package com.twitter.visibility.interfaces.tweets

import com.twitter.spam.rtf.{thriftscala => t}

import com.twitter.context.TwitterContext

import com.twitter.context.thriftscala.Viewer

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.stitch.Stitch

import com.twitter.strato.catalog.Fetch

import com.twitter.strato.client.Client

import com.twitter.strato.client.Fetcher

import com.twitter.strato.thrift.ScroogeConvImplicits.\_

import com.twitter.visibility.builder.VisibilityResult

import com.twitter.visibility.common.tweets.TweetVisibilityResultMapper

import com.twitter.visibility.models.SafetyLevel.toThrift

import com.twitter.visibility.models.ViewerContext

import com.twitter.visibility.thriftscala.TweetVisibilityResult

class TweetVisibilityLibraryParityTest(statsReceiver: StatsReceiver, stratoClient: Client) {

private val parityTestScope = statsReceiver.scope("tweet\_visibility\_library\_parity")

private val requests = parityTestScope.counter("requests")

private val equal = parityTestScope.counter("equal")

private val incorrect = parityTestScope.counter("incorrect")

private val empty = parityTestScope.counter("empty")

private val failures = parityTestScope.counter("failures")

private val fetcher: Fetcher[Long, t.SafetyLevel, TweetVisibilityResult] =

stratoClient.fetcher[Long, t.SafetyLevel, TweetVisibilityResult](

"visibility/service/TweetVisibilityResult.Tweet"

)

def runParityTest(

req: TweetVisibilityRequest,

resp: VisibilityResult

): Stitch[Unit] = {

requests.incr()

val twitterContext = TwitterContext(TwitterContextPermit)

val viewer: Option[Viewer] = {

val remoteViewerContext = ViewerContext.fromContext

if (remoteViewerContext != req.viewerContext) {

val updatedRemoteViewerContext = remoteViewerContext.copy(

userId = req.viewerContext.userId

)

if (updatedRemoteViewerContext == req.viewerContext) {

twitterContext() match {

case None =>

Some(Viewer(userId = req.viewerContext.userId))

case Some(v) =>

Some(v.copy(userId = req.viewerContext.userId))

}

} else {

None

}

} else {

None

}

}

val tweetypieContext = TweetypieContext(

isQuotedTweet = req.isInnerQuotedTweet,

isRetweet = req.isRetweet,

hydrateConversationControl = req.hydrateConversationControl

)

val parityCheck: Stitch[Fetch.Result[TweetVisibilityResult]] = {

Stitch.callFuture {

TweetypieContext.let(tweetypieContext) {

viewer match {

case Some(viewer) =>

twitterContext.let(viewer) {

Stitch.run(fetcher.fetch(req.tweet.id, toThrift(req.safetyLevel)))

}

case None =>

Stitch.run(fetcher.fetch(req.tweet.id, toThrift(req.safetyLevel)))

}

}

}

}

parityCheck

.flatMap { parityResponse =>

val tvr = TweetVisibilityResultMapper.fromAction(resp.verdict.toActionThrift())

parityResponse.v match {

case Some(ptvr) =>

if (tvr == ptvr) {

equal.incr()

} else {

incorrect.incr()

}

case None =>

empty.incr()

}

Stitch.Done

}.rescue {

case t: Throwable =>

failures.incr()

Stitch.Done

}.unit

}

}