package com.twitter.home\_mixer.util.tweetypie.content

import com.twitter.home\_mixer.model.ContentFeatures

import com.twitter.mediaservices.commons.mediainformation.{thriftscala => mi}

import com.twitter.mediaservices.commons.tweetmedia.{thriftscala => tm}

import com.twitter.mediaservices.commons.{thriftscala => ms}

import com.twitter.tweetypie.{thriftscala => tp}

import scala.collection.Map

object TweetMediaFeaturesExtractor {

private val ImageCategories = Set(

ms.MediaCategory.TweetImage.value,

ms.MediaCategory.TweetGif.value

)

private val VideoCategories = Set(

ms.MediaCategory.TweetVideo.value,

ms.MediaCategory.AmplifyVideo.value

)

def hasImage(tweet: tp.Tweet): Boolean = hasMediaByCategory(tweet, ImageCategories)

def hasVideo(tweet: tp.Tweet): Boolean = hasMediaByCategory(tweet, VideoCategories)

private def hasMediaByCategory(tweet: tp.Tweet, categories: Set[Int]): Boolean = {

tweet.media.exists { mediaEntities =>

mediaEntities.exists { mediaEntity =>

mediaEntity.mediaKey.map(\_.mediaCategory).exists { mediaCategory =>

categories.contains(mediaCategory.value)

}

}

}

}

def addMediaFeaturesFromTweet(

inputFeatures: ContentFeatures,

tweet: tp.Tweet,

): ContentFeatures = {

val featuresWithMediaEntity = tweet.media

.map { mediaEntities =>

val sizeFeatures = getSizeFeatures(mediaEntities)

val playbackFeatures = getPlaybackFeatures(mediaEntities)

val mediaWidths = sizeFeatures.map(\_.width.toShort)

val mediaHeights = sizeFeatures.map(\_.height.toShort)

val resizeMethods = sizeFeatures.map(\_.resizeMethod.toShort)

val faceMapAreas = getFaceMapAreas(mediaEntities)

val sortedColorPalette = getSortedColorPalette(mediaEntities)

val stickerFeatures = getStickerFeatures(mediaEntities)

val mediaOriginProviders = getMediaOriginProviders(mediaEntities)

val isManaged = getIsManaged(mediaEntities)

val is360 = getIs360(mediaEntities)

val viewCount = getViewCount(mediaEntities)

val userDefinedProductMetadataFeatures =

getUserDefinedProductMetadataFeatures(mediaEntities)

val isMonetizable =

getOptBooleanFromSeqOpt(userDefinedProductMetadataFeatures.map(\_.isMonetizable))

val isEmbeddable =

getOptBooleanFromSeqOpt(userDefinedProductMetadataFeatures.map(\_.isEmbeddable))

val hasSelectedPreviewImage =

getOptBooleanFromSeqOpt(userDefinedProductMetadataFeatures.map(\_.hasSelectedPreviewImage))

val hasTitle = getOptBooleanFromSeqOpt(userDefinedProductMetadataFeatures.map(\_.hasTitle))

val hasDescription =

getOptBooleanFromSeqOpt(userDefinedProductMetadataFeatures.map(\_.hasDescription))

val hasVisitSiteCallToAction = getOptBooleanFromSeqOpt(

userDefinedProductMetadataFeatures.map(\_.hasVisitSiteCallToAction))

val hasAppInstallCallToAction = getOptBooleanFromSeqOpt(

userDefinedProductMetadataFeatures.map(\_.hasAppInstallCallToAction))

val hasWatchNowCallToAction =

getOptBooleanFromSeqOpt(userDefinedProductMetadataFeatures.map(\_.hasWatchNowCallToAction))

