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

import com.twitter.ads.cards.thriftscala.CardEvent

import com.twitter.ads.eventstream.thriftscala.EngagementEvent

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

import com.twitter.adserver.thriftscala.ImpressionDataNeededAtEngagementTime

import com.twitter.adserver.thriftscala.ClientInfo

import com.twitter.adserver.thriftscala.EngagementType

import com.twitter.adserver.thriftscala.DisplayLocation

import com.twitter.clientapp.thriftscala.AmplifyDetails

import com.twitter.clientapp.thriftscala.CardDetails

import com.twitter.clientapp.thriftscala.EventDetails

import com.twitter.clientapp.thriftscala.EventNamespace

import com.twitter.clientapp.thriftscala.ImpressionDetails

import com.twitter.clientapp.thriftscala.ItemType

import com.twitter.clientapp.thriftscala.LogEvent

import com.twitter.clientapp.thriftscala.MediaDetails

import com.twitter.clientapp.thriftscala.MediaDetailsV2

import com.twitter.clientapp.thriftscala.MediaType

import com.twitter.clientapp.thriftscala.NotificationDetails

import com.twitter.clientapp.thriftscala.NotificationTabDetails

import com.twitter.clientapp.thriftscala.PerformanceDetails

import com.twitter.clientapp.thriftscala.ReportDetails

import com.twitter.clientapp.thriftscala.SearchDetails

import com.twitter.clientapp.thriftscala.SuggestionDetails

import com.twitter.clientapp.thriftscala.{Item => LogEventItem}

import com.twitter.clientapp.thriftscala.{TweetDetails => LogEventTweetDetails}

import com.twitter.gizmoduck.thriftscala.UserModification

import com.twitter.gizmoduck.thriftscala.Profile

import com.twitter.gizmoduck.thriftscala.Auth

import com.twitter.gizmoduck.thriftscala.UpdateDiffItem

import com.twitter.gizmoduck.thriftscala.User

import com.twitter.gizmoduck.thriftscala.UserType

import com.twitter.ibis.thriftscala.NotificationScribe

import com.twitter.ibis.thriftscala.NotificationScribeType

import com.twitter.iesource.thriftscala.ClientEventContext

import com.twitter.iesource.thriftscala.TweetImpression

import com.twitter.iesource.thriftscala.ClientType

import com.twitter.iesource.thriftscala.ContextualEventNamespace

import com.twitter.iesource.thriftscala.EngagingContext

import com.twitter.iesource.thriftscala.EventSource

import com.twitter.iesource.thriftscala.InteractionDetails

import com.twitter.iesource.thriftscala.InteractionEvent

import com.twitter.iesource.thriftscala.InteractionType

import com.twitter.iesource.thriftscala.InteractionTargetType

import com.twitter.iesource.thriftscala.{UserIdentifier => UserIdentifierIE}

import com.twitter.logbase.thriftscala.ClientEventReceiver

import com.twitter.logbase.thriftscala.LogBase

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

import com.twitter.notificationservice.api.thriftscala.NotificationClientEventMetadata

import com.twitter.reportflow.thriftscala.ReportType

import com.twitter.suggests.controller\_data.home\_tweets.thriftscala.HomeTweetsControllerData

import com.twitter.suggests.controller\_data.home\_tweets.v1.thriftscala.{

HomeTweetsControllerData => HomeTweetsControllerDataV1

}

import com.twitter.suggests.controller\_data.thriftscala.ControllerData

import com.twitter.suggests.controller\_data.timelines\_topic.thriftscala.TimelinesTopicControllerData

import com.twitter.suggests.controller\_data.timelines\_topic.v1.thriftscala.{

TimelinesTopicControllerData => TimelinesTopicControllerDataV1

}

import com.twitter.suggests.controller\_data.v2.thriftscala.{ControllerData => ControllerDataV2}

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

import com.twitter.util.Time

import com.twitter.video.analytics.thriftscala.ClientMediaEvent

import com.twitter.video.analytics.thriftscala.SessionState

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

import com.twitter.suggests.controller\_data.search\_response.v1.thriftscala.{

SearchResponseControllerData => SearchResponseControllerDataV1

}

import com.twitter.suggests.controller\_data.search\_response.thriftscala.SearchResponseControllerData

import com.twitter.suggests.controller\_data.search\_response.request.thriftscala.RequestControllerData

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

object TestFixtures {

trait CommonFixture {

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

val userId: Long = 123L

val authorId: Long = 112233L

val itemTweetId: Long = 111L

val itemProfileId: Long = 123456L

val retweetingTweetId: Long = 222L

val quotedTweetId: Long = 333L

val quotedAuthorId: Long = 456L

val inReplyToTweetId: Long = 444L

val quotingTweetId: Long = 555L

val topicId: Long = 1234L

val traceId: Long = 5678L

val requestJoinId: Long = 91011L

val notificationId: String = "12345"

val tweetIds: Seq[Long] = Seq[Long](111, 222, 333)

val reportFlowId: String = "report-flow-id"

