woOps! Implementation Report

Group Name: Noobivore

Course: Internet Technology (Msc.IT)

Group Members

Ugochi Ugbomeh	09131120913112u@student.gla.ac.uk
Theresa T. Afolayan	10018941001894a@student.gla.ac.uk
Augustine Kwanashie	10044791004479k@student.gla.ac.uk
Gergo Maksa	10009991000999m@student.gla.ac.uk

ABSTRACT

This report will elaborate on the web application "woOps!" that bares similarities to twitter; the paper will further discuss the framework, architecture, design which adds to its development. The framework will be built with Django [1] that is written in python programming language. The application basically allows users post, update and view messages sent to each other.

AIM OF APPLICATION

The application is a social networking widget sort of like twitter that allows user interaction within a site (WF 15). It is built on the Django Framework, which has been used to construct other services. The main purpose of this web application is to provide a customized interface for users to post and view messages in real-time. The system should be pluggable to any website, providing chatting functionality to users.

Assumptions:

- There can be anonymous viewers within the website, who can view messages on the "woOPs!" application. However they cannot post comments or messages.
- The application is not age dependent since it can be used on any website.
- The application is rendered in English.
- The users of this application created within the site, are restricted to communicate with only the users on the site in which the app is installed. E.g. woOps! Users' on bingo website cannot communicate with woOps! Users' on tesco website.

Constraints of the project

WoOps! Was designed and implemented based on the following constraints:

- The application was built within the time frame of four weeks with the end-date being 18th March'2011.
- It was designed and implemented by four people who divided the development process into 4 parts based on their strengths respectively.

The design goals and objectives of the application include:

- To build a pluggable widget that allows quick and easy integration within websites.
- To provide real-time access to posted messages at acceptable frequencies.
- To provide access to chat information for external clients through RSS feeds.
- To build an application that is not browser dependent that means that the application should perform all functionalities on any browser (Opera, Mozilla, Internet Explorer, Safari).

Specifically, the application performs the following functions:

- Users are allowed to register with the application.
- Users can login into the system.
- Users can update their profiles. Again, this should be easily pluggable to the websites user management system.

- Users can post messages on the application (we chose a maximum of 200 characters).
- Users will receive a continuous live feed of current posts from other connected users.
- Client systems should be able to read RSS feeds.

APPLICATION ARCHITECTURE

The application is designed to have a 3-tier architecture (fig.1) comprising of:

- ***Front-end clients:*** the site can be viewed by the user with desktop browsers like Mozilla, Google chrome, safari, opera, Microsoft internet explorer and RSS readers which links to sites like Google...
- ***Middleware web server:*** responsible for hosting HTML (Hypertext Markup Language) files which is a building block for viewing the web-page; running server-side programs like python, and responding to HTTP(Hypertext Transfer Protocol) requests from clients.
- ***Back-end database management system:*** that will host and in some cases, process the system data.

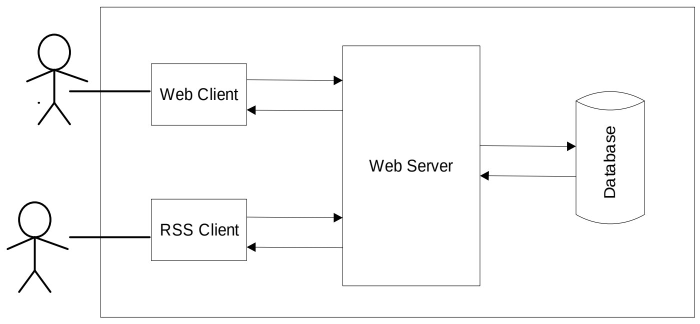


Fig. 1: 3-tier architecture for woOps! Application

The application stores data relating to users, their interaction and messages posted. The main entities in the data model are:

- *Users:* This entity stores basic user information and authentication information (table 1.0).
- *Messages:* This entity represents messages posted by users. It will contain the message content, date and a reference to the user that posted it (table 1.1).
- *Friends:* This relationship object is responsible for the interaction between users (friend requests). It has to attributes that calls the user_id (table 1.2).
- ***PMsg:*** This entity object is responsible for giving users the opportunity to send personal messages to each other. It has two foreign keys that call the user_id (table 1.3)
- A one-to-many relationship is established between the User and Message entity. This will ensure data integrity by making sure all messages created have a valid user associated with it.
- A one-to-many relationship is also established between the User and personal messages (pMsg) entities. This will ensure that a user can send and receive multiple messages to and from other users.
- A many-to-many "Friend" relationship (which will also translate to an implementation object) will be established between users and other users.

Attributes	**Description**	**Type**	**Constraint**
Message_id	Autogenerated	VARCHAR(10)	Primary Key
Message_User_id	Link to user table	VARCHAR(30)	Foreign Key
Content	User posts	VARCHAR(200)	Not Null
Date	Past date	DATE	Not Null
Time	Post time	TIME	Not Null

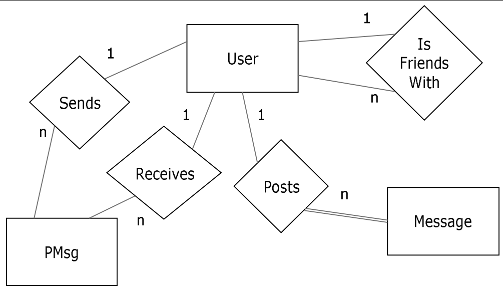


Fig. 2: Entity Relationship Diagram for the Chirper App

Table 1.0 shows the entity (User) attributes types and constraints.

Table 1.2 shows the entity (Friends) attributes types and constraints

Attributes	Description	Type	Constraint
fromCh	Link to user	VARCHAR(30)	Foreign Key
toCh	Link to user	VARCHAR(30)	Foreign key
message	Personal message	VARCHAR(200)	Not Null
dateSent	The day the message was sent	DATE	Not Null

Table 1.3 shows the entity (pMsg) attributes types and constraints

Attributes	Description	Type	Constraint
User_id	Autogenerated	VARCHAR(30)	Primary Key
First_name	First name	VARCHAR(20)	Not Null
Bio	Biography	VARCHAR(250)	
Password	Password	VARCHAR(25)	Not Null
E-mail	E-mail	VARCHAR(50)	Not Null
Chirp_date	Account creation	DATE	Not Null
Other_names	Last name	VARCHAR(20)	

Table 1.1 shows the entity (Message) attributes types and constraints

We are dealing with a web-based application that does not naturally persist state between requests; for this reason, we will employ Django's inbuilt session management utility, to maintain user information across requests and ensure that only valid users will be allowed to post or view (depending on admin settings) posts made on the chat application. We hold user's id numbers, their login credentials and their login-time-in session variables that will persist between requests.

Although all posted messages will be stored in the database, only a certain amount of the most recent posts will be displayed upon loading or refreshing, and subsequent Ajax updates will be incremental. This will prevent performance problems when the message table grows.

