package com.twitter.tweetypie

package hydrator

import com.twitter.stitch.Stitch

import com.twitter.tweetypie.core.\_

import com.twitter.tweetypie.media.Media

import com.twitter.tweetypie.repository.TweetQuery

import com.twitter.tweetypie.serverutil.ExtendedTweetMetadataBuilder

import com.twitter.tweetypie.thriftscala.UrlEntity

import com.twitter.tweetypie.thriftscala.\_

import com.twitter.tweetypie.thriftscala.entities.Implicits.\_

import com.twitter.tweetypie.tweettext.Offset

import com.twitter.tweetypie.tweettext.TextModification

import com.twitter.tweetypie.tweettext.TweetText

import com.twitter.tweetypie.util.EditControlUtil

import com.twitter.tweetypie.util.TweetLenses

/\*\*

\* This hydrator is the backwards-compatibility layer to support QT, Edit Tweets & Mixed Media

\* Tweets rendering on legacy non-updated clients. Legacy rendering provides a way for every client

\* to consume these Tweets until the client is upgraded. For Edit and Mixed Media Tweets, the

\* Tweet's self-permalink is appended to the visible text. For Quoting Tweets, the Quoted Tweet's

\* permalink is appended to the text. For Tweets that meet multiple criteria for legacy rendering

\* (e.g. QT containing Mixed Media), only one permalink is appended and the self-permalink takes

\* precedence.

\*/

object TweetLegacyFormatter {

private[this] val log = Logger(getClass)

import TweetText.\_

def legacyQtPermalink(

td: TweetData,

opts: TweetQuery.Options

): Option[ShortenedUrl] = {

val tweet = td.tweet

val tweetText = TweetLenses.text(tweet)

val urls = TweetLenses.urls(tweet)

val ctx = TweetCtx.from(td, opts)

val qtPermalink: Option[ShortenedUrl] = tweet.quotedTweet.flatMap(\_.permalink)

val qtShortUrl = qtPermalink.map(\_.shortUrl)

def urlsContains(url: String): Boolean =

urls.exists(\_.url == url)

val doLegacyQtFormatting =

!opts.simpleQuotedTweet && !ctx.isRetweet &&

qtPermalink.isDefined && qtShortUrl.isDefined &&

!qtShortUrl.exists(tweetText.contains) &&

!qtShortUrl.exists(urlsContains)

if (doLegacyQtFormatting) qtPermalink else None

}

def legacySelfPermalink(

td: TweetData

): Option[ShortenedUrl] = {

val tweet = td.tweet

val selfPermalink = tweet.selfPermalink

val tweetText = TweetLenses.text(tweet)

val urls = TweetLenses.urls(tweet)

val selfShortUrl = selfPermalink.map(\_.shortUrl)

def urlsContains(url: String): Boolean =

urls.exists(\_.url == url)

val doLegacyFormatting =

selfPermalink.isDefined && selfShortUrl.isDefined &&

!selfShortUrl.exists(tweetText.contains) &&

!selfShortUrl.exists(urlsContains) &&

needsLegacyFormatting(td)

if (doLegacyFormatting) selfPermalink else None

}

def isMixedMediaTweet(tweet: Tweet): Boolean =

tweet.media.exists(Media.isMixedMedia)

def buildUrlEntity(from: Short, to: Short, permalink: ShortenedUrl): UrlEntity =

UrlEntity(

fromIndex = from,

toIndex = to,

url = permalink.shortUrl,

expanded = Some(permalink.longUrl),

display = Some(permalink.displayText)

)

private[this] def isValidVisibleRange(

tweetIdForLogging: TweetId,

textRange: TextRange,

textLength: Int

) = {

val isValid = textRange.fromIndex <= textRange.toIndex && textRange.toIndex <= textLength

if (!isValid) {

log.warn(s"Tweet $tweetIdForLogging has invalid visibleTextRange: $textRange")

}

isValid

}

// This Function checks if legacy formatting is required for Edit & Mixed Media Tweets.

