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

import com.google.common.cache.CacheBuilder

import com.twitter.dynmap.DynMap

import com.twitter.graphql.thriftscala.GraphQlRequest

import com.twitter.graphql.thriftscala.GraphQlResponse

import com.twitter.graphql.thriftscala.GraphqlExecutionService

import com.twitter.inject.Test

import com.twitter.unified\_user\_actions.enricher.EnricherFixture

import com.twitter.unified\_user\_actions.enricher.FatalException

import com.twitter.unified\_user\_actions.enricher.hcache.LocalCache

import com.twitter.unified\_user\_actions.enricher.internal.thriftscala.EnrichmentEnvelop

import com.twitter.unified\_user\_actions.enricher.internal.thriftscala.EnrichmentIdType

import com.twitter.unified\_user\_actions.enricher.internal.thriftscala.EnrichmentInstruction

import com.twitter.unified\_user\_actions.enricher.internal.thriftscala.EnrichmentKey

import com.twitter.unified\_user\_actions.thriftscala.AuthorInfo

import com.twitter.util.Await

import com.twitter.util.Future

import org.mockito.ArgumentMatchers

import org.mockito.MockitoSugar

class DefaultHydratorTest extends Test with MockitoSugar {

trait Fixtures extends EnricherFixture {

val cache = new LocalCache[EnrichmentKey, DynMap](

underlying = CacheBuilder

.newBuilder()

.maximumSize(10)

.build[EnrichmentKey, Future[DynMap]]())

val client = mock[GraphqlExecutionService.FinagledClient]

val key = EnrichmentKey(EnrichmentIdType.TweetId, 1L)

val envelop = EnrichmentEnvelop(123L, mkUUATweetEvent(1L), tweetInfoEnrichmentPlan)

def mkGraphQLResponse(authorId: Long): GraphQlResponse =

GraphQlResponse(

Some(

s"""

|{

| "data": {

| "tweet\_result\_by\_rest\_id": {

| "result": {

| "core": {

| "user": {

| "legacy": {

| "id\_str": "$authorId"

| }

| }

| }

| }

| }

| }

|}

|""".stripMargin

))

}

test("non-fatal errors should proceed as normal") {

new Fixtures {

val hydrator = new DefaultHydrator(cache, client)

// when graphql client encounter any exception

when(client.graphql(ArgumentMatchers.any[GraphQlRequest]))

.thenReturn(Future.exception(new IllegalStateException("any exception")))

val actual =

Await.result(hydrator.hydrate(EnrichmentInstruction.TweetEnrichment, Some(key), envelop))

// then the original envelop is expected

assert(envelop == actual)

}

}

test("fatal errors should return a future exception") {

new Fixtures {

val hydrator = new DefaultHydrator(cache, client)

// when graphql client encounter a fatal exception

when(client.graphql(ArgumentMatchers.any[GraphQlRequest]))

.thenReturn(Future.exception(new FatalException("fatal exception") {}))

val actual = hydrator.hydrate(EnrichmentInstruction.TweetEnrichment, Some(key), envelop)

// then a failed future is expected

assertFailedFuture[FatalException](actual)

}

}

test("author\_id should be hydrated from graphql respond") {

new Fixtures {

val hydrator = new DefaultHydrator(cache, client)

when(client.graphql(ArgumentMatchers.any[GraphQlRequest]))

.thenReturn(Future.value(mkGraphQLResponse(888L)))

val actual = hydrator.hydrate(EnrichmentInstruction.TweetEnrichment, Some(key), envelop)

assertFutureValue(

actual,

envelop.copy(uua = mkUUATweetEvent(1L, Some(AuthorInfo(Some(888L))))))

}

}

test("when AuthorInfo is populated, there should be no hydration") {

new Fixtures {

val hydrator = new DefaultHydrator(cache, client)

when(client.graphql(ArgumentMatchers.any[GraphQlRequest]))

.thenReturn(Future.value(mkGraphQLResponse(333L)))

val expected = envelop.copy(uua =

mkUUATweetEvent(tweetId = 3L, author = Some(AuthorInfo(authorId = Some(222)))))

val actual = hydrator.hydrate(EnrichmentInstruction.TweetEnrichment, Some(key), expected)

assertFutureValue(actual, expected)

}

}

}