package com.twitter.tweetypie

package hydrator

import com.twitter.expandodo.thriftscala.Card2

import com.twitter.expandodo.thriftscala.Card2RequestOptions

import com.twitter.featureswitches.v2.FeatureSwitchResults

import com.twitter.stitch.Stitch

import com.twitter.tweetypie.core.CardReferenceUriExtractor

import com.twitter.tweetypie.core.NonTombstone

import com.twitter.tweetypie.core.ValueState

import com.twitter.tweetypie.repository.\_

import com.twitter.tweetypie.thriftscala.\_

object Card2Hydrator {

type Type = ValueHydrator[Option[Card2], Ctx]

case class Ctx(

urlEntities: Seq[UrlEntity],

mediaEntities: Seq[MediaEntity],

cardReference: Option[CardReference],

underlyingTweetCtx: TweetCtx,

featureSwitchResults: Option[FeatureSwitchResults])

extends TweetCtx.Proxy

val hydratedField: FieldByPath = fieldByPath(Tweet.Card2Field)

val hydrationUrlBlockListKey = "card\_hydration\_blocklist"

def apply(repo: Card2Repository.Type): ValueHydrator[Option[Card2], Ctx] =

ValueHydrator[Option[Card2], Ctx] { (\_, ctx) =>

val repoCtx = requestOptions(ctx)

val filterURLs = ctx.featureSwitchResults

.flatMap(\_.getStringArray(hydrationUrlBlockListKey, false))

.getOrElse(Seq())

val requests =

ctx.cardReference match {

case Some(CardReferenceUriExtractor(cardUri)) =>

cardUri match {

case NonTombstone(uri) if !filterURLs.contains(uri) =>

Seq((UrlCard2Key(uri), repoCtx))

case \_ => Nil

}

case \_ =>

ctx.urlEntities

.filterNot(e => e.expanded.exists(filterURLs.contains))

.map(e => (UrlCard2Key(e.url), repoCtx))

}

Stitch

.traverse(requests) {

case (key, opts) => repo(key, opts).liftNotFoundToOption

}.liftToTry.map {

case Return(results) =>

results.flatten.lastOption match {

case None => ValueState.UnmodifiedNone

case res => ValueState.modified(res)

}

case Throw(\_) => ValueState.partial(None, hydratedField)

}

}.onlyIf { (curr, ctx) =>

curr.isEmpty &&

ctx.tweetFieldRequested(Tweet.Card2Field) &&

ctx.opts.cardsPlatformKey.nonEmpty &&

!ctx.isRetweet &&

ctx.mediaEntities.isEmpty &&

(ctx.cardReference.nonEmpty || ctx.urlEntities.nonEmpty)

}

private[this] def requestOptions(ctx: Ctx) =

Card2RequestOptions(

platformKey = ctx.opts.cardsPlatformKey.get,

perspectiveUserId = ctx.opts.forUserId,

allowNonTcoUrls = ctx.cardReference.nonEmpty,

languageTag = Some(ctx.opts.languageTag)

)

}