package com.twitter.unified\_user\_actions.enricher.graphql

import com.google.common.util.concurrent.RateLimiter

import com.twitter.dynmap.DynMap

import com.twitter.dynmap.json.DynMapJson

import com.twitter.finagle.stats.Counter

import com.twitter.finagle.stats.NullStatsReceiver

import com.twitter.util.logging.Logging

import com.twitter.util.Return

import com.twitter.util.Throw

import com.twitter.util.Try

/\*\*

\* @param dm The DynMap parsed from the returned Json string

\*/

case class GraphqlRspErrors(dm: DynMap) extends Exception {

override def toString: String = dm.toString()

}

object GraphqlRspParser extends Logging {

private val rateLimiter = RateLimiter.create(1.0) // at most 1 log message per second

private def rateLimitedLogError(e: Throwable): Unit =

if (rateLimiter.tryAcquire()) {

error(e.getMessage, e)

}

/\*\*

\* GraphQL's response is a Json string.

\* This function first parses the raw response as a Json string, then it checks if the returned

\* object has the "data" field which means the response is expected. The response could also

\* return a valid Json string but with errors inside it as a list of "errors".

\*/

def toDynMap(

rsp: String,

invalidRspCounter: Counter = NullStatsReceiver.NullCounter,

failedReqCounter: Counter = NullStatsReceiver.NullCounter

): Try[DynMap] = {

val rawRsp: Try[DynMap] = DynMapJson.fromJsonString(rsp)

rawRsp match {

case Return(r) =>

if (r.getMapOpt("data").isDefined) Return(r)

else {

invalidRspCounter.incr()

rateLimitedLogError(GraphqlRspErrors(r))

Throw(GraphqlRspErrors(r))

}

case Throw(e) =>

rateLimitedLogError(e)

failedReqCounter.incr()

Throw(e)

}

}

/\*\*

\* Similar to `toDynMap` above, but returns an Option

\*/

def toDynMapOpt(

rsp: String,

invalidRspCounter: Counter = NullStatsReceiver.NullCounter,

failedReqCounter: Counter = NullStatsReceiver.NullCounter

): Option[DynMap] =

toDynMap(

rsp = rsp,

invalidRspCounter = invalidRspCounter,

failedReqCounter = failedReqCounter).toOption

}