}

trait ClientEventFixture extends CommonFixture {

val timestamp = 1001L

val logBase: LogBase = LogBase(

ipAddress = "",

transactionId = "",

timestamp = 1002L,

driftAdjustedEventCreatedAtMs = Some(1001L),

userId = Some(userId),

clientEventReceiver = Some(ClientEventReceiver.CesHttp)

)

val logBase1: LogBase = LogBase(

ipAddress = "",

transactionId = "",

userId = Some(userId),

guestId = Some(2L),

guestIdMarketing = Some(2L),

timestamp = timestamp

)

def mkSearchResultControllerData(

queryOpt: Option[String],

querySourceOpt: Option[Int] = None,

traceId: Option[Long] = None,

requestJoinId: Option[Long] = None

): ControllerData = {

ControllerData.V2(

ControllerDataV2.SearchResponse(

SearchResponseControllerData.V1(

SearchResponseControllerDataV1(requestControllerData = Some(

RequestControllerData(

rawQuery = queryOpt,

querySource = querySourceOpt,

traceId = traceId,

requestJoinId = requestJoinId

)))

)))

}

val videoEventElementValues: Seq[String] =

Seq[String](

"gif\_player",

"periscope\_player",

"platform\_amplify\_card",

"video\_player",

"vine\_player")

val invalidVideoEventElementValues: Seq[String] =

Seq[String](

"dynamic\_video\_ads",

"live\_video\_player",

"platform\_forward\_card",

"video\_app\_card\_canvas",

"youtube\_player"

)

val clientMediaEvent: ClientMediaEvent = ClientMediaEvent(

sessionState = SessionState(

contentVideoIdentifier = MediaIdentifier.MediaPlatformIdentifier(

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

sessionId = "",

),

mediaClientEventType = MediaEventType.IntentToPlay(IntentToPlay()),

playingMediaState = PlayingMediaState(

videoType = VideoType.Content,

mediaAssetUrl = "",

mediaMetadata = MediaMetadata(publisherIdentifier = PublisherIdentifier

.TwitterPublisherIdentifier(TwitterPublisherIdentifier(123456L)))

),

playerState = PlayerState(isMuted = false)

)

val mediaDetailsV2: MediaDetailsV2 = MediaDetailsV2(

mediaItems = Some(

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 cardDetails =

CardDetails(amplifyDetails = Some(AmplifyDetails(videoType = Some("content"))))

val videoMetadata: TweetActionInfo = TweetActionInfo.TweetVideoWatch(

TweetVideoWatch(

mediaType = Some(MediaType.ConsumerVideo),

isMonetizable = Some(false),

videoType = Some("content")))

val notificationDetails: NotificationDetails =

NotificationDetails(impressionId = Some(notificationId))

val notificationTabTweetEventDetails: NotificationTabDetails =

NotificationTabDetails(

clientEventMetadata = Some(

NotificationClientEventMetadata(

tweetIds = Some(Seq[Long](itemTweetId)),

upstreamId = Some(notificationId),

requestId = "",

notificationId = "",

notificationCount = 0))

)

val notificationTabMultiTweetEventDetails: NotificationTabDetails =

NotificationTabDetails(

clientEventMetadata = Some(

NotificationClientEventMetadata(

tweetIds = Some(tweetIds),

upstreamId = Some(notificationId),

requestId = "",

notificationId = "",

notificationCount = 0))

)

val notificationTabUnknownEventDetails: NotificationTabDetails =

NotificationTabDetails(

clientEventMetadata = Some(

NotificationClientEventMetadata(

upstreamId = Some(notificationId),

requestId = "",

notificationId = "",

notificationCount = 0))

)

val tweetNotificationContent: NotificationContent =

NotificationContent.TweetNotification(TweetNotification(itemTweetId))

val multiTweetNotificationContent: NotificationContent =

NotificationContent.MultiTweetNotification(MultiTweetNotification(tweetIds))

val unknownNotificationContent: NotificationContent =

NotificationContent.UnknownNotification(UnknownNotification())

val reportTweetClick: TweetActionInfo =

TweetActionInfo.ClientTweetReport(ClientTweetReport(isReportTweetDone = false))

val reportTweetDone: TweetActionInfo =

TweetActionInfo.ClientTweetReport(ClientTweetReport(isReportTweetDone = true))

val reportTweetWithReportFlowId: TweetActionInfo =

TweetActionInfo.ClientTweetReport(

ClientTweetReport(isReportTweetDone = true, reportFlowId = Some(reportFlowId)))

