package com.twitter.tweetypie

package handler

import com.twitter.finagle.stats.Counter

import com.twitter.stitch.Stitch

import com.twitter.tweetypie.repository.PlaceKey

import com.twitter.tweetypie.repository.PlaceRepository

import com.twitter.tweetypie.serverutil.ExceptionCounter

import com.twitter.tweetypie.thriftscala.\_

object GeoStats {

val topTenCountryCodes: Set[PlaceLanguage] =

Set("US", "JP", "GB", "ID", "BR", "SA", "TR", "MX", "ES", "CA")

def apply(stats: StatsReceiver): Effect[Option[Place]] = {

val totalCount = stats.counter("total")

val notFoundCount = stats.counter("not\_found")

val countryStats: Map[String, Counter] =

topTenCountryCodes.map(cc => cc -> stats.scope("with\_country\_code").counter(cc)).toMap

val placeTypeStats: Map[PlaceType, Counter] =

Map(

PlaceType.Admin -> stats.counter("admin"),

PlaceType.City -> stats.counter("city"),

PlaceType.Country -> stats.counter("country"),

PlaceType.Neighborhood -> stats.counter("neighborhood"),

PlaceType.Poi -> stats.counter("poi"),

PlaceType.Unknown -> stats.counter("unknown")

)

Effect.fromPartial {

case Some(place) => {

totalCount.incr()

placeTypeStats(place.`type`).incr()

place.countryCode.foreach(cc => countryStats.get(cc).foreach(\_.incr()))

}

case None => notFoundCount.incr()

}

}

}

object GeoBuilder {

case class Request(createGeo: TweetCreateGeo, userGeoEnabled: Boolean, language: String)

case class Result(geoCoordinates: Option[GeoCoordinates], placeId: Option[PlaceId])

type Type = FutureArrow[Request, Result]

def apply(placeRepo: PlaceRepository.Type, rgc: ReverseGeocoder, stats: StatsReceiver): Type = {

val exceptionCounters = ExceptionCounter(stats)

def ignoreFailures[A](future: Future[Option[A]]): Future[Option[A]] =

exceptionCounters(future).handle { case \_ => None }

def isValidPlaceId(placeId: String) = PlaceIdRegex.pattern.matcher(placeId).matches

def isValidLatLon(latitude: Double, longitude: Double): Boolean =

latitude >= -90.0 && latitude <= 90.0 &&

longitude >= -180.0 && longitude <= 180.0 &&

// some clients send (0.0, 0.0) for unknown reasons, but this is highly unlikely to be

// valid and should be treated as if no coordinates were sent. if a place Id is provided,

// that will still be used.

(latitude != 0.0 || longitude != 0.0)

// Count the number of times we erase geo information based on user preferences.

val geoErasedCounter = stats.counter("geo\_erased")

// Count the number of times we override a user's preferences and add geo anyway.

val geoOverriddenCounter = stats.counter("geo\_overridden")

val geoScope = stats.scope("create\_geotagged\_tweet")

// Counter for geo tweets with neither lat lon nor place id data

val noGeoCounter = geoScope.counter("no\_geo\_info")

val invalidCoordinates = geoScope.counter("invalid\_coordinates")

val inValidPlaceId = geoScope.counter("invalid\_place\_id")

val latlonStatsEffect = GeoStats(geoScope.scope("from\_latlon"))

val placeIdStatsEffect = GeoStats(geoScope.scope("from\_place\_id"))

def validateCoordinates(coords: GeoCoordinates): Option[GeoCoordinates] =

if (isValidLatLon(coords.latitude, coords.longitude)) Some(coords)

else {

invalidCoordinates.incr()

None

}

def validatePlaceId(placeId: String): Option[String] =

if (isValidPlaceId(placeId)) Some(placeId)

else {

inValidPlaceId.incr()

None

}

def getPlaceByRGC(coordinates: GeoCoordinates, language: String): Future[Option[Place]] =

ignoreFailures(

rgc((coordinates, language)).onSuccess(latlonStatsEffect)

)

def getPlaceById(placeId: String, language: String): Future[Option[Place]] =

ignoreFailures(

Stitch

.run(placeRepo(PlaceKey(placeId, language)).liftNotFoundToOption)

.onSuccess(placeIdStatsEffect)

)

FutureArrow[Request, Result] { request =>

val createGeo = request.createGeo

val allowGeo = createGeo.overrideUserGeoSetting || request.userGeoEnabled

val overrideGeo = createGeo.overrideUserGeoSetting && !request.userGeoEnabled

if (createGeo.placeId.isEmpty && createGeo.coordinates.isEmpty) {

noGeoCounter.incr()

Future.value(Result(None, None))

} else if (!allowGeo) {

// Record that we had geo information but had to erase it based on user preferences.

geoErasedCounter.incr()

Future.value(Result(None, None))

} else {

if (overrideGeo) geoOverriddenCounter.incr()

// treat invalidate coordinates the same as no-coordinates

val validatedCoordinates = createGeo.coordinates.flatMap(validateCoordinates)

val validatedPlaceId = createGeo.placeId.flatMap(validatePlaceId)

for {

place <- (createGeo.placeId, validatedPlaceId, validatedCoordinates) match {

// if the request contains an invalid place id, we want to return None for the

// place instead of reverse-geocoding the coordinates

case (Some(\_), None, \_) => Future.None

case (\_, Some(placeId), \_) => getPlaceById(placeId, request.language)

case (\_, \_, Some(coords)) => getPlaceByRGC(coords, request.language)

case \_ => Future.None

}

} yield Result(validatedCoordinates, place.map(\_.id))

}

}

}

}