The system will be guaranteed of secure, reliable code in user login and other non-functional requirements by letting the Django framework handle the session management. A typical alternative could be coding this functionality using python directly or any other server-side scripting language like PHP. However, this would have meant more man-hours and no guarantees that the produced code would be reliable and secure. Due to time

Attributes	Description	Type	Constraint
chirper1	Links to user	VARCHAR(30)	Foreign Key
chirper2	Links to user	VARCHAR(30)	Foreign Key

and resource constraints, it was not a good idea to implement low-level functionality (like saving to, and reading from databases) manually. Using a web framework allows us to concentrate on the more functional requirements and leave some of the low level stuff to the framework to handle.

MESSAGE PASSING

Please refer to Appendix A while reading this section.

The clients connect to the middleware web server, mainly by HTTP GET and POST requests. These requests may be made upon loading/refreshing of web pages or in real-time by background JavaScript processes (AJAX).

The RSS readers will typically be other web servers looking to add the chatting content to their served pages; however, any windows or web based application capable of reading RSS feeds can be used. We expect the RSS requests to be simple HTTP GET requests on specified URLs that will provide.

The web server receives HTTP requests, process python code if necessary, thus connecting to the DBMS, and respond with HTML files (or RSS-XML in the case of RSS requests) to the requesting clients.

The database server receives DML (Database Manipulation Language) commands and SQL queries and processes them accordingly.

The messages sent and received between components in the various tiers are:

- ***HTTP POST requests:*** this is used to send information from the browser to the web server in a secure and reliable fashion. The server will retrieve the post information and process them accordingly.
- ***HTTP GET requests:*** this is used to send parameters to the web server for processing. It will be used by RSS clients to send authentication

information to the RSS writer to validate before generating the appropriate XML files.

For the real-time updates of the posted messages, HTTP GET requests will also be made this time from JavaScript and not due to a refresh of form submission. This request will still trigger server-side code to be executed and HTML results delivered.

Due to the fact that the web server and database will be on the Django framework, access to the database which will be handled internally.

CLIENT INTERFACE

Initially, there are three main interfaces for this web application namely login, registration and message page. As a result of extended functionalities, more interfaces are created. Below are brief description of the interfaces and their interaction with the user:

- **WoOps! Default Page (WF 1):** displays messages posted by registered users which can be viewed by both registered and unregistered users. This page includes two ***login*** links located at the top and bottom of the frame, where registered users can click and log into the app through the interface (WoOps! Log-in Page). It also has a ***view more post*** link (WF 12) where users can view a set of post from given dates typed in by the user. There is a ***register free*** link that users can click on to register for the whoops chat.
- **WoOps! Registration Page (WF 2)*:*** the new user can fill in their details which include their first name, last name, e-mail, chosen user-name, password and biography; the user can also upload a picture by clicking on the ***browse*** button. Some of the details marked with asterisks (*****) must be filled in by the user. Once he/she clicks on the ***register*** button, the user will be registered as a member of the woops app. Also, user must accept the terms and condition associated with the app by marking the field directly beneath the field for terms and conditions. There is an additional link (***Already registered? Log in here***), where registered users can log into the app.
- **WoOps! Log-in Page (WF 3):** a registered user can sign in using a previously registered email and password; both fields are compulsory. There is a link that unregistered users can click (***Not a member? Register***) to register their details as new member. There is a link (***Log in as anonymous***) for general users to sign in as anonymous whereby they can only view but cannot post messages.
- **WoOps! Chat Page (User logged in) (WF 4):** this is the main page in which logged in users can view and post messages. It also has a ***view more post*** link where users can view a set of post from given dates typed in by the registered user. It includes a ***log out*** link where users can exit the application when he is done. It has ***my profile*** link where users can click to view their details. It has ***my friends*** link where user can view his list of friends, send message or even delete them. There is a ***RSS Feeds*** link (WF 13); users can click to get to the RSS page.
- **WoOps! Profile Page (WF 5):** registered users can view their details which include first name, last name, email, password, date of birth, and biography; users can also upload pictures by clicking on the ***browse*** button. User's information can be changed by editing the text field and clicking on the ***update my profile*** button. This page also includes ***chat room*** link that can be clicked on to return to the chat page. There is ***my friends*** link where you can view your list of friends, send messages or delete them. Additionally, there is ***RSS feeds*** link that leads to the RSS page.
- **Viewing someone's page and Making a friend (WF 6):** user can view any registered user's profile page. If the person is not your friend, you can make a user your friend by clicking on ***make "given name" your friend*** button. This page also includes ***chat room***, ***my friends*** and ***RSS feeds*** links which performs the same function as mentioned in profile page session above.
- **Viewing Friend Request (WF 7) and Send Message Option (WF 9):** user can view their list of friends, they can accept and reject other users; user can also send messages (WF 10) to friend as well as view messages (WF 11) sent to them. This

page also includes ***chat room***, ***my friends*** and ***RSS feeds*** links which performs the same function as mentioned in profile page session above.

- **Administrative Interface:** manages all information received from the application; these information include the user information (refer to WF 2), personal messages (WF 11), friend relationships (WF 7) and posts (WF 4). This information are located in the chirp section of the site administration (WF 14).

The users will post messages by filling a HTML form which will post messages to the web server in real-time without refreshing the web browser. This will be achieved using AJAX, a technique for making JavaScript connection to server-side scripts without causing a full page refresh. The client will have to run a continuous loop in JavaScript at a predefined frequency to request for additional posted messages from the web server.

Relying on client-side technologies may mean that we do not have total control on the functionality of the system as JavaScript may be disabled or unsupported by some browsers. In this case, the system will only be updated after a page refresh.

RESPONSE TO FEEDBACK

The feedback activity gave the team better understanding, not only on the shortcomings of the specification report, but also on the flaws of the application design. Although a lot of the feedback was not useful, we identified a lot of useful points that guided us through the implementation. Below is a list of some of the useful feedback that was noted:

- Restrictions to the field sizes on certain model fields were pointed out. This meant the users were restricted in the range of valued they could enter for email addresses and passwords. This was fixed by extending the ranges.
- Reviewers thought the search functionality should allow for posts to be searched by poster name as well as date of post. This was also implemented.
- A number of concerns about user management compatibility with the host website. Reviewers did not like the idea that users had to login separately into the application. Although this was not fixed (due to time and resource constraints) we propose a solution where the main website's user management script can call web services exposed by our application to test if the user is also in the app database. This way, if the user is, only one login will be needed and the appropriate session variables for both the host site and the application will be created.

IMPLEMENTATION NOTES

The application, like most built on the Django framework, comprises of a model component (that describes the entities within the application), a view component that implements server-side functionality in python and a presentation component responsible for handling user input (urls.py) and output (Django template system).

The objective of this design is to separate functionality into manageable modules that can be modified without affecting the entire system. Presentation templates can be modified without affecting functionality and the underlying model can be modified with minimal changes to the view component.

The **model component** consisted of a single models.py file that had, defined in it, the entities in the system. Classes like Users, Messages, Friends, PMessages etc were defined with their attributes and relationships specified. The Django framework handles the actual database implementation of these entities and is able to choose from a list of supported database management systems on which to create the tables. Handling queries, DDL and DML SQL commands is done internally by the Django ORM (Object Relational Mapper). This can greatly aid rapid development and ensure database integrity especially in projects where time and technical ability are limited.

