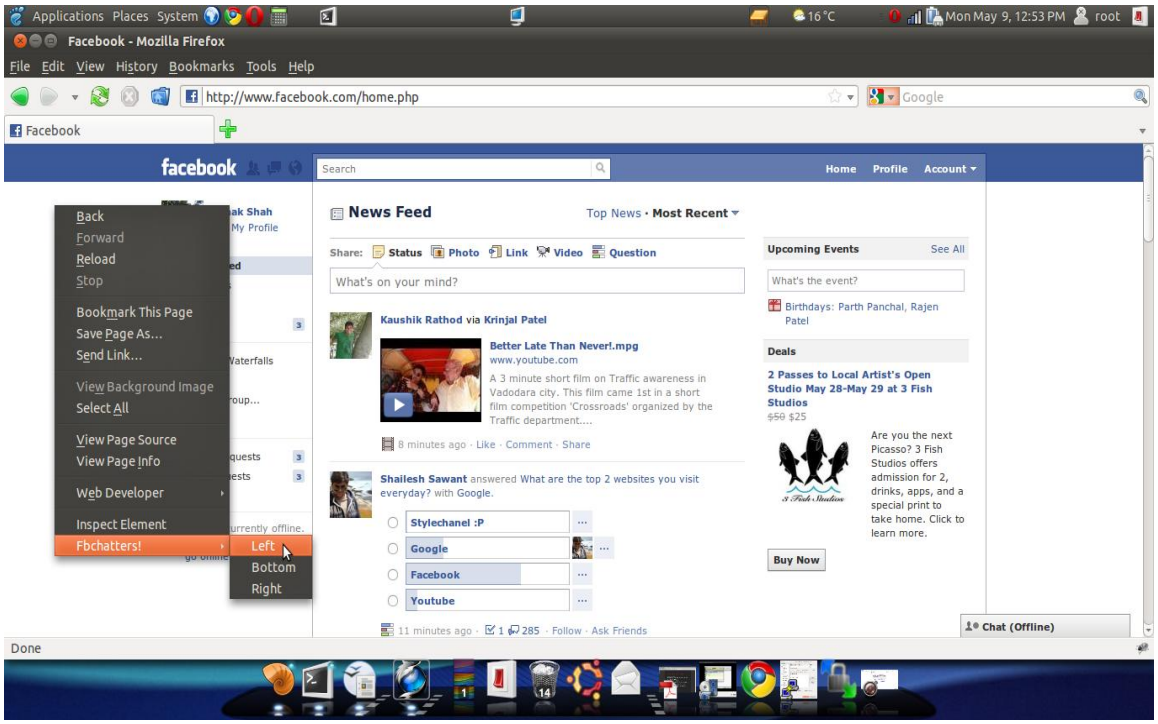


Firefox plug-in screenshots

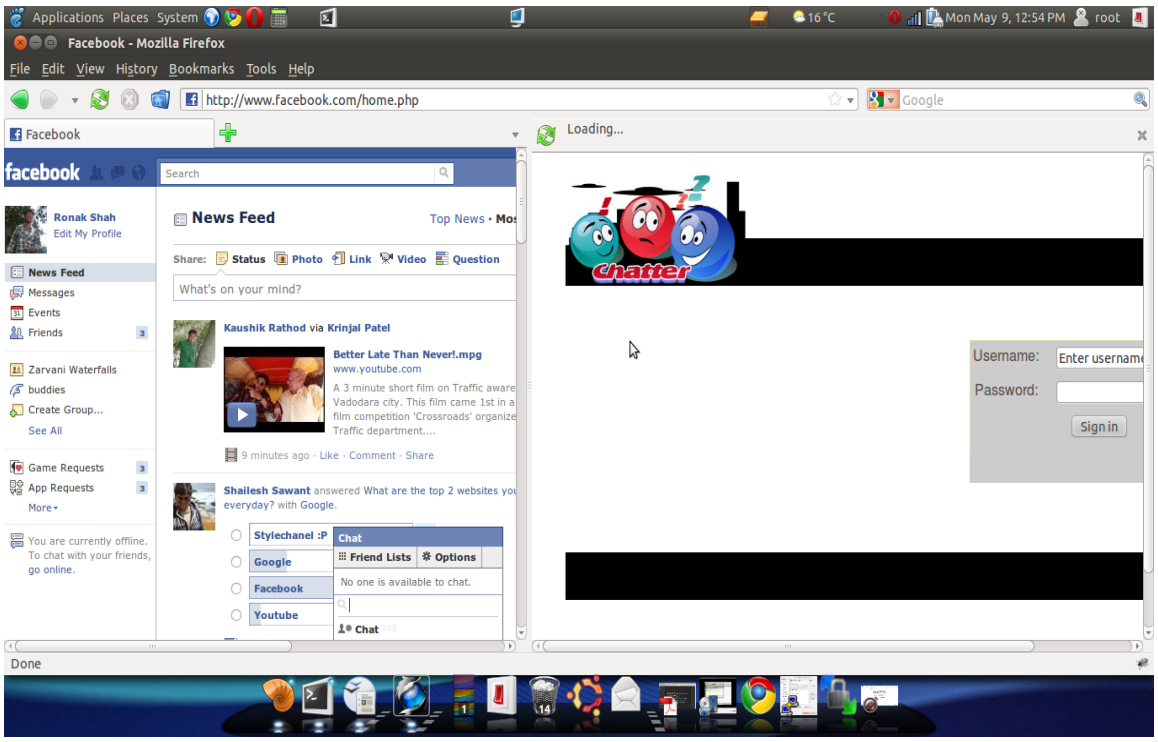
