package com.twitter.usersignalservice.handler

import com.twitter.storehaus.ReadableStore

import com.twitter.usersignalservice.thriftscala.BatchSignalRequest

import com.twitter.usersignalservice.thriftscala.BatchSignalResponse

import com.twitter.util.Future

import com.twitter.conversions.DurationOps.\_

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.util.StatsUtil

import com.twitter.usersignalservice.config.SignalFetcherConfig

import com.twitter.usersignalservice.base.Query

import com.twitter.usersignalservice.thriftscala.ClientIdentifier

import com.twitter.usersignalservice.thriftscala.SignalType

import com.twitter.util.Duration

import com.twitter.util.Timer

import com.twitter.util.TimeoutException

class UserSignalHandler(

signalFetcherConfig: SignalFetcherConfig,

timer: Timer,

stats: StatsReceiver) {

import UserSignalHandler.\_

val statsReceiver: StatsReceiver = stats.scope("user-signal-service/service")

def getBatchSignalsResponse(request: BatchSignalRequest): Future[Option[BatchSignalResponse]] = {

StatsUtil.trackOptionStats(statsReceiver) {

val allSignals = request.signalRequest.map { signalRequest =>

signalFetcherConfig

.SignalFetcherMapper(signalRequest.signalType)

.get(Query(

userId = request.userId,

signalType = signalRequest.signalType,

maxResults = signalRequest.maxResults.map(\_.toInt),

clientId = request.clientId.getOrElse(ClientIdentifier.Unknown)

))

.map(signalOpt => (signalRequest.signalType, signalOpt))

}

Future.collect(allSignals).map { signalsSeq =>

val signalsMap = signalsSeq.map {

case (signalType: SignalType, signalsOpt) =>

(signalType, signalsOpt.getOrElse(EmptySeq))

}.toMap

Some(BatchSignalResponse(signalsMap))

}

}

}

def toReadableStore: ReadableStore[BatchSignalRequest, BatchSignalResponse] = {

new ReadableStore[BatchSignalRequest, BatchSignalResponse] {

override def get(request: BatchSignalRequest): Future[Option[BatchSignalResponse]] = {

getBatchSignalsResponse(request).raiseWithin(UserSignalServiceTimeout)(timer).rescue {

case \_: TimeoutException =>

statsReceiver.counter("endpointGetBatchSignals/failure/timeout").incr()

EmptyResponse

case e =>

statsReceiver.counter("endpointGetBatchSignals/failure/" + e.getClass.getName).incr()

EmptyResponse

}

}

}

}

}

object UserSignalHandler {

val UserSignalServiceTimeout: Duration = 25.milliseconds

val EmptySeq: Seq[Nothing] = Seq.empty

val EmptyResponse: Future[Option[BatchSignalResponse]] = Future.value(Some(BatchSignalResponse()))

}