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

import com.twitter.gizmoduck.thriftscala.User

import com.twitter.gizmoduck.thriftscala.UserType

import com.twitter.inject.Test

import com.twitter.snowflake.id.SnowflakeId

import com.twitter.tweetypie.thriftscala.AdditionalFieldDeleteEvent

import com.twitter.tweetypie.thriftscala.AdditionalFieldUpdateEvent

import com.twitter.tweetypie.thriftscala.AuditDeleteTweet

import com.twitter.tweetypie.thriftscala.DeviceSource

import com.twitter.tweetypie.thriftscala.EditControl

import com.twitter.tweetypie.thriftscala.EditControlEdit

import com.twitter.tweetypie.thriftscala.Language

import com.twitter.tweetypie.thriftscala.Place

import com.twitter.tweetypie.thriftscala.PlaceType

import com.twitter.tweetypie.thriftscala.QuotedTweet

import com.twitter.tweetypie.thriftscala.QuotedTweetDeleteEvent

import com.twitter.tweetypie.thriftscala.QuotedTweetTakedownEvent

import com.twitter.tweetypie.thriftscala.Reply

import com.twitter.tweetypie.thriftscala.Share

import com.twitter.tweetypie.thriftscala.Tweet

import com.twitter.tweetypie.thriftscala.TweetCoreData

import com.twitter.tweetypie.thriftscala.TweetCreateEvent

import com.twitter.tweetypie.thriftscala.TweetDeleteEvent

import com.twitter.tweetypie.thriftscala.TweetEvent

import com.twitter.tweetypie.thriftscala.TweetEventData

import com.twitter.tweetypie.thriftscala.TweetEventFlags

import com.twitter.tweetypie.thriftscala.TweetPossiblySensitiveUpdateEvent

import com.twitter.tweetypie.thriftscala.TweetScrubGeoEvent

import com.twitter.tweetypie.thriftscala.TweetTakedownEvent

import com.twitter.tweetypie.thriftscala.TweetUndeleteEvent

import com.twitter.tweetypie.thriftscala.UserScrubGeoEvent

import com.twitter.unified\_user\_actions.adapter.tweetypie\_event.TweetypieEventAdapter

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

import com.twitter.util.Time

import org.scalatest.prop.TableDrivenPropertyChecks

import org.scalatest.prop.TableFor1

import org.scalatest.prop.TableFor2

import org.scalatest.prop.TableFor3

class TweetypieEventAdapterSpec extends Test with TableDrivenPropertyChecks {

trait Fixture {

val frozenTime: Time = Time.fromMilliseconds(1658949273000L)

val tweetDeleteEventTime: Time = Time.fromMilliseconds(1658949253000L)

val tweetId = 1554576940856246272L

val timestamp: Long = SnowflakeId.unixTimeMillisFromId(tweetId)

val userId = 1L

val user: User = User(

id = userId,

createdAtMsec = 1000L,

updatedAtMsec = 1000L,

userType = UserType.Normal,

)

val actionedTweetId = 1554576940756246333L

val actionedTweetTimestamp: Long = SnowflakeId.unixTimeMillisFromId(actionedTweetId)

val actionedTweetAuthorId = 2L

val actionedByActionedTweetId = 1554566940756246272L

val actionedByActionedTweetTimestamp: Long =

SnowflakeId.unixTimeMillisFromId(actionedByActionedTweetId)

val actionedByActionedTweetAuthorId = 3L

val tweetEventFlags: TweetEventFlags = TweetEventFlags(timestampMs = timestamp)

val language: Option[Language] = Some(Language("EN-US", false))

val deviceSource: Option[DeviceSource] = Some(

DeviceSource(

id = 0,

parameter = "",

internalName = "",

name = "name",

url = "url",

display = "display",

clientAppId = Option(100L)))

val place: Option[Place] = Some(

Place(

id = "id",

`type` = PlaceType.City,

fullName = "San Francisco",

name = "SF",

countryCode = Some("US"),

))

// for TweetDeleteEvent

val auditDeleteTweet = Some(

AuditDeleteTweet(

clientApplicationId = Option(200L)

