package com.twitter.tweetypie

package store

import com.twitter.mediaservices.commons.thriftscala.MediaKey

import com.twitter.servo.util.FutureArrow

import com.twitter.tweetypie.media.\_

import com.twitter.tweetypie.thriftscala.\_

trait MediaServiceStore

extends TweetStoreBase[MediaServiceStore]

with AsyncDeleteTweet.Store

with AsyncUndeleteTweet.Store {

def wrap(w: TweetStore.Wrap): MediaServiceStore =

new TweetStoreWrapper(w, this)

with MediaServiceStore

with AsyncDeleteTweet.StoreWrapper

with AsyncUndeleteTweet.StoreWrapper

}

object MediaServiceStore {

val Action: AsyncWriteAction.MediaDeletion.type = AsyncWriteAction.MediaDeletion

private def ownMedia(t: Tweet): Seq[(MediaKey, TweetId)] =

getMedia(t)

.collect {

case m if Media.isOwnMedia(t.id, m) => (MediaKeyUtil.get(m), t.id)

}

def apply(

deleteMedia: FutureArrow[DeleteMediaRequest, Unit],

undeleteMedia: FutureArrow[UndeleteMediaRequest, Unit]

): MediaServiceStore =

new MediaServiceStore {

override val asyncDeleteTweet: FutureEffect[AsyncDeleteTweet.Event] =

FutureEffect[AsyncDeleteTweet.Event] { e =>

Future.when(!isRetweet(e.tweet)) {

val ownMediaKeys: Seq[(MediaKey, TweetId)] = ownMedia(e.tweet)

val deleteMediaRequests = ownMediaKeys.map(DeleteMediaRequest.tupled)

Future.collect(deleteMediaRequests.map(deleteMedia))

}

}

override val retryAsyncDeleteTweet: FutureEffect[

TweetStoreRetryEvent[AsyncDeleteTweet.Event]

] =

TweetStore.retry(Action, asyncDeleteTweet)

override val asyncUndeleteTweet: FutureEffect[AsyncUndeleteTweet.Event] =

FutureEffect[AsyncUndeleteTweet.Event] { e =>

Future.when(!isRetweet(e.tweet)) {

val ownMediaKeys: Seq[(MediaKey, TweetId)] = ownMedia(e.tweet)

val unDeleteMediaRequests = ownMediaKeys.map(UndeleteMediaRequest.tupled)

Future.collect(unDeleteMediaRequests.map(undeleteMedia))

}

}

override val retryAsyncUndeleteTweet: FutureEffect[

TweetStoreRetryEvent[AsyncUndeleteTweet.Event]

] =

TweetStore.retry(Action, asyncUndeleteTweet)

}

}