package com.twitter.follow\_recommendations.common.candidate\_sources.triangular\_loops

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

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

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

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

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

import com.twitter.hermit.model.Algorithm

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.strato.generated.client.onboarding.userrecs.TriangularLoopsV2OnUserClientColumn

import com.twitter.timelines.configapi.HasParams

import com.twitter.wtf.triangular\_loop.thriftscala.Candidates

import javax.inject.Inject

import javax.inject.Singleton

@Singleton

class TriangularLoopsSource @Inject() (

triangularLoopsV2Column: TriangularLoopsV2OnUserClientColumn)

extends CandidateSource[

HasParams with HasClientContext with HasRecentFollowedByUserIds,

CandidateUser

] {

override val identifier: CandidateSourceIdentifier = TriangularLoopsSource.Identifier

override def apply(

target: HasParams with HasClientContext with HasRecentFollowedByUserIds

): Stitch[Seq[CandidateUser]] = {

val candidates = target.getOptionalUserId

.map { userId =>

val fetcher = triangularLoopsV2Column.fetcher

fetcher

.fetch(userId)

.map { result =>

result.v

.map(TriangularLoopsSource.mapCandidatesToCandidateUsers)

.getOrElse(Nil)

}

}.getOrElse(Stitch.Nil)

// Make sure recentFollowedByUserIds is populated within the RequestBuilder before enable it

if (target.params(TriangularLoopsParams.KeepOnlyCandidatesWhoFollowTargetUser))

filterOutCandidatesNotFollowingTargetUser(candidates, target.recentFollowedByUserIds)

else

candidates

}

def filterOutCandidatesNotFollowingTargetUser(

candidatesStitch: Stitch[Seq[CandidateUser]],

recentFollowings: Option[Seq[Long]]

): Stitch[Seq[CandidateUser]] = {

candidatesStitch.map { candidates =>

val recentFollowingIdsSet = recentFollowings.getOrElse(Nil).toSet

candidates.filter(candidate => recentFollowingIdsSet.contains(candidate.id))

}

}

}

object TriangularLoopsSource {

val Identifier = CandidateSourceIdentifier(Algorithm.TriangularLoop.toString)

val NumResults = 100

def mapCandidatesToCandidateUsers(candidates: Candidates): Seq[CandidateUser] = {

candidates.candidates

.map { candidate =>

CandidateUser(

id = candidate.incomingUserId,

score = Some(1.0 / math

.max(1, candidate.numFollowers.getOrElse(0) + candidate.numFollowings.getOrElse(0))),

reason = Some(

Reason(

Some(

AccountProof(

followProof =

if (candidate.socialProofUserIds.isEmpty) None

else

Some(

FollowProof(

candidate.socialProofUserIds,

candidate.numSocialProof.getOrElse(candidate.socialProofUserIds.size)))

)

)

)

)

).withCandidateSource(Identifier)

}.sortBy(-\_.score.getOrElse(0.0)).take(NumResults)

}

}