package com.twitter.tweetypie

package store

import com.twitter.guano.thriftscala.NsfwTweetActionAction

import com.twitter.tseng.withholding.thriftscala.TakedownReason

import com.twitter.tweetypie.thriftscala.\_

trait GuanoServiceStore

extends TweetStoreBase[GuanoServiceStore]

with AsyncDeleteTweet.Store

with AsyncTakedown.Store

with AsyncUpdatePossiblySensitiveTweet.Store {

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

new TweetStoreWrapper(w, this)

with GuanoServiceStore

with AsyncDeleteTweet.StoreWrapper

with AsyncTakedown.StoreWrapper

with AsyncUpdatePossiblySensitiveTweet.StoreWrapper

}

object GuanoServiceStore {

val Action: AsyncWriteAction.GuanoScribe.type = AsyncWriteAction.GuanoScribe

val toGuanoTakedown: (AsyncTakedown.Event, TakedownReason, Boolean) => Guano.Takedown =

(event: AsyncTakedown.Event, reason: TakedownReason, takendown: Boolean) =>

Guano.Takedown(

tweetId = event.tweet.id,

userId = getUserId(event.tweet),

reason = reason,

takendown = takendown,

note = event.auditNote,

host = event.host,

byUserId = event.byUserId

)

val toGuanoUpdatePossiblySensitiveTweet: (

AsyncUpdatePossiblySensitiveTweet.Event,

Boolean,

NsfwTweetActionAction

) => Guano.UpdatePossiblySensitiveTweet =

(

event: AsyncUpdatePossiblySensitiveTweet.Event,

updatedValue: Boolean,

action: NsfwTweetActionAction

) =>

Guano.UpdatePossiblySensitiveTweet(

tweetId = event.tweet.id,

host = event.host.orElse(Some("unknown")),

userId = event.user.id,

byUserId = event.byUserId,

action = action,

enabled = updatedValue,

note = event.note

)

def apply(guano: Guano, stats: StatsReceiver): GuanoServiceStore = {

val deleteByUserIdCounter = stats.counter("deletes\_with\_by\_user\_id")

val deleteScribeCounter = stats.counter("deletes\_resulting\_in\_scribe")

new GuanoServiceStore {

override val asyncDeleteTweet: FutureEffect[AsyncDeleteTweet.Event] =

FutureEffect[AsyncDeleteTweet.Event] { event =>

val tweet = event.tweet

event.byUserId.foreach(\_ => deleteByUserIdCounter.incr())

// Guano the tweet deletion action not initiated from the RetweetsDeletionStore

event.byUserId match {

case Some(byUserId) =>

deleteScribeCounter.incr()

guano.scribeDestroyTweet(

Guano.DestroyTweet(

tweet = tweet,

userId = getUserId(tweet),

byUserId = byUserId,

passthrough = event.auditPassthrough

)

)

case \_ =>

Future.Unit

}

}.onlyIf(\_.cascadedFromTweetId.isEmpty)

override val retryAsyncDeleteTweet: FutureEffect[

TweetStoreRetryEvent[AsyncDeleteTweet.Event]

] =

TweetStore.retry(Action, asyncDeleteTweet)

override val asyncTakedown: FutureEffect[AsyncTakedown.Event] =

FutureEffect[AsyncTakedown.Event] { event =>

val messages =

event.reasonsToAdd.map(toGuanoTakedown(event, \_, true)) ++

event.reasonsToRemove.map(toGuanoTakedown(event, \_, false))

Future.join(messages.map(guano.scribeTakedown))

}.onlyIf(\_.scribeForAudit)

override val retryAsyncTakedown: FutureEffect[TweetStoreRetryEvent[AsyncTakedown.Event]] =

TweetStore.retry(Action, asyncTakedown)

override val asyncUpdatePossiblySensitiveTweet: FutureEffect[

AsyncUpdatePossiblySensitiveTweet.Event

] =

FutureEffect[AsyncUpdatePossiblySensitiveTweet.Event] { event =>

val messages =

event.nsfwAdminChange.map(

toGuanoUpdatePossiblySensitiveTweet(event, \_, NsfwTweetActionAction.NsfwAdmin)

) ++

event.nsfwUserChange.map(

toGuanoUpdatePossiblySensitiveTweet(event, \_, NsfwTweetActionAction.NsfwUser)

)

Future.join(messages.toSeq.map(guano.scribeUpdatePossiblySensitiveTweet))

}

override val retryAsyncUpdatePossiblySensitiveTweet: FutureEffect[

TweetStoreRetryEvent[AsyncUpdatePossiblySensitiveTweet.Event]

] =

TweetStore.retry(Action, asyncUpdatePossiblySensitiveTweet)

}

}

}