// Calls FeatureSwitches.matchRecipient which is an expensive call,

// so caution is taken to call it only once and only when needed.

def needsLegacyFormatting(

td: TweetData

): Boolean = {

val isEdit = EditControlUtil.isEditTweet(td.tweet)

val isMixedMedia = isMixedMediaTweet(td.tweet)

val isNoteTweet = td.tweet.noteTweet.isDefined

if (isEdit || isMixedMedia || isNoteTweet) {

// These feature switches are disabled unless greater than certain android, ios versions

// & all versions of RWEB.

val TweetEditConsumptionEnabledKey = "tweet\_edit\_consumption\_enabled"

val MixedMediaEnabledKey = "mixed\_media\_enabled"

val NoteTweetConsumptionEnabledKey = "note\_tweet\_consumption\_enabled"

def fsEnabled(fsKey: String): Boolean = {

td.featureSwitchResults

.flatMap(\_.getBoolean(fsKey, shouldLogImpression = false))

.getOrElse(false)

}

val tweetEditConsumptionEnabled = fsEnabled(TweetEditConsumptionEnabledKey)

val mixedMediaEnabled = fsEnabled(MixedMediaEnabledKey)

val noteTweetConsumptionEnabled = fsEnabled(NoteTweetConsumptionEnabledKey)

(isEdit && !tweetEditConsumptionEnabled) ||

(isMixedMedia && !mixedMediaEnabled) ||

(isNoteTweet && !noteTweetConsumptionEnabled)

} else {

false

}

}

//given a permalink, the tweet text gets updated

def updateTextAndURLsAndMedia(

permalink: ShortenedUrl,

tweet: Tweet,

statsReceiver: StatsReceiver

): Tweet = {

val originalText = TweetLenses.text(tweet)

val originalTextLength = codePointLength(originalText)

// Default the visible range to the whole tweet if the existing visible range is invalid.

val visibleRange: TextRange =

TweetLenses

.visibleTextRange(tweet)

.filter((r: TextRange) => isValidVisibleRange(tweet.id, r, originalTextLength))

.getOrElse(TextRange(0, originalTextLength))

val permalinkShortUrl = permalink.shortUrl

val insertAtCodePoint = Offset.CodePoint(visibleRange.toIndex)

/\*

\* Insertion at position 0 implies that the original tweet text has no

\* visible text, so the resulting text should be only the url without

\* leading padding.

\*/

val padLeft = if (insertAtCodePoint.toInt > 0) " " else ""

/\*

\* Empty visible text at position 0 implies that the original tweet text

\* only contains a URL in the hidden suffix area, which would not already

\* be padded.

\*/

val padRight = if (visibleRange == TextRange(0, 0)) " " else ""

val paddedShortUrl = s"$padLeft$permalinkShortUrl$padRight"

val tweetTextModification = TextModification.insertAt(

originalText,

insertAtCodePoint,

paddedShortUrl

)

/\*

\* As we modified tweet text and appended tweet permalink above

\* we have to correct the url and media entities accordingly as they are

\* expected to be present in the hidden suffix of text.

\*

\* - we compute the new (from, to) indices for the url entity

\* - build new url entity for quoted tweet permalink or self permalink for Edit/ MM Tweets

\* - shift url entities which are after visible range end

\* - shift media entities associated with above url entities

\*/

val shortUrlLength = codePointLength(permalinkShortUrl)

val fromIndex = insertAtCodePoint.toInt + codePointLength(padLeft)

val toIndex = fromIndex + shortUrlLength

val tweetUrlEntity = buildUrlEntity(

from = fromIndex.toShort,

to = toIndex.toShort,

permalink = permalink

)

val tweetMedia = if (isMixedMediaTweet(tweet)) {

TweetLenses.media(tweet).take(1)

} else {

TweetLenses.media(tweet)

}

val modifiedMedia = tweetTextModification.reindexEntities(tweetMedia)

val modifiedUrls =

tweetTextModification.reindexEntities(TweetLenses.urls(tweet)) :+ tweetUrlEntity

val modifiedText = tweetTextModification.updated

/\*

\* Visible Text Range computation differs by scenario

\* == Any Tweet with Media ==

\* Tweet text has a media url \*after\* the visible text range

\* original text: [visible text] https://t.co/mediaUrl

\* original range: ^START END^

\*

\* Append the permalink URL to the \*visible text\* so non-upgraded clients can see it