The **view component** consists of a number of functions written in python in a single views.py file. Each function handles one specific task and comprises of the core server-side functionality of the application (a bit of validation and logic was also implemented client-side). View functions may receive extra input required for their operations and after processing, will call the appropriate template file.

The **presentation component** consists of the HTML template files responsible for displaying output, CSS style sheets responsible for formatting, JavaScript files responsible for implementing client-side functionality and other resources like images. A urls.py file also handles user input via HTTP requests by parsing through the requested URL and calling the associated view function.

URL Scheme

In the Django framework, scripts are not accessed by accessing their URLs. Rather, a URL scheme is designed where incoming URLs of incoming HTTP requests are evaluated against a table of URL to view mappings. If a match is found, the related view function is called, if not a "PageNotFound" error page is displayed. Designing a good intuitive URL scheme that would aid maintenance and improve usability is important.

- All URLs start with 'chirp' as the beginning (although this is not necessary, it allows for multiple applications to be run on the same server instance)
- URLs that involved requests for static HTML content (eg forms) with no server side processing are denoted by the name and a '/' eg http://localhost :8000/chirp/login_form/ will display a static HTML page containing the login form.
- URLs that involved requests for server side provessing to be performed were defined with no '/' at the end. Eg http://localhost:8000/chirp/login_form will call a view to process the information submitted by the login form.
- URLs that contained with them extra information required for the views to function had the information placed after the '/' sign. Eg. http://localhost:8000/chirp/profile/12 indicates that the value 12 be passed to the associated view.

Functionality Lists

All the core functionality of the system was implemented along with some extra features. Some of the extra features outlined in the specification report were not implemented due to time and resource constraints. The current system has the following functionalities:

- Users can register, uploading their profile pictures
- Users can update their registration information
- Users can post messages.
- Users and visitors can view posted messages in real-time
- Users can view profiles of other users
- Users and visitors can search for posts by date or poster name.
- Users can make and accept friend requests to/from other users.
- Users can send/receive personal messages to/from other users.
- Users and visitors can subscribe to RSS feeds of the posts.

The functionalities outlined in the specification report that was not working are:

- Users should only be allowed to register once for both the host website and the woops application (too technical to implement given the resources).
- After registration email notification should allow users confirm their account before they are allowed to login (not enough time to implement).

REFLECTIVE SUMMARY

Working in a team can be challenging. The major difficulties encountered were in separating responsibilities between team members and integrating their work product to form one system. The time and effort it took getting the various work products from different team members was enormous. This may have been due to the limited experience in the framework chosen. Eventually a system of peer programming was adopted, where one member of the team mans the console writing the code and the others sit beside providing feedback and ideas. This proved a much more efficient method.

We were able to implement the required functionality and exceed the requirements. All members of the team gained a relatively deep understanding of the relatively new Django framework.

SUMMARY AND FUTURE WORK

The application is a real-time chatting widget making use of a wide range of technologies across 3-tiers. For the client side, JavaScript and AJAX techniques were employed to achieve real-time system updates. On the server side, the python programming language on the Django framework was used to achieve most of the system functionality; a sqllite3 database was used as the data repository.

A summary of the technologies used are as follows:

- JavaScript – used for client side validation, easy date/time selection and AJAX calls.

  Javascript Date/Time Picker [5].

  jQuery JavaScript Library v1.5 [3].
- AJAX – used to make asynchronous calls to the server without the need for a browser refresh.
- jQuery Timer plugin v0.1- used for running the background AJAX processes [4].
- HTTP – main protocol for sending and receiving data over the web. Used for all request and responses from client t server.
- jQuery validation plug-in pre-1.5.2 used for HTML form validation before sending HTTP POST messages [2]
- RSS – used as a standard for providing chat content to external clients or other web servers.
- Python on Django Framework: the web framework used for the development of the application [1]

We identified the uncertainty introduced when systems are built based on client-side settings like JavaScript settings on browsers as one of the limitations of our design; another is the lack of flexibility involved with using web application frameworks.

We propose further development to concentrate on increasing the scope of content that can be shared on the chatting widget. Future modifications may include

- ✓ File transfer, multimedia (photos, videos) and link addresses will be shared across users.
- ✓ Users will be able to customize their profile and chat page. For instance, modify the colors, fonts and so on.
- ✓ Users will be able to customize (color, text) their messages and attach emoticons.
- ✓ A security confirmation message will be sent to the user upon registration.
- ✓ Users will be given the option to delete personal messages and thier posts.
- ✓ The admin should be able to black-list a user or a post, however this is dependent on the who is using the application e.g profane words should not be allowed on kids website that is hosting the application. If this occurs the user that made the post will be blacklisted.
- ✓ Uses will be able to reply to posts in real time.
- ✓ Users should be able to subscribe to SMS updates to receive notifications about posts personal messages.

ACKNOWLEDGMENTS

Thanks to the lecturers and demonstrators for their comments and suggestions.

REFERENCES

1. Django software foundation (2005-2010). [Online] Available from: http://www.djangoproject.com/. [Accessed 1 March 2011].

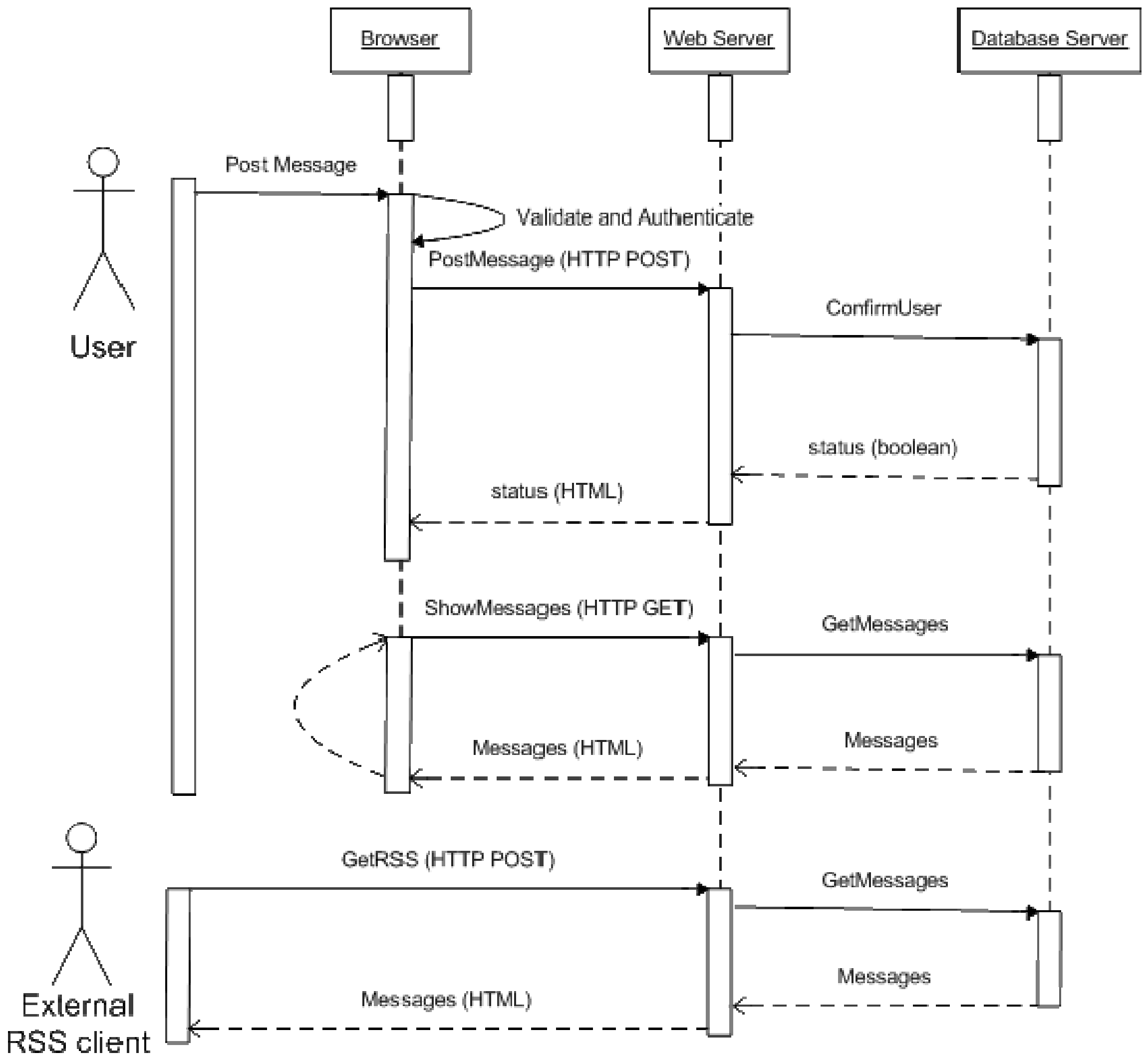
2 Jorn Zaefferer (2009).*bassistance.de* [Online]. Available from: http://bassistance.de/jquery-plugins/jquery-plugin-validation/ [Accessed 6 March 2011].