1) Install plug-in



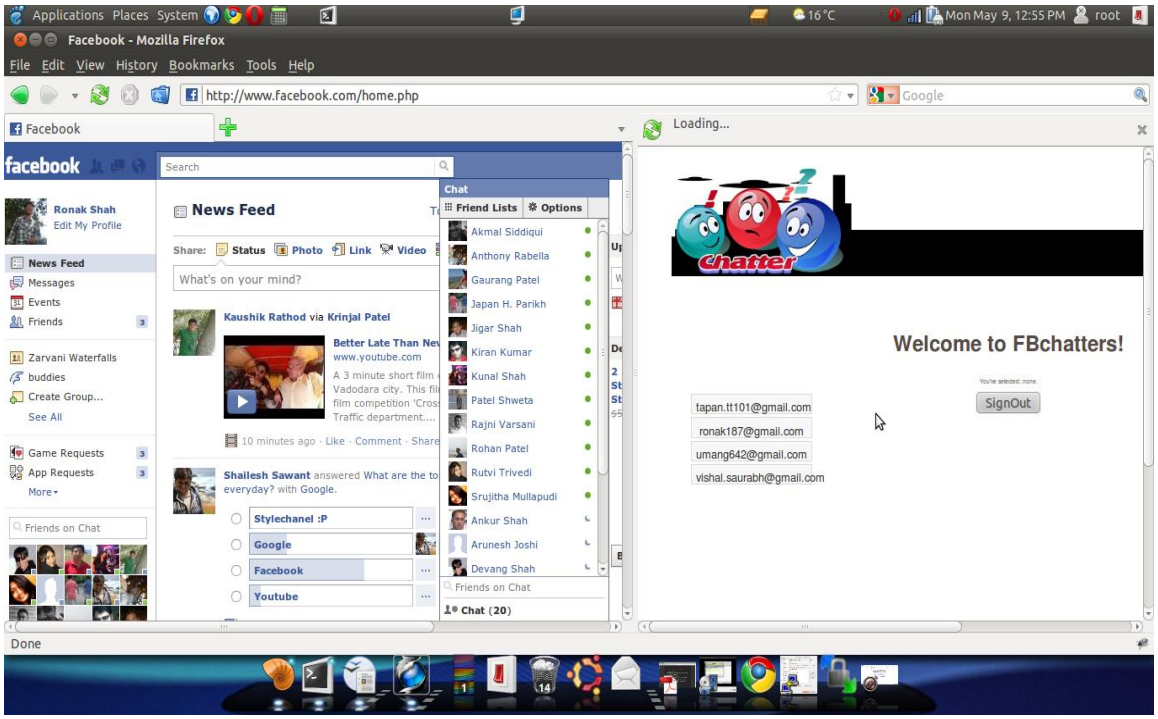
2) Access Plug-in from right-click on window



