package com.twitter.tweetypie

package hydrator

import com.twitter.featureswitches.v2.FeatureSwitchResults

import com.twitter.spam.rtf.thriftscala.SafetyLevel

import com.twitter.stitch.Stitch

import com.twitter.stitch.timelineservice.TimelineService.GetPerspectives.Query

import com.twitter.timelineservice.{thriftscala => tls}

import com.twitter.tweetypie.core.\_

import com.twitter.tweetypie.repository.PerspectiveRepository

import com.twitter.tweetypie.thriftscala.FieldByPath

import com.twitter.tweetypie.thriftscala.StatusPerspective

object PerspectiveHydrator {

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

val hydratedField: FieldByPath = fieldByPath(Tweet.PerspectiveField)

case class Ctx(featureSwitchResults: Option[FeatureSwitchResults], underlyingTweetCtx: TweetCtx)

extends TweetCtx.Proxy

val Types: Set[tls.PerspectiveType] =

Set(

tls.PerspectiveType.Reported,

tls.PerspectiveType.Favorited,

tls.PerspectiveType.Retweeted,

tls.PerspectiveType.Bookmarked

)

val TypesWithoutBookmarked: Set[tls.PerspectiveType] =

Set(

tls.PerspectiveType.Reported,

tls.PerspectiveType.Favorited,

tls.PerspectiveType.Retweeted

)

private[this] val partialResult = ValueState.partial(None, hydratedField)

val bookmarksPerspectiveHydrationEnabledKey = "bookmarks\_perspective\_hydration\_enabled"

def evaluatePerspectiveTypes(

userId: Long,

bookmarksPerspectiveDecider: Gate[Long],

featureSwitchResults: Option[FeatureSwitchResults]

): Set[tls.PerspectiveType] = {

if (bookmarksPerspectiveDecider(userId) ||

featureSwitchResults

.flatMap(\_.getBoolean(bookmarksPerspectiveHydrationEnabledKey, false))

.getOrElse(false))

Types

else

TypesWithoutBookmarked

}

def apply(

repo: PerspectiveRepository.Type,

shouldHydrateBookmarksPerspective: Gate[Long],

stats: StatsReceiver

): Type = {

val statsByLevel =

SafetyLevel.list.map(level => (level, stats.counter(level.name, "calls"))).toMap

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

val res: Stitch[tls.TimelineEntryPerspective] = if (ctx.isRetweet) {

Stitch.value(

tls.TimelineEntryPerspective(

favorited = false,

retweetId = None,

retweeted = false,

reported = false,

bookmarked = None

)

)

} else {

statsByLevel

.getOrElse(ctx.opts.safetyLevel, stats.counter(ctx.opts.safetyLevel.name, "calls"))

.incr()

repo(

Query(

userId = ctx.opts.forUserId.get,

tweetId = ctx.tweetId,

types = evaluatePerspectiveTypes(

ctx.opts.forUserId.get,

shouldHydrateBookmarksPerspective,

ctx.featureSwitchResults)

))

}

res.liftToTry.map {

case Return(perspective) =>

ValueState.modified(

Some(

StatusPerspective(

userId = ctx.opts.forUserId.get,

favorited = perspective.favorited,

retweeted = perspective.retweeted,

retweetId = perspective.retweetId,

reported = perspective.reported,

bookmarked = perspective.bookmarked

)

)

)

case \_ => partialResult

}

}.onlyIf { (curr, ctx) =>

curr.isEmpty &&

ctx.opts.forUserId.nonEmpty &&

(ctx.tweetFieldRequested(Tweet.PerspectiveField) || ctx.opts.excludeReported)

}

}

}