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

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

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

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

/\*\*

\* Exclude conversation modules where Tweets have been dropped by other filters

\*

\* Largest conversation modules have 3 Tweets, so if all 3 are present, module is valid.

\* For 2 Tweet modules, check if the head is the root (not a reply) and the last item

\* is actually replying to the root directly with no missing intermediate tweets

\*/

object InvalidConversationModuleFilter extends Filter[PipelineQuery, TweetCandidate] {

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

val ValidThreeTweetModuleSize = 3

val ValidTwoTweetModuleSize = 2

override def apply(

query: PipelineQuery,

candidates: Seq[CandidateWithFeatures[TweetCandidate]]

): Stitch[FilterResult[TweetCandidate]] = {

val allowedTweetIds = candidates

.groupBy(\_.features.getOrElse(ConversationModuleFocalTweetIdFeature, None))

.map { case (id, candidates) => (id, candidates.sortBy(\_.candidate.id)) }

.filter {

case (Some(\_), conversation) if conversation.size == ValidThreeTweetModuleSize => true

case (Some(focalId), conversation) if conversation.size == ValidTwoTweetModuleSize =>

conversation.head.features.getOrElse(InReplyToTweetIdFeature, None).isEmpty &&

conversation.last.candidate.id == focalId &&

conversation.last.features

.getOrElse(InReplyToTweetIdFeature, None)

.contains(conversation.head.candidate.id)

case (None, \_) => true

case \_ => false

}.values.flatten.toSeq.map(\_.candidate.id).toSet

val (kept, removed) =

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

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

}

}