package controllers;

import com.fasterxml.jackson.databind.JsonNode;

import com.fasterxml.jackson.databind.ObjectMapper;

import com.fasterxml.jackson.databind.node.ArrayNode;

import play.api.mvc.AnyContentAsJson;

import play.libs.F.Promise;

import play.libs.Json;

import play.libs.ws.WS;

import play.libs.ws.WSResponse;

import play.mvc.Controller;

import play.mvc.Http;

import play.mvc.Result;

import play.mvc.WebSocket;

import java.math.BigDecimal;

import java.util.\*;

import java.util.concurrent.CompletionStage;

import java.util.stream.Collectors;

import java.util.stream.Stream;

import com.fasterxml.jackson.databind.node.JsonNodeFactory;

import com.fasterxml.jackson.databind.node.ObjectNode;

import scala.collection.immutable.ListMap;

import utils.Streams;

import views.html.index;

import views.html.\*;

import static java.util.stream.Collectors.toList;

import static utils.Streams.stream;

public class Tweets extends Controller {

public static Result index() {

return ok(views.html.index.render("TweetMap"));

}

public static Promise<Result> search(String query) {

return fetchTweets(query)

.map(jsonNode -> ok(jsonNode));

}

// public static Promise<Result> search1(String query) {

// return Timeline(query)

// .map(jsonNode -> ok(jsonNode));

//}

public static Result Timeline(String name) {

//System.out.println(name);

String token= "AAAAAAAAAAAAAAAAAAAAAHB%2F4wAAAAAAI69NlJD0CNk9SlQRy697nPF5oJQ%3DVYBnbQFjiGCL2WIXFuH3QJmOrGNmEO6kCjcXwZkdL7Z3sZHPhM";

Promise<WSResponse> responsePromise = WS.url("https://api.twitter.com/1.1/users/show.json")

.setHeader("Content-Type", "application/x-www-form-urlencoded;charset=UTF-8")

.setHeader("Authorization", "Bearer "+token)

.setQueryParameter("screen\_name", name)

.setQueryParameter("count", "10").get() ;

Promise<JsonNode> a=responsePromise

.filter(response -> response.getStatus() == Http.Status.OK)

.map(response -> response.asJson());

JsonNode jn=a.get(1000);

String image=jn.get("profile\_image\_url\_https").asText();

String uname=jn.get("name").asText();

String description=jn.get("description").asText();

String following=jn.get("friends\_count").asText();

String followers=jn.get("followers\_count").asText();

Promise<WSResponse> responsePromise1 =WS.url("https://api.twitter.com/1.1/statuses/user\_timeline.json")

.setHeader("Content-Type", "application/x-www-form-urlencoded;charset=UTF-8")

.setHeader("Authorization", "Bearer "+token)

.setQueryParameter("screen\_name", name)

.setQueryParameter("count", "10")

.get() ;

Promise<JsonNode> a1=responsePromise1

.filter(response -> response.getStatus() == Http.Status.OK)

.map(response -> response.asJson());

List<String> text= new ArrayList<>();

for(JsonNode node : a1.get(1000)) {

text.add( node.path("text").asText());

}

return ok(views.html.user.render(image,uname,description,following,followers,text) );

//can also map using method references - WSResponse::asJson\*/

// return responsePromise

// .filter(response -> response.getStatus() == Http.Status.OK)

// .map(response -> response.asJson())

// .recover(Tweets::errorResponse);

}

public static Result Location(String name) {

//System.out.println(name);

String token= "AAAAAAAAAAAAAAAAAAAAAHB%2F4wAAAAAAI69NlJD0CNk9SlQRy697nPF5oJQ%3DVYBnbQFjiGCL2WIXFuH3QJmOrGNmEO6kCjcXwZkdL7Z3sZHPhM";

Promise<WSResponse> responsePromise1 = WS.url("https://api.twitter.com/1.1/search/tweets.json")

.setHeader("Content-Type", "application/x-www-form-urlencoded;charset=UTF-8")

.setHeader("Authorization", "Bearer "+token)

.setQueryParameter("count","10")

.setQueryParameter("q", name)

.setQueryParameter("statuses.user.location",name).get() ;

Promise<JsonNode> a1=responsePromise1

.filter(response -> response.getStatus() == Http.Status.OK)

.map(response -> response.asJson());

List<String> text= new ArrayList<>();

Map<String,String> ltweet= new LinkedHashMap<>();

for(JsonNode node : a1.get(1000).path("statuses")) {

ltweet.put(node.path("text").asText(),node.path("user").path("location").asText());

// System.out.println(node.path("user").path("location").asText());

}

//System.out.println(text);

return ok(views.html.location.render(name, ltweet) );

//can also map using method references - WSResponse::asJson\*/

// return responsePromise

// .filter(response -> response.getStatus() == Http.Status.OK)

// .map(response -> response.asJson())

// .recover(Tweets::errorResponse);

}

public static Result words(String name) {

//System.out.println(name);

String token= "AAAAAAAAAAAAAAAAAAAAAHB%2F4wAAAAAAI69NlJD0CNk9SlQRy697nPF5oJQ%3DVYBnbQFjiGCL2WIXFuH3QJmOrGNmEO6kCjcXwZkdL7Z3sZHPhM";

