package com.twitter.home\_mixer.functional\_component.feature\_hydrator

import com.twitter.conversions.DurationOps.\_

import com.twitter.home\_mixer.model.HomeFeatures.TweetImpressionsFeature

import com.twitter.home\_mixer.model.request.HasSeenTweetIds

import com.twitter.home\_mixer.service.HomeMixerAlertConfig

import com.twitter.product\_mixer.core.feature.Feature

import com.twitter.product\_mixer.core.feature.featuremap.FeatureMap

import com.twitter.product\_mixer.core.feature.featuremap.FeatureMapBuilder

import com.twitter.product\_mixer.core.functional\_component.feature\_hydrator.QueryFeatureHydrator

import com.twitter.product\_mixer.core.model.common.identifier.FeatureHydratorIdentifier

import com.twitter.product\_mixer.core.pipeline.PipelineQuery

import com.twitter.stitch.Stitch

import com.twitter.timelines.impression.{thriftscala => t}

import com.twitter.timelines.impressionstore.store.ManhattanTweetImpressionStoreClient

import com.twitter.util.Duration

import com.twitter.util.Time

import javax.inject.Inject

import javax.inject.Singleton

@Singleton

case class TweetImpressionsQueryFeatureHydrator[

Query <: PipelineQuery with HasSeenTweetIds] @Inject() (

manhattanTweetImpressionStoreClient: ManhattanTweetImpressionStoreClient)

extends QueryFeatureHydrator[Query] {

private val TweetImpressionTTL = 2.days

private val TweetImpressionCap = 5000

override val identifier: FeatureHydratorIdentifier = FeatureHydratorIdentifier("TweetImpressions")

override val features: Set[Feature[\_, \_]] = Set(TweetImpressionsFeature)

override def hydrate(query: Query): Stitch[FeatureMap] = {

manhattanTweetImpressionStoreClient.get(query.getRequiredUserId).map { entriesOpt =>

val entries = entriesOpt.map(\_.entries).toSeq.flatten

val updatedImpressions =

if (query.seenTweetIds.forall(\_.isEmpty)) entries

else updateTweetImpressions(entries, query.seenTweetIds.get)

FeatureMapBuilder().add(TweetImpressionsFeature, updatedImpressions).build()

}

}

override val alerts = Seq(

HomeMixerAlertConfig.BusinessHours.defaultSuccessRateAlert(99.8)

)

/\*\*

\* 1) Check timestamps and remove expired tweets based on [[TweetImpressionTTL]]

\* 2) Filter duplicates between current tweets and those in the impression store (remove older ones)

\* 3) Prepend new (Timestamp, Seq[TweetIds]) to the tweets from the impression store

\* 4) Truncate older tweets if sum of all tweets across timestamps >= [[TweetImpressionCap]],

\*/

private[feature\_hydrator] def updateTweetImpressions(

tweetImpressionsFromStore: Seq[t.TweetImpressionsEntry],

seenIdsFromClient: Seq[Long],

currentTime: Long = Time.now.inMilliseconds,

tweetImpressionTTL: Duration = TweetImpressionTTL,

tweetImpressionCap: Int = TweetImpressionCap,

): Seq[t.TweetImpressionsEntry] = {

val seenIdsFromClientSet = seenIdsFromClient.toSet

val dedupedTweetImpressionsFromStore: Seq[t.TweetImpressionsEntry] = tweetImpressionsFromStore

.collect {

case t.TweetImpressionsEntry(ts, tweetIds)

if Time.fromMilliseconds(ts).untilNow < tweetImpressionTTL =>

t.TweetImpressionsEntry(ts, tweetIds.filterNot(seenIdsFromClientSet.contains))

}.filter { \_.tweetIds.nonEmpty }

val mergedTweetImpressionsEntries =

t.TweetImpressionsEntry(currentTime, seenIdsFromClient) +: dedupedTweetImpressionsFromStore

val initialTweetImpressionsWithCap = (Seq.empty[t.TweetImpressionsEntry], tweetImpressionCap)

val (truncatedTweetImpressionsEntries: Seq[t.TweetImpressionsEntry], \_) =

mergedTweetImpressionsEntries

.foldLeft(initialTweetImpressionsWithCap) {

case (

(tweetImpressions: Seq[t.TweetImpressionsEntry], remainingCap),

t.TweetImpressionsEntry(ts, tweetIds)) if remainingCap > 0 =>

(

t.TweetImpressionsEntry(ts, tweetIds.take(remainingCap)) +: tweetImpressions,

remainingCap - tweetIds.size)

case (tweetImpressionsWithCap, \_) => tweetImpressionsWithCap

}

truncatedTweetImpressionsEntries.reverse

}

}