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

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.graphjet.algorithms.{

RecommendationInfo,

RecommendationType => JavaRecommendationType

}

import com.twitter.graphjet.algorithms.socialproof.{

NodeMetadataSocialProofResult => EntitySocialProofJavaResult,

SocialProofResult => SocialProofJavaResult

}

import com.twitter.recos.decider.UserTweetEntityGraphDecider

import com.twitter.recos.util.Stats

import com.twitter.recos.util.Stats.\_

import com.twitter.recos.recos\_common.thriftscala.{SocialProofType => SocialProofThriftType}

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

HashtagRecommendation,

TweetRecommendation,

UrlRecommendation,

UserTweetEntityRecommendationUnion,

RecommendationSocialProofRequest => SocialProofThriftRequest,

RecommendationSocialProofResponse => SocialProofThriftResponse,

RecommendationType => ThriftRecommendationType

}

import com.twitter.servo.request.RequestHandler

import com.twitter.util.{Future, Try}

import scala.collection.JavaConverters.\_

class SocialProofHandler(

tweetSocialProofRunner: TweetSocialProofRunner,

entitySocialProofRunner: EntitySocialProofRunner,

decider: UserTweetEntityGraphDecider,

statsReceiver: StatsReceiver)

extends RequestHandler[SocialProofThriftRequest, SocialProofThriftResponse] {

private val stats = statsReceiver.scope(this.getClass.getSimpleName)

private def getThriftSocialProof(

entitySocialProof: EntitySocialProofJavaResult

): Map[SocialProofThriftType, Map[Long, Seq[Long]]] = {

val socialProofAttempt = Try(entitySocialProof.getSocialProof)

.onFailure { e =>

stats.counter(e.getClass.getSimpleName).incr()

}

socialProofAttempt.toOption match {

case Some(socialProof) if socialProof.isEmpty =>

stats.counter(Stats.EmptyResult).incr()

Map.empty[SocialProofThriftType, Map[Long, Seq[Long]]]

case Some(socialProof) if !socialProof.isEmpty =>

socialProof.asScala.map {

case (socialProofType, socialProofUserToTweetsMap) =>

val userToTweetsSocialProof = socialProofUserToTweetsMap.asScala.map {

case (socialProofUser, connectingTweets) =>

(socialProofUser.toLong, connectingTweets.asScala.map(Long2long).toSeq)

}.toMap

(SocialProofThriftType(socialProofType.toInt), userToTweetsSocialProof)

}.toMap

case \_ =>

Map.empty[SocialProofThriftType, Map[Long, Seq[Long]]]

}

}

private def getThriftSocialProof(

tweetSocialProof: SocialProofJavaResult

): Map[SocialProofThriftType, Seq[Long]] = {

val socialProofAttempt = Try(tweetSocialProof.getSocialProof)

.onFailure { e =>

stats.counter(e.getClass.getSimpleName).incr()

}

socialProofAttempt.toOption match {

case Some(socialProof) if socialProof.isEmpty =>

stats.counter(Stats.EmptyResult).incr()

Map.empty[SocialProofThriftType, Seq[Long]]

case Some(socialProof) if !socialProof.isEmpty =>

socialProof.asScala.map {

case (socialProofType, connectingUsers) =>

(

SocialProofThriftType(socialProofType.toInt),

connectingUsers.asScala.map { Long2long }.toSeq)

}.toMap

case \_ =>

Map.empty[SocialProofThriftType, Seq[Long]]

}

}

private def getEntitySocialProof(

request: SocialProofThriftRequest

): Future[Seq[UserTweetEntityRecommendationUnion]] = {

val socialProofsFuture = entitySocialProofRunner(request)

socialProofsFuture.map { socialProofs: Seq[RecommendationInfo] =>

stats.counter(Stats.Served).incr(socialProofs.size)

socialProofs.flatMap { entitySocialProof: RecommendationInfo =>

val entitySocialProofJavaResult =

entitySocialProof.asInstanceOf[EntitySocialProofJavaResult]

if (entitySocialProofJavaResult.getRecommendationType == JavaRecommendationType.URL) {

Some(

UserTweetEntityRecommendationUnion.UrlRec(

UrlRecommendation(

entitySocialProofJavaResult.getNodeMetadataId,

entitySocialProofJavaResult.getWeight,

getThriftSocialProof(entitySocialProofJavaResult)

)

)

)

} else if (entitySocialProofJavaResult.getRecommendationType == JavaRecommendationType.HASHTAG) {

Some(

UserTweetEntityRecommendationUnion.HashtagRec(

HashtagRecommendation(

entitySocialProofJavaResult.getNodeMetadataId,

entitySocialProofJavaResult.getWeight,

getThriftSocialProof(entitySocialProofJavaResult)

)

)

)

} else {

None

}

}

}

}

private def getTweetSocialProof(

request: SocialProofThriftRequest

): Future[Seq[UserTweetEntityRecommendationUnion]] = {

val socialProofsFuture = tweetSocialProofRunner(request)

socialProofsFuture.map { socialProofs: Seq[RecommendationInfo] =>

stats.counter(Stats.Served).incr(socialProofs.size)

socialProofs.flatMap { tweetSocialProof: RecommendationInfo =>

val tweetSocialProofJavaResult = tweetSocialProof.asInstanceOf[SocialProofJavaResult]

Some(

UserTweetEntityRecommendationUnion.TweetRec(

TweetRecommendation(

tweetSocialProofJavaResult.getNode,

tweetSocialProofJavaResult.getWeight,

getThriftSocialProof(tweetSocialProofJavaResult)

)

)

)

}

}

}

def apply(request: SocialProofThriftRequest): Future[SocialProofThriftResponse] = {

trackFutureBlockStats(stats) {

val recommendationsWithSocialProofFut = Future

.collect {

request.recommendationIdsForSocialProof.keys.map {

case ThriftRecommendationType.Tweet if decider.tweetSocialProof =>

getTweetSocialProof(request)

case (ThriftRecommendationType.Url | ThriftRecommendationType.Hashtag)

if decider.entitySocialProof =>

getEntitySocialProof(request)

case \_ =>

Future.Nil

}.toSeq

}.map(\_.flatten)

recommendationsWithSocialProofFut.map { recommendationsWithSocialProof =>

SocialProofThriftResponse(recommendationsWithSocialProof)

}

}

}

}