val reportTweetWithoutReportFlowId: TweetActionInfo =

TweetActionInfo.ClientTweetReport(

ClientTweetReport(isReportTweetDone = true, reportFlowId = None))

val reportTweetSubmit: TweetActionInfo =

TweetActionInfo.ServerTweetReport(

ServerTweetReport(reportFlowId = Some(reportFlowId), reportType = Some(ReportType.Abuse)))

val notificationTabProductSurfaceInfo: ProductSurfaceInfo =

ProductSurfaceInfo.NotificationTabInfo(NotificationTabInfo(notificationId = notificationId))

val clientOpenLinkWithUrl: TweetActionInfo =

TweetActionInfo.ClientTweetOpenLink(ClientTweetOpenLink(url = Some("go/url")))

val clientOpenLinkWithoutUrl: TweetActionInfo =

TweetActionInfo.ClientTweetOpenLink(ClientTweetOpenLink(url = None))

val clientTakeScreenshot: TweetActionInfo =

TweetActionInfo.ClientTweetTakeScreenshot(

ClientTweetTakeScreenshot(percentVisibleHeight100k = Some(100)))

// client-event event\_namespace

val ceLingerEventNamespace: EventNamespace = EventNamespace(

component = Some("stream"),

element = Some("linger"),

action = Some("results")

)

val ceRenderEventNamespace: EventNamespace = EventNamespace(

component = Some("stream"),

action = Some("results")

)

val ceTweetDetailsEventNamespace1: EventNamespace = EventNamespace(

page = Some("tweet"),

section = None,

component = Some("tweet"),

element = None,

action = Some("impression")

)

val ceGalleryEventNamespace: EventNamespace = EventNamespace(

component = Some("gallery"),

element = Some("photo"),

action = Some("impression")

)

val ceFavoriteEventNamespace: EventNamespace = EventNamespace(action = Some("favorite"))

val ceHomeFavoriteEventNamespace: EventNamespace =

EventNamespace(page = Some("home"), action = Some("favorite"))

val ceHomeLatestFavoriteEventNamespace: EventNamespace =

EventNamespace(page = Some("home\_latest"), action = Some("favorite"))

val ceSearchFavoriteEventNamespace: EventNamespace =

EventNamespace(page = Some("search"), action = Some("favorite"))

val ceClickReplyEventNamespace: EventNamespace = EventNamespace(action = Some("reply"))

val ceReplyEventNamespace: EventNamespace = EventNamespace(action = Some("send\_reply"))

val ceRetweetEventNamespace: EventNamespace = EventNamespace(action = Some("retweet"))

val ceVideoPlayback25: EventNamespace = EventNamespace(action = Some("playback\_25"))

val ceVideoPlayback50: EventNamespace = EventNamespace(action = Some("playback\_50"))

val ceVideoPlayback75: EventNamespace = EventNamespace(action = Some("playback\_75"))

val ceVideoPlayback95: EventNamespace = EventNamespace(action = Some("playback\_95"))

val ceVideoPlayFromTap: EventNamespace = EventNamespace(action = Some("play\_from\_tap"))

val ceVideoQualityView: EventNamespace = EventNamespace(action = Some("video\_quality\_view"))

val ceVideoView: EventNamespace = EventNamespace(action = Some("video\_view"))

val ceVideoMrcView: EventNamespace = EventNamespace(action = Some("video\_mrc\_view"))

val ceVideoViewThreshold: EventNamespace = EventNamespace(action = Some("view\_threshold"))

val ceVideoCtaUrlClick: EventNamespace = EventNamespace(action = Some("cta\_url\_click"))

val ceVideoCtaWatchClick: EventNamespace = EventNamespace(action = Some("cta\_watch\_click"))

val cePhotoExpand: EventNamespace =

EventNamespace(element = Some("platform\_photo\_card"), action = Some("click"))

val ceCardClick: EventNamespace =

EventNamespace(element = Some("platform\_card"), action = Some("click"))

val ceCardOpenApp: EventNamespace = EventNamespace(action = Some("open\_app"))

val ceCardAppInstallAttempt: EventNamespace = EventNamespace(action = Some("install\_app"))

val cePollCardVote1: EventNamespace =

EventNamespace(element = Some("platform\_card"), action = Some("vote"))

val cePollCardVote2: EventNamespace =

EventNamespace(element = Some("platform\_forward\_card"), action = Some("vote"))

val ceMentionClick: EventNamespace =

EventNamespace(element = Some("mention"), action = Some("click"))

val ceVideoPlaybackStart: EventNamespace = EventNamespace(action = Some("playback\_start"))

val ceVideoPlaybackComplete: EventNamespace = EventNamespace(action = Some("playback\_complete"))

val ceClickHashtag: EventNamespace = EventNamespace(action = Some("hashtag\_click"))

val ceTopicFollow1: EventNamespace =

EventNamespace(element = Some("topic"), action = Some("follow"))