))

val tweetCoreData: TweetCoreData =

TweetCoreData(userId, text = "text", createdVia = "created\_via", createdAtSecs = timestamp)

val baseTweet: Tweet = Tweet(

tweetId,

coreData = Some(tweetCoreData),

language = language,

deviceSource = deviceSource,

place = place)

def getCreateTweetCoreData(userId: Long, timestamp: Long): TweetCoreData =

tweetCoreData.copy(userId = userId, createdAtSecs = timestamp)

def getRetweetTweetCoreData(

userId: Long,

retweetedTweetId: Long,

retweetedAuthorId: Long,

parentStatusId: Long,

timestamp: Long

): TweetCoreData = tweetCoreData.copy(

userId = userId,

share = Some(

Share(

sourceStatusId = retweetedTweetId,

sourceUserId = retweetedAuthorId,

parentStatusId = parentStatusId

)),

createdAtSecs = timestamp

)

def getReplyTweetCoreData(

userId: Long,

repliedTweetId: Long,

repliedAuthorId: Long,

timestamp: Long

): TweetCoreData = tweetCoreData.copy(

userId = userId,

reply = Some(

Reply(

inReplyToStatusId = Some(repliedTweetId),

inReplyToUserId = repliedAuthorId,

)

),

createdAtSecs = timestamp)

def getQuoteTweetCoreData(userId: Long, timestamp: Long): TweetCoreData =

tweetCoreData.copy(userId = userId, createdAtSecs = timestamp)

def getTweet(tweetId: Long, userId: Long, timestamp: Long): Tweet =

baseTweet.copy(id = tweetId, coreData = Some(getCreateTweetCoreData(userId, timestamp)))

def getRetweet(

tweetId: Long,

userId: Long,

timestamp: Long,

retweetedTweetId: Long,

retweetedUserId: Long,

parentStatusId: Option[Long] = None

): Tweet =

baseTweet.copy(

id = tweetId,

coreData = Some(

getRetweetTweetCoreData(

userId,

retweetedTweetId,

retweetedUserId,

parentStatusId.getOrElse(retweetedTweetId),

timestamp)))

def getQuote(

tweetId: Long,

userId: Long,

timestamp: Long,

quotedTweetId: Long,

quotedUserId: Long

): Tweet =

baseTweet.copy(

id = tweetId,

coreData = Some(getQuoteTweetCoreData(userId, timestamp)),

quotedTweet = Some(QuotedTweet(quotedTweetId, quotedUserId)))

def getReply(

tweetId: Long,

userId: Long,

repliedTweetId: Long,

repliedAuthorId: Long,

timestamp: Long

): Tweet =

baseTweet.copy(

id = tweetId,

coreData = Some(getReplyTweetCoreData(userId, repliedTweetId, repliedAuthorId, timestamp)),

)

// ignored tweet events

val additionalFieldUpdateEvent: TweetEvent = TweetEvent(

TweetEventData.AdditionalFieldUpdateEvent(AdditionalFieldUpdateEvent(baseTweet)),

tweetEventFlags)

val additionalFieldDeleteEvent: TweetEvent = TweetEvent(

TweetEventData.AdditionalFieldDeleteEvent(

AdditionalFieldDeleteEvent(Map(tweetId -> Seq.empty))

),

tweetEventFlags

)

val tweetUndeleteEvent: TweetEvent = TweetEvent(

TweetEventData.TweetUndeleteEvent(TweetUndeleteEvent(baseTweet)),

tweetEventFlags

)

val tweetScrubGeoEvent: TweetEvent = TweetEvent(

TweetEventData.TweetScrubGeoEvent(TweetScrubGeoEvent(tweetId, userId)),

tweetEventFlags)

val tweetTakedownEvent: TweetEvent = TweetEvent(

TweetEventData.TweetTakedownEvent(TweetTakedownEvent(tweetId, userId)),

tweetEventFlags

)

val userScrubGeoEvent: TweetEvent = TweetEvent(

TweetEventData.UserScrubGeoEvent(UserScrubGeoEvent(userId = userId, maxTweetId = tweetId)),

