package com.twitter.follow\_recommendations.common.candidate\_sources.geo

import com.google.inject.Singleton

import com.twitter.escherbird.util.stitchcache.StitchCache

import com.twitter.finagle.stats.StatsReceiver

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

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

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

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

import com.twitter.hermit.model.Algorithm

import com.twitter.hermit.pop\_geo.thriftscala.PopUsersInPlace

import com.twitter.product\_mixer.core.functional\_component.candidate\_source.CandidateSource

import com.twitter.product\_mixer.core.model.common.identifier.CandidateSourceIdentifier

import com.twitter.stitch.Stitch

import com.twitter.strato.generated.client.onboarding.userrecs.UniquePopQualityFollowUsersInPlaceClientColumn

import com.twitter.util.Duration

import javax.inject.Inject

@Singleton

class PopGeohashQualityFollowSource @Inject() (

popGeoSource: PopGeoQualityFollowSource,

statsReceiver: StatsReceiver)

extends BasePopGeohashSource(

popGeoSource = popGeoSource,

statsReceiver = statsReceiver.scope("PopGeohashQualityFollowSource"),

) {

override val identifier: CandidateSourceIdentifier = PopGeohashQualityFollowSource.Identifier

override def maxResults(target: Target): Int = {

target.params(PopGeoQualityFollowSourceParams.PopGeoSourceMaxResultsPerPrecision)

}

override def minGeohashLength(target: Target): Int = {

target.params(PopGeoQualityFollowSourceParams.PopGeoSourceGeoHashMinPrecision)

}

override def maxGeohashLength(target: Target): Int = {

target.params(PopGeoQualityFollowSourceParams.PopGeoSourceGeoHashMaxPrecision)

}

override def returnResultFromAllPrecision(target: Target): Boolean = {

target.params(PopGeoQualityFollowSourceParams.PopGeoSourceReturnFromAllPrecisions)

}

override def candidateSourceEnabled(target: Target): Boolean = {

target.params(PopGeoQualityFollowSourceParams.CandidateSourceEnabled)

}

}

object PopGeohashQualityFollowSource {

val Identifier: CandidateSourceIdentifier = CandidateSourceIdentifier(

Algorithm.PopGeohashQualityFollow.toString)

}

object PopGeoQualityFollowSource {

val MaxCacheSize = 20000

val CacheTTL: Duration = Duration.fromHours(24)

val MaxResults = 200

}

@Singleton

class PopGeoQualityFollowSource @Inject() (

popGeoQualityFollowClientColumn: UniquePopQualityFollowUsersInPlaceClientColumn,

statsReceiver: StatsReceiver,

) extends CandidateSource[String, CandidateUser] {

/\*\* @see [[CandidateSourceIdentifier]] \*/

override val identifier: CandidateSourceIdentifier = CandidateSourceIdentifier(

"PopGeoQualityFollowSource")

private val cache = StitchCache[String, Option[PopUsersInPlace]](

maxCacheSize = PopGeoQualityFollowSource.MaxCacheSize,

ttl = PopGeoQualityFollowSource.CacheTTL,

statsReceiver = statsReceiver.scope(identifier.name, "cache"),

underlyingCall = (k: String) => {

popGeoQualityFollowClientColumn.fetcher

.fetch(k)

.map { result => result.v }

}

)

override def apply(target: String): Stitch[Seq[CandidateUser]] = {

val result: Stitch[Option[PopUsersInPlace]] = cache.readThrough(target)

result.map { pu =>

pu.map { candidates =>

candidates.popUsers.sortBy(-\_.score).take(PopGeoQualityFollowSource.MaxResults).map {

candidate =>

CandidateUser(

id = candidate.userId,

score = Some(candidate.score),

reason = Some(

Reason(

Some(

AccountProof(

popularInGeoProof = Some(PopularInGeoProof(location = candidates.place))

)

)

)

)

)

}

}.getOrElse(Nil)

}

}

}