val ceOpenLink: EventNamespace = EventNamespace(action = Some("open\_link"))

val ceTakeScreenshot: EventNamespace = EventNamespace(action = Some("take\_screenshot"))

val ceTopicFollow2: EventNamespace =

EventNamespace(element = Some("social\_proof"), action = Some("follow"))

val ceTopicFollow3: EventNamespace =

EventNamespace(element = Some("feedback\_follow\_topic"), action = Some("click"))

val ceTopicUnfollow1: EventNamespace =

EventNamespace(element = Some("topic"), action = Some("unfollow"))

val ceTopicUnfollow2: EventNamespace =

EventNamespace(element = Some("social\_proof"), action = Some("unfollow"))

val ceTopicUnfollow3: EventNamespace =

EventNamespace(element = Some("feedback\_unfollow\_topic"), action = Some("click"))

val ceTopicNotInterestedIn1: EventNamespace =

EventNamespace(element = Some("topic"), action = Some("not\_interested"))

val ceTopicNotInterestedIn2: EventNamespace =

EventNamespace(element = Some("feedback\_not\_interested\_in\_topic"), action = Some("click"))

val ceTopicUndoNotInterestedIn1: EventNamespace =

EventNamespace(element = Some("topic"), action = Some("un\_not\_interested"))

val ceTopicUndoNotInterestedIn2: EventNamespace =

EventNamespace(element = Some("feedback\_not\_interested\_in\_topic"), action = Some("undo"))

val ceProfileFollowAttempt: EventNamespace =

EventNamespace(action = Some("follow\_attempt"))

val ceTweetFavoriteAttempt: EventNamespace =

EventNamespace(action = Some("favorite\_attempt"))

val ceTweetRetweetAttempt: EventNamespace =

EventNamespace(action = Some("retweet\_attempt"))

val ceTweetReplyAttempt: EventNamespace =

EventNamespace(action = Some("reply\_attempt"))

val ceClientCTALoginClick: EventNamespace =

EventNamespace(action = Some("login"))

val ceClientCTALoginStart: EventNamespace =

EventNamespace(page = Some("login"), action = Some("show"))

val ceClientCTALoginSuccess: EventNamespace =

EventNamespace(page = Some("login"), action = Some("success"))

val ceClientCTASignupClick: EventNamespace =

EventNamespace(action = Some("signup"))

val ceClientCTASignupSuccess: EventNamespace =

EventNamespace(page = Some("signup"), action = Some("success"))

val ceNotificationOpen: EventNamespace = EventNamespace(

page = Some("notification"),

section = Some("status\_bar"),

component = None,

action = Some("open"))

val ceNotificationClick: EventNamespace = EventNamespace(

page = Some("ntab"),

section = Some("all"),

component = Some("urt"),

element = Some("users\_liked\_your\_tweet"),

action = Some("navigate"))

val ceTypeaheadClick: EventNamespace =

EventNamespace(element = Some("typeahead"), action = Some("click"))

val ceTweetReport: EventNamespace = EventNamespace(element = Some("report\_tweet"))

def ceEventNamespace(element: String, action: String): EventNamespace =

EventNamespace(element = Some(element), action = Some(action))

def ceTweetReportFlow(page: String, action: String): EventNamespace =

EventNamespace(element = Some("ticket"), page = Some(page), action = Some(action))

val ceNotificationSeeLessOften: EventNamespace = EventNamespace(

page = Some("ntab"),

section = Some("all"),

component = Some("urt"),

action = Some("see\_less\_often"))

val ceNotificationDismiss: EventNamespace = EventNamespace(

page = Some("notification"),

section = Some("status\_bar"),

component = None,

action = Some("dismiss"))

val ceSearchResultsRelevant: EventNamespace = EventNamespace(

page = Some("search"),

component = Some("did\_you\_find\_it\_module"),

element = Some("is\_relevant"),

action = Some("click")

)

val ceSearchResultsNotRelevant: EventNamespace = EventNamespace(

page = Some("search"),

component = Some("did\_you\_find\_it\_module"),

element = Some("not\_relevant"),

action = Some("click")

)

val ceTweetRelevantToSearch: EventNamespace = EventNamespace(

page = Some("search"),

component = Some("relevance\_prompt\_module"),

element = Some("is\_relevant"),

action = Some("click"))

val ceTweetNotRelevantToSearch: EventNamespace = EventNamespace(

page = Some("search"),

component = Some("relevance\_prompt\_module"),

element = Some("not\_relevant"),

action = Some("click"))

val ceProfileBlock: EventNamespace =

EventNamespace(page = Some("profile"), action = Some("block"))

val ceProfileUnblock: EventNamespace =

EventNamespace(page = Some("profile"), action = Some("unblock"))

val ceProfileMute: EventNamespace =

EventNamespace(page = Some("profile"), action = Some("mute\_user"))

val ceProfileReport: EventNamespace =

EventNamespace(page = Some("profile"), action = Some("report"))

val ceProfileShow: EventNamespace =

EventNamespace(page = Some("profile"), action = Some("show"))

val ceProfileFollow: EventNamespace =

EventNamespace(action = Some("follow"))

val ceProfileClick: EventNamespace =

EventNamespace(action = Some("profile\_click"))

val ceTweetFollowAuthor1: EventNamespace = EventNamespace(

action = Some("click"),

element = Some("follow")

