package com.twitter.tweetypie.storage

import com.twitter.conversions.DurationOps.\_

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.stitch.Stitch

import com.twitter.stitch.StitchSeqGroup

import com.twitter.tweetypie.storage.TweetStorageClient.GetStoredTweet

import com.twitter.tweetypie.storage.TweetStorageClient.GetStoredTweet.Error

import com.twitter.tweetypie.storage.TweetStorageClient.GetStoredTweet.Response.\_

import com.twitter.tweetypie.storage.TweetUtils.\_

import com.twitter.tweetypie.thriftscala.Tweet

import com.twitter.util.Time

import com.twitter.util.Try

import scala.collection.mutable

object GetStoredTweetHandler {

private[this] object DeletedState {

def unapply(stateRecord: Option[TweetStateRecord]): Option[TweetStateRecord] =

stateRecord match {

case state @ (Some(\_: TweetStateRecord.SoftDeleted) | Some(

\_: TweetStateRecord.HardDeleted) | Some(\_: TweetStateRecord.BounceDeleted)) =>

state

case \_ => None

}

}

private[this] def deletedAtMs(stateRecord: Option[TweetStateRecord]): Option[Long] =

stateRecord match {

case Some(d: TweetStateRecord.SoftDeleted) => Some(d.createdAt)

case Some(d: TweetStateRecord.BounceDeleted) => Some(d.createdAt)

case Some(d: TweetStateRecord.HardDeleted) => Some(d.deletedAt)

case \_ => None

}

private[this] def tweetResponseFromRecords(

tweetId: TweetId,

mhRecords: Seq[TweetManhattanRecord],

statsReceiver: StatsReceiver,

): GetStoredTweet.Response = {

val errs =

mutable.Buffer[Error]()

val hasStoredTweetFields: Boolean = mhRecords.exists {

case TweetManhattanRecord(TweetKey(\_, \_: TweetKey.LKey.FieldKey), \_) => true

case \_ => false

}

val storedTweet = if (hasStoredTweetFields) {

Try(buildStoredTweet(tweetId, mhRecords, includeScrubbed = true))

.onFailure(\_ => errs.append(Error.TweetIsCorrupt))

.toOption

} else {

None

}

val scrubbedFields: Set[FieldId] = extractScrubbedFields(mhRecords)

val tweet: Option[Tweet] = storedTweet.map(StorageConversions.fromStoredTweetAllowInvalid)

val stateRecords: Seq[TweetStateRecord] = TweetStateRecord.fromTweetMhRecords(mhRecords)

val tweetState: Option[TweetStateRecord] = TweetStateRecord.mostRecent(mhRecords)

storedTweet.foreach { storedTweet =>

val storedExpectedFields = storedTweet.getFieldBlobs(expectedFields)

val missingExpectedFields = expectedFields.filterNot(storedExpectedFields.contains)

if (missingExpectedFields.nonEmpty || !isValid(storedTweet)) {

errs.append(Error.TweetFieldsMissingOrInvalid)

}

val invalidScrubbedFields = storedTweet.getFieldBlobs(scrubbedFields).keys

if (invalidScrubbedFields.nonEmpty) {

errs.append(Error.ScrubbedFieldsPresent)

}

if (deletedAtMs(tweetState).exists(\_ < Time.now.inMilliseconds - 14.days.inMilliseconds)) {

errs.append(Error.TweetShouldBeHardDeleted)

}

}

val err = Option(errs.toList).filter(\_.nonEmpty)

(tweet, tweetState, err) match {

case (None, None, None) =>

statsReceiver.counter("not\_found").incr()

NotFound(tweetId)

case (None, Some(tweetState: TweetStateRecord.HardDeleted), None) =>

statsReceiver.counter("hard\_deleted").incr()

HardDeleted(tweetId, Some(tweetState), stateRecords, scrubbedFields)

case (None, \_, Some(errs)) =>

statsReceiver.counter("failed").incr()

Failed(tweetId, tweetState, stateRecords, scrubbedFields, errs)

case (Some(tweet), \_, Some(errs)) =>

statsReceiver.counter("found\_invalid").incr()

FoundWithErrors(tweet, tweetState, stateRecords, scrubbedFields, errs)

case (Some(tweet), DeletedState(state), None) =>

statsReceiver.counter("deleted").incr()

FoundDeleted(tweet, Some(state), stateRecords, scrubbedFields)

case (Some(tweet), \_, None) =>

statsReceiver.counter("found").incr()

Found(tweet, tweetState, stateRecords, scrubbedFields)

}

}

def apply(read: ManhattanOperations.Read, statsReceiver: StatsReceiver): GetStoredTweet = {

object mhGroup extends StitchSeqGroup[TweetId, Seq[TweetManhattanRecord]] {

override def run(tweetIds: Seq[TweetId]): Stitch[Seq[Seq[TweetManhattanRecord]]] = {

Stats.addWidthStat("getStoredTweet", "tweetIds", tweetIds.size, statsReceiver)

Stitch.traverse(tweetIds)(read(\_))

}

}

tweetId =>

if (tweetId <= 0) {

Stitch.NotFound

} else {

Stitch

.call(tweetId, mhGroup)

.map(mhRecords =>

tweetResponseFromRecords(tweetId, mhRecords, statsReceiver.scope("getStoredTweet")))

}

}

}