package com.twitter.cr\_mixer

package featureswitch

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.abdecider.LoggingABDecider

import com.twitter.abdecider.Recipient

import com.twitter.abdecider.Bucket

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

import com.twitter.util.Local

import scala.collection.concurrent.{Map => ConcurrentMap}

/\*\*

\* Wraps a LoggingABDecider, so all impressed buckets are recorded to a 'LocalContext' on a given request.

\*

\* Contexts (https://twitter.github.io/finagle/guide/Contexts.html) are Finagle's mechanism for

\* storing state/variables without having to pass these variables all around the request.

\*

\* In order for this class to be used the [[SetImpressedBucketsLocalContextFilter]] must be applied

\* at the beginning of the request, to initialize a concurrent map used to store impressed buckets.

\*

\* Whenever we get an a/b impression, the bucket information is logged to the concurrent hashmap.

\*/

case class CrMixerLoggingABDecider(

loggingAbDecider: LoggingABDecider,

statsReceiver: StatsReceiver)

extends LoggingABDecider {

private val scopedStatsReceiver = statsReceiver.scope("cr\_logging\_ab\_decider")

override def impression(

experimentName: String,

recipient: Recipient

): Option[Bucket] = {

StatsUtil.trackNonFutureBlockStats(scopedStatsReceiver.scope("log\_impression")) {

val maybeBuckets = loggingAbDecider.impression(experimentName, recipient)

maybeBuckets.foreach { b =>

scopedStatsReceiver.counter("impressions").incr()

CrMixerImpressedBuckets.recordImpressedBucket(b)

}

maybeBuckets

}

}

override def track(

experimentName: String,

eventName: String,

recipient: Recipient

): Unit = {

loggingAbDecider.track(experimentName, eventName, recipient)

}

override def bucket(

experimentName: String,

recipient: Recipient

): Option[Bucket] = {

loggingAbDecider.bucket(experimentName, recipient)

}

override def experiments: Seq[String] = loggingAbDecider.experiments

override def experiment(experimentName: String) =

loggingAbDecider.experiment(experimentName)

}

object CrMixerImpressedBuckets {

private[featureswitch] val localImpressedBucketsMap = new Local[ConcurrentMap[Bucket, Boolean]]

/\*\*

\* Gets all impressed buckets for this request.

\*\*/

def getAllImpressedBuckets: Option[List[Bucket]] = {

localImpressedBucketsMap.apply().map(\_.map { case (k, \_) => k }.toList)

}

private[featureswitch] def recordImpressedBucket(bucket: Bucket) = {

localImpressedBucketsMap().foreach { m => m += bucket -> true }

}

}