package com.twitter.unified\_user\_actions.adapter

import com.twitter.clientapp.thriftscala.AmplifyDetails

import com.twitter.clientapp.thriftscala.MediaDetails

import com.twitter.clientapp.thriftscala.MediaType

import com.twitter.mediaservices.commons.thriftscala.MediaCategory

import com.twitter.unified\_user\_actions.adapter.client\_event.VideoClientEventUtils.getVideoMetadata

import com.twitter.unified\_user\_actions.adapter.client\_event.VideoClientEventUtils.videoIdFromMediaIdentifier

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

import com.twitter.util.mock.Mockito

import com.twitter.video.analytics.thriftscala.\_

import org.junit.runner.RunWith

import org.scalatest.funsuite.AnyFunSuite

import org.scalatest.matchers.should.Matchers

import org.scalatest.prop.TableDrivenPropertyChecks

import org.scalatestplus.junit.JUnitRunner

@RunWith(classOf[JUnitRunner])

class VideoClientEventUtilsSpec

extends AnyFunSuite

with Matchers

with Mockito

with TableDrivenPropertyChecks {

trait Fixture {

val mediaDetails = Seq[MediaDetails](

MediaDetails(

contentId = Some("456"),

mediaType = Some(MediaType.ConsumerVideo),

dynamicAds = Some(false)),

MediaDetails(

contentId = Some("123"),

mediaType = Some(MediaType.ConsumerVideo),

dynamicAds = Some(false)),

MediaDetails(

contentId = Some("789"),

mediaType = Some(MediaType.ConsumerVideo),

dynamicAds = Some(false))

)

val videoMetadata: TweetActionInfo = TweetActionInfo.TweetVideoWatch(

TweetVideoWatch(mediaType = Some(MediaType.ConsumerVideo), isMonetizable = Some(false)))

val videoMetadataWithAmplifyDetailsVideoType: TweetActionInfo = TweetActionInfo.TweetVideoWatch(

TweetVideoWatch(

mediaType = Some(MediaType.ConsumerVideo),

isMonetizable = Some(false),

videoType = Some("content")))

val validMediaIdentifier: MediaIdentifier = MediaIdentifier.MediaPlatformIdentifier(

MediaPlatformIdentifier(mediaId = 123L, mediaCategory = MediaCategory.TweetVideo))

val invalidMediaIdentifier: MediaIdentifier = MediaIdentifier.AmplifyCardIdentifier(

AmplifyCardIdentifier(vmapUrl = "", contentId = "")

)

}

test("findVideoMetadata") {

new Fixture {

val testData = Table(

("testType", "mediaId", "mediaItems", "amplifyDetails", "expectedOutput"),

("emptyMediaDetails", "123", Seq[MediaDetails](), None, None),

("mediaIdNotFound", "111", mediaDetails, None, None),

("mediaIdFound", "123", mediaDetails, None, Some(videoMetadata)),

(

"mediaIdFound",

"123",

mediaDetails,

Some(AmplifyDetails(videoType = Some("content"))),

Some(videoMetadataWithAmplifyDetailsVideoType))

)

forEvery(testData) {

(

\_: String,

mediaId: String,

mediaItems: Seq[MediaDetails],

amplifyDetails: Option[AmplifyDetails],

expectedOutput: Option[TweetActionInfo]

) =>

val actual = getVideoMetadata(mediaId, mediaItems, amplifyDetails)

assert(expectedOutput === actual)

}

}

}

test("videoIdFromMediaIdentifier") {

new Fixture {

val testData = Table(

("testType", "mediaIdentifier", "expectedOutput"),

("validMediaIdentifierType", validMediaIdentifier, Some("123")),

("invalidMediaIdentifierType", invalidMediaIdentifier, None)

)

forEvery(testData) {

(\_: String, mediaIdentifier: MediaIdentifier, expectedOutput: Option[String]) =>

val actual = videoIdFromMediaIdentifier(mediaIdentifier)

assert(expectedOutput === actual)

}

}

}

}