tweetEventFlags

)

val tweetPossiblySensitiveUpdateEvent: TweetEvent = TweetEvent(

TweetEventData.TweetPossiblySensitiveUpdateEvent(

TweetPossiblySensitiveUpdateEvent(

tweetId = tweetId,

userId = userId,

nsfwAdmin = false,

nsfwUser = false)),

tweetEventFlags

)

val quotedTweetDeleteEvent: TweetEvent = TweetEvent(

TweetEventData.QuotedTweetDeleteEvent(

QuotedTweetDeleteEvent(

quotingTweetId = tweetId,

quotingUserId = userId,

quotedTweetId = tweetId,

quotedUserId = userId)),

tweetEventFlags

)

val quotedTweetTakedownEvent: TweetEvent = TweetEvent(

TweetEventData.QuotedTweetTakedownEvent(

QuotedTweetTakedownEvent(

quotingTweetId = tweetId,

quotingUserId = userId,

quotedTweetId = tweetId,

quotedUserId = userId,

takedownCountryCodes = Seq.empty,

takedownReasons = Seq.empty

)

),

tweetEventFlags

)

val replyOnlyTweet =

getReply(tweetId, userId, actionedTweetId, actionedTweetAuthorId, timestamp)

val replyAndRetweetTweet = replyOnlyTweet.copy(coreData = replyOnlyTweet.coreData.map(

\_.copy(share = Some(

Share(

sourceStatusId = actionedTweetId,

sourceUserId = actionedTweetAuthorId,

parentStatusId = actionedTweetId

)))))

val replyRetweetPresentEvent: TweetEvent = TweetEvent(

TweetEventData.TweetCreateEvent(

TweetCreateEvent(

tweet = replyAndRetweetTweet,

user = user,

sourceTweet =

Some(getTweet(actionedTweetId, actionedTweetAuthorId, actionedTweetTimestamp))

)),

tweetEventFlags

)

def getExpectedUUA(

userId: Long,

actionTweetId: Long,

actionTweetAuthorId: Long,

sourceTimestampMs: Long,

actionType: ActionType,

replyingTweetId: Option[Long] = None,

quotingTweetId: Option[Long] = None,

retweetingTweetId: Option[Long] = None,

inReplyToTweetId: Option[Long] = None,

quotedTweetId: Option[Long] = None,

retweetedTweetId: Option[Long] = None,

editedTweetId: Option[Long] = None,

appId: Option[Long] = None,

): UnifiedUserAction = UnifiedUserAction(

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

item = Item.TweetInfo(

TweetInfo(

actionTweetId = actionTweetId,

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

replyingTweetId = replyingTweetId,

quotingTweetId = quotingTweetId,

retweetingTweetId = retweetingTweetId,

inReplyToTweetId = inReplyToTweetId,

quotedTweetId = quotedTweetId,

retweetedTweetId = retweetedTweetId,

editedTweetId = editedTweetId

)

),

actionType = actionType,

eventMetadata = EventMetadata(

sourceTimestampMs = sourceTimestampMs,

receivedTimestampMs = frozenTime.inMilliseconds,

sourceLineage = SourceLineage.ServerTweetypieEvents,

language = None,

countryCode = Some("US"),

clientAppId = appId,

)

)

/\* Note: This is a deprecated field {ActionTweetType}.

\* We keep this here to document the behaviors of each unit test.

/\*

\* Types of tweets on which actions can take place.

\* Note that retweets are not included because actions can NOT take place

\* on retweets. They can only take place on source tweets of retweets,

\* which are one of the ActionTweetTypes listed below.