3 Jquery(2010). *Jquery right less do more* [Online] Available from: http://jquery.com/[Accessed 10 March 2011

4 Matt Schmidt (2011). Matt_ptr*.[Online]. Available from: http://www.mattptr.net [Accessed 6 March 2011].

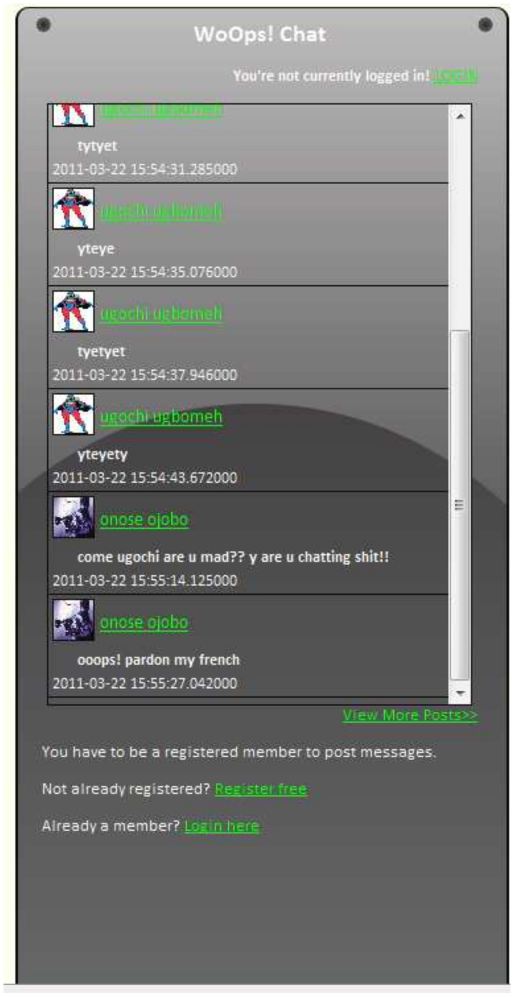
5 Nsf tools. *NotesTips* [Online]. Available from: http://www.nsftools.com/tips/NotesTips.htm#datepicker [Accessed 10 March 2011].

Appendix A: Sequence Diagram



Appendix B: Wireframe Diagrams

***WF - WireFrame**



WoOps! Chat

You're not currently logged in! [illegible]

[illegible]

tytyet

2011-03-22 15:54:31.285000

ugochi ugbomeh

yteye

2011-03-22 15:54:35.076000

ugochi ugbomeh

tyetyet

2011-03-22 15:54:37.946000

ugochi ugbomeh

yteyety

2011-03-22 15:54:43.672000

onose ojobo

come ugochi are u mad?? y are u chatting shit!!

2011-03-22 15:55:14.125000

onose ojobo

ooops! pardon my french

2011-03-22 15:55:27.042000

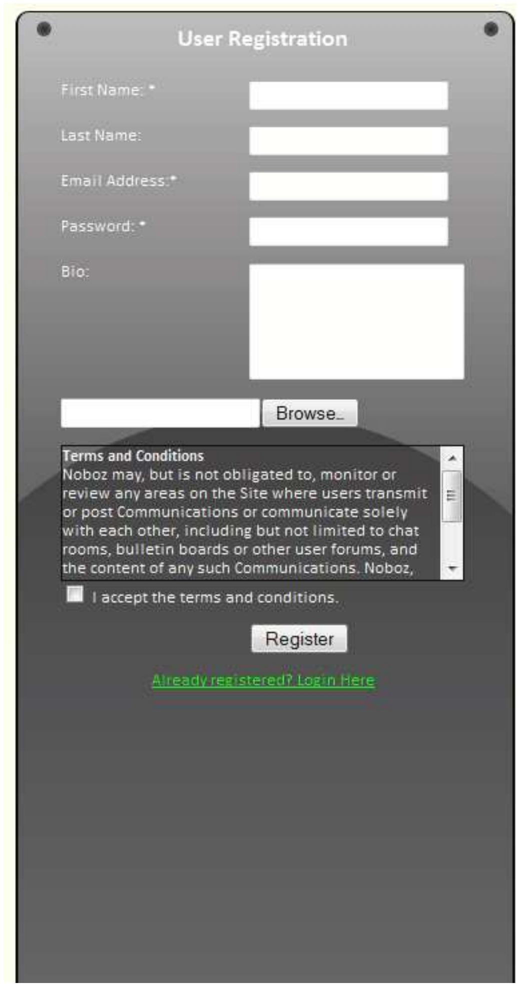
View More Posts>>

You have to be a registered member to post messages.