inputFeatures.copy(

videoDurationMs = playbackFeatures.durationMs,

bitRate = playbackFeatures.bitRate,

aspectRatioNum = playbackFeatures.aspectRatioNum,

aspectRatioDen = playbackFeatures.aspectRatioDen,

widths = Some(mediaWidths),

heights = Some(mediaHeights),

resizeMethods = Some(resizeMethods),

faceAreas = Some(faceMapAreas),

dominantColorRed = sortedColorPalette.headOption.map(\_.rgb.red),

dominantColorBlue = sortedColorPalette.headOption.map(\_.rgb.blue),

dominantColorGreen = sortedColorPalette.headOption.map(\_.rgb.green),

dominantColorPercentage = sortedColorPalette.headOption.map(\_.percentage),

numColors = Some(sortedColorPalette.size.toShort),

stickerIds = Some(stickerFeatures),

mediaOriginProviders = Some(mediaOriginProviders),

isManaged = Some(isManaged),

is360 = Some(is360),

viewCount = viewCount,

isMonetizable = isMonetizable,

isEmbeddable = isEmbeddable,

hasSelectedPreviewImage = hasSelectedPreviewImage,

hasTitle = hasTitle,

hasDescription = hasDescription,

hasVisitSiteCallToAction = hasVisitSiteCallToAction,

hasAppInstallCallToAction = hasAppInstallCallToAction,

hasWatchNowCallToAction = hasWatchNowCallToAction

)

}

.getOrElse(inputFeatures)

val featuresWithMediaTags = tweet.mediaTags

.map { mediaTags =>

val mediaTagScreenNames = getMediaTagScreenNames(mediaTags.tagMap)

val numMediaTags = mediaTagScreenNames.size

featuresWithMediaEntity.copy(

mediaTagScreenNames = Some(mediaTagScreenNames),

numMediaTags = Some(numMediaTags.toShort)

)

}

.getOrElse(featuresWithMediaEntity)

featuresWithMediaTags

.copy(media = tweet.media)

}

private def getSizeFeatures(mediaEntities: Seq[tp.MediaEntity]): Seq[MediaSizeFeatures] = {

mediaEntities.map { mediaEntity =>

mediaEntity.sizes.foldLeft(MediaSizeFeatures(0, 0, 0))((accDimensions, dimensions) =>

MediaSizeFeatures(

width = math.max(dimensions.width, accDimensions.width),

height = math.max(dimensions.height, accDimensions.height),

resizeMethod = math.max(dimensions.resizeMethod.getValue, accDimensions.resizeMethod)

))

}

}

private def getPlaybackFeatures(mediaEntities: Seq[tp.MediaEntity]): PlaybackFeatures = {

val allPlaybackFeatures = mediaEntities

.flatMap { mediaEntity =>

mediaEntity.mediaInfo map {

case videoEntity: tm.MediaInfo.VideoInfo =>

PlaybackFeatures(

durationMs = Some(videoEntity.videoInfo.durationMillis),

bitRate = videoEntity.videoInfo.variants.maxBy(\_.bitRate).bitRate,

aspectRatioNum = Some(videoEntity.videoInfo.aspectRatio.numerator),

aspectRatioDen = Some(videoEntity.videoInfo.aspectRatio.denominator)

)

case gifEntity: tm.MediaInfo.AnimatedGifInfo =>

PlaybackFeatures(

durationMs = None,

bitRate = gifEntity.animatedGifInfo.variants.maxBy(\_.bitRate).bitRate,

aspectRatioNum = Some(gifEntity.animatedGifInfo.aspectRatio.numerator),

aspectRatioDen = Some(gifEntity.animatedGifInfo.aspectRatio.denominator)

)

case \_ => PlaybackFeatures(None, None, None, None)

}

}

.collect {

case playbackFeatures: PlaybackFeatures => playbackFeatures

}

if (allPlaybackFeatures.nonEmpty) allPlaybackFeatures.maxBy(\_.durationMs)

else PlaybackFeatures(None, None, None, None)

}

private def getMediaTagScreenNames(tagMap: Map[Long, Seq[tp.MediaTag]]): Seq[String] =

tagMap.values

.flatMap(seqMediaTag => seqMediaTag.flatMap(\_.screenName))

.toSeq

// Areas of the faces identified in the media entities

private def getFaceMapAreas(mediaEntities: Seq[tp.MediaEntity]): Seq[Int] = {

for {

mediaEntity <- mediaEntities

metadata <- mediaEntity.additionalMetadata.toSeq

faceData <- metadata.faceData

faces <- faceData.faces

} yield {

faces

.getOrElse("orig", Seq.empty[mi.Face])

.flatMap(f => f.boundingBox.map(bb => bb.width \* bb.height))