\*/

enum ActionTweetType {

/\* Is a standard (non-retweet, non-reply, non-quote) tweet \*/

Default = 0

/\*

\* Is a tweet in a reply chain (this includes tweets

\* without a leading @mention, as long as they are in reply

\* to some tweet id)

\*/

Reply = 1

/\* Is a retweet with comment \*/

Quote = 2

}(persisted='true', hasPersonalData='false')

\*/

// tweet create

val tweetCreateEvent: TweetEvent = TweetEvent(

TweetEventData.TweetCreateEvent(

TweetCreateEvent(

tweet = getTweet(tweetId, userId, timestamp),

user = user,

)

),

tweetEventFlags)

val expectedUUACreate = getExpectedUUA(

userId = userId,

actionTweetId = tweetId,

/\* @see comment above for ActionTweetType

actionTweetType = Some(ActionTweetType.Default),

\*/

actionTweetAuthorId = userId,

sourceTimestampMs = timestamp,

actionType = ActionType.ServerTweetCreate,

appId = deviceSource.flatMap(\_.clientAppId)

)

// tweet reply to a default

val tweetReplyDefaultEvent: TweetEvent = TweetEvent(

TweetEventData.TweetCreateEvent(

TweetCreateEvent(

tweet = getReply(tweetId, userId, actionedTweetId, actionedTweetAuthorId, timestamp),

user = user

)

),

tweetEventFlags

)

val expectedUUAReplyDefault = getExpectedUUA(

userId = userId,

actionTweetId = actionedTweetId,

/\* @see comment above for ActionTweetType

actionTweetType = None,

\*/

actionTweetAuthorId = actionedTweetAuthorId,

sourceTimestampMs = timestamp,

actionType = ActionType.ServerTweetReply,

replyingTweetId = Some(tweetId),

appId = deviceSource.flatMap(\_.clientAppId)

)

// tweet reply to a reply

val tweetReplyToReplyEvent: TweetEvent = TweetEvent(

TweetEventData.TweetCreateEvent(

TweetCreateEvent(

tweet = getReply(tweetId, userId, actionedTweetId, actionedTweetAuthorId, timestamp),

user = user

)

),

tweetEventFlags

)

// tweet reply to a quote

val tweetReplyToQuoteEvent: TweetEvent = TweetEvent(

TweetEventData.TweetCreateEvent(

TweetCreateEvent(

tweet = getReply(tweetId, userId, actionedTweetId, actionedTweetAuthorId, timestamp),

user = user

)

),

tweetEventFlags

)

// tweet quote a default

val tweetQuoteDefaultEvent: TweetEvent = TweetEvent(

TweetEventData.TweetCreateEvent(

TweetCreateEvent(

tweet = getQuote(tweetId, userId, timestamp, actionedTweetId, actionedTweetAuthorId),

user = user,

quotedTweet =

Some(getTweet(actionedTweetId, actionedTweetAuthorId, actionedTweetTimestamp))

)

),

tweetEventFlags

)

val expectedUUAQuoteDefault: UnifiedUserAction = getExpectedUUA(

userId = userId,

actionTweetId = actionedTweetId,

/\* @see comment above for ActionTweetType

actionTweetType = Some(ActionTweetType.Default),

\*/

actionTweetAuthorId = actionedTweetAuthorId,

sourceTimestampMs = timestamp,

actionType = ActionType.ServerTweetQuote,

quotingTweetId = Some(tweetId),

appId = deviceSource.flatMap(\_.clientAppId)

)

// tweet quote a reply

val tweetQuoteReplyEvent: TweetEvent = TweetEvent(

TweetEventData.TweetCreateEvent(

TweetCreateEvent(

tweet = getQuote(tweetId, userId, timestamp, actionedTweetId, actionedTweetAuthorId),

user = user,

quotedTweet = Some(

getReply(

tweetId = actionedTweetId,

userId = actionedTweetAuthorId,

repliedTweetId = actionedByActionedTweetId,

repliedAuthorId = actionedByActionedTweetAuthorId,

timestamp = actionedTweetTimestamp

))

)

),

tweetEventFlags

)

val expectedUUAQuoteReply: UnifiedUserAction = getExpectedUUA(

userId = userId,

actionTweetId = actionedTweetId,

/\* @see comment above for ActionTweetType

actionTweetType = Some(ActionTweetType.Reply),

\*/

actionTweetAuthorId = actionedTweetAuthorId,

sourceTimestampMs = timestamp,

actionType = ActionType.ServerTweetQuote,

quotingTweetId = Some(tweetId),

appId = deviceSource.flatMap(\_.clientAppId)

