package com.twitter.tweetypie

package handler

import com.twitter.stitch.Stitch

import com.twitter.tweetypie.core.InternalServerError

import com.twitter.tweetypie.core.OverCapacity

import com.twitter.tweetypie.storage.Response.TweetResponseCode

import com.twitter.tweetypie.storage.TweetStorageClient.GetTweet

import com.twitter.tweetypie.storage.DeleteState

import com.twitter.tweetypie.storage.DeletedTweetResponse

import com.twitter.tweetypie.storage.RateLimited

import com.twitter.tweetypie.storage.TweetStorageClient

import com.twitter.tweetypie.thriftscala.\_

/\*\*

\* Allow access to raw, unhydrated deleted tweet fields from storage backends (currently Manhattan)

\*/

object GetDeletedTweetsHandler {

type Type = FutureArrow[GetDeletedTweetsRequest, Seq[GetDeletedTweetResult]]

type TweetsExist = Seq[TweetId] => Stitch[Set[TweetId]]

def processTweetResponse(response: Try[GetTweet.Response]): Stitch[Option[Tweet]] = {

import GetTweet.Response.\_

response match {

case Return(Found(tweet)) => Stitch.value(Some(tweet))

case Return(Deleted | NotFound | BounceDeleted(\_)) => Stitch.None

case Throw(\_: RateLimited) => Stitch.exception(OverCapacity("manhattan"))

case Throw(exception) => Stitch.exception(exception)

}

}

def convertDeletedTweetResponse(

r: DeletedTweetResponse,

extantIds: Set[TweetId]

): GetDeletedTweetResult = {

val id = r.tweetId

if (extantIds.contains(id) || r.deleteState == DeleteState.NotDeleted) {

GetDeletedTweetResult(id, DeletedTweetState.NotDeleted)

} else {

r.overallResponse match {

case TweetResponseCode.Success =>

GetDeletedTweetResult(id, convertState(r.deleteState), r.tweet)

case TweetResponseCode.OverCapacity => throw OverCapacity("manhattan")

case \_ =>

throw InternalServerError(

s"Unhandled response ${r.overallResponse} from getDeletedTweets for tweet $id"

)

}

}

}

def convertState(d: DeleteState): DeletedTweetState = d match {

case DeleteState.NotFound => DeletedTweetState.NotFound

case DeleteState.NotDeleted => DeletedTweetState.NotDeleted

case DeleteState.SoftDeleted => DeletedTweetState.SoftDeleted

// Callers of this endpoint treat BounceDeleted tweets the same as SoftDeleted

case DeleteState.BounceDeleted => DeletedTweetState.SoftDeleted

case DeleteState.HardDeleted => DeletedTweetState.HardDeleted

}

/\*\*

\* Converts [[TweetStorageClient.GetTweet]] into a FutureArrow that returns extant tweet ids from

\* the original list. This method is used to check underlying storage againt cache, preferring

\* cache if a tweet exists there.

\*/

def tweetsExist(getTweet: TweetStorageClient.GetTweet): TweetsExist =

(tweetIds: Seq[TweetId]) =>

for {

response <- Stitch.traverse(tweetIds) { tweetId => getTweet(tweetId).liftToTry }

tweets <- Stitch.collect(response.map(processTweetResponse))

} yield tweets.flatten.map(\_.id).toSet.filter(tweetIds.contains)

def apply(

getDeletedTweets: TweetStorageClient.GetDeletedTweets,

tweetsExist: TweetsExist,

stats: StatsReceiver

): Type = {

val notFound = stats.counter("not\_found")

val notDeleted = stats.counter("not\_deleted")

val softDeleted = stats.counter("soft\_deleted")

val hardDeleted = stats.counter("hard\_deleted")

val unknown = stats.counter("unknown")

def trackState(results: Seq[GetDeletedTweetResult]): Unit =

results.foreach { r =>

r.state match {

case DeletedTweetState.NotFound => notFound.incr()

case DeletedTweetState.NotDeleted => notDeleted.incr()

case DeletedTweetState.SoftDeleted => softDeleted.incr()

case DeletedTweetState.HardDeleted => hardDeleted.incr()

case \_ => unknown.incr()

}

}

FutureArrow { request =>

Stitch.run {

Stitch

.join(

getDeletedTweets(request.tweetIds),

tweetsExist(request.tweetIds)

)

.map {

case (deletedTweetResponses, extantIds) =>

val responseIds = deletedTweetResponses.map(\_.tweetId)

assert(

responseIds == request.tweetIds,

s"getDeletedTweets response does not match order of request: Request ids " +

s"(${request.tweetIds.mkString(", ")}) != response ids (${responseIds

.mkString(", ")})"

)

deletedTweetResponses.map { r => convertDeletedTweetResponse(r, extantIds) }

}

}

}

}

}