package com.twitter.tweetypie

package handler

import com.twitter.servo.util.FutureArrow

import com.twitter.stitch.Stitch

import com.twitter.tweetypie.core.FilteredState

import com.twitter.tweetypie.core.TweetHydrationError

import com.twitter.tweetypie.repository.ParentUserIdRepository

import com.twitter.tweetypie.storage.TweetStorageClient.Undelete

import com.twitter.tweetypie.storage.DeleteState

import com.twitter.tweetypie.storage.DeletedTweetResponse

import com.twitter.tweetypie.storage.TweetStorageClient

import com.twitter.tweetypie.store.UndeleteTweet

import com.twitter.tweetypie.thriftscala.UndeleteTweetState.{Success => TweetypieSuccess, \_}

import com.twitter.tweetypie.thriftscala.\_

import com.twitter.tweetypie.thriftscala.entities.EntityExtractor

import scala.util.control.NoStackTrace

trait UndeleteException extends Exception with NoStackTrace

/\*\*

\* Exceptions we return to the user, things that we don't expect to ever happen unless there is a

\* problem with the underlying data in Manhattan or a bug in [[com.twitter.tweetypie.storage.TweetStorageClient]]

\*/

object NoDeletedAtTimeException extends UndeleteException

object NoCreatedAtTimeException extends UndeleteException

object NoStatusWithSuccessException extends UndeleteException

object NoUserIdWithTweetException extends UndeleteException

object NoDeletedTweetException extends UndeleteException

object SoftDeleteUserIdNotFoundException extends UndeleteException

/\*\*

\* represents a problem that we choose to return to the user as a response state

\* rather than as an exception.

\*/

case class ResponseException(state: UndeleteTweetState) extends Exception with NoStackTrace {

def toResponse: UndeleteTweetResponse = UndeleteTweetResponse(state = state)

}

private[this] object SoftDeleteExpiredException extends ResponseException(SoftDeleteExpired)

private[this] object BounceDeleteException extends ResponseException(TweetIsBounceDeleted)

private[this] object SourceTweetNotFoundException extends ResponseException(SourceTweetNotFound)

private[this] object SourceUserNotFoundException extends ResponseException(SourceUserNotFound)

private[this] object TweetExistsException extends ResponseException(TweetAlreadyExists)

private[this] object TweetNotFoundException extends ResponseException(TweetNotFound)

private[this] object U13TweetException extends ResponseException(TweetIsU13Tweet)

private[this] object UserNotFoundException extends ResponseException(UserNotFound)

/\*\*

\* Undelete Notes:

\*

\* If request.force is set to true, then the undelete will take place even if the undeleted tweet

\* is already present in Manhattan. This is useful if a tweet was recently restored to the backend,

\* but the async actions portion of the undelete failed and you want to retry them.

\*

\* Before undeleting the tweet we check if it's a retweet, in which case we require that the sourceTweet

\* and sourceUser exist.

\*

\* Tweets can only be undeleted for N days where N is the number of days before tweets marked with

\* the soft\_delete\_state flag are deleted permanently by the cleanup job

\*

\*/

object UndeleteTweetHandler {

type Type = FutureArrow[UndeleteTweetRequest, UndeleteTweetResponse]

/\*\* Extract an optional value inside a future or throw if it's missing. \*/

def required[T](option: Future[Option[T]], ex: => Exception): Future[T] =

option.flatMap {

case None => Future.exception(ex)

case Some(i) => Future.value(i)

}

def apply(

undelete: TweetStorageClient.Undelete,

tweetExists: FutureArrow[TweetId, Boolean],

getUser: FutureArrow[UserId, Option[User]],

getDeletedTweets: TweetStorageClient.GetDeletedTweets,

parentUserIdRepo: ParentUserIdRepository.Type,

save: FutureArrow[UndeleteTweet.Event, Tweet]

): Type = {

def getParentUserId(tweet: Tweet): Future[Option[UserId]] =

Stitch.run {

parentUserIdRepo(tweet)

.handle {

case ParentUserIdRepository.ParentTweetNotFound(id) => None

}

}

val entityExtractor = EntityExtractor.mutationAll.endo

val getDeletedTweet: Long => Future[DeletedTweetResponse] =

id => Stitch.run(getDeletedTweets(Seq(id)).map(\_.head))

def getRequiredUser(userId: Option[UserId]): Future[User] =

userId match {

case None => Future.exception(SoftDeleteUserIdNotFoundException)

case Some(id) => required(getUser(id), UserNotFoundException)

