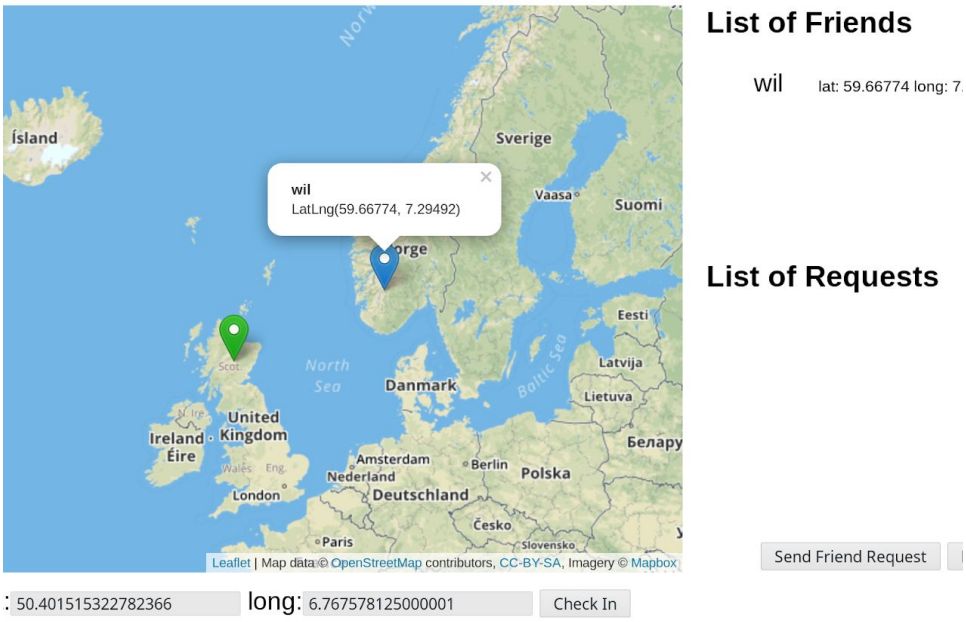
CM4108 - Cloud Computing

Submission 1

Friend Finding Application on AWS



Introduction

The Web App is described in more technical terms below but I will briefly describe the function here. The app is made up of two components, a front end and a back end. The back end takes the form of a REST API that interfaces with a DynamoDB. The user interface is developed in Javascript/JQuery and communicates with the server by sending HTTP requests to the API.

The design of the server was to have a simple but powerful set of tools for a client to extend and user in a number of ways. To this end, the API points were made as simple as possible and the jquery handles all the building of the friend and requests lists. This in principle means that any number of different apps could target the same set of basic tools and expand them to their own toolset as opposed to providing only niche methods of accessing.

Finally, particular note was taken in designing the front-end to keep traffic, requests to the API, to a minimum to preserve not only bandwidth but server time as in a production environment this could seriously hamper the performance of the server for all users.

For the DB, it should be named “rgu-cloud-cw” with a string hash key named “name” and no other changes made.

Statement of Compliance

Task 1

Java class “User” has been implemented to handle the DynamoDB entries and the “UserResource” class allows the mapping of elements in the DB.

User	- Implemented
UserResource	- Implemented

Task 2

Send / Approve / Deny requests	- Implemented
Update users location	- Implemented
Get a list of subscriptions	- Implemented
Get a specific users location	- Implemented
Get all friends locations	- Implemented

Task 3

User Interface	Implemented
Using HTML, CSS, JS and jQuery	- Implemented

Description Of Java Classes

The Java classes used are the “User.java” and the “UserResource.java” aside from the two nescaccery files for accessing the DynamoDB (“Config.java” and “DynamoDBUtil”).

User.java

The User class handles the structure of the data coming and going to the DynamoDB and defines its structure, which is as follows.

- String name

The name represents our Primary Key or Hash. This has the added benefit of eliminating the possibility for duplicate usernames as uniqueness is enforced.

- double latitude, longitude

The latitude and longitude are simply two doubles which get mapped to numbers inside the DynamoDB.

- List friends
- List sentRequests
- List receivedRequests

The three Lists store the users friends, sent and received friend requests. This method at first glance seems backwards as in SQL we would expect this to be a Relational Table. However as DynamoDB is a NoSQL database we are encouraged to keep everything in the minimal amount of tables. The alternate approach would require parsing entire other tables for the names we want. By storing the requests and friends this way it makes our handler code somewhat more verbose however when actually searching for the users in these lists we reduce the amount of queries done on the database (abstracted from us by the load() function from DynamoDBMapper class) and, in a larger deployment, would have an effect on performance.

UserResource.java

The UserResource class handles all of the HTTP calls to the server that is passed along from Jersey and interfaces with the DB, creating our RESTful API. This class is made up of a number of methods accepting a number of parameters, as explained below.

Method Name	HTTP Type	Paramaters	URL	Return type
addUser	POST	name, latitude, longitude, friends, sentRequests, recievedRequests	/	TEXT
This method allows users to send a POST request to create a new user that can be logged into, send friend requests and set its locations.				
getOneUser	GET	name	/$name	JSON
This method allows users to send a GET and retrieve a single user defined in the URL parameter $name. This works by searching the database for a user with the name provided, if one is not found a 404 is thrown back otherwise the user is returned as JSON.				
updateLocation	POST	name, latitude, longitude	/$name	JSON
This method allows for the updating of a created user. The $name is passed in through the URL parameter and the latitude and longitude is expected from JSON. We handle a number of potential issues by catching expected errors from some common issues. For example if NPE is thrown we know this is because a user is not found and thus we can return a 404 User not found.				
newFriendRequest	POST	sender, receiver	/$sender/$receiver	JSON
This method allows users to add another user as a friend. This is done by adding the “sender” to the receiver’s received list and the receiver to the sender’s sent list, the reasoning for this implementation is explained above. One potential issue lies with accepting two URL parameters. While not functionally an issue and semantically making more sense as we POST to /$user thus one would expect to /$sender/$receiver for a request, the argument could me made it should be a POST request to the /$sender.				
getAllUsers	GET		/	JSON

This method returns all the users in the table and is used more for testing as it doesn't allow for friend lists.

cancelRequest	POST	sender, receiver	/$sender/$receiver	JSON

This method follows the same structure as newFriendRequest() but in the reverse. It only removes the "sender" from the receiver's sent list and the receiver from the sender's received list.

There are some miscellaneous methods in the class that exist to remove duplicate code and self explanatory in function, these are:

- isRequested
- hasSent
- isFriend
- acceptFriendRequest
- removeFriendRequset

Description Of The API

All URL's are assumed to be from http://localhost:8080/cw/

URL Map

URL	HTTP	Response
/	POST	201, 400
/$name	GET	404
/$name	POST	201, 400, 404, 500
/$sender/$receiver	POST	200, 201, 403, 404, 418, 500
/	GET	404
/$sender/$receiver	POST	200, 403, 404, 500

HTTP Code Explanation

200	Indicates that a user has successfully accepted a friend request
201	Indicates the success of an operation where possible
400	Indicates an error with the data a user has submitted
403	Indicates that a user is already your friend (and thus the request can be denied in the cancel method)
404	Indicates that the requested User does not exist or there are no users (when getting the whole list)
418	Indicates that you cannot add yourself as a friend
500	Indicates that an error has occurred with the server in the form of an unhandled catch-all exception