package com.twitter.home\_mixer.functional\_component.filter

import com.twitter.home\_mixer.model.HomeFeatures.IsRetweetFeature

import com.twitter.home\_mixer.util.CandidatesUtil

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

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.stitch.Stitch

import scala.collection.mutable

object RetweetDeduplicationFilter extends Filter[PipelineQuery, TweetCandidate] {

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

override def apply(

query: PipelineQuery,

candidates: Seq[CandidateWithFeatures[TweetCandidate]]

): Stitch[FilterResult[TweetCandidate]] = {

// If there are 2 retweets of the same native tweet, we will choose the first one

// The tweets are returned in descending score order, so we will choose the higher scored tweet

val dedupedTweetIdsSet =

candidates.partition(\_.features.getOrElse(IsRetweetFeature, false)) match {

case (retweets, nativeTweets) =>

val nativeTweetIds = nativeTweets.map(\_.candidate.id)

val seenTweetIds = mutable.Set[Long]() ++ nativeTweetIds

val dedupedRetweets = retweets.filter { retweet =>

val tweetIdAndSourceId = CandidatesUtil.getTweetIdAndSourceId(retweet)

val retweetIsUnique = tweetIdAndSourceId.forall(!seenTweetIds.contains(\_))

if (retweetIsUnique) {

seenTweetIds ++= tweetIdAndSourceId

}

retweetIsUnique

}

(nativeTweets ++ dedupedRetweets).map(\_.candidate.id).toSet

}

val (kept, removed) =

candidates

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

Stitch.value(FilterResult(kept = kept, removed = removed))

}

}