package com.twitter.tweetypie

package handler

import com.twitter.snowflake.id.SnowflakeId

import com.twitter.tweetutil.DmDeepLink

import com.twitter.tweetutil.TweetPermalink

import com.twitter.tweetypie.core.CardReferenceUriExtractor

import com.twitter.tweetypie.core.NonTombstone

import com.twitter.tweetypie.core.TweetCreateFailure

import com.twitter.tweetypie.repository.TweetQuery

import com.twitter.tweetypie.repository.TweetRepository

import com.twitter.tweetypie.thriftscala.CardReference

import com.twitter.tweetypie.thriftscala.DeviceSource

import com.twitter.tweetypie.thriftscala.QuotedTweet

import com.twitter.tweetypie.thriftscala.ShortenedUrl

import com.twitter.tweetypie.thriftscala.Tweet

import com.twitter.tweetypie.thriftscala.TweetCreateState

case class AttachmentBuilderRequest(

tweetId: TweetId,

user: User,

mediaUploadIds: Option[Seq[Long]],

cardReference: Option[CardReference],

attachmentUrl: Option[String],

remoteHost: Option[String],

darkTraffic: Boolean,

deviceSource: DeviceSource) {

val ctx: ValidationContext = ValidationContext(

user = user,

mediaUploadIds = mediaUploadIds,

cardReference = cardReference

)

val passThroughResponse: AttachmentBuilderResult =

AttachmentBuilderResult(attachmentUrl = attachmentUrl, validationContext = ctx)

}

case class ValidationContext(

user: User,

mediaUploadIds: Option[Seq[Long]],

cardReference: Option[CardReference])

case class AttachmentBuilderResult(

attachmentUrl: Option[String] = None,

quotedTweet: Option[QuotedTweet] = None,

extraChars: Int = 0,

validationContext: ValidationContext)

object AttachmentBuilder {

private[this] val log = Logger(getClass)

private[this] val attachmentCountLogger = Logger(

"com.twitter.tweetypie.handler.CreateAttachmentCount"

)

type Type = FutureArrow[AttachmentBuilderRequest, AttachmentBuilderResult]

type ValidationType = FutureEffect[AttachmentBuilderResult]

def validateAttachmentUrl(attachmentUrl: Option[String]): Unit.type =

attachmentUrl match {

case None => Unit

case Some(TweetPermalink(\_, \_)) => Unit

case Some(DmDeepLink(\_)) => Unit

case \_ => throw TweetCreateFailure.State(TweetCreateState.InvalidAttachmentUrl)

}

def validateAttachments(

stats: StatsReceiver,

validateCardRef: Gate[Option[String]]

): AttachmentBuilder.ValidationType =

FutureEffect { result: AttachmentBuilderResult =>

validateAttachmentUrl(result.attachmentUrl)

val ctx = result.validationContext

val cardRef = ctx.cardReference.filter {

case CardReferenceUriExtractor(NonTombstone(\_)) => true

case \_ => false

}

if (result.quotedTweet.isDefined && cardRef.isEmpty) {

Future.Unit

} else {

val attachmentCount =

Seq(

ctx.mediaUploadIds,

result.attachmentUrl,

result.quotedTweet

).count(\_.nonEmpty)

val userAgent = TwitterContext().flatMap(\_.userAgent)

if (attachmentCount + cardRef.count(\_ => true) > 1) {

attachmentCountLogger.warn(

s"Too many attachment types on tweet create from user: ${ctx.user.id}, " +

s"agent: '${userAgent}', media: ${ctx.mediaUploadIds}, " +

s"attachmentUrl: ${result.attachmentUrl}, cardRef: $cardRef"

)

stats.counter("too\_many\_attachment\_types\_with\_cardref").incr()

}

Future.when(attachmentCount + cardRef.count(\_ => validateCardRef(userAgent)) > 1) {

Future.exception(TweetCreateFailure.State(TweetCreateState.TooManyAttachmentTypes))

}

}

}

private val queryInclude = TweetQuery.Include(Set(Tweet.CoreDataField.id))

private val queryOptions = TweetQuery.Options(include = queryInclude)

def buildUrlShortenerCtx(request: AttachmentBuilderRequest): UrlShortener.Context =

UrlShortener.Context(

tweetId = request.tweetId,

userId = request.user.id,

createdAt = SnowflakeId(request.tweetId).time,

userProtected = request.user.safety.get.isProtected,

clientAppId = request.deviceSource.clientAppId,

remoteHost = request.remoteHost,

dark = request.darkTraffic

)

def asQuotedTweet(tweet: Tweet, shortenedUrl: ShortenedUrl): QuotedTweet =

getShare(tweet) match {

case None => QuotedTweet(tweet.id, getUserId(tweet), Some(shortenedUrl))

case Some(share) => QuotedTweet(share.sourceStatusId, share.sourceUserId, Some(shortenedUrl))

}

def tweetPermalink(request: AttachmentBuilderRequest): Option[TweetPermalink] =

request.attachmentUrl.collectFirst {

// prevent tweet-quoting cycles

case TweetPermalink(screenName, quotedTweetId) if request.tweetId > quotedTweetId =>

TweetPermalink(screenName, quotedTweetId)

}

def apply(

tweetRepo: TweetRepository.Optional,

urlShortener: UrlShortener.Type,

validateAttachments: AttachmentBuilder.ValidationType,

stats: StatsReceiver,

denyNonTweetPermalinks: Gate[Unit] = Gate.False

): Type = {

val tweetGetter = TweetRepository.tweetGetter(tweetRepo, queryOptions)

val attachmentNotPermalinkCounter = stats.counter("attachment\_url\_not\_tweet\_permalink")

val quotedTweetFoundCounter = stats.counter("quoted\_tweet\_found")

val quotedTweetNotFoundCounter = stats.counter("quoted\_tweet\_not\_found")

def buildAttachmentResult(request: AttachmentBuilderRequest) =

tweetPermalink(request) match {

case Some(qtPermalink) =>

tweetGetter(qtPermalink.tweetId).flatMap {

case Some(tweet) =>

quotedTweetFoundCounter.incr()

val ctx = buildUrlShortenerCtx(request)

urlShortener((qtPermalink.url, ctx)).map { shortenedUrl =>

AttachmentBuilderResult(

quotedTweet = Some(asQuotedTweet(tweet, shortenedUrl)),

extraChars = shortenedUrl.shortUrl.length + 1,

validationContext = request.ctx

)

}

case None =>

quotedTweetNotFoundCounter.incr()

log.warn(

s"unable to extract quote tweet from attachment builder request: $request"

)

if (denyNonTweetPermalinks()) {

throw TweetCreateFailure.State(

TweetCreateState.SourceTweetNotFound,

Some(s"quoted tweet is not found from given permalink: $qtPermalink")

)

} else {

Future.value(request.passThroughResponse)

}

}

case \_ =>

attachmentNotPermalinkCounter.incr()

Future.value(request.passThroughResponse)

}

FutureArrow { request =>

for {

result <- buildAttachmentResult(request)

() <- validateAttachments(result)

} yield result

}

}

}