)

val ceTweetFollowAuthor2: EventNamespace = EventNamespace(

action = Some("follow")

)

val ceTweetUnfollowAuthor1: EventNamespace = EventNamespace(

action = Some("click"),

element = Some("unfollow")

)

val ceTweetUnfollowAuthor2: EventNamespace = EventNamespace(

action = Some("unfollow")

)

val ceTweetBlockAuthor: EventNamespace = EventNamespace(

page = Some("profile"),

section = Some("tweets"),

component = Some("tweet"),

action = Some("click"),

element = Some("block")

)

val ceTweetUnblockAuthor: EventNamespace = EventNamespace(

section = Some("tweets"),

component = Some("tweet"),

action = Some("click"),

element = Some("unblock")

)

val ceTweetMuteAuthor: EventNamespace = EventNamespace(

component = Some("suggest\_sc\_tweet"),

action = Some("click"),

element = Some("mute")

)

val ceTweetClick: EventNamespace =

EventNamespace(element = Some("tweet"), action = Some("click"))

val ceTweetClickProfile: EventNamespace = EventNamespace(

component = Some("tweet"),

element = Some("user"),

action = Some("profile\_click"))

val ceAppExit: EventNamespace =

EventNamespace(page = Some("app"), action = Some("become\_inactive"))

// UUA client\_event\_namespace

val uuaLingerClientEventNamespace: ClientEventNamespace = ClientEventNamespace(

component = Some("stream"),

element = Some("linger"),

action = Some("results")

)

val uuaRenderClientEventNamespace: ClientEventNamespace = ClientEventNamespace(

component = Some("stream"),

action = Some("results")

)

val ceTweetDetailsClientEventNamespace1: ClientEventNamespace = ClientEventNamespace(

page = Some("tweet"),

section = None,

component = Some("tweet"),

element = None,

action = Some("impression")

)

val ceTweetDetailsClientEventNamespace2: ClientEventNamespace = ClientEventNamespace(

page = Some("tweet"),

section = None,

component = Some("suggest\_ranked\_list\_tweet"),

element = None,

action = Some("impression")

)

val ceTweetDetailsClientEventNamespace3: ClientEventNamespace = ClientEventNamespace(

page = Some("tweet"),

section = None,

component = None,

element = None,

action = Some("impression")

)

val ceTweetDetailsClientEventNamespace4: ClientEventNamespace = ClientEventNamespace(

page = Some("tweet"),

section = None,

component = None,

element = None,

action = Some("show")

)

val ceTweetDetailsClientEventNamespace5: ClientEventNamespace = ClientEventNamespace(

page = Some("tweet"),

section = Some("landing"),

component = None,

element = None,

action = Some("show")

)

val ceGalleryClientEventNamespace: ClientEventNamespace = ClientEventNamespace(

component = Some("gallery"),

element = Some("photo"),

action = Some("impression")

)

val uuaFavoriteClientEventNamespace: ClientEventNamespace =

ClientEventNamespace(action = Some("favorite"))

val uuaHomeFavoriteClientEventNamespace: ClientEventNamespace =

ClientEventNamespace(page = Some("home"), action = Some("favorite"))

val uuaSearchFavoriteClientEventNamespace: ClientEventNamespace =

ClientEventNamespace(page = Some("search"), action = Some("favorite"))

val uuaHomeLatestFavoriteClientEventNamespace: ClientEventNamespace =

ClientEventNamespace(page = Some("home\_latest"), action = Some("favorite"))

val uuaClickReplyClientEventNamespace: ClientEventNamespace =

ClientEventNamespace(action = Some("reply"))

val uuaReplyClientEventNamespace: ClientEventNamespace =

ClientEventNamespace(action = Some("send\_reply"))

val uuaRetweetClientEventNamespace: ClientEventNamespace =

ClientEventNamespace(action = Some("retweet"))

val uuaVideoPlayback25ClientEventNamespace: ClientEventNamespace =

ClientEventNamespace(action = Some("playback\_25"))

val uuaVideoPlayback50ClientEventNamespace: ClientEventNamespace =

ClientEventNamespace(action = Some("playback\_50"))