Not already registered? Register free

Already a member? Login here

WF 1: WoOps! Default Page



User Registration

First Name: *

Last Name:

Email Address:*

Password: *

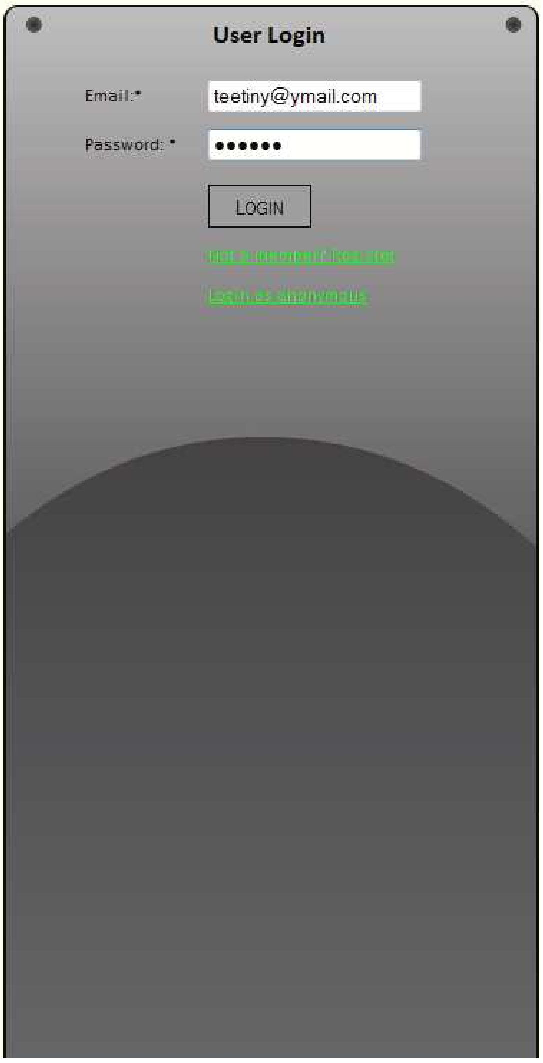
Bio:

Terms and Conditions
Noboz may, but is not obligated to, monitor or review any areas on the Site where users transmit or post Communications or communicate solely with each other, including but not limited to chat rooms, bulletin boards or other user forums, and the content of any such Communications. Noboz,

I accept the terms and conditions.

Already registered? Login Here

WF 2: WoOps! Registration Page

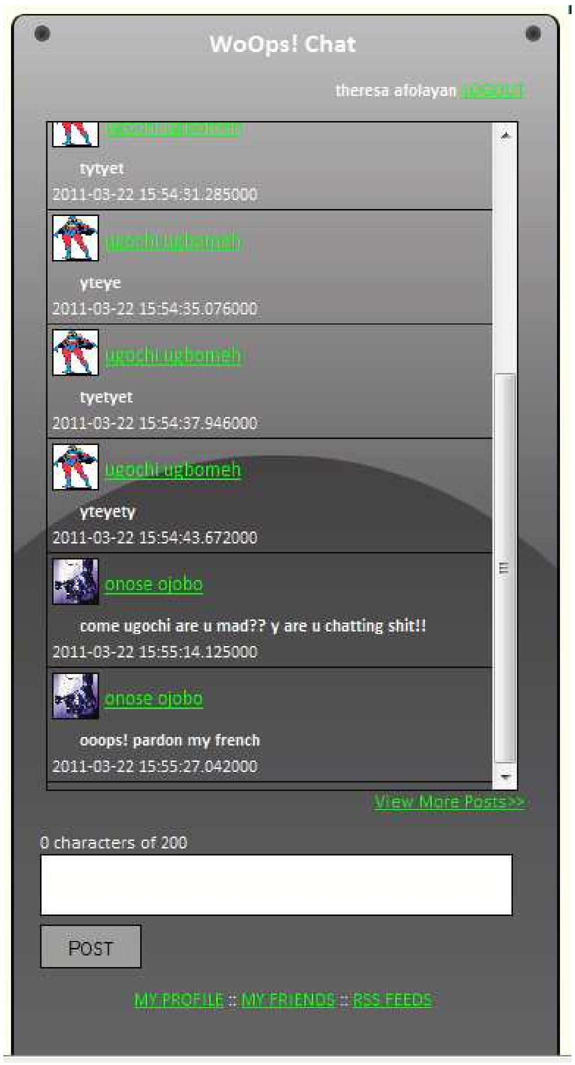


User Login

Email:* teetiny@ymail.com

Password: * ●●●●●●

WF 3: WoOps! Log-in Page



WoOps! Chat

theresa afolayan LOGOUT

ugochi ugbomeh
tytyet
2011-03-22 15:54:31.285000

ugochi ugbomeh
yteye
2011-03-22 15:54:35.076000

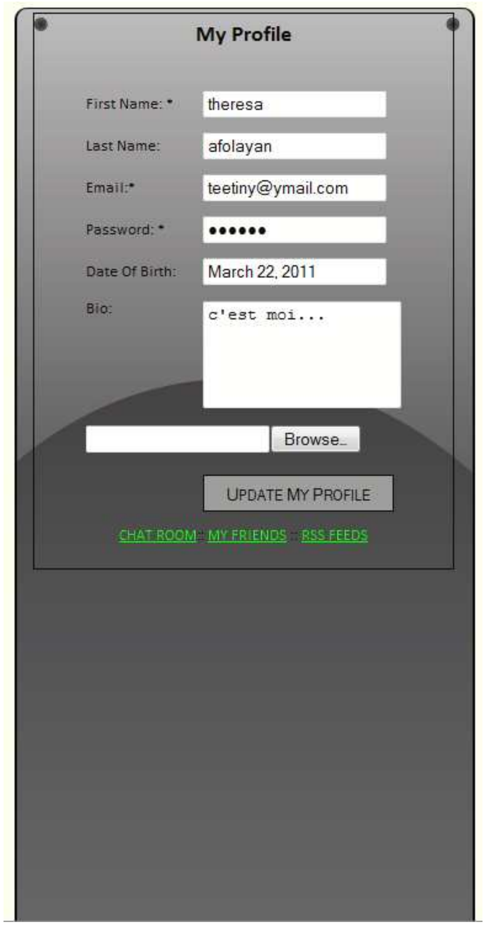
ugochi ugbomeh
tyetyet
2011-03-22 15:54:37.946000

ugochi ugbomeh
yteyety
2011-03-22 15:54:43.672000

onose ojobo
come ugochi are u mad?? y are u chatting shit!!
2011-03-22 15:55:14.125000

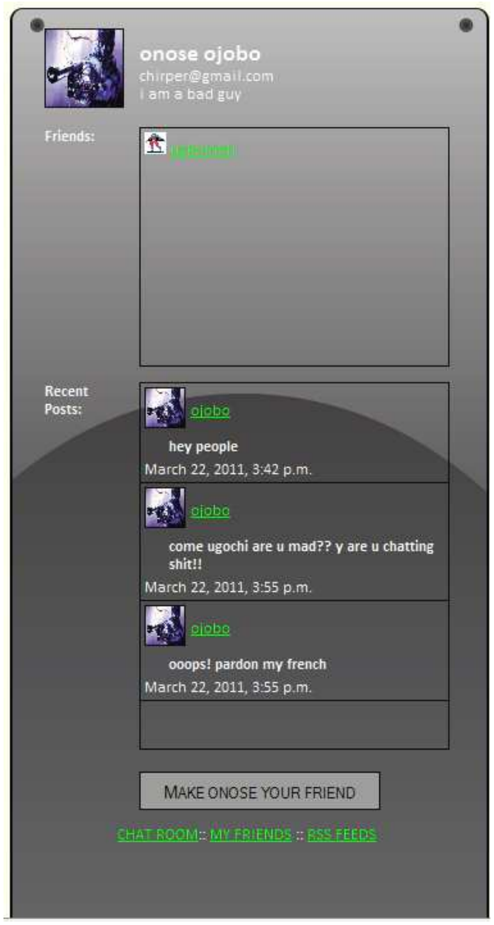
onose ojobo
ooops! pardon my french
2011-03-22 15:55:27.042000

WF 4:WoOps! Chat Page (User logged in)



First Name: *	theresa
Last Name:	afolayan
Email:*	teetiny@ymail.com
Password: *	••••••
Date Of Birth:	March 22, 2011
Bio:	c'est moi...

