package com.twitter.tweetypie

package handler

import com.twitter.service.talon.thriftscala.\_

import com.twitter.servo.util.FutureArrow

import com.twitter.tco\_util.DisplayUrl

import com.twitter.tco\_util.TcoUrl

import com.twitter.tweetypie.backends.Talon

import com.twitter.tweetypie.core.OverCapacity

import com.twitter.tweetypie.store.Guano

import com.twitter.tweetypie.thriftscala.ShortenedUrl

import scala.util.control.NoStackTrace

object UrlShortener {

type Type = FutureArrow[(String, Context), ShortenedUrl]

case class Context(

tweetId: TweetId,

userId: UserId,

createdAt: Time,

userProtected: Boolean,

clientAppId: Option[Long] = None,

remoteHost: Option[String] = None,

dark: Boolean = false)

object MalwareUrlError extends Exception with NoStackTrace

object InvalidUrlError extends Exception with NoStackTrace

/\*\*

\* Returns a new UrlShortener that checks the response from the underlying shortner

\* and, if the request is not dark but fails with a MalwareUrlError, scribes request

\* info to guano.

\*/

def scribeMalware(guano: Guano)(underlying: Type): Type =

FutureArrow {

case (longUrl, ctx) =>

underlying((longUrl, ctx)).onFailure {

case MalwareUrlError if !ctx.dark =>

guano.scribeMalwareAttempt(

Guano.MalwareAttempt(

longUrl,

ctx.userId,

ctx.clientAppId,

ctx.remoteHost

)

)

case \_ =>

}

}

def fromTalon(talonShorten: Talon.Shorten): Type = {

val log = Logger(getClass)

FutureArrow {

case (longUrl, ctx) =>

val request =

ShortenRequest(

userId = ctx.userId,

longUrl = longUrl,

auditMsg = "tweetypie",

directMessage = Some(false),

protectedAccount = Some(ctx.userProtected),

maxShortUrlLength = None,

tweetData = Some(TweetData(ctx.tweetId, ctx.createdAt.inMilliseconds)),

trafficType =

if (ctx.dark) ShortenTrafficType.Testing

else ShortenTrafficType.Production

)

talonShorten(request).flatMap { res =>

res.responseCode match {

case ResponseCode.Ok =>

if (res.malwareStatus == MalwareStatus.UrlBlocked) {

Future.exception(MalwareUrlError)

} else {

val shortUrl =

res.fullShortUrl.getOrElse {

// fall back to fromSlug if talon response does not have the full short url

// Could be replaced with an exception once the initial integration on production

// is done

TcoUrl.fromSlug(res.shortUrl, TcoUrl.isHttps(res.longUrl))

}

Future.value(

ShortenedUrl(

shortUrl = shortUrl,

longUrl = res.longUrl,

displayText = DisplayUrl(shortUrl, Some(res.longUrl), true)

)

)

}

case ResponseCode.BadInput =>

log.warn(s"Talon rejected URL that Extractor thought was fine: $longUrl")

Future.exception(InvalidUrlError)

// we shouldn't see other ResponseCodes, because Talon.Shorten translates them to

// exceptions, but we have this catch-all just in case.

case resCode =>

log.warn(s"Unexpected response code $resCode for '$longUrl'")

Future.exception(OverCapacity("talon"))

}

}

}

}

}