val uuaVideoPlayback75ClientEventNamespace: ClientEventNamespace =

ClientEventNamespace(action = Some("playback\_75"))

val uuaVideoPlayback95ClientEventNamespace: ClientEventNamespace =

ClientEventNamespace(action = Some("playback\_95"))

val uuaOpenLinkClientEventNamespace: ClientEventNamespace =

ClientEventNamespace(action = Some("open\_link"))

val uuaTakeScreenshotClientEventNamespace: ClientEventNamespace =

ClientEventNamespace(action = Some("take\_screenshot"))

val uuaVideoPlayFromTapClientEventNamespace: ClientEventNamespace =

ClientEventNamespace(action = Some("play\_from\_tap"))

val uuaVideoQualityViewClientEventNamespace: ClientEventNamespace =

ClientEventNamespace(action = Some("video\_quality\_view"))

val uuaVideoViewClientEventNamespace: ClientEventNamespace =

ClientEventNamespace(action = Some("video\_view"))

val uuaVideoMrcViewClientEventNamespace: ClientEventNamespace =

ClientEventNamespace(action = Some("video\_mrc\_view"))

val uuaVideoViewThresholdClientEventNamespace: ClientEventNamespace =

ClientEventNamespace(action = Some("view\_threshold"))

val uuaVideoCtaUrlClickClientEventNamespace: ClientEventNamespace =

ClientEventNamespace(action = Some("cta\_url\_click"))

val uuaVideoCtaWatchClickClientEventNamespace: ClientEventNamespace =

ClientEventNamespace(action = Some("cta\_watch\_click"))

val uuaPhotoExpandClientEventNamespace: ClientEventNamespace =

ClientEventNamespace(element = Some("platform\_photo\_card"), action = Some("click"))

val uuaCardClickClientEventNamespace: ClientEventNamespace =

ClientEventNamespace(element = Some("platform\_card"), action = Some("click"))

val uuaCardOpenAppClientEventNamespace: ClientEventNamespace =

ClientEventNamespace(action = Some("open\_app"))

val uuaCardAppInstallAttemptClientEventNamespace: ClientEventNamespace =

ClientEventNamespace(action = Some("install\_app"))

val uuaPollCardVote1ClientEventNamespace: ClientEventNamespace =

ClientEventNamespace(element = Some("platform\_card"), action = Some("vote"))

val uuaPollCardVote2ClientEventNamespace: ClientEventNamespace =

ClientEventNamespace(element = Some("platform\_forward\_card"), action = Some("vote"))

val uuaMentionClickClientEventNamespace: ClientEventNamespace =

ClientEventNamespace(element = Some("mention"), action = Some("click"))

val uuaVideoPlaybackStartClientEventNamespace: ClientEventNamespace =

ClientEventNamespace(action = Some("playback\_start"))

val uuaVideoPlaybackCompleteClientEventNamespace: ClientEventNamespace =

ClientEventNamespace(action = Some("playback\_complete"))

val uuaClickHashtagClientEventNamespace: ClientEventNamespace =

ClientEventNamespace(action = Some("hashtag\_click"))

val uuaTopicFollowClientEventNamespace1: ClientEventNamespace =

ClientEventNamespace(element = Some("topic"), action = Some("follow"))

val uuaTopicFollowClientEventNamespace2: ClientEventNamespace =

ClientEventNamespace(element = Some("social\_proof"), action = Some("follow"))

val uuaTopicFollowClientEventNamespace3: ClientEventNamespace =

ClientEventNamespace(element = Some("feedback\_follow\_topic"), action = Some("click"))

val uuaTopicUnfollowClientEventNamespace1: ClientEventNamespace =

ClientEventNamespace(element = Some("topic"), action = Some("unfollow"))

val uuaTopicUnfollowClientEventNamespace2: ClientEventNamespace =

ClientEventNamespace(element = Some("social\_proof"), action = Some("unfollow"))

val uuaTopicUnfollowClientEventNamespace3: ClientEventNamespace =

ClientEventNamespace(element = Some("feedback\_unfollow\_topic"), action = Some("click"))

val uuaTopicNotInterestedInClientEventNamespace1: ClientEventNamespace =

ClientEventNamespace(element = Some("topic"), action = Some("not\_interested"))

val uuaTopicNotInterestedInClientEventNamespace2: ClientEventNamespace =

ClientEventNamespace(

element = Some("feedback\_not\_interested\_in\_topic"),

action = Some("click"))

val uuaTopicUndoNotInterestedInClientEventNamespace1: ClientEventNamespace =

ClientEventNamespace(element = Some("topic"), action = Some("un\_not\_interested"))

val uuaTopicUndoNotInterestedInClientEventNamespace2: ClientEventNamespace =

ClientEventNamespace(

element = Some("feedback\_not\_interested\_in\_topic"),

action = Some("undo"))

