package com.twitter.home\_mixer.product.scored\_tweets.filter

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

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

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

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

/\*\*

\* This filter removes source tweets of retweets, added via second EB call in TLR

\*/

object RetweetSourceTweetRemovingFilter extends Filter[PipelineQuery, TweetCandidate] {

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

override def apply(

query: PipelineQuery,

candidates: Seq[CandidateWithFeatures[TweetCandidate]]

): Stitch[FilterResult[TweetCandidate]] = {

val (kept, removed) =

candidates.partition(

\_.features.getOrElse(EarlybirdFeature, None).exists(\_.isSourceTweet)) match {

case (sourceTweets, nonSourceTweets) =>

val inReplyToTweetIds: Set[Long] =

nonSourceTweets

.filter(ReplyRetweetUtil.isEligibleReply(\_)).flatMap(

\_.features.getOrElse(InReplyToTweetIdFeature, None)).toSet

val (keptSourceTweets, removedSourceTweets) = sourceTweets

.map(\_.candidate)

.partition(candidate => inReplyToTweetIds.contains(candidate.id))

(nonSourceTweets.map(\_.candidate) ++ keptSourceTweets, removedSourceTweets)

}

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

}

}