package com.twitter.follow\_recommendations.common.predicates

import com.google.inject.name.Named

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.follow\_recommendations.common.base.Predicate

import com.twitter.follow\_recommendations.common.base.PredicateResult

import com.twitter.follow\_recommendations.common.constants.GuiceNamedConstants

import com.twitter.follow\_recommendations.common.models.FilterReason.CuratedAccountsCompetitorList

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

import com.twitter.stitch.Stitch

import com.twitter.strato.client.Fetcher

import javax.inject.Inject

import javax.inject.Singleton

import com.twitter.conversions.DurationOps.\_

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

@Singleton

case class CuratedCompetitorListPredicate @Inject() (

statsReceiver: StatsReceiver,

@Named(GuiceNamedConstants.CURATED\_COMPETITOR\_ACCOUNTS\_FETCHER) competitorAccountFetcher: Fetcher[

String,

Unit,

Seq[Long]

]) extends Predicate[CandidateUser] {

private val stats: StatsReceiver = statsReceiver.scope(this.getClass.getName)

private val cacheStats = stats.scope("cache")

private val cache = StitchCache[String, Set[Long]](

maxCacheSize = CuratedCompetitorListPredicate.CacheNumberOfEntries,

ttl = CuratedCompetitorListPredicate.CacheTTL,

statsReceiver = cacheStats,

underlyingCall = (competitorListPrefix: String) => query(competitorListPrefix)

)

private def query(prefix: String): Stitch[Set[Long]] =

competitorAccountFetcher.fetch(prefix).map(\_.v.getOrElse(Nil).toSet)

/\*\*

\* Caveat here is that though the similarToUserIds allows for a Seq[Long], in practice we would

\* only return 1 userId. Multiple userId's would result in filtering candidates associated with

\* a different similarToUserId. For example:

\* - similarToUser1 -> candidate1, candidate2

\* - similarToUser2 -> candidate3

\* and in the competitorList store we have:

\* - similarToUser1 -> candidate3

\* we'll be filtering candidate3 on account of similarToUser1, even though it was generated

\* with similarToUser2. This might still be desirable at a product level (since we don't want

\* to show these accounts anyway), but might not achieve what you intend to code-wise.

\*/

override def apply(candidate: CandidateUser): Stitch[PredicateResult] = {

cache.readThrough(CuratedCompetitorListPredicate.DefaultKey).map { competitorListAccounts =>

if (competitorListAccounts.contains(candidate.id)) {

PredicateResult.Invalid(Set(CuratedAccountsCompetitorList))

} else {

PredicateResult.Valid

}

}

}

}

object CuratedCompetitorListPredicate {

val DefaultKey: String = "default\_list"

val CacheTTL = 5.minutes

val CacheNumberOfEntries = 5

}