Project Report

Ming Wang (mw2724)
Jian Wang (jw2862)
Lixing Pan (lp2441)

Project Name: Web2.0 Conferencing Room (link: panlixing04.appspot.com)
Cloud Platform: Google AppEngine for Python
Programming Language: Python2.7, Html, javascript (Ajax)
Framework: Django-nonrel (an independent Django branch with NoSQL support for ORM)
Database: Google's High Replication Datastore (highly reliable non-relational database)

Google App Engine:

Google App Engine (http://code.google.com/appengine/) is a cloud platform which enables you to deploy your web applications on it. It supports java and python. We decided to deploy our application on App Engine for the following reasons:

1. It saves us from considerable amount of work in setting up a web server, getting a public IP address and registering for domain name.
2. It is highly reliable and do not need additional effort of maintenance and back up of data.
3. Resources are scalable, it is not necessary to have a estimation of how much resources we need exactly.
4. It charges for the amount you used. Most importantly, It gives you some free quote that is actually enough in our development phase.
5. Cloud is a hot topic these days, so we just want to try it out.

Python On Django:

Python is an interpreted, interactive, object-oriented programming language. Django is a high-level Python web application framework designed to support the deployment of dynamic web sites. It is specially designed to promote rapid web development. So It helps you build Web applications of high-performance elegantly and quickly. It is becoming widely used and accepted by lots of web developers, so we think it worth the time to learn a new cool language and get familiar with the Django frame through this course project.

Django-nonrel: Google AppEngine itself actually provides a framework called Webapp for Python, now It also supports Django. One problem for us is appEngine uses a object-oriented database. The good thing is finally we found a project called Django-nonrel, which adds NoSQL support for ORM (http://www.allbuttonspressed.com/projects/django-nonrel). But this solution still does not support many-to-many relationship, so we have to create additional tables to work around.

Project Design Object Description:

As the name implied, the basic goal of this project is to create a conference/meeting environment for people whenever have access to Internet.Sometimes, we may want everyone to join the group and have a discussion, sometimes, we may want to have control over who can join. So we initially decided we will have two types of conference room: **public room** and **private room**. Anyone can join a public room and password will be required to join a private room. Each room has a comment field, which is written by the owner of the room and it can be a simple description of the room.

Users can see a list of all rooms at the home page and choose a room to join. A user can also create a new room and launch a discussion.

After join a room, you can see a large window which will show all the contents after you join the room. You can send messages by type into the input bar and then sent it out. You can see the message history by selecting different options in the history drop down menu, this enables you to see messages before you join the room. There is also a member list where you can see who have actually joined room so far. You can update the list as you want.

If you are the owner of the room, you will have few more options. You can set the room to be public or private by selecting between the two radio button on the top. You can also edit the room comments and submit the change. If you do not want everyone to be able to talk, you can type in his account name and disable him. You can enable him later in the same manner.

Features not implemented:
All the feature above has been implemented in our application now. We actually want have more features want to add, but run out of time. Yet, we still want to list them below.

1. Voting for user-defined poll. Give group members the ability to create polls in a room. Links to polls will be listed in the room. Group members can go and voting for it.
2. File sharing among room members. Group members can upload files. Links of uploaded files will be listed in the room. Group members can click the link to download the file.
3. Scratch board possibly for all group members. It features “What you do is what you see”.
4. Multiple administrator of single room. Enhance the administration over rooms and on different granularity.
5. Voice/Video … ...

Code Review:

Django uses the MVC(Model, View, Controller) model.
Model is the data model we used in the application. They are written in the models.py file.
Controller is the corresponding actions to user requests. They are written in the views.py file.
View is the presentation layer, they are used to render the dynamically created web pages. They are the html files stored in the templates folder.

Data Model
Room

Field	owner	created	roomName	comment	public	password	deleted
Type	ForeignKey	DateTime	Char	Char	Boolean	Char	Boolean

owner refers to the user who created the room.
created is the time when the room is created.
roomName is the name of the room.
comment is can be any simple description you want to add to the room.
public marks if the room is private or public.
password is the password need to enter the room if it is private.
deleted marks if the table has been deleted

Message

Field	room	type	author	message	timestamp
Type	ForeignKey	Char	ForeignKey	Char	DateTime

room refers to the room the message belongs to.
type use different characters to represent the type of the message
author tells who created the message.
timestamp records the time the message is created.

RoomUser

Field	rname	uname	talk_permission	inroom
Type	Char	Char	Boolean	Boolean

This table is used to indicate if a user is in a room and if he can talk in that room.
rname is the name of a room.
uname is the name of a user.
talk_permission marks if a user can talk.
inroom marks if a user is in the room.

Templates:
The html file defines how the web pages will look like. They also contains the javascript which is very important in our implementation. Here we focus on explain how Ajax is used in our implementation.
We explain how we get messages from sever, other features using ajax are actually implemented in a very similar way.

```
function sync_messages() {
        $.ajax({
        type: 'POST',
```

```
            data: {id:window.chat_room_id},
            url:'/rooms/sync/',
            dataType: 'json',
            success: function (json) {
                last_received = json.last_message_id;
                last_time = json.last_message_time;
            }
            });

            setTimeout("get_messages()", 1000);
        }
```

This sync_messages() function is called after the page have been loaded. We set the request type to be 'POST', we pass the chat room id as a parameter, which can be used by the controller. The url will be mapped to a controller. 'json' is the data type we want to use. 'success' tells what to do after the ajax request is successfully done. Here it set the last received message id and the created time of that message. The parameters are pass back by the controller. At last, the use the setTimeout function to make the get_messages() function to be called every 1 second.

Controller

To be consistent with the previous javascript function, we use the sync function as an example, which is mapped to url '/rooms/sync/'.

```
@login_required
def sync(request):
    if request.method != 'POST':
        raise Http404
    post = request.POST

    if not post.get('id', None):
        raise Http404

    r = Room.objects.get(id=post['id'])

    lmid = r.last_message_id()
    lmtm = float(time.mktime(r.last_message_time().timetuple()))

  return HttpResponse(jsonify({'last_message_id':lmid, 'last_message_time':lmtm}))
```

The function get the room according to its room id. It then get the id of the last message by calling the last_message_id() function and get the timestamp of the last message by calling the last_message_time() function. It then returns the id and time parameter to the page.

All other features are implemented is the same principle. The details of how every thing implemented can be seen in the code.