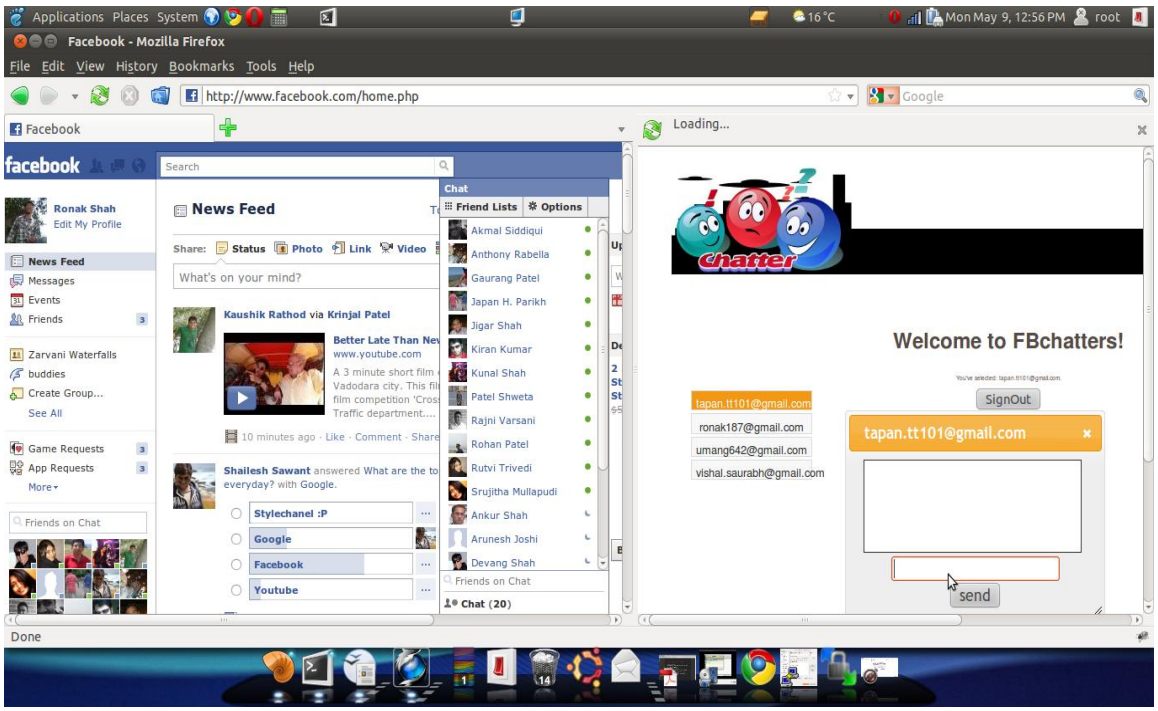
3) Plug-in login page (appear on right side of window)



