package com.twitter.recosinjector.util

import com.twitter.frigate.common.base.TweetUtil

import com.twitter.gizmoduck.thriftscala.User

import com.twitter.recos.util.Action.Action

import com.twitter.tweetypie.thriftscala.Tweet

/\*\*

\* This is used to store information about a newly created tweet

\* @param validEntityUserIds For users mentioned or mediatagged in the tweet, these follow the

\* engage user and only they are are considered valid

\* @param sourceTweetDetails For Reply, Quote, or RT, source tweet is the tweet being actioned on

\*/

case class TweetCreateEventDetails(

userTweetEngagement: UserTweetEngagement,

validEntityUserIds: Seq[Long],

sourceTweetDetails: Option[TweetDetails]) {

// A mention is only valid if the mentioned user follows the source user

val validMentionUserIds: Option[Seq[Long]] = {

userTweetEngagement.tweetDetails.flatMap(\_.mentionUserIds.map(\_.intersect(validEntityUserIds)))

}

// A mediatag is only valid if the mediatagged user follows the source user

val validMediatagUserIds: Option[Seq[Long]] = {

userTweetEngagement.tweetDetails.flatMap(\_.mediatagUserIds.map(\_.intersect(validEntityUserIds)))

}

}

/\*\*

\* Stores information about a favorite/unfav engagement.

\* NOTE: This could either be Likes, or UNLIKEs (i.e. when user cancels the Like)

\* @param userTweetEngagement the engagement details

\*/

case class TweetFavoriteEventDetails(

userTweetEngagement: UserTweetEngagement)

/\*\*

\* Stores information about a unified user action engagement.

\* @param userTweetEngagement the engagement details

\*/

case class UuaEngagementEventDetails(

userTweetEngagement: UserTweetEngagement)

/\*\*

\* Details about a user-tweet engagement, like when a user tweeted/liked a tweet

\* @param engageUserId User that engaged with the tweet

\* @param action The action the user took on the tweet

\* @param tweetId The type of engagement the user took on the tweet

\*/

case class UserTweetEngagement(

engageUserId: Long,

engageUser: Option[User],

action: Action,

engagementTimeMillis: Option[Long],

tweetId: Long,

tweetDetails: Option[TweetDetails])

/\*\*

\* Helper class that decomposes a tweet object and provides related details about this tweet

\*/

case class TweetDetails(tweet: Tweet) {

val authorId: Option[Long] = tweet.coreData.map(\_.userId)

val urls: Option[Seq[String]] = tweet.urls.map(\_.map(\_.url))

val mediaUrls: Option[Seq[String]] = tweet.media.map(\_.map(\_.expandedUrl))

val hashtags: Option[Seq[String]] = tweet.hashtags.map(\_.map(\_.text))

// mentionUserIds include reply user ids at the beginning of a tweet

val mentionUserIds: Option[Seq[Long]] = tweet.mentions.map(\_.flatMap(\_.userId))

val mediatagUserIds: Option[Seq[Long]] = tweet.mediaTags.map {

\_.tagMap.flatMap {

case (\_, mediaTag) => mediaTag.flatMap(\_.userId)

}.toSeq

}

val replySourceId: Option[Long] = tweet.coreData.flatMap(\_.reply.flatMap(\_.inReplyToStatusId))

val replyUserId: Option[Long] = tweet.coreData.flatMap(\_.reply.map(\_.inReplyToUserId))

val retweetSourceId: Option[Long] = tweet.coreData.flatMap(\_.share.map(\_.sourceStatusId))

val retweetUserId: Option[Long] = tweet.coreData.flatMap(\_.share.map(\_.sourceUserId))

val quoteSourceId: Option[Long] = tweet.quotedTweet.map(\_.tweetId)

val quoteUserId: Option[Long] = tweet.quotedTweet.map(\_.userId)

val quoteTweetUrl: Option[String] = tweet.quotedTweet.flatMap(\_.permalink.map(\_.shortUrl))

//If the tweet is retweet/reply/quote, this is the tweet that the new tweet responds to

val (sourceTweetId, sourceTweetUserId) = {

(replySourceId, retweetSourceId, quoteSourceId) match {

case (Some(replyId), \_, \_) =>

(Some(replyId), replyUserId)

case (\_, Some(retweetId), \_) =>

(Some(retweetId), retweetUserId)

case (\_, \_, Some(quoteId)) =>

(Some(quoteId), quoteUserId)

case \_ =>

(None, None)

}

}

// Boolean information

val hasPhoto: Boolean = TweetUtil.containsPhotoTweet(tweet)

val hasVideo: Boolean = TweetUtil.containsVideoTweet(tweet)

// TweetyPie does not populate url fields in a quote tweet create event, even though we

// consider quote tweets as url tweets. This boolean helps make up for it.

// Details: https://groups.google.com/a/twitter.com/d/msg/eng/BhK1XAcSSWE/F8Gc4\_5uDwAJ

val hasQuoteTweetUrl: Boolean = tweet.quotedTweet.exists(\_.permalink.isDefined)

val hasUrl: Boolean = this.urls.exists(\_.nonEmpty) || hasQuoteTweetUrl

val hasHashtag: Boolean = this.hashtags.exists(\_.nonEmpty)

val isCard: Boolean = hasUrl | hasPhoto | hasVideo

implicit def bool2Long(b: Boolean): Long = if (b) 1L else 0L

// Return a hashed long that contains card type information of the tweet

val cardInfo: Long = isCard | (hasUrl << 1) | (hasPhoto << 2) | (hasVideo << 3)

// nullcast tweet is one that is purposefully not broadcast to followers, ex. an ad tweet.

val isNullCastTweet: Boolean = tweet.coreData.exists(\_.nullcast)

}