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

import com.twitter.inject.Test

import com.twitter.tweetypie.thriftscala.RetweetArchivalEvent

import com.twitter.unified\_user\_actions.adapter.retweet\_archival\_events.RetweetArchivalEventsAdapter

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

import com.twitter.util.Time

import org.scalatest.prop.TableDrivenPropertyChecks

class RetweetArchivalEventsAdapterSpec extends Test with TableDrivenPropertyChecks {

trait Fixture {

val frozenTime = Time.fromMilliseconds(1658949273000L)

val authorId = 1L

val tweetId = 101L

val retweetId = 102L

val retweetAuthorId = 2L

val retweetArchivalEvent = RetweetArchivalEvent(

retweetId = retweetId,

srcTweetId = tweetId,

retweetUserId = retweetAuthorId,

srcTweetUserId = authorId,

timestampMs = 0L,

isArchivingAction = Some(true),

)

val retweetUnarchivalEvent = RetweetArchivalEvent(

retweetId = retweetId,

srcTweetId = tweetId,

retweetUserId = retweetAuthorId,

srcTweetUserId = authorId,

timestampMs = 0L,

isArchivingAction = Some(false),

)

val expectedUua1 = UnifiedUserAction(

userIdentifier = UserIdentifier(userId = Some(retweetAuthorId)),

item = Item.TweetInfo(

TweetInfo(

actionTweetId = tweetId,

actionTweetAuthorInfo = Some(AuthorInfo(authorId = Some(authorId))),

retweetingTweetId = Some(retweetId)

)

),

actionType = ActionType.ServerTweetArchiveRetweet,

eventMetadata = EventMetadata(

sourceTimestampMs = 0L,

receivedTimestampMs = frozenTime.inMilliseconds,

sourceLineage = SourceLineage.ServerRetweetArchivalEvents,

)

)

val expectedUua2 = UnifiedUserAction(

userIdentifier = UserIdentifier(userId = Some(retweetAuthorId)),

item = Item.TweetInfo(

TweetInfo(

actionTweetId = tweetId,

actionTweetAuthorInfo = Some(AuthorInfo(authorId = Some(authorId))),

retweetingTweetId = Some(retweetId)

)

),

actionType = ActionType.ServerTweetUnarchiveRetweet,

eventMetadata = EventMetadata(

sourceTimestampMs = 0L,

receivedTimestampMs = frozenTime.inMilliseconds,

sourceLineage = SourceLineage.ServerRetweetArchivalEvents,

)

)

}

test("all tests") {

new Fixture {

Time.withTimeAt(frozenTime) { \_ =>

val table = Table(

("event", "expected"),

(retweetArchivalEvent, expectedUua1),

(retweetUnarchivalEvent, expectedUua2),

)

forEvery(table) { (event: RetweetArchivalEvent, expected: UnifiedUserAction) =>

val actual = RetweetArchivalEventsAdapter.adaptEvent(event)

assert(Seq(expected) === actual)

}

}

}

}

}