package com.twitter.follow\_recommendations.common.candidate\_sources.user\_user\_graph

import com.twitter.finagle.stats.Counter

import com.twitter.finagle.stats.StatsReceiver

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

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

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.recos.recos\_common.thriftscala.UserSocialProofType

import com.twitter.recos.user\_user\_graph.thriftscala.RecommendUserDisplayLocation

import com.twitter.recos.user\_user\_graph.thriftscala.RecommendUserRequest

import com.twitter.recos.user\_user\_graph.thriftscala.RecommendUserResponse

import com.twitter.recos.user\_user\_graph.thriftscala.RecommendedUser

import com.twitter.stitch.Stitch

import com.twitter.strato.client.Fetcher

import com.twitter.timelines.configapi.HasParams

import javax.inject.Inject

import javax.inject.Named

import javax.inject.Singleton

@Singleton

class UserUserGraphCandidateSource @Inject() (

@Named(GuiceNamedConstants.USER\_USER\_GRAPH\_FETCHER)

fetcher: Fetcher[RecommendUserRequest, Unit, RecommendUserResponse],

statsReceiver: StatsReceiver)

extends CandidateSource[

UserUserGraphCandidateSource.Target,

CandidateUser

] {

override val identifier: CandidateSourceIdentifier = UserUserGraphCandidateSource.Identifier

val stats: StatsReceiver = statsReceiver.scope("UserUserGraph")

val requestCounter: Counter = stats.counter("requests")

override def apply(

target: UserUserGraphCandidateSource.Target

): Stitch[Seq[CandidateUser]] = {

if (target.params(UserUserGraphParams.UserUserGraphCandidateSourceEnabledInWeightMap)) {

requestCounter.incr()

buildRecommendUserRequest(target)

.map { request =>

fetcher

.fetch(request)

.map(\_.v)

.map { responseOpt =>

responseOpt

.map { response =>

response.recommendedUsers

.sortBy(-\_.score)

.map(convertToCandidateUsers)

.map(\_.withCandidateSource(identifier))

}.getOrElse(Nil)

}

}.getOrElse(Stitch.Nil)

} else {

Stitch.Nil

}

}

private[this] def buildRecommendUserRequest(

target: UserUserGraphCandidateSource.Target

): Option[RecommendUserRequest] = {

(target.getOptionalUserId, target.recentFollowedUserIds) match {

case (Some(userId), Some(recentFollowedUserIds)) =>

// use recentFollowedUserIds as seeds for initial experiment

val seedsWithWeights: Map[Long, Double] = recentFollowedUserIds.map {

recentFollowedUserId =>

recentFollowedUserId -> UserUserGraphCandidateSource.DefaultSeedWeight

}.toMap

val request = RecommendUserRequest(

requesterId = userId,

displayLocation = UserUserGraphCandidateSource.DisplayLocation,

seedsWithWeights = seedsWithWeights,

excludedUserIds = Some(target.excludedUserIds),

maxNumResults = Some(target.params.getInt(UserUserGraphParams.MaxCandidatesToReturn)),

maxNumSocialProofs = Some(UserUserGraphCandidateSource.MaxNumSocialProofs),

minUserPerSocialProof = Some(UserUserGraphCandidateSource.MinUserPerSocialProof),

socialProofTypes = Some(Seq(UserUserGraphCandidateSource.SocialProofType))

)

Some(request)

case \_ => None

}

}

private[this] def convertToCandidateUsers(

recommendedUser: RecommendedUser

): CandidateUser = {

val socialProofUserIds =

recommendedUser.socialProofs.getOrElse(UserUserGraphCandidateSource.SocialProofType, Nil)

val reasonOpt = if (socialProofUserIds.nonEmpty) {

Some(

Reason(

Some(AccountProof(followProof =

Some(FollowProof(socialProofUserIds, socialProofUserIds.size)))))

)

} else {

None

}

CandidateUser(

id = recommendedUser.userId,

score = Some(recommendedUser.score),

reason = reasonOpt)

}

}

object UserUserGraphCandidateSource {

type Target = HasParams

with HasClientContext

with HasRecentFollowedUserIds

with HasExcludedUserIds

val Identifier: CandidateSourceIdentifier = CandidateSourceIdentifier(

Algorithm.UserUserGraph.toString)

//Use HomeTimeline for experiment

val DisplayLocation: RecommendUserDisplayLocation = RecommendUserDisplayLocation.HomeTimeLine

//Default params used in MagicRecs

val DefaultSeedWeight: Double = 1.0

val SocialProofType = UserSocialProofType.Follow

val MaxNumSocialProofs = 10

val MinUserPerSocialProof: Map[UserSocialProofType, Int] =

Map[UserSocialProofType, Int]((SocialProofType, 2))

}