package com.twitter.home\_mixer.product.scored\_tweets.marshaller

import com.twitter.home\_mixer.model.HomeFeatures.\_

import com.twitter.home\_mixer.product.scored\_tweets.model.ScoredTweetsResponse

import com.twitter.home\_mixer.{thriftscala => t}

import com.twitter.product\_mixer.core.feature.featuremap.FeatureMap

import com.twitter.product\_mixer.core.functional\_component.marshaller.TransportMarshaller

import com.twitter.product\_mixer.core.functional\_component.marshaller.response.urt.metadata.TopicContextFunctionalityTypeMarshaller

import com.twitter.product\_mixer.core.model.common.identifier.TransportMarshallerIdentifier

/\*\*

\* Marshall the domain model into our transport (Thrift) model.

\*/

object ScoredTweetsResponseTransportMarshaller

extends TransportMarshaller[ScoredTweetsResponse, t.ScoredTweetsResponse] {

override val identifier: TransportMarshallerIdentifier =

TransportMarshallerIdentifier("ScoredTweetsResponse")

override def apply(input: ScoredTweetsResponse): t.ScoredTweetsResponse = {

val scoredTweets = input.scoredTweets.map { tweet =>

mkScoredTweet(tweet.candidateIdLong, tweet.features)

}

t.ScoredTweetsResponse(scoredTweets)

}

private def mkScoredTweet(tweetId: Long, features: FeatureMap): t.ScoredTweet = {

val topicFunctionalityType = features

.getOrElse(TopicContextFunctionalityTypeFeature, None)

.map(TopicContextFunctionalityTypeMarshaller(\_))

t.ScoredTweet(

tweetId = tweetId,

authorId = features.get(AuthorIdFeature).get,

score = features.get(ScoreFeature),

suggestType = features.get(SuggestTypeFeature),

sourceTweetId = features.getOrElse(SourceTweetIdFeature, None),

sourceUserId = features.getOrElse(SourceUserIdFeature, None),

quotedTweetId = features.getOrElse(QuotedTweetIdFeature, None),

quotedUserId = features.getOrElse(QuotedUserIdFeature, None),

inReplyToTweetId = features.getOrElse(InReplyToTweetIdFeature, None),

inReplyToUserId = features.getOrElse(InReplyToUserIdFeature, None),

directedAtUserId = features.getOrElse(DirectedAtUserIdFeature, None),

inNetwork = Some(features.getOrElse(InNetworkFeature, true)),

sgsValidLikedByUserIds = Some(features.getOrElse(SGSValidLikedByUserIdsFeature, Seq.empty)),

sgsValidFollowedByUserIds =

Some(features.getOrElse(SGSValidFollowedByUserIdsFeature, Seq.empty)),

topicId = features.getOrElse(TopicIdSocialContextFeature, None),

topicFunctionalityType = topicFunctionalityType,

ancestors = Some(features.getOrElse(AncestorsFeature, Seq.empty)),

isReadFromCache = Some(features.getOrElse(IsReadFromCacheFeature, false)),

streamToKafka = Some(features.getOrElse(StreamToKafkaFeature, false)),

exclusiveConversationAuthorId =

features.getOrElse(ExclusiveConversationAuthorIdFeature, None),

authorMetadata = Some(

t.AuthorMetadata(

blueVerified = features.getOrElse(AuthorIsBlueVerifiedFeature, false),

goldVerified = features.getOrElse(AuthorIsGoldVerifiedFeature, false),

grayVerified = features.getOrElse(AuthorIsGrayVerifiedFeature, false),

legacyVerified = features.getOrElse(AuthorIsLegacyVerifiedFeature, false),

creator = features.getOrElse(AuthorIsCreatorFeature, false)

)),

lastScoredTimestampMs = None,

candidatePipelineIdentifier = None,

tweetUrls = None,

perspectiveFilteredLikedByUserIds =

Some(features.getOrElse(PerspectiveFilteredLikedByUserIdsFeature, Seq.empty)),

)

}

}