)

// tweet quote a quote

val tweetQuoteQuoteEvent: TweetEvent = TweetEvent(

TweetEventData.TweetCreateEvent(

TweetCreateEvent(

tweet = getQuote(tweetId, userId, timestamp, actionedTweetId, actionedTweetAuthorId),

user = user,

quotedTweet = Some(

getQuote(

tweetId = actionedTweetId,

userId = actionedTweetAuthorId,

timestamp = actionedTweetTimestamp,

quotedTweetId = actionedByActionedTweetId,

quotedUserId = actionedByActionedTweetAuthorId,

))

)

),

tweetEventFlags

)

val expectedUUAQuoteQuote: UnifiedUserAction = getExpectedUUA(

userId = userId,

actionTweetId = actionedTweetId,

/\* @see comment above for ActionTweetType

actionTweetType = Some(ActionTweetType.Quote),

\*/

actionTweetAuthorId = actionedTweetAuthorId,

sourceTimestampMs = timestamp,

actionType = ActionType.ServerTweetQuote,

quotingTweetId = Some(tweetId),

appId = deviceSource.flatMap(\_.clientAppId)

)

// tweet retweet a default

val tweetRetweetDefaultEvent: TweetEvent = TweetEvent(

TweetEventData.TweetCreateEvent(

TweetCreateEvent(

tweet = getRetweet(tweetId, userId, timestamp, actionedTweetId, actionedTweetAuthorId),

user = user,

sourceTweet =

Some(getTweet(actionedTweetId, actionedTweetAuthorId, actionedTweetTimestamp))

)

),

tweetEventFlags

)

val expectedUUARetweetDefault: UnifiedUserAction = getExpectedUUA(

userId = userId,

actionTweetId = actionedTweetId,

/\* @see comment above for ActionTweetType

actionTweetType = Some(ActionTweetType.Default),

\*/

actionTweetAuthorId = actionedTweetAuthorId,

sourceTimestampMs = timestamp,

actionType = ActionType.ServerTweetRetweet,

retweetingTweetId = Some(tweetId),

appId = deviceSource.flatMap(\_.clientAppId)

)

// tweet retweet a reply

val tweetRetweetReplyEvent: TweetEvent = TweetEvent(

TweetEventData.TweetCreateEvent(

TweetCreateEvent(

tweet = getRetweet(tweetId, userId, timestamp, actionedTweetId, actionedTweetAuthorId),

user = user,

sourceTweet = Some(

getReply(

actionedTweetId,

actionedTweetAuthorId,

actionedByActionedTweetId,

actionedByActionedTweetAuthorId,

actionedTweetTimestamp))

)

),

tweetEventFlags

)

val expectedUUARetweetReply: UnifiedUserAction = getExpectedUUA(

userId = userId,

actionTweetId = actionedTweetId,

/\* @see comment above for ActionTweetType

actionTweetType = Some(ActionTweetType.Reply),

\*/

actionTweetAuthorId = actionedTweetAuthorId,

sourceTimestampMs = timestamp,

actionType = ActionType.ServerTweetRetweet,

retweetingTweetId = Some(tweetId),

appId = deviceSource.flatMap(\_.clientAppId)

)

// tweet retweet a quote

val tweetRetweetQuoteEvent: TweetEvent = TweetEvent(

TweetEventData.TweetCreateEvent(

TweetCreateEvent(

tweet = getRetweet(tweetId, userId, timestamp, actionedTweetId, actionedTweetAuthorId),

user = user,

sourceTweet = Some(

getQuote(

actionedTweetId,

actionedTweetAuthorId,

actionedTweetTimestamp,

actionedByActionedTweetId,

actionedByActionedTweetAuthorId

))

)

