package com.twitter.tweetypie.storage

import com.twitter.stitch.Stitch

import com.twitter.tweetypie.storage.Response.TweetResponse

import com.twitter.tweetypie.thriftscala.Tweet

import com.twitter.util.Future

/\*\*

\* Interface for reading and writing tweet data in Manhattan

\*/

trait TweetStorageClient {

import TweetStorageClient.\_

def addTweet: AddTweet

def deleteAdditionalFields: DeleteAdditionalFields

def getTweet: GetTweet

def getStoredTweet: GetStoredTweet

def getDeletedTweets: GetDeletedTweets

def undelete: Undelete

def updateTweet: UpdateTweet

def scrub: Scrub

def softDelete: SoftDelete

def bounceDelete: BounceDelete

def hardDeleteTweet: HardDeleteTweet

def ping: Ping

}

object TweetStorageClient {

type GetTweet = TweetId => Stitch[GetTweet.Response]

object GetTweet {

sealed trait Response

object Response {

case class Found(tweet: Tweet) extends Response

object NotFound extends Response

object Deleted extends Response

// On BounceDeleted, provide the full Tweet so that implementations

// (i.e. ManhattanTweetStorageClient) don't not need to be aware of the specific tweet

// fields required by callers for proper processing of bounced deleted tweets.

case class BounceDeleted(tweet: Tweet) extends Response

}

}

type GetStoredTweet = TweetId => Stitch[GetStoredTweet.Response]

object GetStoredTweet {

sealed abstract class Error(val message: String) {

override def toString: String = message

}

object Error {

case object TweetIsCorrupt extends Error("stored tweet data is corrupt and cannot be decoded")

case object ScrubbedFieldsPresent

extends Error("stored tweet fields that should be scrubbed are still present")

case object TweetFieldsMissingOrInvalid

extends Error("expected tweet fields are missing or contain invalid values")

case object TweetShouldBeHardDeleted

extends Error("stored tweet that should be hard deleted is still present")

}

sealed trait Response

object Response {

sealed trait StoredTweetMetadata {

def state: Option[TweetStateRecord]

def allStates: Seq[TweetStateRecord]

def scrubbedFields: Set[FieldId]

}

sealed trait StoredTweetErrors {

def errs: Seq[Error]

}

/\*\*

\* Tweet data was found, possibly state records and/or scrubbed field records.

\*/

sealed trait FoundAny extends Response with StoredTweetMetadata {

def tweet: Tweet

}

object FoundAny {

def unapply(

response: Response

): Option[

(Tweet, Option[TweetStateRecord], Seq[TweetStateRecord], Set[FieldId], Seq[Error])

] =

response match {

case f: FoundWithErrors =>

Some((f.tweet, f.state, f.allStates, f.scrubbedFields, f.errs))

case f: FoundAny => Some((f.tweet, f.state, f.allStates, f.scrubbedFields, Seq.empty))

case \_ => None

}

}

/\*\*

\* No records for this tweet id were found in storage

\*/

case class NotFound(id: TweetId) extends Response

/\*\*

\* Data related to the Tweet id was found but could not be loaded successfully. The

\* errs array contains details of the problems.

\*/

case class Failed(

id: TweetId,

state: Option[TweetStateRecord],

allStates: Seq[TweetStateRecord],

scrubbedFields: Set[FieldId],

errs: Seq[Error],

) extends Response

with StoredTweetMetadata

with StoredTweetErrors

/\*\*

\* No Tweet data was found, and the most recent state record found is HardDeleted

\*/

case class HardDeleted(

id: TweetId,

state: Option[TweetStateRecord.HardDeleted],

allStates: Seq[TweetStateRecord],

scrubbedFields: Set[FieldId],

) extends Response

with StoredTweetMetadata

/\*\*

\* Tweet data was found, and the most recent state record found, if any, is not

\* any form of deletion record.

\*/

case class Found(

tweet: Tweet,

state: Option[TweetStateRecord],

allStates: Seq[TweetStateRecord],

scrubbedFields: Set[FieldId],

) extends FoundAny

/\*\*

\* Tweet data was found, and the most recent state record found indicates deletion.

\*/

case class FoundDeleted(

tweet: Tweet,

state: Option[TweetStateRecord],

allStates: Seq[TweetStateRecord],

scrubbedFields: Set[FieldId],

) extends FoundAny

/\*\*

\* Tweet data was found, however errors were detected in the stored data. Required

\* fields may be missing from the Tweet struct (e.g. CoreData), stored fields that

\* should be scrubbed remain present, or Tweets that should be hard-deleted remain

\* in storage. The errs array contains details of the problems.

\*/

case class FoundWithErrors(

tweet: Tweet,

state: Option[TweetStateRecord],

allStates: Seq[TweetStateRecord],

scrubbedFields: Set[FieldId],

errs: Seq[Error],

) extends FoundAny

with StoredTweetErrors

}

}

type HardDeleteTweet = TweetId => Stitch[HardDeleteTweet.Response]

type SoftDelete = TweetId => Stitch[Unit]

type BounceDelete = TweetId => Stitch[Unit]

object HardDeleteTweet {

sealed trait Response

object Response {

case class Deleted(deletedAtMillis: Option[Long], createdAtMillis: Option[Long])

extends Response

case class NotDeleted(id: TweetId, ineligibleLKey: Option[TweetKey.LKey])

extends Throwable

with Response

}

}

type Undelete = TweetId => Stitch[Undelete.Response]

object Undelete {

case class Response(

code: UndeleteResponseCode,

tweet: Option[Tweet] = None,

createdAtMillis: Option[Long] = None,

archivedAtMillis: Option[Long] = None)

sealed trait UndeleteResponseCode

object UndeleteResponseCode {

object Success extends UndeleteResponseCode

object BackupNotFound extends UndeleteResponseCode

object NotCreated extends UndeleteResponseCode

}

}

type AddTweet = Tweet => Stitch[Unit]

type UpdateTweet = (Tweet, Seq[Field]) => Stitch[TweetResponse]

type GetDeletedTweets = Seq[TweetId] => Stitch[Seq[DeletedTweetResponse]]

type DeleteAdditionalFields = (Seq[TweetId], Seq[Field]) => Stitch[Seq[TweetResponse]]

type Scrub = (Seq[TweetId], Seq[Field]) => Stitch[Unit]

type Ping = () => Future[Unit]

}