WF 5: WoOps! Profile Page


onose ojobo
chirper@gmail.com
I am a bad guy

Friends: [illegible]

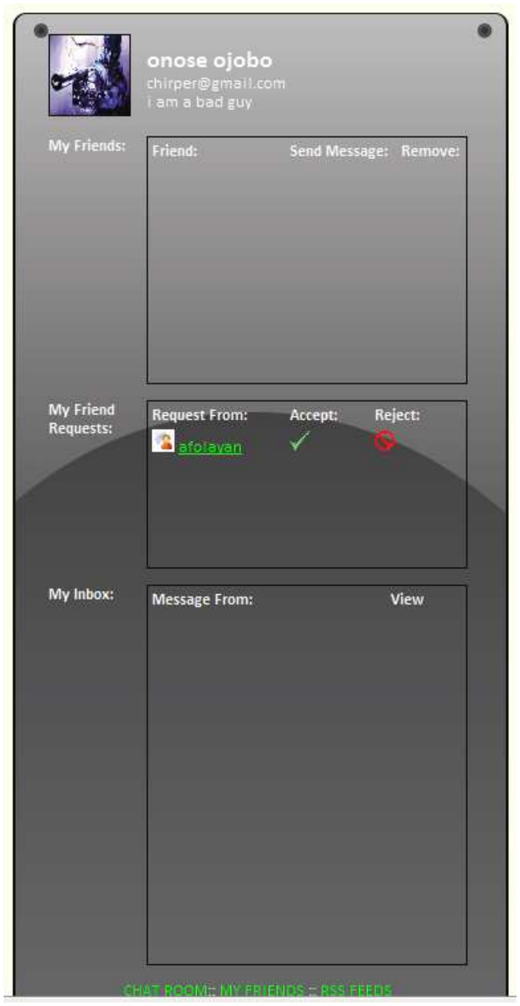
Recent Posts:

ojobo
hey people
March 22, 2011, 3:42 p.m.

ojobo
come ugochi are u mad?? y are u chatting shit!!
March 22, 2011, 3:55 p.m.

ojobo
ooops! pardon my french
March 22, 2011, 3:55 p.m.

WF 6: Viewing someone's page and Making a friend


onose ojobo
chirper@gmail.com
i am a bad guy

My Friends:

Friend:	Send Message:	Remove:

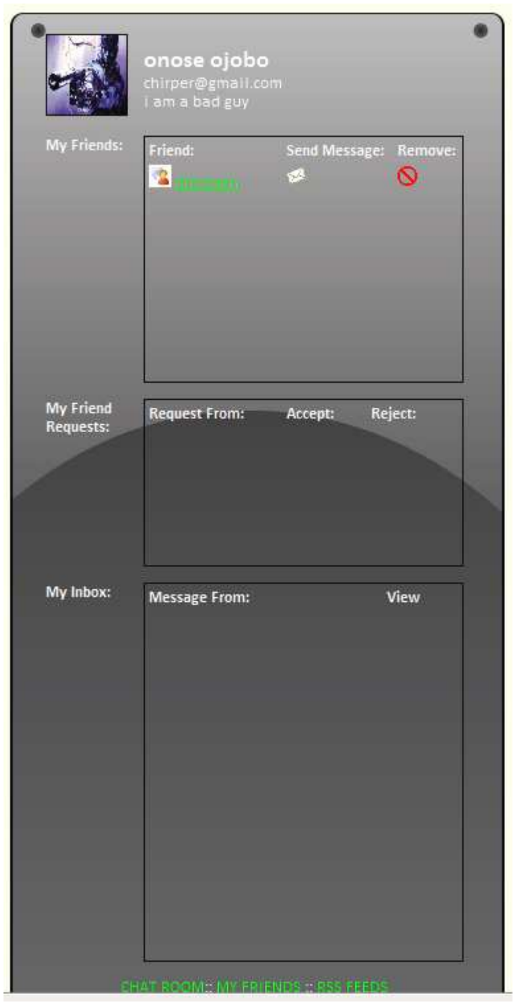
My Friend Requests:

Request From:	Accept:	Reject:
afolayan	✓	🚫

My Inbox:

Message From:	View

WF 7: Viewing Friend Request


onose ojobo
chirper@gmail.com
i am a bad guy

My Friends:

Friend:	Send Message:	Remove:
[illegible]	✉	🚫

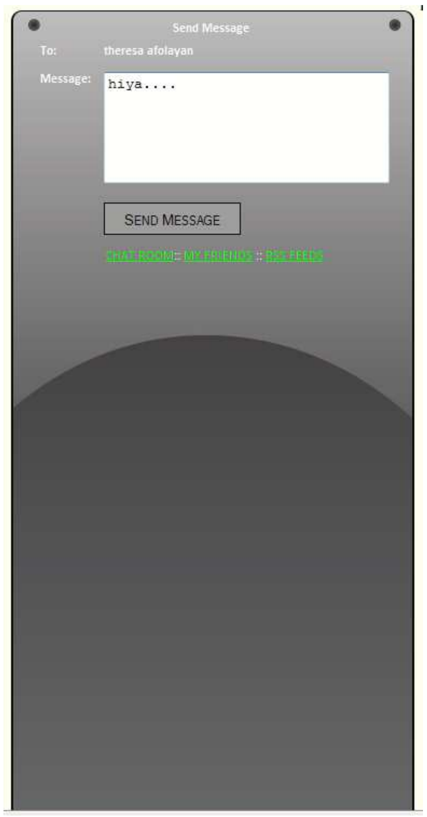
My Friend Requests:

Request From:	Accept:	Reject:

My Inbox:

Message From:	View

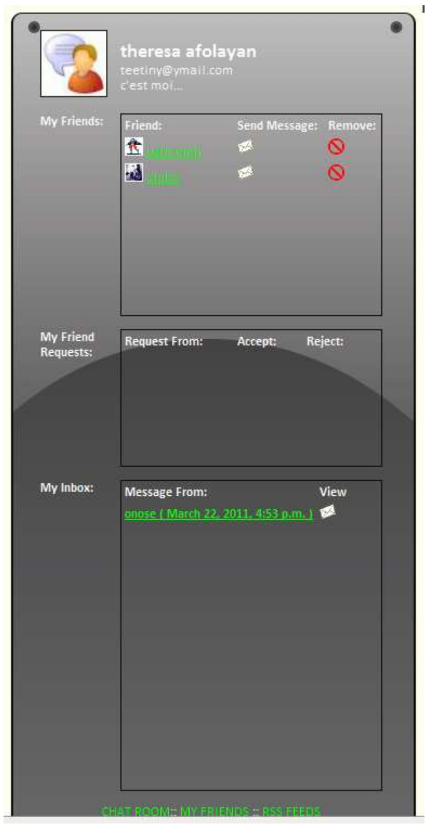
WF 8: Send Message Option


Send Message

To: theresa afolayan

Message: hiya....

