package com.twitter.tweetypie

package handler

import com.twitter.tco\_util.TcoUrl

import com.twitter.tweetypie.core.\_

import com.twitter.tweetypie.thriftscala.entities.EntityExtractor

import com.twitter.tweetypie.thriftscala.\_

import com.twitter.tweetypie.tweettext.IndexConverter

import com.twitter.tweetypie.tweettext.Offset

import com.twitter.tweetypie.tweettext.Preprocessor.\_

object UrlEntityBuilder {

import UpstreamFailure.UrlShorteningFailure

import UrlShortener.Context

/\*\*

\* Extracts URLs from the given tweet text, shortens them, and returns an updated tweet

\* text that contains the shortened URLs, along with the generated `UrlEntity`s.

\*/

type Type = FutureArrow[(String, Context), (String, Seq[UrlEntity])]

def fromShortener(shortener: UrlShortener.Type): Type =

FutureArrow {

case (text, ctx) =>

Future

.collect(EntityExtractor.extractAllUrls(text).map(shortenEntity(shortener, \_, ctx)))

.map(\_.flatMap(\_.toSeq))

.map(updateTextAndUrls(text, \_)(replaceInvisiblesWithWhitespace))

}

/\*\*

\* Update a url entity with tco-ed url

\*

\* @param urlEntity an url entity with long url in the `url` field

\* @param ctx additional data needed to build the shortener request

\* @return an updated url entity with tco-ed url in the `url` field,

\* and long url in the `expanded` field

\*/

private def shortenEntity(

shortener: UrlShortener.Type,

entity: UrlEntity,

ctx: Context

): Future[Option[UrlEntity]] =

shortener((TcoUrl.normalizeProtocol(entity.url), ctx))

.map { urlData =>

Some(

entity.copy(

url = urlData.shortUrl,

expanded = Some(urlData.longUrl),

display = Some(urlData.displayText)

)

)

}

.rescue {

// fail tweets with invalid urls

case UrlShortener.InvalidUrlError =>

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

// fail tweets with malware urls

case UrlShortener.MalwareUrlError =>

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

// propagate OverCapacity

case e @ OverCapacity(\_) => Future.exception(e)

// convert any other failure into UrlShorteningFailure

case e => Future.exception(UrlShorteningFailure(e))

}

/\*\*

\* Applies a text-modification function to all parts of the text not found within a UrlEntity,

\* and then updates all the UrlEntity indices as necessary.

\*/

def updateTextAndUrls(

text: String,

urlEntities: Seq[UrlEntity]

)(

textMod: String => String

): (String, Seq[UrlEntity]) = {

var offsetInText = Offset.CodePoint(0)

var offsetInNewText = Offset.CodePoint(0)

val newText = new StringBuilder

val newUrlEntities = Seq.newBuilder[UrlEntity]

val indexConverter = new IndexConverter(text)

urlEntities.foreach { e =>

val nonUrl = textMod(indexConverter.substringByCodePoints(offsetInText.toInt, e.fromIndex))

newText.append(nonUrl)

newText.append(e.url)

offsetInText = Offset.CodePoint(e.toIndex.toInt)

val urlFrom = offsetInNewText + Offset.CodePoint.length(nonUrl)

val urlTo = urlFrom + Offset.CodePoint.length(e.url)

val newEntity =

e.copy(fromIndex = urlFrom.toShort, toIndex = urlTo.toShort)

newUrlEntities += newEntity

offsetInNewText = urlTo

}

newText.append(textMod(indexConverter.substringByCodePoints(offsetInText.toInt)))

(newText.toString, newUrlEntities.result())

}

}