package com.twitter.product\_mixer.component\_library.pipeline.candidate.ads

import com.twitter.product\_mixer.component\_library.model.candidate.ads.AdsCandidate

import com.twitter.product\_mixer.component\_library.model.candidate.ads.AdsTweetCandidate

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

case class PromotedTweetsOnlyFilter[Query <: PipelineQuery](

underlyingFilter: Filter[Query, AdsTweetCandidate])

extends Filter[Query, AdsCandidate] {

override val identifier: FilterIdentifier =

FilterIdentifier(s"PromotedTweets${underlyingFilter.identifier.name}")

override def apply(

query: Query,

candidatesWithFeatures: Seq[CandidateWithFeatures[AdsCandidate]]

): Stitch[FilterResult[AdsCandidate]] = {

val adsTweetCandidates: Seq[CandidateWithFeatures[AdsTweetCandidate]] =

candidatesWithFeatures.flatMap {

case tweetCandidateWithFeatures @ CandidateWithFeatures(\_: AdsTweetCandidate, \_) =>

Some(tweetCandidateWithFeatures.asInstanceOf[CandidateWithFeatures[AdsTweetCandidate]])

case \_ => None

}

underlyingFilter

.apply(query, adsTweetCandidates)

.map { filterResult =>

val removedSet = filterResult.removed.toSet[AdsCandidate]

val (removed, kept) = candidatesWithFeatures.map(\_.candidate).partition(removedSet.contains)

FilterResult(kept, removed)

}

}

}