}

}.flatten

// All ColorPalettes in the media sorted by the percentage in descending order

private def getSortedColorPalette(

mediaEntities: Seq[tp.MediaEntity]

): Seq[mi.ColorPaletteItem] = {

for {

mediaEntity <- mediaEntities

metadata <- mediaEntity.additionalMetadata.toSeq

colorInfo <- metadata.colorInfo

} yield {

colorInfo.palette

}

}.flatten.sortBy(-\_.percentage)

// Id's of stickers applied by the user

private def getStickerFeatures(mediaEntities: Seq[tp.MediaEntity]): Seq[Long] = {

for {

mediaEntity <- mediaEntities

metadata <- mediaEntity.additionalMetadata.toSeq

stickerInfo <- metadata.stickerInfo

} yield {

stickerInfo.stickers.map(\_.id)

}

}.flatten

// 3rd party media providers. eg. giphy for gifs

private def getMediaOriginProviders(mediaEntities: Seq[tp.MediaEntity]): Seq[String] =

for {

mediaEntity <- mediaEntities

metadata <- mediaEntity.additionalMetadata.toSeq

mediaOrigin <- metadata.foundMediaOrigin

} yield {

mediaOrigin.provider

}

private def getIsManaged(mediaEntities: Seq[tp.MediaEntity]): Boolean = {

for {

mediaEntity <- mediaEntities

metadata <- mediaEntity.additionalMetadata.toSeq

managementInfo <- metadata.managementInfo

} yield {

managementInfo.managed

}

}.contains(true)

private def getIs360(mediaEntities: Seq[tp.MediaEntity]): Boolean = {

for {

mediaEntity <- mediaEntities

metadata <- mediaEntity.additionalMetadata.toSeq

info360 <- metadata.info360

} yield {

info360.is360

}

}.contains(Some(true))

private def getViewCount(mediaEntities: Seq[tp.MediaEntity]): Option[Long] = {

for {

mediaEntity <- mediaEntities

metadata <- mediaEntity.additionalMetadata.toSeq

engagementInfo <- metadata.engagementInfo

viewCounts <- engagementInfo.viewCount

} yield {

viewCounts

}

}.reduceOption(\_ max \_)

// metadata defined by the user when uploading the image

private def getUserDefinedProductMetadataFeatures(

mediaEntities: Seq[tp.MediaEntity]

): Seq[UserDefinedProductMetadataFeatures] =

for {

mediaEntity <- mediaEntities

userDefinedMetadata <- mediaEntity.metadata

} yield {

UserDefinedProductMetadataFeatures(

isMonetizable = userDefinedMetadata.monetizable,

isEmbeddable = userDefinedMetadata.embeddable,

hasSelectedPreviewImage = Some(userDefinedMetadata.previewImage.nonEmpty),

hasTitle = userDefinedMetadata.title.map(\_.nonEmpty),

hasDescription = userDefinedMetadata.description.map(\_.nonEmpty),

hasVisitSiteCallToAction = userDefinedMetadata.callToActions.map(\_.visitSite.nonEmpty),

hasAppInstallCallToAction = userDefinedMetadata.callToActions.map(\_.appInstall.nonEmpty),

hasWatchNowCallToAction = userDefinedMetadata.callToActions.map(\_.watchNow.nonEmpty)

)

}

private def getOptBooleanFromSeqOpt(

seqOpt: Seq[Option[Boolean]]

): Option[Boolean] = Some(

seqOpt.exists(boolOpt => boolOpt.contains(true))

)

}

case class MediaSizeFeatures(width: Int, height: Int, resizeMethod: Int)

case class PlaybackFeatures(

durationMs: Option[Int],

bitRate: Option[Int],

aspectRatioNum: Option[Short],

aspectRatioDen: Option[Short])

case class UserDefinedProductMetadataFeatures(

isMonetizable: Option[Boolean],

isEmbeddable: Option[Boolean],

hasSelectedPreviewImage: Option[Boolean],

hasTitle: Option[Boolean],

hasDescription: Option[Boolean],

hasVisitSiteCallToAction: Option[Boolean],

hasAppInstallCallToAction: Option[Boolean],

hasWatchNowCallToAction: Option[Boolean])