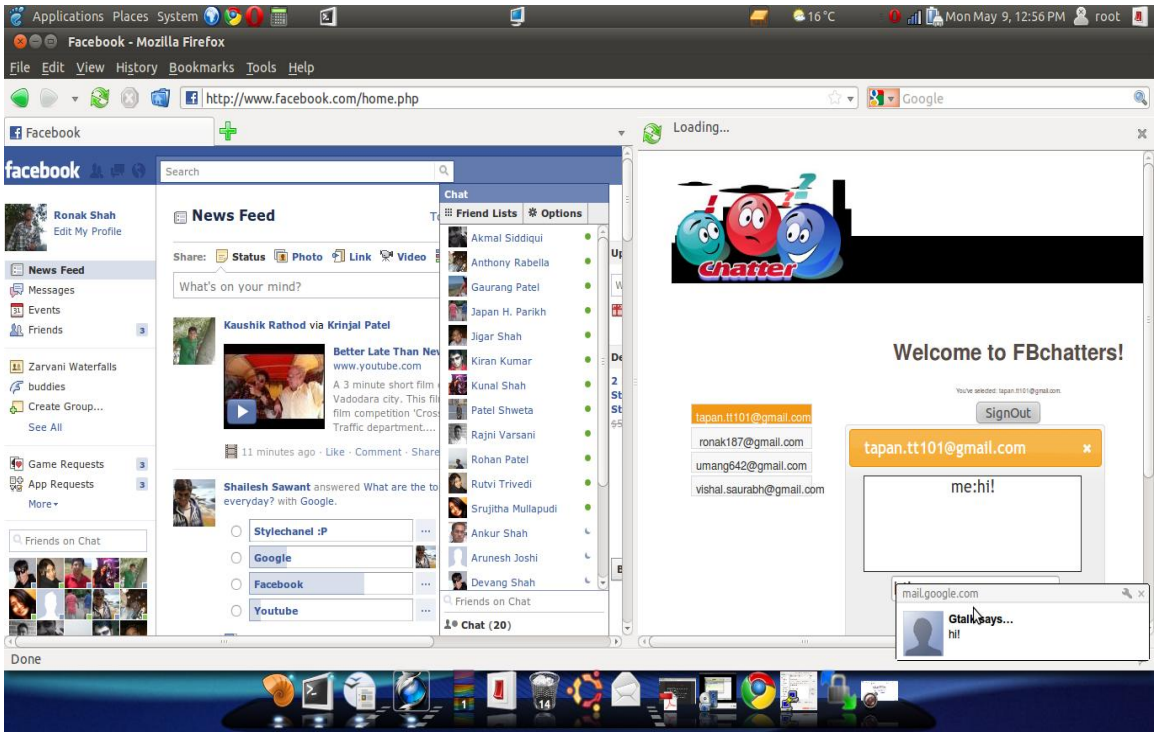
4) Plug-in home page



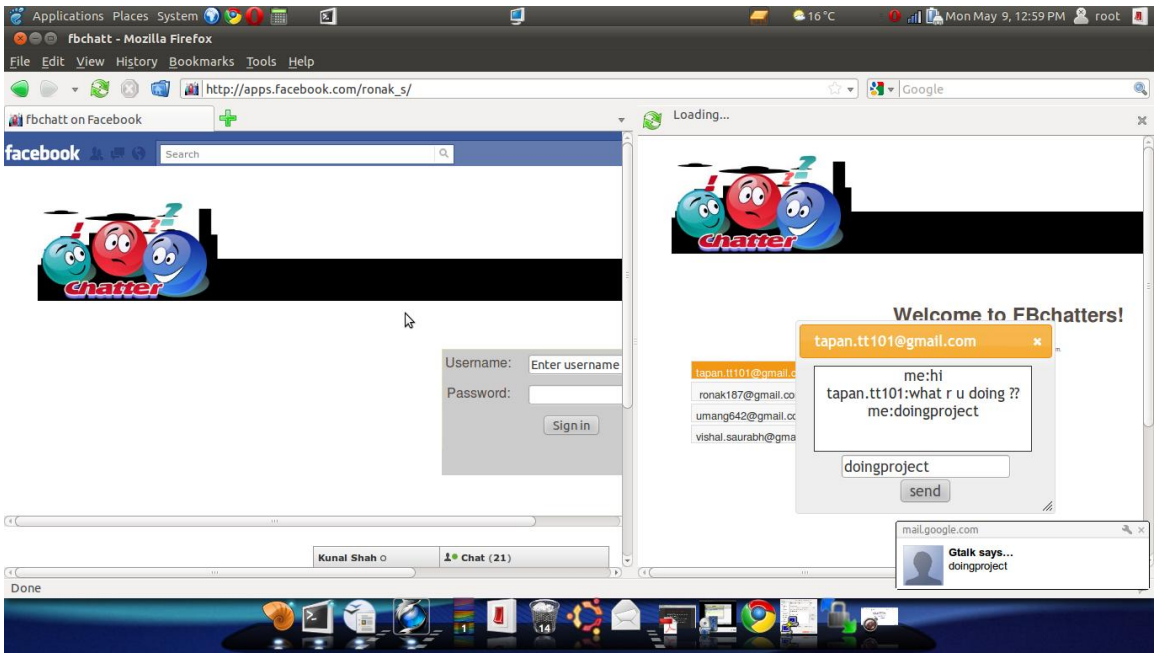
5) Plug-in chat box



6) Send message through plug-in chatbox



7) Send – Receive using plugin chatbox



References

[1] Statistics http://www.facebook.com/press/info.php?statistics

[2] Facebook http://en.wikipedia.org/wiki/Facebook

[3] Extensible Messaging and Presence Protocol

http://en.wikipedia.org/wiki/Extensible_Messaging_and_Presence_Protocol

[4] Amazon Machine Image http://en.wikipedia.org/wiki/Amazon_Machine_Image

[5] Django FAQ: General http://docs.djangoproject.com/en/dev/faq/general/#django-appears-to-be-a-mvc-framework-but-you-call-the-controller-the-view-and-the-view-the-template-how-come-you-don-t-use-the-standard-names

[6] Mod_python http://en.wikipedia.org/wiki/Mod_python

[7] Amazon EC2 http://aws.amazon.com/ec2/

[8] Amazon Elastic Compute Cloud (EC2) Documentation

http://aws.amazon.com/documentation/ec2/