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

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

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

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

object PreviouslyServedTweetsFilter

extends Filter[PipelineQuery, TweetCandidate]

with Filter.Conditionally[PipelineQuery, TweetCandidate] {

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

override def onlyIf(

query: PipelineQuery,

candidates: Seq[CandidateWithFeatures[TweetCandidate]]

): Boolean = {

query.features.exists(\_.getOrElse(GetOlderFeature, false))

}

override def apply(

query: PipelineQuery,

candidates: Seq[CandidateWithFeatures[TweetCandidate]]

): Stitch[FilterResult[TweetCandidate]] = {

val servedTweetIds =

query.features.map(\_.getOrElse(ServedTweetIdsFeature, Seq.empty)).toSeq.flatten.toSet

val (removed, kept) = candidates.partition { candidate =>

val tweetIdAndSourceId = CandidatesUtil.getTweetIdAndSourceId(candidate)

tweetIdAndSourceId.exists(servedTweetIds.contains)

}

Stitch.value(FilterResult(kept = kept.map(\_.candidate), removed = removed.map(\_.candidate)))

}

}