package com.twitter.tsp.stores

import com.twitter.conversions.DurationOps.\_

import com.twitter.tsp.thriftscala.TspTweetInfo

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.thriftscala.TweetHealthScores

import com.twitter.frigate.thriftscala.UserAgathaScores

import com.twitter.logging.Logger

import com.twitter.mediaservices.commons.thriftscala.MediaCategory

import com.twitter.mediaservices.commons.tweetmedia.thriftscala.MediaInfo

import com.twitter.mediaservices.commons.tweetmedia.thriftscala.MediaSizeType

import com.twitter.simclusters\_v2.common.TweetId

import com.twitter.simclusters\_v2.common.UserId

import com.twitter.spam.rtf.thriftscala.SafetyLevel

import com.twitter.stitch.Stitch

import com.twitter.stitch.storehaus.ReadableStoreOfStitch

import com.twitter.stitch.tweetypie.TweetyPie

import com.twitter.stitch.tweetypie.TweetyPie.TweetyPieException

import com.twitter.storehaus.ReadableStore

import com.twitter.topiclisting.AnnotationRuleProvider

import com.twitter.tsp.utils.HealthSignalsUtils

import com.twitter.tweetypie.thriftscala.TweetInclude

import com.twitter.tweetypie.thriftscala.{Tweet => TTweet}

import com.twitter.tweetypie.thriftscala.\_

import com.twitter.util.Duration

import com.twitter.util.Future

import com.twitter.util.TimeoutException

import com.twitter.util.Timer

object TweetyPieFieldsStore {

// Tweet fields options. Only fields specified here will be hydrated in the tweet

private val CoreTweetFields: Set[TweetInclude] = Set[TweetInclude](

TweetInclude.TweetFieldId(TTweet.IdField.id),

TweetInclude.TweetFieldId(TTweet.CoreDataField.id), // needed for the authorId

TweetInclude.TweetFieldId(TTweet.LanguageField.id),

TweetInclude.CountsFieldId(StatusCounts.FavoriteCountField.id),

TweetInclude.CountsFieldId(StatusCounts.RetweetCountField.id),

TweetInclude.TweetFieldId(TTweet.QuotedTweetField.id),

TweetInclude.TweetFieldId(TTweet.MediaKeysField.id),

TweetInclude.TweetFieldId(TTweet.EscherbirdEntityAnnotationsField.id),

TweetInclude.TweetFieldId(TTweet.MediaField.id),

TweetInclude.TweetFieldId(TTweet.UrlsField.id)

)

private val gtfo: GetTweetFieldsOptions = GetTweetFieldsOptions(

tweetIncludes = CoreTweetFields,

safetyLevel = Some(SafetyLevel.Recommendations)

)

def getStoreFromTweetyPie(

tweetyPie: TweetyPie,

convertExceptionsToNotFound: Boolean = true

): ReadableStore[Long, GetTweetFieldsResult] = {

val log = Logger("TweetyPieFieldsStore")

ReadableStoreOfStitch { tweetId: Long =>

tweetyPie

.getTweetFields(tweetId, options = gtfo)

.rescue {

case ex: TweetyPieException if convertExceptionsToNotFound =>

log.error(ex, s"Error while hitting tweetypie ${ex.result}")

Stitch.NotFound

}

}

}

}

object TweetInfoStore {

case class IsPassTweetHealthFilters(tweetStrictest: Option[Boolean])

case class IsPassAgathaHealthFilters(agathaStrictest: Option[Boolean])

private val HealthStoreTimeout: Duration = 40.milliseconds

private val isPassTweetHealthFilters: IsPassTweetHealthFilters = IsPassTweetHealthFilters(None)

private val isPassAgathaHealthFilters: IsPassAgathaHealthFilters = IsPassAgathaHealthFilters(None)

}

case class TweetInfoStore(

tweetFieldsStore: ReadableStore[TweetId, GetTweetFieldsResult],

tweetHealthModelStore: ReadableStore[TweetId, TweetHealthScores],

userHealthModelStore: ReadableStore[UserId, UserAgathaScores],

timer: Timer

)(

statsReceiver: StatsReceiver)

extends ReadableStore[TweetId, TspTweetInfo] {

import TweetInfoStore.\_

private[this] def toTweetInfo(

tweetFieldsResult: GetTweetFieldsResult

): Future[Option[TspTweetInfo]] = {

tweetFieldsResult.tweetResult match {

case result: TweetFieldsResultState.Found if result.found.suppressReason.isEmpty =>

val tweet = result.found.tweet

val authorIdOpt = tweet.coreData.map(\_.userId)

val favCountOpt = tweet.counts.flatMap(\_.favoriteCount)

val languageOpt = tweet.language.map(\_.language)

val hasImageOpt =

tweet.mediaKeys.map(\_.map(\_.mediaCategory).exists(\_ == MediaCategory.TweetImage))

val hasGifOpt =

tweet.mediaKeys.map(\_.map(\_.mediaCategory).exists(\_ == MediaCategory.TweetGif))

val isNsfwAuthorOpt = Some(

tweet.coreData.exists(\_.nsfwUser) || tweet.coreData.exists(\_.nsfwAdmin))

val isTweetReplyOpt = tweet.coreData.map(\_.reply.isDefined)

val hasMultipleMediaOpt =

tweet.mediaKeys.map(\_.map(\_.mediaCategory).size > 1)

val isKGODenylist = Some(

tweet.escherbirdEntityAnnotations

.exists(\_.entityAnnotations.exists(AnnotationRuleProvider.isSuppressedTopicsDenylist)))

val isNullcastOpt = tweet.coreData.map(\_.nullcast) // These are Ads. go/nullcast

val videoDurationOpt = tweet.media.flatMap(\_.flatMap {

\_.mediaInfo match {

case Some(MediaInfo.VideoInfo(info)) =>

Some((info.durationMillis + 999) / 1000) // video playtime always round up

case \_ => None

}

}.headOption)

