package com.twitter.search.common.relevance.entities;

import java.util.List;

import java.util.Optional;

import com.google.common.annotations.VisibleForTesting;

import com.twitter.search.common.indexing.thriftjava.ThriftGeoLocationSource;

import com.twitter.search.common.indexing.thriftjava.ThriftGeoTags;

import com.twitter.tweetypie.thriftjava.GeoCoordinates;

import com.twitter.tweetypie.thriftjava.Place;

import geo.google.datamodel.GeoAddressAccuracy;

/\*\*

\* A GeoObject, extending a GeoCoordinate to include radius and accuracy

\*/

public class GeoObject {

public static final int INT\_FIELD\_NOT\_PRESENT = -1;

public static final double DOUBLE\_FIELD\_NOT\_PRESENT = -1.0;

private double latitude = DOUBLE\_FIELD\_NOT\_PRESENT;

private double longitude = DOUBLE\_FIELD\_NOT\_PRESENT;

private double radius = DOUBLE\_FIELD\_NOT\_PRESENT;

private final ThriftGeoLocationSource source;

// Valid range is 0-9. With 0 being unknown and 9 being most accurate.

// If this GeoObject is valid, this should be set to INT\_FIELD\_NOT\_PRESENT

private int accuracy = 0;

/\*\* Creates a new GeoObject instance. \*/

public GeoObject(double lat, double lon, ThriftGeoLocationSource source) {

this(lat, lon, 0, source);

}

/\*\* Creates a new GeoObject instance. \*/

public GeoObject(double lat, double lon, int acc, ThriftGeoLocationSource source) {

latitude = lat;

longitude = lon;

accuracy = acc;

this.source = source;

}

/\*\* Creates a new GeoObject instance. \*/

public GeoObject(ThriftGeoLocationSource source) {

this.source = source;

}

/\*\*

\* Tries to create a {@code GeoObject} instance from a given TweetyPie {@code Place} struct based

\* on its bounding box coordinates.

\*

\* @param place

\* @return {@code Optional} instance with {@code GeoObject} if bounding box coordinates are

\* available, or an empty {@code Optional}.

\*/

public static Optional<GeoObject> fromPlace(Place place) {

// Can't use place.centroid: from the sample of data, centroid seems to always be null

// (as of May 17 2016).

if (place.isSetBounding\_box() && place.getBounding\_boxSize() > 0) {

int pointsCount = place.getBounding\_boxSize();

if (pointsCount == 1) {

GeoCoordinates point = place.getBounding\_box().get(0);

return Optional.of(createForIngester(point.getLatitude(), point.getLongitude()));

} else {

double sumLatitude = 0.0;

double sumLongitude = 0.0;

List<GeoCoordinates> box = place.getBounding\_box();

// Drop the last point if it's the same as the first point.

// The same logic is present in several other classes dealing with places.

// See e.g. birdherd/src/main/scala/com/twitter/birdherd/tweetypie/TweetyPiePlace.scala

if (box.get(pointsCount - 1).equals(box.get(0))) {

pointsCount--;

}

for (int i = 0; i < pointsCount; i++) {

GeoCoordinates coords = box.get(i);

sumLatitude += coords.getLatitude();

sumLongitude += coords.getLongitude();

}

double averageLatitude = sumLatitude / pointsCount;

double averageLongitude = sumLongitude / pointsCount;

return Optional.of(GeoObject.createForIngester(averageLatitude, averageLongitude));

}

}

return Optional.empty();

}

public void setRadius(double radius) {

this.radius = radius;

}

public Double getRadius() {

return radius;

}

public void setLatitude(double latitude) {

this.latitude = latitude;

}

public Double getLatitude() {

return latitude;

}

public void setLongitude(double longitude) {

this.longitude = longitude;

}

public Double getLongitude() {

return longitude;

}

public int getAccuracy() {

return accuracy;

}

public void setAccuracy(int accuracy) {

this.accuracy = accuracy;

}

public ThriftGeoLocationSource getSource() {

return source;

}

/\*\* Convers this GeoObject instance to a ThriftGeoTags instance. \*/

public ThriftGeoTags toThriftGeoTags(long twitterMessageId) {

ThriftGeoTags geoTags = new ThriftGeoTags();

geoTags.setStatusId(twitterMessageId);

geoTags.setLatitude(getLatitude());

geoTags.setLongitude(getLongitude());

geoTags.setAccuracy(accuracy);

geoTags.setGeoLocationSource(source);

return geoTags;

}

private static final double COORDS\_EQUALITY\_THRESHOLD = 1e-7;

/\*\*

\* Performs an approximate comparison between the two GeoObject instances.

\*

\* @deprecated This code is not performant and should not be used in

\* production code. Use only for tests. See SEARCH-5148.

\*/

@Deprecated

@VisibleForTesting

public static boolean approxEquals(GeoObject a, GeoObject b) {

if (a == null && b == null) {

return true;

}

if ((a == null && b != null) || (a != null && b == null)) {

return false;

}

if (a.accuracy != b.accuracy) {

return false;

}

if (Math.abs(a.latitude - b.latitude) > COORDS\_EQUALITY\_THRESHOLD) {

return false;

}

if (Math.abs(a.longitude - b.longitude) > COORDS\_EQUALITY\_THRESHOLD) {

return false;

}

if (Double.compare(a.radius, b.radius) != 0) {

return false;

}

if (a.source != b.source) {

return false;

}

return true;

}

@Override

public String toString() {

return "GeoObject{"

+ "latitude=" + latitude

+ ", longitude=" + longitude

+ ", radius=" + radius

+ ", source=" + source

+ ", accuracy=" + accuracy

+ '}';

}

/\*\*

\* Convenience factory method for ingester purposes.

\*/

public static GeoObject createForIngester(double latitude, double longitude) {

return new GeoObject(

latitude,

longitude,

// store with highest level of accuracy: POINT\_LEVEL

GeoAddressAccuracy.POINT\_LEVEL.getCode(),

ThriftGeoLocationSource.GEOTAG);

}

}