package com.twitter.usersignalservice.base

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.base.Stats

import com.twitter.storehaus.ReadableStore

import com.twitter.twistly.common.UserId

import com.twitter.usersignalservice.base.BaseSignalFetcher.Timeout

import com.twitter.usersignalservice.thriftscala.Signal

import com.twitter.usersignalservice.thriftscala.SignalType

import com.twitter.util.Future

import com.twitter.util.Timer

case class AggregatedSignalController(

signalsAggregationInfo: Seq[SignalAggregatedInfo],

signalsWeightMapInfo: Map[SignalType, Double],

stats: StatsReceiver,

timer: Timer)

extends ReadableStore[Query, Seq[Signal]] {

val name: String = this.getClass.getCanonicalName

val statsReceiver: StatsReceiver = stats.scope(name)

override def get(query: Query): Future[Option[Seq[Signal]]] = {

Stats

.trackItems(statsReceiver) {

val allSignalsFut =

Future

.collect(signalsAggregationInfo.map(\_.getSignals(query.userId))).map(\_.flatten.flatten)

val aggregatedSignals =

allSignalsFut.map { allSignals =>

allSignals

.groupBy(\_.targetInternalId).collect {

case (Some(internalId), signals) =>

val mostRecentEnagementTime = signals.map(\_.timestamp).max

val totalWeight =

signals

.map(signal => signalsWeightMapInfo.getOrElse(signal.signalType, 0.0)).sum

(Signal(query.signalType, mostRecentEnagementTime, Some(internalId)), totalWeight)

}.toSeq.sortBy { case (signal, weight) => (-weight, -signal.timestamp) }

.map(\_.\_1)

.take(query.maxResults.getOrElse(Int.MaxValue))

}

aggregatedSignals.map(Some(\_))

}.raiseWithin(Timeout)(timer).handle {

case e =>

statsReceiver.counter(e.getClass.getCanonicalName).incr()

Some(Seq.empty[Signal])

}

}

}

case class SignalAggregatedInfo(

signalType: SignalType,

signalFetcher: ReadableStore[Query, Seq[Signal]]) {

def getSignals(userId: UserId): Future[Option[Seq[Signal]]] = {

signalFetcher.get(Query(userId, signalType, None))

}

}