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

import com.twitter.conversions.DurationOps.\_

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

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

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

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

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

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

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

import com.twitter.product\_mixer.core.pipeline.PipelineQuery

import com.twitter.stitch.Stitch

import com.twitter.timelines.common.thriftscala.FeedbackEntity

import com.twitter.timelineservice.model.FeedbackEntry

import com.twitter.timelineservice.{thriftscala => tls}

object FeedbackFatigueFilter

extends Filter[PipelineQuery, TweetCandidate]

with Filter.Conditionally[PipelineQuery, TweetCandidate] {

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

override def onlyIf(

query: PipelineQuery,

candidates: Seq[CandidateWithFeatures[TweetCandidate]]

): Boolean =

query.features.exists(\_.getOrElse(FeedbackHistoryFeature, Seq.empty).nonEmpty)

private val DurationForFiltering = 14.days

override def apply(

query: pipeline.PipelineQuery,

candidates: Seq[CandidateWithFeatures[TweetCandidate]]

): Stitch[FilterResult[TweetCandidate]] = {

val feedbackEntriesByEngagementType =

query.features

.getOrElse(FeatureMap.empty).getOrElse(FeedbackHistoryFeature, Seq.empty)

.filter { entry =>

val timeSinceFeedback = query.queryTime.minus(entry.timestamp)

timeSinceFeedback < DurationForFiltering &&

entry.feedbackType == tls.FeedbackType.SeeFewer

}.groupBy(\_.engagementType)

val authorsToFilter =

getUserIds(

feedbackEntriesByEngagementType.getOrElse(tls.FeedbackEngagementType.Tweet, Seq.empty))

val likersToFilter =

getUserIds(

feedbackEntriesByEngagementType.getOrElse(tls.FeedbackEngagementType.Like, Seq.empty))

val followersToFilter =

getUserIds(

feedbackEntriesByEngagementType.getOrElse(tls.FeedbackEngagementType.Follow, Seq.empty))

val retweetersToFilter =

getUserIds(

feedbackEntriesByEngagementType.getOrElse(tls.FeedbackEngagementType.Retweet, Seq.empty))

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

val originalAuthorId = CandidatesUtil.getOriginalAuthorId(candidate.features)

val authorId = candidate.features.getOrElse(AuthorIdFeature, None)

val likers = candidate.features.getOrElse(SGSValidLikedByUserIdsFeature, Seq.empty)

val eligibleLikers = likers.filterNot(likersToFilter.contains)

val followers = candidate.features.getOrElse(SGSValidFollowedByUserIdsFeature, Seq.empty)

val eligibleFollowers = followers.filterNot(followersToFilter.contains)

originalAuthorId.exists(authorsToFilter.contains) ||

(likers.nonEmpty && eligibleLikers.isEmpty) ||

(followers.nonEmpty && eligibleFollowers.isEmpty && likers.isEmpty) ||

(candidate.features.getOrElse(IsRetweetFeature, false) &&

authorId.exists(retweetersToFilter.contains))

}

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

}

private def getUserIds(

feedbackEntries: Seq[FeedbackEntry],

): Set[Long] =

feedbackEntries.collect {

case FeedbackEntry(\_, \_, FeedbackEntity.UserId(userId), \_, \_) => userId

}.toSet

}