WF 9: Send Message Content


theresa afolayan
teetiny@ymail.com
c'est moi...

My Friends:

Friend:	Send Message:	Remove:
[illegible]	✉	🚫
[illegible]	✉	🚫

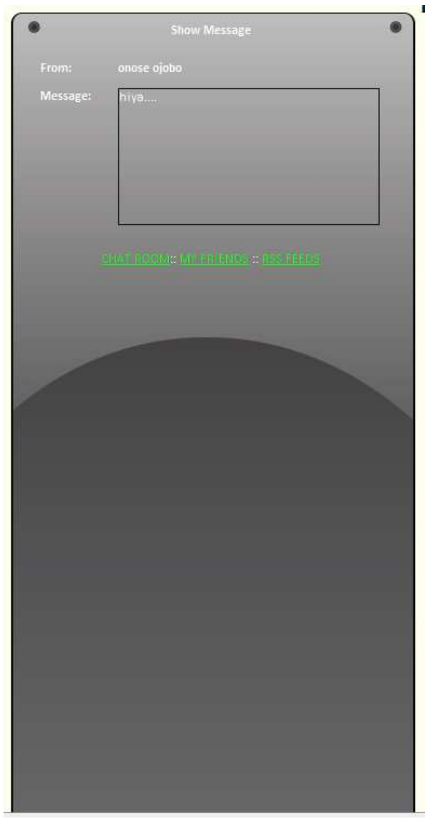
My Friend Requests:

Request From:	Accept:	Reject:

My Inbox:

Message From:	View
onose (March 22, 2011, 4:53 p.m.)	✉

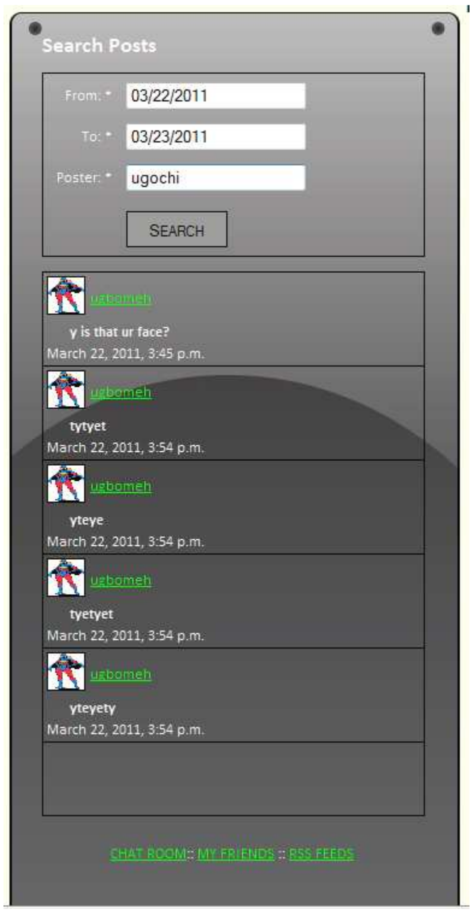
WF 10: Message Notification


Show Message

From: onose ojobo

Message: hiya....

WF 11: View Message


Search Posts

From: * 03/22/2011

To: * 03/23/2011

Poster: * ugochi

SEARCH

ugbomeh

y is that ur face?

March 22, 2011, 3:45 p.m.

ugbomeh

tytyet

March 22, 2011, 3:54 p.m.

ugbomeh

yteye

March 22, 2011, 3:54 p.m.

ugbomeh

tyetyet

March 22, 2011, 3:54 p.m.

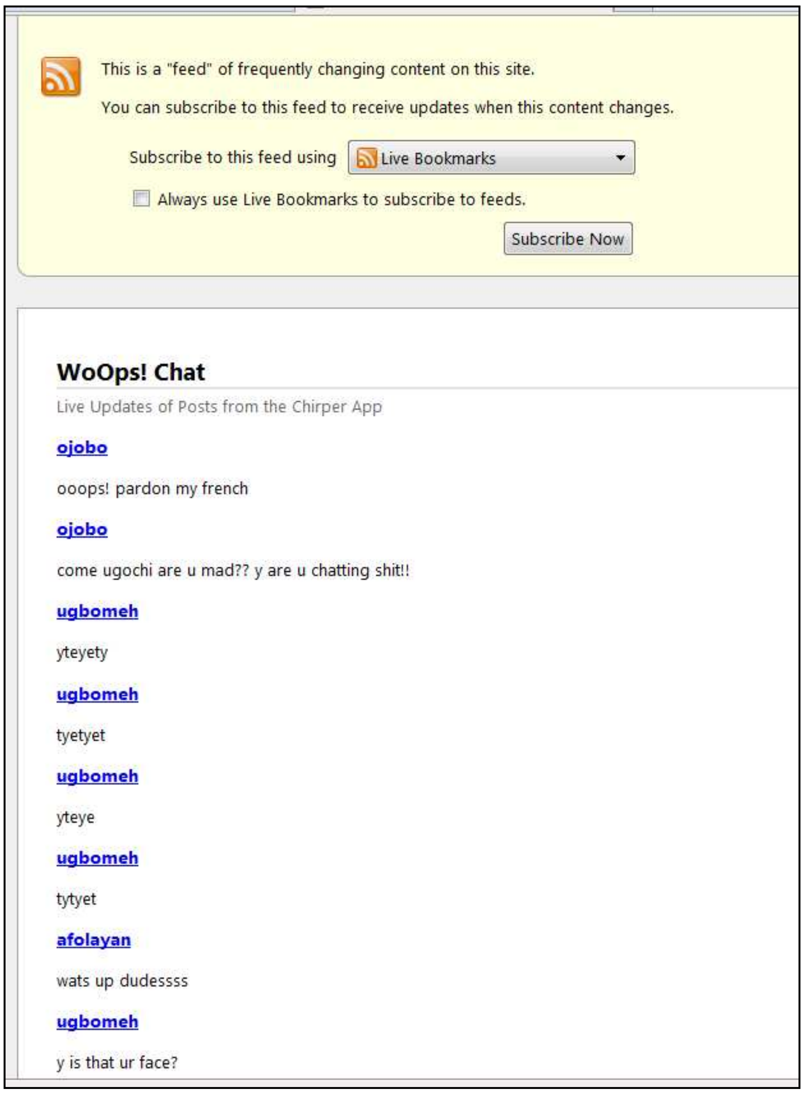
ugbomeh

yteyety

March 22, 2011, 3:54 p.m.

CHAT ROOM :: MY FRIENDS :: RSS FEEDS

WF 12: View Older Post


This is a "feed" of frequently changing content on this site.

You can subscribe to this feed to receive updates when this content changes.

Subscribe to this feed using Live Bookmarks

☐ Always use Live Bookmarks to subscribe to feeds.

