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

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

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

import com.twitter.stitch.Stitch

/\*\*

\* base trait for tweet authors based algorithms, e.g. topical tweet authors, twistly, ...

\*

\* @tparam Target target type

\* @tparam Candidate output candidate types

\*/

trait TweetAuthorsCandidateSource[-Target, +Candidate] extends CandidateSource[Target, Candidate] {

/\*\*

\* fetch Tweet candidates

\*/

def getTweetCandidates(target: Target): Stitch[Seq[TweetCandidate]]

/\*\*

\* fetch authorId

\*/

def getTweetAuthorId(tweetCandidate: TweetCandidate): Stitch[Option[Long]]

/\*\*

\* wrap candidate ID and TweetAuthorProof in Candidate

\*/

def toCandidate(authorId: Long, tweetIds: Seq[Long], score: Option[Double]): Candidate

/\*\*

\* aggregate scores, default to the first score

\*/

def aggregator(scores: Seq[Double]): Double =

scores.headOption.getOrElse(TweetAuthorsCandidateSource.DefaultScore)

/\*\*

\* aggregation method for a group of tweet candidates

\*/

def aggregateAndScore(

target: Target,

tweetCandidates: Seq[TweetCandidate]

): Seq[Candidate]

/\*\*

\* generate a list of candidates for the target

\*/

def build(

target: Target

): Stitch[Seq[Candidate]] = {

// Fetch Tweet candidates and hydrate author IDs

val tweetCandidatesStitch = for {

tweetCandidates <- getTweetCandidates(target)

authorIds <- Stitch.collect(tweetCandidates.map(getTweetAuthorId(\_)))

} yield {

for {

(authorIdOpt, tweetCandidate) <- authorIds.zip(tweetCandidates)

authorId <- authorIdOpt

} yield tweetCandidate.copy(authorId = authorId)

}

// Aggregate and score, convert to candidate

tweetCandidatesStitch.map(aggregateAndScore(target, \_))

}

def apply(target: Target): Stitch[Seq[Candidate]] =

build(target)

}

object TweetAuthorsCandidateSource {

final val DefaultScore: Double = 0.0

}