package com.twitter.follow\_recommendations.common.candidate\_sources.real\_graph

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

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

import com.twitter.hermit.model.Algorithm

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.realGraph.UserRealgraphOonV2ClientColumn

import com.twitter.timelines.configapi.HasParams

import com.twitter.wtf.candidate.thriftscala.CandidateSeq

import javax.inject.Inject

import javax.inject.Singleton

@Singleton

class RealGraphOonV2Source @Inject() (

realGraphClientColumn: UserRealgraphOonV2ClientColumn)

extends CandidateSource[HasParams with HasClientContext, CandidateUser] {

override val identifier: CandidateSourceIdentifier =

RealGraphOonV2Source.Identifier

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

request.getOptionalUserId

.map { userId =>

realGraphClientColumn.fetcher

.fetch(userId)

.map { result =>

result.v

.map { candidates => parseStratoResults(request, candidates) }

.getOrElse(Nil)

// returned candidates are sorted by score in descending order

.take(request.params(RealGraphOonParams.MaxResults))

.map(\_.withCandidateSource(identifier))

}

}.getOrElse(Stitch(Seq.empty))

}

private def parseStratoResults(

request: HasParams with HasClientContext,

candidateSeqThrift: CandidateSeq

): Seq[CandidateUser] = {

candidateSeqThrift.candidates.collect {

case candidate if candidate.score >= request.params(RealGraphOonParams.ScoreThreshold) =>

CandidateUser(

candidate.userId,

Some(candidate.score)

)

}

}

}

object RealGraphOonV2Source {

val Identifier: CandidateSourceIdentifier = CandidateSourceIdentifier(

Algorithm.RealGraphOonV2.toString

)

}