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

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.finagle.tracing.Trace

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

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.socialgraph.{thriftscala => sg}

import com.twitter.stitch.Stitch

import com.twitter.stitch.socialgraph.SocialGraph

import com.twitter.util.logging.Logging

import javax.inject.Inject

import javax.inject.Singleton

/\*\*

\* Exclude invalid subscription tweets - cases where the viewer is not subscribed to the author

\*

\* If SGS hydration fails, `SGSInvalidSubscriptionTweetFeature` will be set to None for

\* subscription tweets, so we explicitly filter those tweets out.

\*/

@Singleton

case class InvalidSubscriptionTweetFilter @Inject() (

socialGraphClient: SocialGraph,

statsReceiver: StatsReceiver)

extends Filter[PipelineQuery, TweetCandidate]

with Logging {

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

private val scopedStatsReceiver = statsReceiver.scope(identifier.toString)

private val validCounter = scopedStatsReceiver.counter("validExclusiveTweet")

private val invalidCounter = scopedStatsReceiver.counter("invalidExclusiveTweet")

override def apply(

query: PipelineQuery,

candidates: Seq[CandidateWithFeatures[TweetCandidate]]

): Stitch[FilterResult[TweetCandidate]] = Stitch

.traverse(candidates) { candidate =>

val exclusiveAuthorId =

candidate.features.getOrElse(ExclusiveConversationAuthorIdFeature, None)

if (exclusiveAuthorId.isDefined) {

val request = sg.ExistsRequest(

source = query.getRequiredUserId,

target = exclusiveAuthorId.get,

relationships =

Seq(sg.Relationship(sg.RelationshipType.TierOneSuperFollowing, hasRelationship = true)),

)

socialGraphClient.exists(request).map(\_.exists).map { valid =>

if (!valid) invalidCounter.incr() else validCounter.incr()

valid

}

} else Stitch.value(true)

}.map { validResults =>

val (kept, removed) = candidates

.map(\_.candidate)

.zip(validResults)

.partition { case (candidate, valid) => valid }

val keptCandidates = kept.map { case (candidate, \_) => candidate }

val removedCandidates = removed.map { case (candidate, \_) => candidate }

FilterResult(kept = keptCandidates, removed = removedCandidates)

}

}