Promise<WSResponse> responsePromise1 = WS.url("https://api.twitter.com/1.1/search/tweets.json")

.setHeader("Content-Type", "application/x-www-form-urlencoded;charset=UTF-8")

.setHeader("Authorization", "Bearer "+token)

.setQueryParameter("count","100")

.setQueryParameter("q", name)

.get() ;

Promise<JsonNode> a1=responsePromise1

.filter(response -> response.getStatus() == Http.Status.OK)

.map(response -> response.asJson());

List<String> text= new ArrayList<>();

List<String> utext= new ArrayList<>();

for(JsonNode node : a1.get(10000).path("statuses")) {

// System.out.println(node.path("user").path("location").asText());

text.add(node.path("text").asText());

utext.add(node.path("user").path("screen\_name").asText());

}

//System.out.println(text);

List<String> text1=text.stream().map(w -> w.split(" "))

.flatMap(Arrays::stream)

.distinct()

.collect(Collectors.toList());

text1.remove(" ");

text1.remove("");

//System.out.println(text1);

return ok(views.html.words.render(text,text1,utext) );

}

public static Result stats(String words){

System.out.println(words);

String words1=words.replace(",","").replace("[",

"").replace("]","");

List<String> words2=Stream.of(words1).sorted((String o1,String o2) -> o1.compareTo(o2)).collect(toList());

//System.out.println(words2);

return ok(views.html.stats.render( words1));

}

public static Result words1(){return ok(views.html.words1.render());}

public static WebSocket<JsonNode> ws() {

return WebSocket.whenReady((in, out) -> {

in.onMessage(jsonNode -> {

String query = jsonNode.findPath("query").textValue();

fetchTweets(query).onRedeem(json -> out.write(json));

});

in.onClose(() -> {

});

});

}

public static Result fetchTweet(String name) {

System.out.println(name);

String token= "AAAAAAAAAAAAAAAAAAAAAHB%2F4wAAAAAAI69NlJD0CNk9SlQRy697nPF5oJQ%3DVYBnbQFjiGCL2WIXFuH3QJmOrGNmEO6kCjcXwZkdL7Z3sZHPhM";

Promise<WSResponse> responsePromise1 =WS.url("https://api.twitter.com/1.1/statuses/user\_timeline.json")

.setHeader("Content-Type", "application/x-www-form-urlencoded;charset=UTF-8")

.setHeader("Authorization", "Bearer "+token)

.setQueryParameter("screen\_name", name)

.setQueryParameter("count", "10")

.get() ;

Promise<JsonNode> a1=responsePromise1

.filter(response -> response.getStatus() == Http.Status.OK)

.map(response -> response.asJson());

String text="";

for(JsonNode node : a1.get(1000)) {

text = node.path("text").asText();

System.out.println(text);

}

//can also map using method references - WSResponse::asJson

return ok(views.html.test1.render("Tweets",text));

}

/\*\*

\* Fetch the latest tweets and return the Promise of the json results.

\* This fetches the tweets asynchronously and fulfills the promise when the results are returned.

\* The results are first filtered and only returned if the result status was OK.

\* Then the results are mapped (or transformed) to JSON.

\* Finally a recover is added to the Promise to return an error Json if the tweets couldn't be fetched.

\*

\* @param query

\* @return

\*/

public static Promise<JsonNode> fetchTweets(String query1) {

String token= "AAAAAAAAAAAAAAAAAAAAAHB%2F4wAAAAAAI69NlJD0CNk9SlQRy697nPF5oJQ%3DVYBnbQFjiGCL2WIXFuH3QJmOrGNmEO6kCjcXwZkdL7Z3sZHPhM";

//"http://twitter-search-proxy.herokuapp.com/search/tweets"

Promise<WSResponse> responsePromise = WS.url("https://api.twitter.com/1.1/search/tweets.json")

.setHeader("Content-Type", "application/x-www-form-urlencoded;charset=UTF-8")

.setHeader("Authorization", "Bearer "+token)

.setQueryParameter("count","10")

.setQueryParameter("q", query1).get() ;

//can also map using method references - WSResponse::asJson

return responsePromise

.filter(response -> response.getStatus() == Http.Status.OK)

.map(response -> response.asJson())

.recover(Tweets::errorResponse);

}

private static JsonNode transformStatusResponses(JsonNode jsonNode) {

//create a stream view of the jsonNode iterator

List<JsonNode> newJsonList = stream(jsonNode.findPath("statuses"))

//map the stream of json to update the values to have the geo-info

.map(json -> json.path("id"))

.collect(toList());

ObjectNode objectNode = JsonNodeFactory.instance.objectNode();

objectNode.putArray("statuses").addAll(newJsonList);

return objectNode;

}

private static ObjectNode setCoordinates(ObjectNode nextStatus) {

nextStatus.putArray("coordinates").add(randomLat()).add(randomLon());

return nextStatus;

}

private static Random rand = new java.util.Random();

private static double randomLat() {

return (rand.nextDouble() \* 180) - 90;

}

private static double randomLon() {

return (rand.nextDouble() \* 360) - 180;

}

/\*\*

\* The error response when the twitter search fails.

\*

\* @param ignored

\* @return

\*/

public static JsonNode errorResponse(Throwable ignored) {

return Json.newObject().put("error", "Could not fetch the tweets");

}

}