val uuaProfileFollowAttempt: ClientEventNamespace =

ClientEventNamespace(action = Some("follow\_attempt"))

val uuaTweetFavoriteAttempt: ClientEventNamespace =

ClientEventNamespace(action = Some("favorite\_attempt"))

val uuaTweetRetweetAttempt: ClientEventNamespace =

ClientEventNamespace(action = Some("retweet\_attempt"))

val uuaTweetReplyAttempt: ClientEventNamespace =

ClientEventNamespace(action = Some("reply\_attempt"))

val uuaClientCTALoginClick: ClientEventNamespace =

ClientEventNamespace(action = Some("login"))

val uuaClientCTALoginStart: ClientEventNamespace =

ClientEventNamespace(page = Some("login"), action = Some("show"))

val uuaClientCTALoginSuccess: ClientEventNamespace =

ClientEventNamespace(page = Some("login"), action = Some("success"))

val uuaClientCTASignupClick: ClientEventNamespace =

ClientEventNamespace(action = Some("signup"))

val uuaClientCTASignupSuccess: ClientEventNamespace =

ClientEventNamespace(page = Some("signup"), action = Some("success"))

val uuaNotificationOpen: ClientEventNamespace =

ClientEventNamespace(

page = Some("notification"),

section = Some("status\_bar"),

component = None,

action = Some("open"))

val uuaNotificationClick: ClientEventNamespace =

ClientEventNamespace(

page = Some("ntab"),

section = Some("all"),

component = Some("urt"),

element = Some("users\_liked\_your\_tweet"),

action = Some("navigate"))

val uuaTweetReport: ClientEventNamespace = ClientEventNamespace(element = Some("report\_tweet"))

val uuaTweetFollowAuthor1: ClientEventNamespace =

ClientEventNamespace(element = Some("follow"), action = Some("click"))

val uuaTweetFollowAuthor2: ClientEventNamespace =

ClientEventNamespace(action = Some("follow"))

val uuaTweetUnfollowAuthor1: ClientEventNamespace =

ClientEventNamespace(element = Some("unfollow"), action = Some("click"))

val uuaTweetUnfollowAuthor2: ClientEventNamespace =

ClientEventNamespace(action = Some("unfollow"))

val uuaNotificationSeeLessOften: ClientEventNamespace = ClientEventNamespace(

page = Some("ntab"),

section = Some("all"),

component = Some("urt"),

action = Some("see\_less\_often"))

def uuaClientEventNamespace(element: String, action: String): ClientEventNamespace =

ClientEventNamespace(element = Some(element), action = Some(action))

def uuaTweetReportFlow(page: String, action: String): ClientEventNamespace =

ClientEventNamespace(element = Some("ticket"), page = Some(page), action = Some(action))

val uuaTweetClick: ClientEventNamespace =

ClientEventNamespace(element = Some("tweet"), action = Some("click"))

def uuaTweetClickProfile: ClientEventNamespace = ClientEventNamespace(

component = Some("tweet"),

element = Some("user"),

action = Some("profile\_click"))

val uuaNotificationDismiss: ClientEventNamespace = ClientEventNamespace(

page = Some("notification"),

section = Some("status\_bar"),

component = None,

action = Some("dismiss"))

val uuaTypeaheadClick: ClientEventNamespace =

ClientEventNamespace(element = Some("typeahead"), action = Some("click"))

val uuaSearchResultsRelevant: ClientEventNamespace = ClientEventNamespace(

page = Some("search"),

component = Some("did\_you\_find\_it\_module"),

element = Some("is\_relevant"),

action = Some("click")

)

val uuaSearchResultsNotRelevant: ClientEventNamespace = ClientEventNamespace(

page = Some("search"),

component = Some("did\_you\_find\_it\_module"),

element = Some("not\_relevant"),

action = Some("click")

)

val uuaTweetRelevantToSearch: ClientEventNamespace = ClientEventNamespace(

page = Some("search"),

component = Some("relevance\_prompt\_module"),

element = Some("is\_relevant"),

action = Some("click"))

val uuaTweetNotRelevantToSearch: ClientEventNamespace = ClientEventNamespace(

page = Some("search"),

component = Some("relevance\_prompt\_module"),

element = Some("not\_relevant"),

action = Some("click"))

val uuaProfileBlock: ClientEventNamespace =

ClientEventNamespace(page = Some("profile"), action = Some("block"))

val uuaProfileUnblock: ClientEventNamespace =

ClientEventNamespace(page = Some("profile"), action = Some("unblock"))

val uuaProfileMute: ClientEventNamespace =

ClientEventNamespace(page = Some("profile"), action = Some("mute\_user"))

val uuaProfileReport: ClientEventNamespace =

ClientEventNamespace(page = Some("profile"), action = Some("report"))

val uuaProfileShow: ClientEventNamespace =

ClientEventNamespace(page = Some("profile"), action = Some("show"))

val uuaProfileFollow: ClientEventNamespace =

ClientEventNamespace(action = Some("follow"))

val uuaProfileClick: ClientEventNamespace =

ClientEventNamespace(action = Some("profile\_click"))

val uuaTweetBlockAuthor: ClientEventNamespace = ClientEventNamespace(

page = Some("profile"),

section = Some("tweets"),

component = Some("tweet"),

action = Some("click"),

element = Some("block")

