package com.twitter.recosinjector.edges

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.recos.util.Action

import com.twitter.recosinjector.util.TweetFavoriteEventDetails

import com.twitter.util.Future

class TimelineEventToUserTweetEntityGraphBuilder(

userTweetEntityEdgeBuilder: UserTweetEntityEdgeBuilder

)(

override implicit val statsReceiver: StatsReceiver)

extends EventToMessageBuilder[TweetFavoriteEventDetails, UserTweetEntityEdge] {

private val numFavEdgeCounter = statsReceiver.counter("num\_favorite\_edge")

private val numUnfavEdgeCounter = statsReceiver.counter("num\_unfavorite\_edge")

override def shouldProcessEvent(event: TweetFavoriteEventDetails): Future[Boolean] = {

Future(true)

}

override def buildEdges(details: TweetFavoriteEventDetails): Future[Seq[UserTweetEntityEdge]] = {

val engagement = details.userTweetEngagement

val tweetDetails = engagement.tweetDetails

val entitiesMapFut = userTweetEntityEdgeBuilder.getEntitiesMapAndUpdateCache(

tweetId = engagement.tweetId,

tweetDetails = tweetDetails

)

entitiesMapFut

.map { entitiesMap =>

UserTweetEntityEdge(

sourceUser = engagement.engageUserId,

targetTweet = engagement.tweetId,

action = engagement.action,

metadata = engagement.engagementTimeMillis,

cardInfo = engagement.tweetDetails.map(\_.cardInfo.toByte),

entitiesMap = entitiesMap,

tweetDetails = tweetDetails

)

}

.map { edge =>

edge match {

case fav if fav.action == Action.Favorite =>

numFavEdgeCounter.incr()

case unfav if unfav.action == Action.Unfavorite =>

numUnfavEdgeCounter.incr()

case \_ =>

}

Seq(edge)

}

}

override def filterEdges(

event: TweetFavoriteEventDetails,

edges: Seq[UserTweetEntityEdge]

): Future[Seq[UserTweetEntityEdge]] = {

Future(edges)

}

}