}

def getValidatedDeletedTweet(

tweetId: TweetId,

allowNotDeleted: Boolean

): Future[DeletedTweet] = {

import DeleteState.\_

val deletedTweet = getDeletedTweet(tweetId).map { response =>

response.deleteState match {

case SoftDeleted => response.tweet

// BounceDeleted tweets violated Twitter Rules and may not be undeleted

case BounceDeleted => throw BounceDeleteException

case HardDeleted => throw SoftDeleteExpiredException

case NotDeleted => if (allowNotDeleted) response.tweet else throw TweetExistsException

case NotFound => throw TweetNotFoundException

}

}

required(deletedTweet, NoDeletedTweetException)

}

/\*\*

\* Fetch the source tweet's user for a deleted share

\*/

def getSourceUser(share: Option[DeletedTweetShare]): Future[Option[User]] =

share match {

case None => Future.value(None)

case Some(s) => required(getUser(s.sourceUserId), SourceUserNotFoundException).map(Some(\_))

}

/\*\*

\* Ensure that the undelete response contains all the required information to continue with

\* the tweetypie undelete.

\*/

def validateUndeleteResponse(response: Undelete.Response, force: Boolean): Future[Tweet] =

Future {

(response.code, response.tweet) match {

case (Undelete.UndeleteResponseCode.NotCreated, \_) => throw TweetNotFoundException

case (Undelete.UndeleteResponseCode.BackupNotFound, \_) => throw SoftDeleteExpiredException

case (Undelete.UndeleteResponseCode.Success, None) => throw NoStatusWithSuccessException

case (Undelete.UndeleteResponseCode.Success, Some(tweet)) =>

// archivedAtMillis is required on the response unless force is present

// or the tweet is a retweet. retweets have no favs or retweets to clean up

// of their own so the original deleted at time is not needed

if (response.archivedAtMillis.isEmpty && !force && !isRetweet(tweet))

throw NoDeletedAtTimeException

else

tweet

case (code, \_) => throw new Exception(s"Unknown UndeleteResponseCode $code")

}

}

def enforceU13Compliance(user: User, deletedTweet: DeletedTweet): Future[Unit] =

Future.when(U13ValidationUtil.wasTweetCreatedBeforeUserTurned13(user, deletedTweet)) {

throw U13TweetException

}

/\*\*

\* Fetch required data and perform before/after validations for undelete.

\* If everything looks good with the undelete, kick off the tweetypie undelete

\* event.

\*/

FutureArrow { request =>

val hydrationOptions = request.hydrationOptions.getOrElse(WritePathHydrationOptions())

val force = request.force.getOrElse(false)

val tweetId = request.tweetId

(for {

// we must be able to query the tweet from the soft delete table

deletedTweet <- getValidatedDeletedTweet(tweetId, allowNotDeleted = force)

// we always require the user

user <- getRequiredUser(deletedTweet.userId)

// Make sure we're not restoring any u13 tweets.

() <- enforceU13Compliance(user, deletedTweet)

// if a retweet, then sourceUser is required; sourceTweet will be hydrated in save()

sourceUser <- getSourceUser(deletedTweet.share)

// validations passed, perform the undelete.

undeleteResponse <- Stitch.run(undelete(tweetId))

// validate the response

tweet <- validateUndeleteResponse(undeleteResponse, force)

// Extract entities from tweet text

tweetWithEntities = entityExtractor(tweet)

// If a retweet, get user id of parent retweet

parentUserId <- getParentUserId(tweet)

// undeletion was successful, hydrate the tweet and

// kick off tweetypie async undelete actions

hydratedTweet <- save(

UndeleteTweet.Event(

tweet = tweetWithEntities,

user = user,

timestamp = Time.now,

hydrateOptions = hydrationOptions,

deletedAt = undeleteResponse.archivedAtMillis.map(Time.fromMilliseconds),

sourceUser = sourceUser,

parentUserId = parentUserId

)

)

} yield {

UndeleteTweetResponse(TweetypieSuccess, Some(hydratedTweet))

}).handle {

case TweetHydrationError(\_, Some(FilteredState.Unavailable.SourceTweetNotFound(\_))) =>

SourceTweetNotFoundException.toResponse

case ex: ResponseException =>

ex.toResponse

}

}

}

}