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

import com.twitter.finagle.stats.StatsReceiver

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

import com.twitter.follow\_recommendations.common.transforms.modify\_social\_proof.ModifySocialProof

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

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

import com.twitter.product\_mixer.core.model.marshalling.request.HasClientContext

import com.twitter.stitch.Stitch

import com.twitter.timelines.configapi.HasParams

import com.twitter.util.Duration

abstract class SocialProofEnforcedCandidateSource(

candidateSource: CandidateSource[HasClientContext with HasParams, CandidateUser],

modifySocialProof: ModifySocialProof,

minNumSocialProofsRequired: Int,

override val identifier: CandidateSourceIdentifier,

baseStatsReceiver: StatsReceiver)

extends CandidateSource[HasClientContext with HasParams, CandidateUser] {

val statsReceiver = baseStatsReceiver.scope(identifier.name)

override def apply(target: HasClientContext with HasParams): Stitch[Seq[CandidateUser]] = {

val mustCallSgs: Boolean = target.params(SocialProofEnforcedCandidateSourceParams.MustCallSgs)

val callSgsCachedColumn: Boolean =

target.params(SocialProofEnforcedCandidateSourceParams.CallSgsCachedColumn)

val QueryIntersectionIdsNum: Int =

target.params(SocialProofEnforcedCandidateSourceParams.QueryIntersectionIdsNum)

val MaxNumCandidatesToAnnotate: Int =

target.params(SocialProofEnforcedCandidateSourceParams.MaxNumCandidatesToAnnotate)

val gfsIntersectionIdsNum: Int =

target.params(SocialProofEnforcedCandidateSourceParams.GfsIntersectionIdsNum)

val sgsIntersectionIdsNum: Int =

target.params(SocialProofEnforcedCandidateSourceParams.SgsIntersectionIdsNum)

val gfsLagDuration: Duration =

target.params(SocialProofEnforcedCandidateSourceParams.GfsLagDurationInDays)

candidateSource(target)

.flatMap { candidates =>

val candidatesWithoutEnoughSocialProof = candidates

.collect {

case candidate if !candidate.followedBy.exists(\_.size >= minNumSocialProofsRequired) =>

candidate

}

statsReceiver

.stat("candidates\_with\_no\_social\_proofs").add(candidatesWithoutEnoughSocialProof.size)

val candidatesToAnnotate =

candidatesWithoutEnoughSocialProof.take(MaxNumCandidatesToAnnotate)

statsReceiver.stat("candidates\_to\_annotate").add(candidatesToAnnotate.size)

val annotatedCandidatesMapStitch = target.getOptionalUserId

.map { userId =>

modifySocialProof

.hydrateSocialProof(

userId,

candidatesToAnnotate,

Some(QueryIntersectionIdsNum),

mustCallSgs,

callSgsCachedColumn,

gfsLagDuration = gfsLagDuration,

gfsIntersectionIds = gfsIntersectionIdsNum,

sgsIntersectionIds = sgsIntersectionIdsNum

).map { annotatedCandidates =>

annotatedCandidates

.map(annotatedCandidate => (annotatedCandidate.id, annotatedCandidate)).toMap

}

}.getOrElse(Stitch.value(Map.empty[Long, CandidateUser]))

annotatedCandidatesMapStitch.map { annotatedCandidatesMap =>

candidates

.flatMap { candidate =>

if (candidate.followedBy.exists(\_.size >= minNumSocialProofsRequired)) {

Some(candidate)

} else {

annotatedCandidatesMap.get(candidate.id).collect {

case annotatedCandidate

if annotatedCandidate.followedBy.exists(

\_.size >= minNumSocialProofsRequired) =>

annotatedCandidate

}

}

}.map(\_.withCandidateSource(identifier))

}

}

}

}