// There many different types of videos. To be robust to new types being added, we just use

// the videoDurationOpt to keep track of whether the item has a video or not.

val hasVideo = videoDurationOpt.isDefined

val mediaDimensionsOpt =

tweet.media.flatMap(\_.headOption.flatMap(

\_.sizes.find(\_.sizeType == MediaSizeType.Orig).map(size => (size.width, size.height))))

val mediaWidth = mediaDimensionsOpt.map(\_.\_1).getOrElse(1)

val mediaHeight = mediaDimensionsOpt.map(\_.\_2).getOrElse(1)

// high resolution media's width is always greater than 480px and height is always greater than 480px

val isHighMediaResolution = mediaHeight > 480 && mediaWidth > 480

val isVerticalAspectRatio = mediaHeight >= mediaWidth && mediaWidth > 1

val hasUrlOpt = tweet.urls.map(\_.nonEmpty)

(authorIdOpt, favCountOpt) match {

case (Some(authorId), Some(favCount)) =>

hydrateHealthScores(tweet.id, authorId).map {

case (isPassAgathaHealthFilters, isPassTweetHealthFilters) =>

Some(

TspTweetInfo(

authorId = authorId,

favCount = favCount,

language = languageOpt,

hasImage = hasImageOpt,

hasVideo = Some(hasVideo),

hasGif = hasGifOpt,

isNsfwAuthor = isNsfwAuthorOpt,

isKGODenylist = isKGODenylist,

isNullcast = isNullcastOpt,

videoDurationSeconds = videoDurationOpt,

isHighMediaResolution = Some(isHighMediaResolution),

isVerticalAspectRatio = Some(isVerticalAspectRatio),

isPassAgathaHealthFilterStrictest = isPassAgathaHealthFilters.agathaStrictest,

isPassTweetHealthFilterStrictest = isPassTweetHealthFilters.tweetStrictest,

isReply = isTweetReplyOpt,

hasMultipleMedia = hasMultipleMediaOpt,

hasUrl = hasUrlOpt

))

}

case \_ =>

statsReceiver.counter("missingFields").incr()

Future.None // These values should always exist.

}

case \_: TweetFieldsResultState.NotFound =>

statsReceiver.counter("notFound").incr()

Future.None

case \_: TweetFieldsResultState.Failed =>

statsReceiver.counter("failed").incr()

Future.None

case \_: TweetFieldsResultState.Filtered =>

statsReceiver.counter("filtered").incr()

Future.None

case \_ =>

statsReceiver.counter("unknown").incr()

Future.None

}

}

private[this] def hydrateHealthScores(

tweetId: TweetId,

authorId: Long

): Future[(IsPassAgathaHealthFilters, IsPassTweetHealthFilters)] = {

Future

.join(

tweetHealthModelStore

.multiGet(Set(tweetId))(tweetId),

userHealthModelStore

.multiGet(Set(authorId))(authorId)

).map {

case (tweetHealthScoresOpt, userAgathaScoresOpt) =>

// This stats help us understand empty rate for AgathaCalibratedNsfw / NsfwTextUserScore

statsReceiver.counter("totalCountAgathaScore").incr()

if (userAgathaScoresOpt.getOrElse(UserAgathaScores()).agathaCalibratedNsfw.isEmpty)

statsReceiver.counter("emptyCountAgathaCalibratedNsfw").incr()

if (userAgathaScoresOpt.getOrElse(UserAgathaScores()).nsfwTextUserScore.isEmpty)

statsReceiver.counter("emptyCountNsfwTextUserScore").incr()

val isPassAgathaHealthFilters = IsPassAgathaHealthFilters(

agathaStrictest =

Some(HealthSignalsUtils.isTweetAgathaModelQualified(userAgathaScoresOpt)),

)

val isPassTweetHealthFilters = IsPassTweetHealthFilters(

tweetStrictest =

Some(HealthSignalsUtils.isTweetHealthModelQualified(tweetHealthScoresOpt))

)

(isPassAgathaHealthFilters, isPassTweetHealthFilters)

}.raiseWithin(HealthStoreTimeout)(timer).rescue {

case \_: TimeoutException =>

statsReceiver.counter("hydrateHealthScoreTimeout").incr()

Future.value((isPassAgathaHealthFilters, isPassTweetHealthFilters))

case \_ =>

statsReceiver.counter("hydrateHealthScoreFailure").incr()

Future.value((isPassAgathaHealthFilters, isPassTweetHealthFilters))

}

}

override def multiGet[K1 <: TweetId](ks: Set[K1]): Map[K1, Future[Option[TspTweetInfo]]] = {

statsReceiver.counter("tweetFieldsStore").incr(ks.size)

tweetFieldsStore

.multiGet(ks).mapValues(\_.flatMap { \_.map { v => toTweetInfo(v) }.getOrElse(Future.None) })

}

}