package com.twitter.follow\_recommendations.common.clients.geoduck

import com.twitter.follow\_recommendations.common.models.GeohashAndCountryCode

import com.twitter.geoduck.common.thriftscala.Location

import com.twitter.geoduck.common.thriftscala.PlaceQuery

import com.twitter.geoduck.common.thriftscala.ReverseGeocodeIPRequest

import com.twitter.geoduck.service.thriftscala.GeoContext

import com.twitter.geoduck.thriftscala.ReverseGeocoder

import com.twitter.stitch.Stitch

import javax.inject.Inject

import javax.inject.Singleton

@Singleton

class ReverseGeocodeClient @Inject() (rgcService: ReverseGeocoder.MethodPerEndpoint) {

def getGeohashAndCountryCode(ipAddress: String): Stitch[GeohashAndCountryCode] = {

Stitch

.callFuture {

rgcService

.reverseGeocodeIp(

ReverseGeocodeIPRequest(

Seq(ipAddress),

PlaceQuery(None),

simpleReverseGeocode = true

) // note: simpleReverseGeocode means that country code will be included in response

).map { response =>

response.found.get(ipAddress) match {

case Some(location) => getGeohashAndCountryCodeFromLocation(location)

case \_ => GeohashAndCountryCode(None, None)

}

}

}

}

private def getGeohashAndCountryCodeFromLocation(location: Location): GeohashAndCountryCode = {

val countryCode: Option[String] = location.simpleRgcResult.flatMap { \_.countryCodeAlpha2 }

val geohashString: Option[String] = location.geohash.flatMap { hash =>

hash.stringGeohash.flatMap { hashString =>

Some(ReverseGeocodeClient.truncate(hashString))

}

}

GeohashAndCountryCode(geohashString, countryCode)

}

}

object ReverseGeocodeClient {

val DefaultGeoduckIPRequestContext: GeoContext =

GeoContext(allPlaceTypes = true, includeGeohash = true, includeCountryCode = true)

// All these geohashes are guessed by IP (Logical Location Source).

// So take the four letters to make sure it is consistent with LocationServiceClient

val GeohashLengthAfterTruncation = 4

def truncate(geohash: String): String = geohash.take(GeohashLengthAfterTruncation)

}