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

import com.twitter.ads.spendserver.thriftscala.SpendServerEvent

import com.twitter.adserver.thriftscala.EngagementType

import com.twitter.clientapp.thriftscala.AmplifyDetails

import com.twitter.inject.Test

import com.twitter.unified\_user\_actions.adapter.TestFixtures.AdsCallbackEngagementsFixture

import com.twitter.unified\_user\_actions.adapter.ads\_callback\_engagements.AdsCallbackEngagementsAdapter

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

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

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

import com.twitter.util.Time

import org.scalatest.prop.TableDrivenPropertyChecks

class AdsCallbackEngagementsAdapterSpec extends Test with TableDrivenPropertyChecks {

test("Test basic conversion for ads callback engagement type fav") {

new AdsCallbackEngagementsFixture {

Time.withTimeAt(frozenTime) { \_ =>

val events = Table(

("inputEvent", "expectedUuaOutput"),

( // Test with authorId

createSpendServerEvent(EngagementType.Fav),

Seq(

createExpectedUua(

ActionType.ServerPromotedTweetFav,

createTweetInfoItem(authorInfo = Some(authorInfo)))))

)

forEvery(events) { (event: SpendServerEvent, expected: Seq[UnifiedUserAction]) =>

val actual = AdsCallbackEngagementsAdapter.adaptEvent(event)

assert(expected === actual)

}

}

}

}

test("Test basic conversion for different engagement types") {

new AdsCallbackEngagementsFixture {

Time.withTimeAt(frozenTime) { \_ =>

val mappings = Table(

("engagementType", "actionType"),

(EngagementType.Unfav, ActionType.ServerPromotedTweetUnfav),

(EngagementType.Reply, ActionType.ServerPromotedTweetReply),

(EngagementType.Retweet, ActionType.ServerPromotedTweetRetweet),

(EngagementType.Block, ActionType.ServerPromotedTweetBlockAuthor),

(EngagementType.Unblock, ActionType.ServerPromotedTweetUnblockAuthor),

(EngagementType.Send, ActionType.ServerPromotedTweetComposeTweet),

(EngagementType.Detail, ActionType.ServerPromotedTweetClick),

(EngagementType.Report, ActionType.ServerPromotedTweetReport),

(EngagementType.Mute, ActionType.ServerPromotedTweetMuteAuthor),

(EngagementType.ProfilePic, ActionType.ServerPromotedTweetClickProfile),

(EngagementType.ScreenName, ActionType.ServerPromotedTweetClickProfile),

(EngagementType.UserName, ActionType.ServerPromotedTweetClickProfile),

(EngagementType.Hashtag, ActionType.ServerPromotedTweetClickHashtag),

(EngagementType.CarouselSwipeNext, ActionType.ServerPromotedTweetCarouselSwipeNext),

(

EngagementType.CarouselSwipePrevious,

ActionType.ServerPromotedTweetCarouselSwipePrevious),

(EngagementType.DwellShort, ActionType.ServerPromotedTweetLingerImpressionShort),

(EngagementType.DwellMedium, ActionType.ServerPromotedTweetLingerImpressionMedium),

(EngagementType.DwellLong, ActionType.ServerPromotedTweetLingerImpressionLong),

(EngagementType.DismissSpam, ActionType.ServerPromotedTweetDismissSpam),

(EngagementType.DismissWithoutReason, ActionType.ServerPromotedTweetDismissWithoutReason),

(EngagementType.DismissUninteresting, ActionType.ServerPromotedTweetDismissUninteresting),

(EngagementType.DismissRepetitive, ActionType.ServerPromotedTweetDismissRepetitive),

)

forEvery(mappings) { (engagementType: EngagementType, actionType: ActionType) =>

val event = createSpendServerEvent(engagementType)

val actual = AdsCallbackEngagementsAdapter.adaptEvent(event)

val expected =

Seq(createExpectedUua(actionType, createTweetInfoItem(authorInfo = Some(authorInfo))))

assert(expected === actual)

}

}

}

}

test("Test conversion for ads callback engagement type spotlight view and click") {

new AdsCallbackEngagementsFixture {

Time.withTimeAt(frozenTime) { \_ =>

val input = Table(

("adsEngagement", "uuaAction"),

(EngagementType.SpotlightClick, ActionType.ServerPromotedTweetClickSpotlight),

(EngagementType.SpotlightView, ActionType.ServerPromotedTweetViewSpotlight),

(EngagementType.TrendView, ActionType.ServerPromotedTrendView),

(EngagementType.TrendClick, ActionType.ServerPromotedTrendClick),

)

forEvery(input) { (engagementType: EngagementType, actionType: ActionType) =>

val adsEvent = createSpendServerEvent(engagementType)

val expected = Seq(createExpectedUua(actionType, trendInfoItem))

val actual = AdsCallbackEngagementsAdapter.adaptEvent(adsEvent)

assert(expected === actual)

}

}

}

}

test("Test basic conversion for ads callback engagement open link with or without url") {

new AdsCallbackEngagementsFixture {

Time.withTimeAt(frozenTime) { \_ =>

val input = Table(

("url", "tweetActionInfo"),

(Some("go/url"), openLinkWithUrl),

(None, openLinkWithoutUrl)

)

forEvery(input) { (url: Option[String], tweetActionInfo: TweetActionInfo) =>

val event = createSpendServerEvent(engagementType = EngagementType.Url, url = url)

val actual = AdsCallbackEngagementsAdapter.adaptEvent(event)

val expected = Seq(createExpectedUua(

ActionType.ServerPromotedTweetOpenLink,

createTweetInfoItem(authorInfo = Some(authorInfo), actionInfo = Some(tweetActionInfo))))

assert(expected === actual)

