package com.twitter.product\_mixer.component\_library.filter

import com.twitter.util.logging.Logging

import com.twitter.product\_mixer.component\_library.filter.TweetVisibilityFilter.\_

import com.twitter.product\_mixer.component\_library.model.candidate.BaseTweetCandidate

import com.twitter.product\_mixer.core.functional\_component.filter.Filter

import com.twitter.product\_mixer.core.functional\_component.filter.FilterResult

import com.twitter.product\_mixer.core.model.common.CandidateWithFeatures

import com.twitter.product\_mixer.core.model.common.identifier.FilterIdentifier

import com.twitter.product\_mixer.core.pipeline.PipelineQuery

import com.twitter.spam.rtf.thriftscala.SafetyLevel

import com.twitter.stitch.Stitch

import com.twitter.stitch.tweetypie.{TweetyPie => TweetypieStitchClient}

import com.twitter.tweetypie.{thriftscala => TP}

import com.twitter.util.Return

import com.twitter.util.Try

object TweetVisibilityFilter {

val DefaultTweetIncludes = Set(TP.TweetInclude.TweetFieldId(TP.Tweet.IdField.id))

private final val getTweetFieldsFailureMessage = "TweetyPie.getTweetFields failed: "

}

case class TweetVisibilityFilter[Candidate <: BaseTweetCandidate](

tweetypieStitchClient: TweetypieStitchClient,

tweetVisibilityPolicy: TP.TweetVisibilityPolicy,

safetyLevel: SafetyLevel,

tweetIncludes: Set[TP.TweetInclude.TweetFieldId] = DefaultTweetIncludes)

extends Filter[PipelineQuery, Candidate]

with Logging {

override val identifier: FilterIdentifier = FilterIdentifier("TweetVisibility")

def apply(

query: PipelineQuery,

candidates: Seq[CandidateWithFeatures[Candidate]]

): Stitch[FilterResult[Candidate]] = {

Stitch

.traverse(candidates.map(\_.candidate.id)) { tweetId =>

tweetypieStitchClient

.getTweetFields(tweetId, getTweetFieldsOptions(query.getOptionalUserId))

.liftToTry

}

.map { getTweetFieldsResults: Seq[Try[TP.GetTweetFieldsResult]] =>

val (checkedSucceeded, checkFailed) = getTweetFieldsResults.partition(\_.isReturn)

checkFailed.foreach(e => warn(() => getTweetFieldsFailureMessage, e.throwable))

if (checkFailed.nonEmpty) {

warn(() =>

s"TweetVisibilityFilter dropped ${checkFailed.size} candidates due to tweetypie failure.")

}

val allowedTweets = checkedSucceeded.collect {

case Return(TP.GetTweetFieldsResult(\_, TP.TweetFieldsResultState.Found(found), \_, \_)) =>

found.tweet.id

}.toSet

val (kept, removed) =

candidates.map(\_.candidate).partition(candidate => allowedTweets.contains(candidate.id))

FilterResult(kept = kept, removed = removed)

}

}

private def getTweetFieldsOptions(userId: Option[Long]) =

TP.GetTweetFieldsOptions(

forUserId = userId,

tweetIncludes = tweetIncludes.toSet,

doNotCache = true,

visibilityPolicy = tweetVisibilityPolicy,

safetyLevel = Some(safetyLevel)

)

}