package com.twitter.tweetypie

package handler

import com.twitter.servo.util.FutureArrow

import com.twitter.tweetypie.store.Takedown

import com.twitter.tweetypie.thriftscala.DataError

import com.twitter.tweetypie.thriftscala.DataErrorCause

import com.twitter.tweetypie.thriftscala.SetTweetUserTakedownRequest

trait UserTakedownHandler {

val setTweetUserTakedownRequest: FutureArrow[SetTweetUserTakedownRequest, Unit]

}

/\*\*

\* This handler processes SetTweetUserTakedownRequest objects sent to Tweetypie's

\* setTweetUserTakedown endpoint. These requests originate from tweetypie daemon and the

\* request object specifies the user ID of the user who is being modified, and a boolean value

\* to indicate whether takedown is being added or removed.

\*

\* If takedown is being added, the hasTakedown bit is set on all of the user's tweets.

\* If takedown is being removed, we can't automatically unset the hasTakedown bit on all tweets

\* since some of the tweets might have tweet-specific takedowns, in which case the hasTakedown bit

\* needs to remain set. Instead, we flush the user's tweets from cache, and let the repairer

\* unset the bit when hydrating tweets where the bit is set but no user or tweet

\* takedown country codes are present.

\*/

object UserTakedownHandler {

type Type = FutureArrow[SetTweetUserTakedownRequest, Unit]

def takedownEvent(userHasTakedown: Boolean): Tweet => Option[Takedown.Event] =

tweet => {

val tweetHasTakedown =

TweetLenses.tweetypieOnlyTakedownCountryCodes(tweet).exists(\_.nonEmpty) ||

TweetLenses.tweetypieOnlyTakedownReasons(tweet).exists(\_.nonEmpty)

val updatedHasTakedown = userHasTakedown || tweetHasTakedown

if (updatedHasTakedown == TweetLenses.hasTakedown(tweet))

None

else

Some(

Takedown.Event(

tweet = TweetLenses.hasTakedown.set(tweet, updatedHasTakedown),

timestamp = Time.now,

eventbusEnqueue = false,

scribeForAudit = false,

updateCodesAndReasons = false

)

)

}

def setHasTakedown(

tweetTakedown: FutureEffect[Takedown.Event],

userHasTakedown: Boolean

): FutureEffect[Seq[Tweet]] =

tweetTakedown.contramapOption(takedownEvent(userHasTakedown)).liftSeq

def verifyTweetUserId(expectedUserId: Option[UserId], tweet: Tweet): Unit = {

val tweetUserId: UserId = getUserId(tweet)

val tweetId: Long = tweet.id

expectedUserId.filter(\_ != tweetUserId).foreach { u =>

throw DataError(

message =

s"SetTweetUserTakedownRequest userId $u does not match userId $tweetUserId for Tweet: $tweetId",

errorCause = Some(DataErrorCause.UserTweetRelationship),

)

}

}

def apply(

getTweet: FutureArrow[TweetId, Option[Tweet]],

tweetTakedown: FutureEffect[Takedown.Event],

): Type =

FutureArrow { request =>

for {

tweet <- getTweet(request.tweetId)

\_ = tweet.foreach(t => verifyTweetUserId(request.userId, t))

\_ <- setHasTakedown(tweetTakedown, request.hasTakedown)(tweet.toSeq)

} yield ()

}

}