}

}

}

}

test("Test basic conversion for different engagement types with profile info") {

new AdsCallbackEngagementsFixture {

Time.withTimeAt(frozenTime) { \_ =>

val mappings = Table(

("engagementType", "actionType"),

(EngagementType.Follow, ActionType.ServerPromotedProfileFollow),

(EngagementType.Unfollow, ActionType.ServerPromotedProfileUnfollow)

)

forEvery(mappings) { (engagementType: EngagementType, actionType: ActionType) =>

val event = createSpendServerEvent(engagementType)

val actual = AdsCallbackEngagementsAdapter.adaptEvent(event)

val expected = Seq(createExpectedUuaWithProfileInfo(actionType))

assert(expected === actual)

}

}

}

}

test("Test basic conversion for ads callback engagement type video\_content\_\*") {

new AdsCallbackEngagementsFixture {

Time.withTimeAt(frozenTime) { \_ =>

val events = Table(

("engagementType", "amplifyDetails", "actionType", "tweetActionInfo"),

//For video\_content\_\* events on promoted tweets when there is no preroll ad played

(

EngagementType.VideoContentPlayback25,

amplifyDetailsPromotedTweetWithoutAd,

ActionType.ServerPromotedTweetVideoPlayback25,

tweetActionInfoPromotedTweetWithoutAd),

(

EngagementType.VideoContentPlayback50,

amplifyDetailsPromotedTweetWithoutAd,

ActionType.ServerPromotedTweetVideoPlayback50,

tweetActionInfoPromotedTweetWithoutAd),

(

EngagementType.VideoContentPlayback75,

amplifyDetailsPromotedTweetWithoutAd,

ActionType.ServerPromotedTweetVideoPlayback75,

tweetActionInfoPromotedTweetWithoutAd),

//For video\_content\_\* events on promoted tweets when there is a preroll ad

(

EngagementType.VideoContentPlayback25,

amplifyDetailsPromotedTweetWithAd,

ActionType.ServerPromotedTweetVideoPlayback25,

tweetActionInfoPromotedTweetWithAd),

(

EngagementType.VideoContentPlayback50,

amplifyDetailsPromotedTweetWithAd,

ActionType.ServerPromotedTweetVideoPlayback50,

tweetActionInfoPromotedTweetWithAd),

(

EngagementType.VideoContentPlayback75,

amplifyDetailsPromotedTweetWithAd,

ActionType.ServerPromotedTweetVideoPlayback75,

tweetActionInfoPromotedTweetWithAd),

)

forEvery(events) {

(

engagementType: EngagementType,

amplifyDetails: Option[AmplifyDetails],

actionType: ActionType,

actionInfo: Option[TweetActionInfo]

) =>

val spendEvent =

createVideoSpendServerEvent(engagementType, amplifyDetails, promotedTweetId, None)

val expected = Seq(createExpectedVideoUua(actionType, actionInfo, promotedTweetId))

val actual = AdsCallbackEngagementsAdapter.adaptEvent(spendEvent)

assert(expected === actual)

}

}

}

}

test("Test basic conversion for ads callback engagement type video\_ad\_\*") {

new AdsCallbackEngagementsFixture {

Time.withTimeAt(frozenTime) { \_ =>

val events = Table(

(

"engagementType",

"amplifyDetails",

"actionType",

"tweetActionInfo",

"promotedTweetId",

"organicTweetId"),

//For video\_ad\_\* events when the preroll ad is on a promoted tweet.

(

EngagementType.VideoAdPlayback25,

amplifyDetailsPrerollAd,

ActionType.ServerPromotedTweetVideoAdPlayback25,

tweetActionInfoPrerollAd,

promotedTweetId,

None

),

(

EngagementType.VideoAdPlayback50,

amplifyDetailsPrerollAd,

ActionType.ServerPromotedTweetVideoAdPlayback50,

tweetActionInfoPrerollAd,

promotedTweetId,

None

),

(

EngagementType.VideoAdPlayback75,

amplifyDetailsPrerollAd,

ActionType.ServerPromotedTweetVideoAdPlayback75,

tweetActionInfoPrerollAd,

promotedTweetId,

None

),

// For video\_ad\_\* events when the preroll ad is on an organic tweet.

(

EngagementType.VideoAdPlayback25,

amplifyDetailsPrerollAd,

ActionType.ServerTweetVideoAdPlayback25,

tweetActionInfoPrerollAd,

None,

organicTweetId

),

(

EngagementType.VideoAdPlayback50,

amplifyDetailsPrerollAd,

ActionType.ServerTweetVideoAdPlayback50,

tweetActionInfoPrerollAd,

None,

organicTweetId

),

(

EngagementType.VideoAdPlayback75,

amplifyDetailsPrerollAd,

ActionType.ServerTweetVideoAdPlayback75,

tweetActionInfoPrerollAd,

None,

organicTweetId

),

)

forEvery(events) {

(

engagementType: EngagementType,

amplifyDetails: Option[AmplifyDetails],

actionType: ActionType,

actionInfo: Option[TweetActionInfo],

promotedTweetId: Option[Long],

organicTweetId: Option[Long],

) =>

val spendEvent =

createVideoSpendServerEvent(

engagementType,

amplifyDetails,

promotedTweetId,

organicTweetId)

val actionTweetId = if (organicTweetId.isDefined) organicTweetId else promotedTweetId

val expected = Seq(createExpectedVideoUua(actionType, actionInfo, actionTweetId))

val actual = AdsCallbackEngagementsAdapter.adaptEvent(spendEvent)

assert(expected === actual)

}

}

}

}

}