Mike Nguyen

DePaul University

CSC536 Final Project

**How to run my code**

Project was built on a Window 7

1). Navigate to the server folder

2). Open up a cmd shell

3). Type in sbt run, make sure the shell points to the current directory

4). Open up a browser and type in <http://localhost:8888/html>

5). You will be prompted for a username, enter that and you’re ready to go

6). There are simple instructions on the pages that will guide you

7). For most of the requests you will need to type in your input in the text area and then hit the button associated next to the requests

8). You can open up more than one browser and tweets to your followers

**System Design**

The system contains seven files. Two of the files were copy from the example (MyTwitterApp, MyTwitterHandler). One of the file contains the messages that are shared between files and needed to send requests/replies (Messages). The rest of the files are used to accessed and processed the user requests by communicating each other for the necessary data (ClientsHandler, NetworkHandler, Supervisor, TweetHandler).

The way the system works is by a browser interface. The MyTwitterApp will set up the interface between the user and the server. The MyTwitterHandler will processed all incoming and outgoing data. When there’s an incoming “event”, MyTwitterHandler will look for an identifier, which was added in the browser before being send, and send the “event” to the Supervisor. The Supervisor strip away all unnecessary add-ons before sending the data to the appropriate file. The supervisor is the coordinator that manages most of the allocation of data and most of the fault tolerance. Depending on the requests of the user it will either send the data to NetworkHandler, TweetHandler, or ClientHandler. And these three files will send the data back to the Supervisor. The Supervisor add an identifier if needed before sending all outgoing data to the MyTwitterHandler, which is than display on the browser. The ClientsHandler stores all connected client and forward tweets to the NetworkHandlers. It also contains the webSocketID for each client, which is needed to identify clients for some of the requests. The NetworkHandler stores the followers and followings of a client. The TweetHandler stores the tweets by either user or tag.

I took an synchronize approach when designing the system. I wanted the system to wait for an reply for every request before continuing. If the reply doesn’t come the system will tell the user and it should be able to continue. I also tried to log everything for easier bug handling. I used Futures with the ExcutionContext to block and wait for the responds, I needed the responds data before moving on so it seems like a good fits. I also set a Timeout just in case the responds never comes. The tradeoff for this is performance. The great thing about this is I know right away where the problems is, and the code only execute if a future was successful. If not successful there will be a warning to the user, and the system is able to run properly. Other things I did was override the OneForOneStrategy to restart on exceptions and a Deathwatch for my actors. This is to inform me if any of the actors were terminated and if I needed to restart one of them.

For the testing I prevented the reply from coming through. I also tested inputting invalid data into the data structures that store all clients, followers/followings, and tweets. I also prevented the requests from getting through. I wasn’t able to properly have a test file.