package com.twitter.tweetypie

package store

import com.twitter.geoduck.backend.relevance.thriftscala.ReportFailure

import com.twitter.geoduck.backend.relevance.thriftscala.ReportResult

import com.twitter.geoduck.backend.relevance.thriftscala.ConversionReport

import com.twitter.geoduck.backend.searchrequestid.thriftscala.SearchRequestID

import com.twitter.geoduck.backend.tweetid.thriftscala.TweetID

import com.twitter.geoduck.common.thriftscala.GeoduckException

import com.twitter.geoduck.service.identifier.thriftscala.PlaceIdentifier

import com.twitter.servo.util.FutureArrow

import com.twitter.tweetypie.thriftscala.\_

trait GeoSearchRequestIDStore

extends TweetStoreBase[GeoSearchRequestIDStore]

with AsyncInsertTweet.Store {

def wrap(w: TweetStore.Wrap): GeoSearchRequestIDStore =

new TweetStoreWrapper[GeoSearchRequestIDStore](w, this)

with GeoSearchRequestIDStore

with AsyncInsertTweet.StoreWrapper

}

object GeoSearchRequestIDStore {

type ConversionReporter = FutureArrow[ConversionReport, ReportResult]

val Action: AsyncWriteAction.GeoSearchRequestId.type = AsyncWriteAction.GeoSearchRequestId

private val log = Logger(getClass)

object FailureHandler {

def translateException(failure: ReportResult.Failure): GeoduckException = {

failure.failure match {

case ReportFailure.Failure(exception) => exception

case \_ => GeoduckException("Unknown failure: " + failure.toString)

}

}

}

def apply(conversionReporter: ConversionReporter): GeoSearchRequestIDStore =

new GeoSearchRequestIDStore {

val conversionEffect: FutureEffect[ConversionReport] =

FutureEffect

.fromPartial[ReportResult] {

case unionFailure: ReportResult.Failure =>

Future.exception(FailureHandler.translateException(unionFailure))

}

.contramapFuture(conversionReporter)

override val asyncInsertTweet: FutureEffect[AsyncInsertTweet.Event] =

conversionEffect.contramapOption[AsyncInsertTweet.Event] { event =>

for {

isUserProtected <- event.user.safety.map(\_.isProtected)

geoSearchRequestID <- event.geoSearchRequestId

placeType <- event.tweet.place.map(\_.`type`)

placeId <- event.tweet.coreData.flatMap(\_.placeId)

placeIdLong <- Try(java.lang.Long.parseUnsignedLong(placeId, 16)).toOption

if placeType == PlaceType.Poi && isUserProtected == false

} yield {

ConversionReport(

requestID = SearchRequestID(requestID = geoSearchRequestID),

tweetID = TweetID(event.tweet.id),

placeID = PlaceIdentifier(placeIdLong)

)

}

}

override val retryAsyncInsertTweet: FutureEffect[

TweetStoreRetryEvent[AsyncInsertTweet.Event]

] =

TweetStore.retry(Action, asyncInsertTweet)

}

}