package com.twitter.recos.user\_tweet\_entity\_graph

import com.twitter.finagle.tracing.Trace

import com.twitter.logging.Logger

import com.twitter.recos.user\_tweet\_entity\_graph.thriftscala.\_

import com.twitter.util.Future

trait LoggingUserTweetEntityGraph extends thriftscala.UserTweetEntityGraph.MethodPerEndpoint {

private[this] val accessLog = Logger("access")

abstract override def recommendTweets(

request: RecommendTweetEntityRequest

): Future[RecommendTweetEntityResponse] = {

val time = System.currentTimeMillis

super.recommendTweets(request) onSuccess { resp =>

accessLog.info(

"%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\tRecommendTweetResponse size: %s\t%s in %d ms"

.format(

time,

Trace.id.toString(),

request.requesterId,

request.displayLocation,

request.recommendationTypes,

request.maxResultsByType,

request.excludedTweetIds.map(\_.take(5)),

request.excludedTweetIds.map(\_.size),

request.seedsWithWeights.take(5),

request.seedsWithWeights.size,

request.maxTweetAgeInMillis,

request.maxUserSocialProofSize,

request.maxTweetSocialProofSize,

request.minUserSocialProofSizes,

request.tweetTypes,

request.socialProofTypes,

request.socialProofTypeUnions,

resp.recommendations.size,

resp.recommendations.take(20).toList map {

case UserTweetEntityRecommendationUnion.TweetRec(tweetRec) =>

(tweetRec.tweetId, tweetRec.socialProofByType.map { case (k, v) => (k, v.size) })

case UserTweetEntityRecommendationUnion.HashtagRec(hashtagRec) =>

(hashtagRec.id, hashtagRec.socialProofByType.map { case (k, v) => (k, v.size) })

case UserTweetEntityRecommendationUnion.UrlRec(urlRec) =>

(urlRec.id, urlRec.socialProofByType.map { case (k, v) => (k, v.size) })

case \_ =>

throw new Exception("Unsupported recommendation types")

},

System.currentTimeMillis - time

)

)

} onFailure { exc =>

accessLog.error(

"%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s in %d ms".format(

time,

Trace.id.toString(),

request.requesterId,

request.displayLocation,

request.recommendationTypes,

request.maxResultsByType,

request.excludedTweetIds.map(\_.take(5)),

request.excludedTweetIds.map(\_.size),

request.seedsWithWeights.take(5),

request.seedsWithWeights.size,

request.maxTweetAgeInMillis,

request.maxUserSocialProofSize,

request.maxTweetSocialProofSize,

request.minUserSocialProofSizes,

request.tweetTypes,

request.socialProofTypes,

request.socialProofTypeUnions,

exc,

System.currentTimeMillis - time

)

)

}

}

abstract override def findTweetSocialProofs(

request: SocialProofRequest

): Future[SocialProofResponse] = {

val time = System.currentTimeMillis

super.findTweetSocialProofs(request) onSuccess { resp =>

accessLog.info(

"%s\t%s\t%d\tResponse: %s\tin %d ms".format(

Trace.id.toString,

request.requesterId,

request.seedsWithWeights.size,

resp.socialProofResults.toList,

System.currentTimeMillis - time

)

)

} onFailure { exc =>

accessLog.info(

"%s\t%s\t%d\tException: %s\tin %d ms".format(

Trace.id.toString,

request.requesterId,

request.seedsWithWeights.size,

exc,

System.currentTimeMillis - time

)

)

}

}

}