package com.twitter.recosinjector.event\_processors

import com.twitter.finagle.mtls.authentication.ServiceIdentifier

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.recos.util.Action

import com.twitter.recosinjector.clients.Gizmoduck

import com.twitter.recosinjector.clients.Tweetypie

import com.twitter.recosinjector.decider.RecosInjectorDecider

import com.twitter.recosinjector.decider.RecosInjectorDeciderConstants

import com.twitter.recosinjector.edges.TimelineEventToUserTweetEntityGraphBuilder

import com.twitter.recosinjector.filters.TweetFilter

import com.twitter.recosinjector.filters.UserFilter

import com.twitter.recosinjector.publishers.KafkaEventPublisher

import com.twitter.recosinjector.util.TweetDetails

import com.twitter.recosinjector.util.TweetFavoriteEventDetails

import com.twitter.recosinjector.util.UserTweetEngagement

import com.twitter.scrooge.ThriftStructCodec

import com.twitter.timelineservice.thriftscala.FavoriteEvent

import com.twitter.timelineservice.thriftscala.UnfavoriteEvent

import com.twitter.timelineservice.thriftscala.{Event => TimelineEvent}

import com.twitter.util.Future

/\*\*

\* Processor for Timeline events, such as Favorite (liking) tweets

\*/

class TimelineEventProcessor(

override val eventBusStreamName: String,

override val thriftStruct: ThriftStructCodec[TimelineEvent],

override val serviceIdentifier: ServiceIdentifier,

kafkaEventPublisher: KafkaEventPublisher,

userTweetEntityGraphTopic: String,

userTweetEntityGraphMessageBuilder: TimelineEventToUserTweetEntityGraphBuilder,

decider: RecosInjectorDecider,

gizmoduck: Gizmoduck,

tweetypie: Tweetypie

)(

override implicit val statsReceiver: StatsReceiver)

extends EventBusProcessor[TimelineEvent] {

private val processEventDeciderCounter = statsReceiver.counter("num\_process\_timeline\_event")

private val numFavoriteEventCounter = statsReceiver.counter("num\_favorite\_event")

private val numUnFavoriteEventCounter = statsReceiver.counter("num\_unfavorite\_event")

private val numNotFavoriteEventCounter = statsReceiver.counter("num\_not\_favorite\_event")

private val numSelfFavoriteCounter = statsReceiver.counter("num\_self\_favorite\_event")

private val numNullCastTweetCounter = statsReceiver.counter("num\_null\_cast\_tweet")

private val numTweetFailSafetyLevelCounter = statsReceiver.counter("num\_fail\_tweetypie\_safety")

private val numFavoriteUserUnsafeCounter = statsReceiver.counter("num\_favorite\_user\_unsafe")

private val engageUserFilter = new UserFilter(gizmoduck)(statsReceiver.scope("engage\_user"))

private val tweetFilter = new TweetFilter(tweetypie)

private val numProcessFavorite = statsReceiver.counter("num\_process\_favorite")

private val numNoProcessFavorite = statsReceiver.counter("num\_no\_process\_favorite")

private def getFavoriteEventDetails(

favoriteEvent: FavoriteEvent

): TweetFavoriteEventDetails = {

val engagement = UserTweetEngagement(

engageUserId = favoriteEvent.userId,

engageUser = favoriteEvent.user,

action = Action.Favorite,

engagementTimeMillis = Some(favoriteEvent.eventTimeMs),

tweetId = favoriteEvent.tweetId, // the tweet, or source tweet if target tweet is a retweet

tweetDetails = favoriteEvent.tweet.map(TweetDetails) // tweet always exists

)

TweetFavoriteEventDetails(userTweetEngagement = engagement)

}

private def getUnfavoriteEventDetails(

unfavoriteEvent: UnfavoriteEvent

): TweetFavoriteEventDetails = {

val engagement = UserTweetEngagement(

engageUserId = unfavoriteEvent.userId,

engageUser = unfavoriteEvent.user,

action = Action.Unfavorite,

engagementTimeMillis = Some(unfavoriteEvent.eventTimeMs),

tweetId = unfavoriteEvent.tweetId, // the tweet, or source tweet if target tweet is a retweet

tweetDetails = unfavoriteEvent.tweet.map(TweetDetails) // tweet always exists

)

TweetFavoriteEventDetails(userTweetEngagement = engagement)

}

private def shouldProcessFavoriteEvent(event: TweetFavoriteEventDetails): Future[Boolean] = {

val engagement = event.userTweetEngagement

val engageUserId = engagement.engageUserId

val tweetId = engagement.tweetId

val authorIdOpt = engagement.tweetDetails.flatMap(\_.authorId)

val isSelfFavorite = authorIdOpt.contains(engageUserId)

val isNullCastTweet = engagement.tweetDetails.forall(\_.isNullCastTweet)

val isEngageUserSafeFut = engageUserFilter.filterByUserId(engageUserId)

val isTweetPassSafetyFut = tweetFilter.filterForTweetypieSafetyLevel(tweetId)

Future.join(isEngageUserSafeFut, isTweetPassSafetyFut).map {

case (isEngageUserSafe, isTweetPassSafety) =>

if (isSelfFavorite) numSelfFavoriteCounter.incr()

if (isNullCastTweet) numNullCastTweetCounter.incr()

if (!isEngageUserSafe) numFavoriteUserUnsafeCounter.incr()

if (!isTweetPassSafety) numTweetFailSafetyLevelCounter.incr()

!isSelfFavorite && !isNullCastTweet && isEngageUserSafe && isTweetPassSafety

}

}

private def processFavoriteEvent(favoriteEvent: FavoriteEvent): Future[Unit] = {

val eventDetails = getFavoriteEventDetails(favoriteEvent)

shouldProcessFavoriteEvent(eventDetails).map {

case true =>

numProcessFavorite.incr()

// Convert the event for UserTweetEntityGraph

userTweetEntityGraphMessageBuilder.processEvent(eventDetails).map { edges =>

edges.foreach { edge =>

kafkaEventPublisher.publish(edge.convertToRecosHoseMessage, userTweetEntityGraphTopic)

}

}

case false =>

numNoProcessFavorite.incr()

}

}

private def processUnFavoriteEvent(unFavoriteEvent: UnfavoriteEvent): Future[Unit] = {

if (decider.isAvailable(RecosInjectorDeciderConstants.EnableUnfavoriteEdge)) {

val eventDetails = getUnfavoriteEventDetails(unFavoriteEvent)

// Convert the event for UserTweetEntityGraph

userTweetEntityGraphMessageBuilder.processEvent(eventDetails).map { edges =>

edges.foreach { edge =>

kafkaEventPublisher.publish(edge.convertToRecosHoseMessage, userTweetEntityGraphTopic)

}

}

} else {

Future.Unit

}

}

override def processEvent(event: TimelineEvent): Future[Unit] = {

processEventDeciderCounter.incr()

event match {

case TimelineEvent.Favorite(favoriteEvent: FavoriteEvent) =>

numFavoriteEventCounter.incr()

processFavoriteEvent(favoriteEvent)

case TimelineEvent.Unfavorite(unFavoriteEvent: UnfavoriteEvent) =>

numUnFavoriteEventCounter.incr()

processUnFavoriteEvent(unFavoriteEvent)

case \_ =>

numNotFavoriteEventCounter.incr()

Future.Unit

}

}

}