),

tweetEventFlags

)

val expectedUUARetweetQuote: UnifiedUserAction = getExpectedUUA(

userId = userId,

actionTweetId = actionedTweetId,

/\* @see comment above for ActionTweetType

actionTweetType = Some(ActionTweetType.Quote),

\*/

actionTweetAuthorId = actionedTweetAuthorId,

sourceTimestampMs = timestamp,

actionType = ActionType.ServerTweetRetweet,

retweetingTweetId = Some(tweetId),

appId = deviceSource.flatMap(\_.clientAppId)

)

// tweet retweet a retweet

val tweetRetweetRetweetEvent: TweetEvent = TweetEvent(

TweetEventData.TweetCreateEvent(

TweetCreateEvent(

tweet = getRetweet(

tweetId,

userId,

timestamp,

actionedByActionedTweetId,

actionedByActionedTweetAuthorId,

Some(actionedTweetId)),

user = user,

sourceTweet = Some(

getTweet(

actionedByActionedTweetId,

actionedByActionedTweetAuthorId,

actionedByActionedTweetTimestamp,

))

)

),

tweetEventFlags

)

val expectedUUARetweetRetweet: UnifiedUserAction = getExpectedUUA(

userId = userId,

actionTweetId = actionedByActionedTweetId,

/\* @see comment above for ActionTweetType

actionTweetType = Some(ActionTweetType.Default),

\*/

actionTweetAuthorId = actionedByActionedTweetAuthorId,

sourceTimestampMs = timestamp,

actionType = ActionType.ServerTweetRetweet,

retweetingTweetId = Some(tweetId),

appId = deviceSource.flatMap(\_.clientAppId)

)

// delete a tweet

val tweetDeleteEvent: TweetEvent = TweetEvent(

TweetEventData.TweetDeleteEvent(

TweetDeleteEvent(

tweet = getTweet(tweetId, userId, timestamp),

user = Some(user),

audit = auditDeleteTweet

)

),

tweetEventFlags.copy(timestampMs = tweetDeleteEventTime.inMilliseconds)

)

val expectedUUADeleteDefault: UnifiedUserAction = getExpectedUUA(

userId = user.id,

actionTweetId = tweetId,

actionTweetAuthorId = userId,

sourceTimestampMs = tweetDeleteEventTime.inMilliseconds,

actionType = ActionType.ServerTweetDelete,

appId = auditDeleteTweet.flatMap(\_.clientApplicationId)

)

// delete a reply - Unreply

val tweetUnreplyEvent: TweetEvent = TweetEvent(

TweetEventData.TweetDeleteEvent(

TweetDeleteEvent(

tweet = getReply(tweetId, userId, actionedTweetId, actionedTweetAuthorId, timestamp),

user = Some(user),

audit = auditDeleteTweet

)

),

tweetEventFlags.copy(timestampMs = tweetDeleteEventTime.inMilliseconds)

)

val expectedUUAUnreply: UnifiedUserAction = getExpectedUUA(

userId = user.id,

actionTweetId = actionedTweetId,

actionTweetAuthorId = actionedTweetAuthorId,

sourceTimestampMs = tweetDeleteEventTime.inMilliseconds,

actionType = ActionType.ServerTweetUnreply,

replyingTweetId = Some(tweetId),

appId = auditDeleteTweet.flatMap(\_.clientApplicationId)

)

// delete a quote - Unquote

val tweetUnquoteEvent: TweetEvent = TweetEvent(

TweetEventData.TweetDeleteEvent(

TweetDeleteEvent(

tweet = getQuote(tweetId, userId, timestamp, actionedTweetId, actionedTweetAuthorId),

user = Some(user),

audit = auditDeleteTweet

)

),

tweetEventFlags.copy(timestampMs = tweetDeleteEventTime.inMilliseconds)

)

val expectedUUAUnquote: UnifiedUserAction = getExpectedUUA(

userId = user.id,

actionTweetId = actionedTweetId,

actionTweetAuthorId = actionedTweetAuthorId,

sourceTimestampMs = tweetDeleteEventTime.inMilliseconds,

actionType = ActionType.ServerTweetUnquote,

quotingTweetId = Some(tweetId),

appId = auditDeleteTweet.flatMap(\_.clientApplicationId)

