package com.twitter.timelineranker.model

import com.twitter.search.earlybird.thriftscala.ThriftSearchResult

import com.twitter.timelines.model.tweet.HydratedTweet

import com.twitter.timelines.model.TweetId

import com.twitter.timelines.model.UserId

import com.twitter.timelines.util.SnowflakeSortIndexHelper

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

object PartiallyHydratedTweet {

private val InvalidValue = "Invalid value"

/\*\*

\* Creates an instance of PartiallyHydratedTweet based on the given search result.

\*/

def fromSearchResult(result: ThriftSearchResult): PartiallyHydratedTweet = {

val tweetId = result.id

val metadata = result.metadata.getOrElse(

throw new IllegalArgumentException(

s"cannot initialize PartiallyHydratedTweet $tweetId without ThriftSearchResult metadata."

)

)

val extraMetadataOpt = metadata.extraMetadata

val userId = metadata.fromUserId

// The value of referencedTweetAuthorId and sharedStatusId is only considered valid if it is greater than 0.

val referencedTweetAuthorId =

if (metadata.referencedTweetAuthorId > 0) Some(metadata.referencedTweetAuthorId) else None

val sharedStatusId = if (metadata.sharedStatusId > 0) Some(metadata.sharedStatusId) else None

val isRetweet = metadata.isRetweet.getOrElse(false)

val retweetSourceTweetId = if (isRetweet) sharedStatusId else None

val retweetSourceUserId = if (isRetweet) referencedTweetAuthorId else None

// The fields sharedStatusId and referencedTweetAuthorId have overloaded meaning when

// this tweet is not a retweet (for retweet, there is only 1 meaning).

// When not a retweet,

// if referencedTweetAuthorId and sharedStatusId are both set, it is considered a reply

// if referencedTweetAuthorId is set and sharedStatusId is not set, it is a directed at tweet.

// References: SEARCH-8561 and SEARCH-13142

val inReplyToTweetId = if (!isRetweet) sharedStatusId else None

val inReplyToUserId = if (!isRetweet) referencedTweetAuthorId else None

val isReply = metadata.isReply.contains(true)

val quotedTweetId = extraMetadataOpt.flatMap(\_.quotedTweetId)

val quotedUserId = extraMetadataOpt.flatMap(\_.quotedUserId)

val isNullcast = metadata.isNullcast.contains(true)

val conversationId = extraMetadataOpt.flatMap(\_.conversationId)

// Root author id for the user who posts an exclusive tweet

val exclusiveConversationAuthorId = extraMetadataOpt.flatMap(\_.exclusiveConversationAuthorId)

// Card URI associated with an attached card to this tweet, if it contains one

val cardUri = extraMetadataOpt.flatMap(\_.cardUri)

val tweet = makeTweetyPieTweet(

tweetId,

userId,

inReplyToTweetId,

inReplyToUserId,

retweetSourceTweetId,

retweetSourceUserId,

quotedTweetId,

quotedUserId,

isNullcast,

isReply,

conversationId,

exclusiveConversationAuthorId,

cardUri

)

new PartiallyHydratedTweet(tweet)

}

def makeTweetyPieTweet(

tweetId: TweetId,

userId: UserId,

inReplyToTweetId: Option[TweetId],

inReplyToUserId: Option[TweetId],

retweetSourceTweetId: Option[TweetId],

retweetSourceUserId: Option[UserId],

quotedTweetId: Option[TweetId],

quotedUserId: Option[UserId],

isNullcast: Boolean,

isReply: Boolean,

conversationId: Option[Long],

exclusiveConversationAuthorId: Option[Long] = None,

cardUri: Option[String] = None

): tweetypie.Tweet = {

val isDirectedAt = inReplyToUserId.isDefined

val isRetweet = retweetSourceTweetId.isDefined && retweetSourceUserId.isDefined

val reply = if (isReply) {

Some(

tweetypie.Reply(

inReplyToStatusId = inReplyToTweetId,

inReplyToUserId = inReplyToUserId.getOrElse(0L) // Required

)

)

} else None

val directedAt = if (isDirectedAt) {

Some(

tweetypie.DirectedAtUser(

userId = inReplyToUserId.get,

screenName = "" // not available from search

)

)

} else None

val share = if (isRetweet) {

Some(

tweetypie.Share(

sourceStatusId = retweetSourceTweetId.get,

sourceUserId = retweetSourceUserId.get,

parentStatusId =

retweetSourceTweetId.get // Not always correct (eg, retweet of a retweet).

)

)

} else None

val quotedTweet =

for {

tweetId <- quotedTweetId

userId <- quotedUserId

} yield tweetypie.QuotedTweet(tweetId = tweetId, userId = userId)

val coreData = tweetypie.TweetCoreData(

userId = userId,

text = InvalidValue,

createdVia = InvalidValue,

createdAtSecs = SnowflakeSortIndexHelper.idToTimestamp(tweetId).inSeconds,

directedAtUser = directedAt,

reply = reply,

share = share,

nullcast = isNullcast,

conversationId = conversationId

)

// Hydrate exclusiveTweetControl which determines whether the user is able to view an exclusive / SuperFollow tweet.

val exclusiveTweetControl = exclusiveConversationAuthorId.map { authorId =>

tweetypie.ExclusiveTweetControl(conversationAuthorId = authorId)

}

val cardReference = cardUri.map { cardUriFromEB =>

tweetypie.CardReference(cardUri = cardUriFromEB)

}

tweetypie.Tweet(

id = tweetId,

quotedTweet = quotedTweet,

coreData = Some(coreData),

exclusiveTweetControl = exclusiveTweetControl,

cardReference = cardReference

)

}

}

/\*\*

\* Represents an instance of HydratedTweet that is hydrated using search result

\* (instead of being hydrated using TweetyPie service).

\*

\* Not all fields are available using search therefore such fields if accessed

\* throw UnsupportedOperationException to ensure that they are not inadvertently

\* accessed and relied upon.

\*/

class PartiallyHydratedTweet(tweet: tweetypie.Tweet) extends HydratedTweet(tweet) {

override def parentTweetId: Option[TweetId] = throw notSupported("parentTweetId")

override def mentionedUserIds: Seq[UserId] = throw notSupported("mentionedUserIds")

override def takedownCountryCodes: Set[String] = throw notSupported("takedownCountryCodes")

override def hasMedia: Boolean = throw notSupported("hasMedia")

override def isNarrowcast: Boolean = throw notSupported("isNarrowcast")

override def hasTakedown: Boolean = throw notSupported("hasTakedown")

override def isNsfw: Boolean = throw notSupported("isNsfw")

override def isNsfwUser: Boolean = throw notSupported("isNsfwUser")

override def isNsfwAdmin: Boolean = throw notSupported("isNsfwAdmin")

private def notSupported(name: String): UnsupportedOperationException = {

new UnsupportedOperationException(s"Not supported: $name")

}

}