package com.twitter.frigate.pushservice.util

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.util.TimeUtil

import com.twitter.frigate.pushservice.model.PushTypes.Target

import com.twitter.frigate.pushservice.params.PushConstants

import com.twitter.frigate.pushservice.params.{PushFeatureSwitchParams => FSParams}

import com.twitter.util.Future

import com.twitter.util.Time

import java.util.Calendar

import java.util.TimeZone

case class MinDurationModifierCalculator() {

private def mapCountryCodeToTimeZone(

countryCode: String,

stats: StatsReceiver

): Option[Calendar] = {

PushConstants.countryCodeToTimeZoneMap

.get(countryCode.toUpperCase).map(timezone =>

Calendar.getInstance(TimeZone.getTimeZone(timezone)))

}

private def transformToHour(

dayOfHour: Int

): Int = {

if (dayOfHour < 0) dayOfHour + 24

else dayOfHour

}

private def getMinDurationByHourOfDay(

hourOfDay: Int,

startTimeList: Seq[Int],

endTimeList: Seq[Int],

minDurationTimeModifierConst: Seq[Int],

stats: StatsReceiver

): Option[Int] = {

val scopedStats = stats.scope("getMinDurationByHourOfDay")

scopedStats.counter("request").incr()

val durationOpt = (startTimeList, endTimeList, minDurationTimeModifierConst).zipped.toList

.filter {

case (startTime, endTime, \_) =>

if (startTime <= endTime) hourOfDay >= startTime && hourOfDay < endTime

else (hourOfDay >= startTime) || hourOfDay < endTime

case \_ => false

}.map {

case (\_, \_, modifier) => modifier

}.headOption

durationOpt match {

case Some(duration) => scopedStats.counter(s"$duration.minutes").incr()

case \_ => scopedStats.counter("none").incr()

}

durationOpt

}

def getMinDurationModifier(

target: Target,

calendar: Calendar,

stats: StatsReceiver

): Option[Int] = {

val startTimeList = target.params(FSParams.MinDurationModifierStartHourList)

val endTimeList = target.params(FSParams.MinDurationModifierEndHourList)

val minDurationTimeModifierConst = target.params(FSParams.MinDurationTimeModifierConst)

if (startTimeList.length != endTimeList.length || minDurationTimeModifierConst.length != startTimeList.length) {

None

} else {

val hourOfDay = calendar.get(Calendar.HOUR\_OF\_DAY)

getMinDurationByHourOfDay(

hourOfDay,

startTimeList,

endTimeList,

minDurationTimeModifierConst,

stats)

}

}

def getMinDurationModifier(

target: Target,

countryCodeOpt: Option[String],

stats: StatsReceiver

): Option[Int] = {

val scopedStats = stats

.scope("getMinDurationModifier")

scopedStats.counter("total\_requests").incr()

countryCodeOpt match {

case Some(countryCode) =>

scopedStats

.counter("country\_code\_exists").incr()

val calendarOpt = mapCountryCodeToTimeZone(countryCode, scopedStats)

calendarOpt.flatMap(calendar => getMinDurationModifier(target, calendar, scopedStats))

case \_ => None

}

}

def getMinDurationModifier(target: Target, stats: StatsReceiver): Future[Option[Int]] = {

val scopedStats = stats

.scope("getMinDurationModifier")

scopedStats.counter("total\_requests").incr()

val startTimeList = target.params(FSParams.MinDurationModifierStartHourList)

val endTimeList = target.params(FSParams.MinDurationModifierEndHourList)

val minDurationTimeModifierConst = target.params(FSParams.MinDurationTimeModifierConst)

if (startTimeList.length != endTimeList.length || minDurationTimeModifierConst.length != startTimeList.length) {

Future.value(None)

} else {

target.localTimeInHHMM.map {

case (hourOfDay, \_) =>

getMinDurationByHourOfDay(

hourOfDay,

startTimeList,

endTimeList,

minDurationTimeModifierConst,

scopedStats)

case \_ => None

}

}

}

def getMinDurationModifierByUserOpenedHistory(

target: Target,

openedPushByHourAggregatedOpt: Option[Map[Int, Int]],

stats: StatsReceiver

): Option[Int] = {

val scopedStats = stats

.scope("getMinDurationModifierByUserOpenedHistory")

scopedStats.counter("total\_requests").incr()

openedPushByHourAggregatedOpt match {

case Some(openedPushByHourAggregated) =>

if (openedPushByHourAggregated.isEmpty) {

scopedStats.counter("openedPushByHourAggregated\_empty").incr()

None

} else {

val currentUTCHour = TimeUtil.hourOfDay(Time.now)

val utcHourWithMaxOpened = if (target.params(FSParams.EnableRandomHourForQuickSend)) {

(target.targetId % 24).toInt

} else {

openedPushByHourAggregated.maxBy(\_.\_2).\_1

}

val numOfMaxOpened = openedPushByHourAggregated.maxBy(\_.\_2).\_2

if (numOfMaxOpened >= target.params(FSParams.SendTimeByUserHistoryMaxOpenedThreshold)) {

scopedStats.counter("pass\_experiment\_bucket\_threshold").incr()

if (numOfMaxOpened >= target

.params(FSParams.SendTimeByUserHistoryMaxOpenedThreshold)) { // only update if number of opened pushes meet threshold

scopedStats.counter("pass\_max\_threshold").incr()

val quickSendBeforeHours =

target.params(FSParams.SendTimeByUserHistoryQuickSendBeforeHours)

val quickSendAfterHours =

target.params(FSParams.SendTimeByUserHistoryQuickSendAfterHours)

val hoursToLessSend = target.params(FSParams.SendTimeByUserHistoryNoSendsHours)

val quickSendTimeMinDurationInMinute =

target.params(FSParams.SendTimeByUserHistoryQuickSendMinDurationInMinute)

val noSendTimeMinDuration =

target.params(FSParams.SendTimeByUserHistoryNoSendMinDuration)

val startTimeForNoSend = transformToHour(

utcHourWithMaxOpened - quickSendBeforeHours - hoursToLessSend)

val startTimeForQuickSend = transformToHour(

utcHourWithMaxOpened - quickSendBeforeHours)

val endTimeForNoSend =

transformToHour(utcHourWithMaxOpened - quickSendBeforeHours)

val endTimeForQuickSend =

transformToHour(utcHourWithMaxOpened + quickSendAfterHours) + 1

val startTimeList = Seq(startTimeForNoSend, startTimeForQuickSend)

val endTimeList = Seq(endTimeForNoSend, endTimeForQuickSend)

val minDurationTimeModifierConst =

Seq(noSendTimeMinDuration, quickSendTimeMinDurationInMinute)

getMinDurationByHourOfDay(

currentUTCHour,

startTimeList,

endTimeList,

minDurationTimeModifierConst,

scopedStats)

} else None

} else None

}

case \_ =>

None

}

}

}