package com.twitter.home\_mixer.product.for\_you.filter

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

import com.twitter.home\_mixer.product.for\_you.param.ForYouParam

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

import com.twitter.product\_mixer.core.feature.featuremap.FeatureMap

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 com.twitter.timelineservice.suggests.{thriftscala => st}

object SocialContextFilter extends Filter[PipelineQuery, TweetCandidate] {

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

// Tweets from candidate sources which don't need generic like/follow/topic proof

private val AllowedSources: Set[st.SuggestType] = Set(

st.SuggestType.RankedListTweet,

st.SuggestType.RecommendedTrendTweet,

st.SuggestType.MediaTweet

)

override def apply(

query: PipelineQuery,

candidates: Seq[CandidateWithFeatures[TweetCandidate]]

): Stitch[FilterResult[TweetCandidate]] = {

val enableIsVerifiedAuthorFilter =

query.params(ForYouParam.EnableVerifiedAuthorSocialContextBypassParam)

val enableTopicSocialContextFilter =

query.params(ForYouParam.EnableTopicSocialContextFilterParam)

val validTweetIds = candidates

.filter { candidate =>

candidate.features.getOrElse(InNetworkFeature, true) ||

candidate.features.getOrElse(SuggestTypeFeature, None).exists(AllowedSources.contains) ||

candidate.features.getOrElse(ConversationModuleFocalTweetIdFeature, None).isDefined ||

(enableIsVerifiedAuthorFilter && isVerifiedAuthor(candidate.features)) ||

hasLikedBySocialContext(candidate.features) ||

hasFollowedBySocialContext(candidate.features) ||

(enableTopicSocialContextFilter && hasTopicSocialContext(candidate.features))

}.map(\_.candidate.id).toSet

val (kept, removed) =

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

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

}

private def isVerifiedAuthor(candidateFeatures: FeatureMap): Boolean = {

candidateFeatures.getOrElse(AuthorIsBlueVerifiedFeature, false) ||

candidateFeatures.getOrElse(AuthorIsGoldVerifiedFeature, false) ||

candidateFeatures.getOrElse(AuthorIsGrayVerifiedFeature, false) ||

candidateFeatures.getOrElse(AuthorIsLegacyVerifiedFeature, false)

}

private def hasLikedBySocialContext(candidateFeatures: FeatureMap): Boolean =

candidateFeatures

.getOrElse(SGSValidLikedByUserIdsFeature, Seq.empty)

.exists(

candidateFeatures

.getOrElse(PerspectiveFilteredLikedByUserIdsFeature, Seq.empty)

.toSet.contains

)

private def hasFollowedBySocialContext(candidateFeatures: FeatureMap): Boolean =

candidateFeatures.getOrElse(SGSValidFollowedByUserIdsFeature, Seq.empty).nonEmpty

private def hasTopicSocialContext(candidateFeatures: FeatureMap): Boolean = {

candidateFeatures.getOrElse(TopicIdSocialContextFeature, None).isDefined &&

candidateFeatures.getOrElse(TopicContextFunctionalityTypeFeature, None).isDefined

}

}