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

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.Reason

import com.twitter.hermit.stp.thriftscala.STPResult

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

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

import com.twitter.stitch.Stitch

import com.twitter.strato.client.Fetcher

import com.twitter.timelines.configapi.HasParams

/\*\* Base class that all variants of our offline stp dataset can extend. Assumes the same STPResult

\* value in the key and converts the result into the necessary internal model.

\*/

abstract class OfflineStrongTiePredictionBaseSource(

fetcher: Fetcher[Long, Unit, STPResult])

extends CandidateSource[HasParams with HasClientContext, CandidateUser] {

def fetch(

target: Long,

): Stitch[Seq[CandidateUser]] = {

fetcher

.fetch(target)

.map { result =>

result.v

.map { candidates => OfflineStrongTiePredictionBaseSource.map(target, candidates) }

.getOrElse(Nil)

.map(\_.withCandidateSource(identifier))

}

}

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

request.getOptionalUserId.map(fetch).getOrElse(Stitch.Nil)

}

}

object OfflineStrongTiePredictionBaseSource {

def map(target: Long, candidates: STPResult): Seq[CandidateUser] = {

for {

candidate <- candidates.strongTieUsers.sortBy(-\_.score)

} yield CandidateUser(

id = candidate.userId,

score = Some(candidate.score),

reason = Some(

Reason(

Some(

AccountProof(

followProof = candidate.socialProof.map(proof => FollowProof(proof, proof.size))

)

)

)

)

)

}

}