)

// delete a retweet / unretweet

val tweetUnretweetEvent: TweetEvent = TweetEvent(

TweetEventData.TweetDeleteEvent(

TweetDeleteEvent(

tweet = getRetweet(

tweetId,

userId,

timestamp,

actionedTweetId,

actionedTweetAuthorId,

Some(actionedTweetId)),

user = Some(user),

audit = auditDeleteTweet

)

),

tweetEventFlags.copy(timestampMs = tweetDeleteEventTime.inMilliseconds)

)

val expectedUUAUnretweet: UnifiedUserAction = getExpectedUUA(

userId = user.id,

actionTweetId = actionedTweetId,

actionTweetAuthorId = actionedTweetAuthorId,

sourceTimestampMs = tweetDeleteEventTime.inMilliseconds,

actionType = ActionType.ServerTweetUnretweet,

retweetingTweetId = Some(tweetId),

appId = auditDeleteTweet.flatMap(\_.clientApplicationId)

)

// edit a tweet, the new tweet from edit is a default tweet (not reply/quote/retweet)

val regularTweetFromEditEvent: TweetEvent = TweetEvent(

TweetEventData.TweetCreateEvent(

TweetCreateEvent(

tweet = getTweet(

tweetId,

userId,

timestamp

).copy(editControl =

Some(EditControl.Edit(EditControlEdit(initialTweetId = actionedTweetId)))),

user = user,

)

),

tweetEventFlags

)

val expectedUUARegularTweetFromEdit: UnifiedUserAction = getExpectedUUA(

userId = user.id,

actionTweetId = tweetId,

actionTweetAuthorId = userId,

sourceTimestampMs = tweetEventFlags.timestampMs,

actionType = ActionType.ServerTweetEdit,

editedTweetId = Some(actionedTweetId),

appId = deviceSource.flatMap(\_.clientAppId)

)

// edit a tweet, the new tweet from edit is a Quote

val quoteFromEditEvent: TweetEvent = TweetEvent(

TweetEventData.TweetCreateEvent(

TweetCreateEvent(

tweet = getQuote(

tweetId,

userId,

timestamp,

actionedTweetId,

actionedTweetAuthorId

).copy(editControl =

Some(EditControl.Edit(EditControlEdit(initialTweetId = actionedByActionedTweetId)))),

user = user,

)

),

tweetEventFlags

)

val expectedUUAQuoteFromEdit: UnifiedUserAction = getExpectedUUA(

userId = user.id,

actionTweetId = tweetId,

actionTweetAuthorId = userId,

sourceTimestampMs = tweetEventFlags.timestampMs,

actionType = ActionType.ServerTweetEdit,

editedTweetId = Some(actionedByActionedTweetId),

quotedTweetId = Some(actionedTweetId),

appId = deviceSource.flatMap(\_.clientAppId)

)

}

test("ignore non-TweetCreate / non-TweetDelete events") {

new Fixture {

val ignoredTweetEvents: TableFor1[TweetEvent] = Table(

"ignoredTweetEvents",

additionalFieldUpdateEvent,

additionalFieldDeleteEvent,

tweetUndeleteEvent,

tweetScrubGeoEvent,

tweetTakedownEvent,

userScrubGeoEvent,

tweetPossiblySensitiveUpdateEvent,

quotedTweetDeleteEvent,

quotedTweetTakedownEvent

)

forEvery(ignoredTweetEvents) { tweetEvent: TweetEvent =>

val actual = TweetypieEventAdapter.adaptEvent(tweetEvent)

assert(actual.isEmpty)

}

}

}

test("ignore invalid TweetCreate events") {

new Fixture {

val ignoredTweetEvents: TableFor2[String, TweetEvent] = Table(

("invalidType", "event"),

("replyAndRetweetBothPresent", replyRetweetPresentEvent)

)

forEvery(ignoredTweetEvents) { (\_, event) =>

val actual = TweetypieEventAdapter.adaptEvent(event)

assert(actual.isEmpty)

