package com.twitter.visibility.builder.tweets

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.mediaservices.commons.mediainformation.thriftscala.AdditionalMetadata

import com.twitter.mediaservices.media\_util.GenericMediaKey

import com.twitter.stitch.Stitch

import com.twitter.tweetypie.thriftscala.Tweet

import com.twitter.visibility.builder.FeatureMapBuilder

import com.twitter.visibility.common.TweetMediaMetadataSource

import com.twitter.visibility.features.HasDmcaMediaFeature

import com.twitter.visibility.features.MediaGeoRestrictionsAllowList

import com.twitter.visibility.features.MediaGeoRestrictionsDenyList

class TweetMediaMetadataFeatures(

mediaMetadataSource: TweetMediaMetadataSource,

statsReceiver: StatsReceiver) {

private[this] val scopedStatsReceiver = statsReceiver.scope("tweet\_media\_metadata\_features")

private[this] val reportedStats = scopedStatsReceiver.scope("dmcaStats")

def forTweet(

tweet: Tweet,

mediaKeys: Seq[GenericMediaKey],

enableFetchMediaMetadata: Boolean

): FeatureMapBuilder => FeatureMapBuilder = { featureMapBuilder =>

featureMapBuilder.withFeature(

HasDmcaMediaFeature,

mediaIsDmca(tweet, mediaKeys, enableFetchMediaMetadata))

featureMapBuilder.withFeature(

MediaGeoRestrictionsAllowList,

allowlist(tweet, mediaKeys, enableFetchMediaMetadata))

featureMapBuilder.withFeature(

MediaGeoRestrictionsDenyList,

denylist(tweet, mediaKeys, enableFetchMediaMetadata))

}

private def mediaIsDmca(

tweet: Tweet,

mediaKeys: Seq[GenericMediaKey],

enableFetchMediaMetadata: Boolean

) = getMediaAdditionalMetadata(tweet, mediaKeys, enableFetchMediaMetadata)

.map(\_.exists(\_.restrictions.exists(\_.isDmca)))

private def allowlist(

tweet: Tweet,

mediaKeys: Seq[GenericMediaKey],

enableFetchMediaMetadata: Boolean

) = getMediaGeoRestrictions(tweet, mediaKeys, enableFetchMediaMetadata)

.map(\_.flatMap(\_.whitelistedCountryCodes))

private def denylist(

tweet: Tweet,

mediaKeys: Seq[GenericMediaKey],

enableFetchMediaMetadata: Boolean

) = getMediaGeoRestrictions(tweet, mediaKeys, enableFetchMediaMetadata)

.map(\_.flatMap(\_.blacklistedCountryCodes))

private def getMediaGeoRestrictions(

tweet: Tweet,

mediaKeys: Seq[GenericMediaKey],

enableFetchMediaMetadata: Boolean

) = {

getMediaAdditionalMetadata(tweet, mediaKeys, enableFetchMediaMetadata)

.map(additionalMetadatasSeq => {

for {

additionalMetadata <- additionalMetadatasSeq

restrictions <- additionalMetadata.restrictions

geoRestrictions <- restrictions.geoRestrictions

} yield {

geoRestrictions

}

})

}

private def getMediaAdditionalMetadata(

tweet: Tweet,

mediaKeys: Seq[GenericMediaKey],

enableFetchMediaMetadata: Boolean

): Stitch[Seq[AdditionalMetadata]] = {

if (mediaKeys.isEmpty) {

reportedStats.counter("empty").incr()

Stitch.value(Seq.empty)

} else {

tweet.media.flatMap { mediaEntities =>

val alreadyHydratedMetadata = mediaEntities

.filter(\_.mediaKey.isDefined)

.flatMap(\_.additionalMetadata)

if (alreadyHydratedMetadata.nonEmpty) {

Some(alreadyHydratedMetadata)

} else {

None

}

} match {

case Some(additionalMetadata) =>

reportedStats.counter("already\_hydrated").incr()

Stitch.value(additionalMetadata)

case None =>

Stitch

.collect(

mediaKeys.map(fetchAdditionalMetadata(tweet.id, \_, enableFetchMediaMetadata))

).map(maybeMetadatas => {

maybeMetadatas

.filter(\_.isDefined)

.map(\_.get)

})

}

}

}

private def fetchAdditionalMetadata(

tweetId: Long,

genericMediaKey: GenericMediaKey,

enableFetchMediaMetadata: Boolean

): Stitch[Option[AdditionalMetadata]] =

if (enableFetchMediaMetadata) {

genericMediaKey.toThriftMediaKey() match {

case Some(mediaKey) =>

reportedStats.counter("request").incr()

mediaMetadataSource.fetch(tweetId, mediaKey)

case None =>

reportedStats.counter("empty\_key").incr()

Stitch.None

}

} else {

reportedStats.counter("light\_request").incr()

Stitch.None

}

}