\* modified text: [visible text https://t.co/permalink] https://t.co/mediaUrl

\* modified range: ^START END^

\* visible range expanded, permalink is visible

\*

\* == Non-QT Tweet w/o Media ==

\* original text: [visible text]

\* original range: None (default: whole text is visible)

\*

\* modified text: [visible text https://t.co/selfPermalink]

\* modified range: None (default: whole text is visible)

\* trailing self permalink will be visible

\*

\* == QT w/o Media ==

\* original text: [visible text]

\* original range: None (default: whole text is visible)

\*

\* modified text: [visible text] https://t.co/qtPermalink

\* modified range: ^START END^

\* trailing QT permalink is \*hidden\* because legacy clients that process the visible text range know how to display QTs

\*

\* == Non-QT Replies w/o media ==

\* original text: @user [visible text]

\* original range: ^START END^

\*

\* modified text: @user [visible text https://t.co/selfPermalink]

\* modified range: ^START END^

\* visible range expanded, self permalink is visible

\*

\* == QT Replies w/o media ==

\* original text: @user [visible text]

\* original range: ^START END^

\*

\* modified text: @user [visible text] https://t.co/qtPermalink

\* modified range: ^START END^

\* visible range remains the same, trailing QT permalink is hidden

\*

\*/

val modifiedVisibleTextRange =

if (modifiedMedia.nonEmpty ||

EditControlUtil.isEditTweet(tweet) ||

tweet.noteTweet.isDefined) {

Some(

visibleRange.copy(

toIndex = visibleRange.toIndex + codePointLength(padLeft) + shortUrlLength

)

)

} else {

Some(visibleRange)

}

val updatedTweet =

Lens.setAll(

tweet,

TweetLenses.text -> modifiedText,

TweetLenses.urls -> modifiedUrls.sortBy(\_.fromIndex),

TweetLenses.media -> modifiedMedia.sortBy(\_.fromIndex),

TweetLenses.visibleTextRange -> modifiedVisibleTextRange

)

/\*\*

\* compute extended tweet metadata when text length > 140

\* and apply the final lens to return a modified tweet

\*/

val totalDisplayLength = displayLength(modifiedText)

if (totalDisplayLength > OriginalMaxDisplayLength) {

updatedTweet.selfPermalink match {

case Some(permalink) =>

val extendedTweetMetadata = ExtendedTweetMetadataBuilder(updatedTweet, permalink)

updatedTweet.copy(

extendedTweetMetadata = Some(extendedTweetMetadata)

)

case None =>

/\*\*

\* This case shouldn't happen as TweetBuilder currently populates

\* selfPermalink for extended tweets. In QT + Media, we will

\* use AttachmentBuilder to store selfPermalink during writes,

\* if text display length is going to exceed 140 after QT url append.

\*/

log.error(

s"Failed to compute extended metadata for tweet: ${tweet.id} with " +

s"display length: ${totalDisplayLength}, as self-permalink is empty."

)

statsReceiver.counter("self\_permalink\_not\_found").incr()

tweet

}

} else {

updatedTweet

}

}

def apply(

statsReceiver: StatsReceiver

): TweetDataValueHydrator = {

ValueHydrator[TweetData, TweetQuery.Options] { (td, opts) =>

// Prefer any required self permalink rendering over QT permalink rendering because a

// client that doesn't understand the attributes of the Tweet (i.e. Edit, Mixed

// Media) won't be able to render the Tweet properly at all, regardless of whether

// it's a QT. By preferring a visible self-permalink, the viewer is linked to an

// RWeb view of the Tweet which can fully display all of its features.

val permalink: Option[ShortenedUrl] =

legacySelfPermalink(td)

.orElse(legacyQtPermalink(td, opts))

permalink match {

case Some(permalink) =>

val updatedTweet = updateTextAndURLsAndMedia(permalink, td.tweet, statsReceiver)

Stitch(ValueState.delta(td, td.copy(tweet = updatedTweet)))

case \_ =>

Stitch(ValueState.unmodified(td))

}

}

}

}