}

}

}

test("TweetypieCreateEvent") {

new Fixture {

Time.withTimeAt(frozenTime) { \_ =>

val actual = TweetypieEventAdapter.adaptEvent(tweetCreateEvent)

assert(Seq(expectedUUACreate) == actual)

}

}

}

test("TweetypieReplyEvent") {

new Fixture {

Time.withTimeAt(frozenTime) { \_ =>

val tweetReplies: TableFor3[String, TweetEvent, UnifiedUserAction] = Table(

("actionTweetType", "event", "expected"),

("Default", tweetReplyDefaultEvent, expectedUUAReplyDefault),

("Reply", tweetReplyToReplyEvent, expectedUUAReplyDefault),

("Quote", tweetReplyToQuoteEvent, expectedUUAReplyDefault),

)

forEvery(tweetReplies) { (\_: String, event: TweetEvent, expected: UnifiedUserAction) =>

val actual = TweetypieEventAdapter.adaptEvent(event)

assert(Seq(expected) === actual)

}

}

}

}

test("TweetypieQuoteEvent") {

new Fixture {

Time.withTimeAt(frozenTime) { \_ =>

val tweetQuotes: TableFor3[String, TweetEvent, UnifiedUserAction] = Table(

("actionTweetType", "event", "expected"),

("Default", tweetQuoteDefaultEvent, expectedUUAQuoteDefault),

("Reply", tweetQuoteReplyEvent, expectedUUAQuoteReply),

("Quote", tweetQuoteQuoteEvent, expectedUUAQuoteQuote),

)

forEvery(tweetQuotes) { (\_: String, event: TweetEvent, expected: UnifiedUserAction) =>

val actual = TweetypieEventAdapter.adaptEvent(event)

assert(Seq(expected) === actual)

}

}

}

}

test("TweetypieRetweetEvent") {

new Fixture {

Time.withTimeAt(frozenTime) { \_ =>

val tweetRetweets: TableFor3[String, TweetEvent, UnifiedUserAction] = Table(

("actionTweetType", "event", "expected"),

("Default", tweetRetweetDefaultEvent, expectedUUARetweetDefault),

("Reply", tweetRetweetReplyEvent, expectedUUARetweetReply),

("Quote", tweetRetweetQuoteEvent, expectedUUARetweetQuote),

("Retweet", tweetRetweetRetweetEvent, expectedUUARetweetRetweet),

)

forEvery(tweetRetweets) { (\_: String, event: TweetEvent, expected: UnifiedUserAction) =>

val actual = TweetypieEventAdapter.adaptEvent(event)

assert(Seq(expected) === actual)

}

}

}

}

test("TweetypieDeleteEvent") {

new Fixture {

Time.withTimeAt(frozenTime) { \_ =>

val tweetDeletes: TableFor3[String, TweetEvent, UnifiedUserAction] = Table(

("actionTweetType", "event", "expected"),

("Default", tweetDeleteEvent, expectedUUADeleteDefault),

("Reply", tweetUnreplyEvent, expectedUUAUnreply),

("Quote", tweetUnquoteEvent, expectedUUAUnquote),

("Retweet", tweetUnretweetEvent, expectedUUAUnretweet),

)

forEvery(tweetDeletes) { (\_: String, event: TweetEvent, expected: UnifiedUserAction) =>

val actual = TweetypieEventAdapter.adaptEvent(event)

assert(Seq(expected) === actual)

}

}

}

}

test("TweetypieEditEvent") {

new Fixture {

Time.withTimeAt(frozenTime) { \_ =>

val tweetEdits: TableFor3[String, TweetEvent, UnifiedUserAction] = Table(

("actionTweetType", "event", "expected"),

("RegularTweetFromEdit", regularTweetFromEditEvent, expectedUUARegularTweetFromEdit),

("QuoteFromEdit", quoteFromEditEvent, expectedUUAQuoteFromEdit)

)

forEvery(tweetEdits) { (\_: String, event: TweetEvent, expected: UnifiedUserAction) =>

val actual = TweetypieEventAdapter.adaptEvent(event)

assert(Seq(expected) === actual)

}

}

}

}

}