package com.twitter.timelineranker.uteg\_liked\_by\_tweets

import com.twitter.recos.recos\_common.thriftscala.SocialProofType

import com.twitter.recos.user\_tweet\_entity\_graph.thriftscala.TweetEntityDisplayLocation

import com.twitter.recos.user\_tweet\_entity\_graph.thriftscala.TweetRecommendation

import com.twitter.servo.util.FutureArrow

import com.twitter.snowflake.id.SnowflakeId

import com.twitter.timelineranker.core.CandidateEnvelope

import com.twitter.timelineranker.core.DependencyTransformer

import com.twitter.timelineranker.model.RecapQuery

import com.twitter.timelineranker.model.TimeRange

import com.twitter.timelineranker.model.TweetIdRange

import com.twitter.timelineranker.model.RecapQuery.DependencyProvider

import com.twitter.timelines.clients.user\_tweet\_entity\_graph.RecommendTweetEntityQuery

import com.twitter.timelines.clients.user\_tweet\_entity\_graph.UserTweetEntityGraphClient

import com.twitter.util.Future

object UTEGResultsTransform {

val MaxUserSocialProofSize = 10

val MaxTweetSocialProofSize = 10

val MinUserSocialProofSize = 1

def requiredTweetAuthors(query: RecapQuery): Option[Set[Long]] = {

query.utegLikedByTweetsOptions

.filter(\_.isInNetwork)

.map(\_.weightedFollowings.keySet)

}

def makeUTEGQuery(

query: RecapQuery,

socialProofTypes: Seq[SocialProofType],

utegCountProvider: DependencyProvider[Int]

): RecommendTweetEntityQuery = {

val utegLikedByTweetsOpt = query.utegLikedByTweetsOptions

RecommendTweetEntityQuery(

userId = query.userId,

displayLocation = TweetEntityDisplayLocation.HomeTimeline,

seedUserIdsWithWeights = utegLikedByTweetsOpt.map(\_.weightedFollowings).getOrElse(Map.empty),

maxTweetResults = utegCountProvider(query),

maxTweetAgeInMillis = // the "to" in the Range field is not supported by this new endpoint

query.range match {

case Some(TimeRange(from, \_)) => from.map(\_.untilNow.inMillis)

case Some(TweetIdRange(from, \_)) => from.map(SnowflakeId.timeFromId(\_).untilNow.inMillis)

case \_ => None

},

excludedTweetIds = query.excludedTweetIds,

maxUserSocialProofSize = Some(MaxUserSocialProofSize),

maxTweetSocialProofSize = Some(MaxTweetSocialProofSize),

minUserSocialProofSize = Some(MinUserSocialProofSize),

socialProofTypes = socialProofTypes,

tweetAuthors = requiredTweetAuthors(query)

)

}

}

class UTEGResultsTransform(

userTweetEntityGraphClient: UserTweetEntityGraphClient,

utegCountProvider: DependencyProvider[Int],

recommendationsFilter: DependencyTransformer[Seq[TweetRecommendation], Seq[TweetRecommendation]],

socialProofTypes: Seq[SocialProofType])

extends FutureArrow[CandidateEnvelope, CandidateEnvelope] {

override def apply(envelope: CandidateEnvelope): Future[CandidateEnvelope] = {

val utegQuery =

UTEGResultsTransform.makeUTEGQuery(envelope.query, socialProofTypes, utegCountProvider)

userTweetEntityGraphClient.findTweetRecommendations(utegQuery).map { recommendations =>

val filteredRecommendations = recommendationsFilter(envelope.query, recommendations)

val utegResultsMap = filteredRecommendations.map { recommendation =>

recommendation.tweetId -> recommendation

}.toMap

envelope.copy(utegResults = utegResultsMap)

}

}

}