Subscribe Now

WoOps! Chat

Live Updates of Posts from the Chirper App

ojobo

ooops! pardon my french

ojobo

come ugochi are u mad?? y are u chatting shit!!

ugbomeh

yteyety

ugbomeh

tyetyet

ugbomeh

yteye

ugbomeh

tytyet

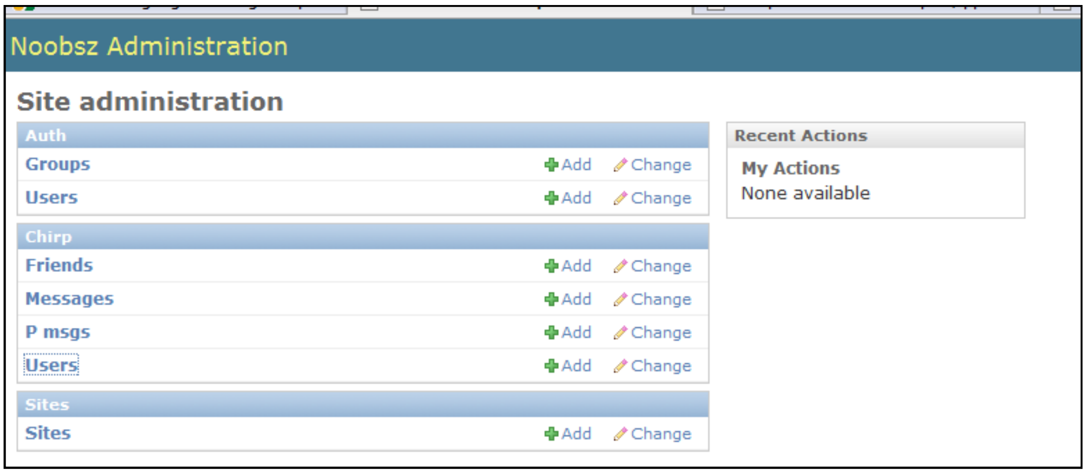
afolayan

wats up dudessss

ugbomeh

y is that ur face?

WF 13: RSS Feeds



Noobsz Administration

Site administration

Auth		
Groups	Add	Change
Users	Add	Change

Chirp		
Friends	Add	Change
Messages	Add	Change
P msgs	Add	Change
Users	Add	Change

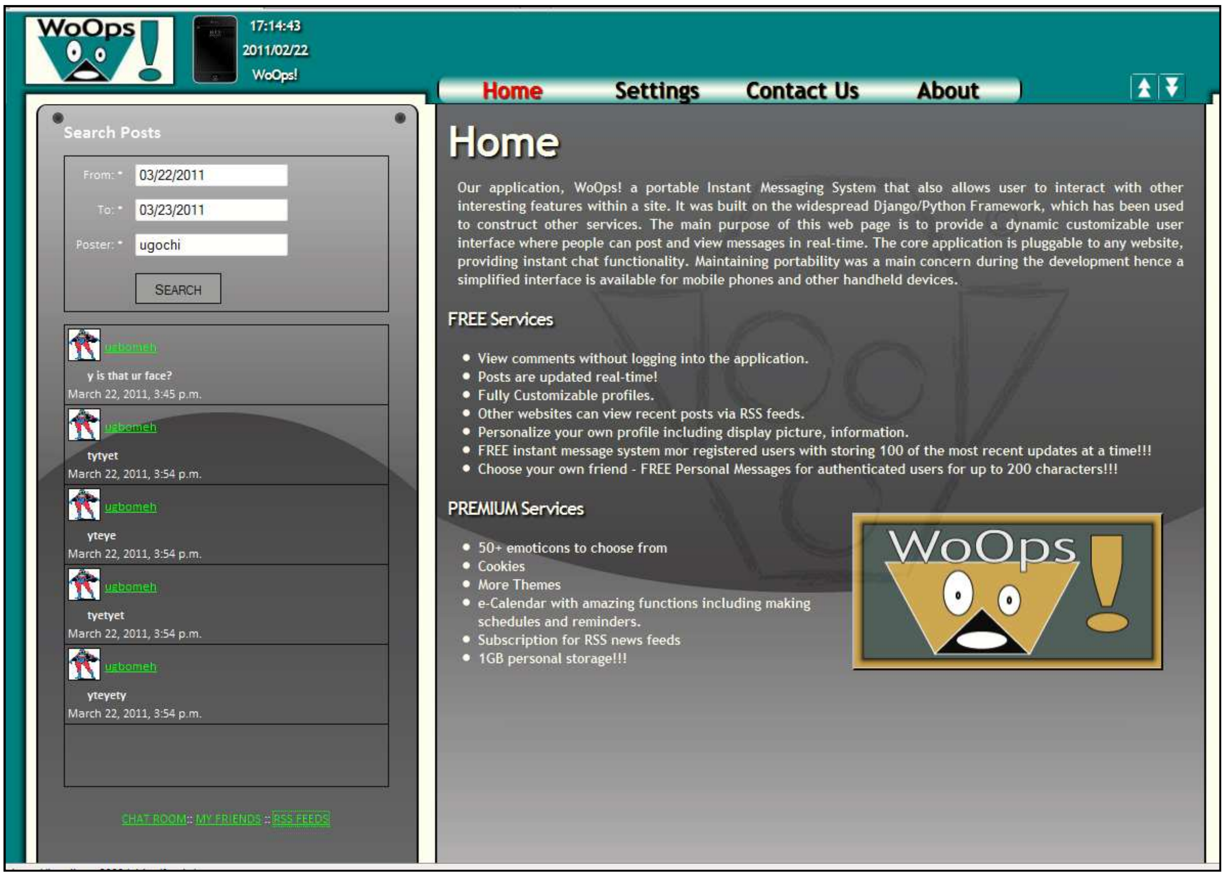
Sites		
Sites	Add	Change

Recent Actions

My Actions

None available

WF 14: Site Administration



Search Posts

From: *	03/22/2011
To: *	03/23/2011
Poster: *	ugochi

SEARCH

ugbomeh
y is that ur face?
March 22, 2011, 3:45 p.m.

ugbomeh
tytyet
March 22, 2011, 3:54 p.m.

ugbomeh
yteye
March 22, 2011, 3:54 p.m.

ugbomeh
tyetyet
March 22, 2011, 3:54 p.m.

ugbomeh
yteyety
March 22, 2011, 3:54 p.m.

CHAT ROOM :: MY FRIENDS :: RSS FEEDS

Home

Our application, WoOps! a portable Instant Messaging System that also allows user to interact with other interesting features within a site. It was built on the widespread Django/Python Framework, which has been used to construct other services. The main purpose of this web page is to provide a dynamic customizable user interface where people can post and view messages in real-time. The core application is pluggable to any website, providing instant chat functionality. Maintaining portability was a main concern during the development hence a simplified interface is available for mobile phones and other handheld devices.

FREE Services

- View comments without logging into the application.
- Posts are updated real-time!
- Fully Customizable profiles.
- Other websites can view recent posts via RSS feeds.
- Personalize your own profile including display picture, information.
- FREE instant message system mor registered users with storing 100 of the most recent updates at a time!!!
- Choose your own friend - FREE Personal Messages for authenticated users for up to 200 characters!!!

PREMIUM Services

- 50+ emoticons to choose from
- Cookies
- More Themes
- e-Calendar with amazing functions including making schedules and reminders.
- Subscription for RSS news feeds
- 1GB personal storage!!!

WF 15: Woops! Website