)

val uuaTweetUnblockAuthor: ClientEventNamespace = ClientEventNamespace(

section = Some("tweets"),

component = Some("tweet"),

action = Some("click"),

element = Some("unblock")

)

val uuaTweetMuteAuthor: ClientEventNamespace = ClientEventNamespace(

component = Some("suggest\_sc\_tweet"),

action = Some("click"),

element = Some("mute")

)

val uuaAppExit: ClientEventNamespace =

ClientEventNamespace(page = Some("app"), action = Some("become\_inactive"))

// helper methods for creating client-events and UUA objects

def mkLogEvent(

eventName: String = "",

eventNamespace: Option[EventNamespace],

eventDetails: Option[EventDetails] = None,

logBase: Option[LogBase] = None,

pushNotificationDetails: Option[NotificationDetails] = None,

reportDetails: Option[ReportDetails] = None,

searchDetails: Option[SearchDetails] = None,

performanceDetails: Option[PerformanceDetails] = None

): LogEvent = LogEvent(

eventName = eventName,

eventNamespace = eventNamespace,

eventDetails = eventDetails,

logBase = logBase,

notificationDetails = pushNotificationDetails,

reportDetails = reportDetails,

searchDetails = searchDetails,

performanceDetails = performanceDetails

)

def actionTowardDefaultTweetEvent(

eventNamespace: Option[EventNamespace],

impressionDetails: Option[ImpressionDetails] = None,

suggestionDetails: Option[SuggestionDetails] = None,

itemId: Option[Long] = Some(itemTweetId),

mediaDetailsV2: Option[MediaDetailsV2] = None,

clientMediaEvent: Option[ClientMediaEvent] = None,

itemTypeOpt: Option[ItemType] = Some(ItemType.Tweet),

authorId: Option[Long] = None,

isFollowedByActingUser: Option[Boolean] = None,

isFollowingActingUser: Option[Boolean] = None,

notificationTabDetails: Option[NotificationTabDetails] = None,

reportDetails: Option[ReportDetails] = None,

logBase: LogBase = logBase,

tweetPosition: Option[Int] = None,

promotedId: Option[String] = None,

url: Option[String] = None,

targets: Option[Seq[LogEventItem]] = None,

percentVisibleHeight100k: Option[Int] = None,

searchDetails: Option[SearchDetails] = None,

cardDetails: Option[CardDetails] = None

): LogEvent =

mkLogEvent(

eventName = "action\_toward\_default\_tweet\_event",

eventNamespace = eventNamespace,

reportDetails = reportDetails,

eventDetails = Some(

EventDetails(

url = url,

items = Some(

Seq(LogEventItem(

id = itemId,

percentVisibleHeight100k = percentVisibleHeight100k,

itemType = itemTypeOpt,

impressionDetails = impressionDetails,

suggestionDetails = suggestionDetails,

mediaDetailsV2 = mediaDetailsV2,

clientMediaEvent = clientMediaEvent,

cardDetails = cardDetails,

tweetDetails = authorId.map { id => LogEventTweetDetails(authorId = Some(id)) },

isViewerFollowsTweetAuthor = isFollowedByActingUser,

isTweetAuthorFollowsViewer = isFollowingActingUser,

notificationTabDetails = notificationTabDetails,

position = tweetPosition,

promotedId = promotedId

))),

targets = targets

)

),

logBase = Some(logBase),

searchDetails = searchDetails

)

def actionTowardReplyEvent(

eventNamespace: Option[EventNamespace],

inReplyToTweetId: Long = inReplyToTweetId,

impressionDetails: Option[ImpressionDetails] = None

): LogEvent =

mkLogEvent(

eventName = "action\_toward\_reply\_event",

eventNamespace = eventNamespace,

eventDetails = Some(

EventDetails(

items = Some(

Seq(

LogEventItem(

id = Some(itemTweetId),

itemType = Some(ItemType.Tweet),

impressionDetails = impressionDetails,

tweetDetails =

Some(LogEventTweetDetails(inReplyToTweetId = Some(inReplyToTweetId)))

))

)

)

),

logBase = Some(logBase)

)

def actionTowardRetweetEvent(

eventNamespace: Option[EventNamespace],

inReplyToTweetId: Option[Long] = None,

impressionDetails: Option[ImpressionDetails] = None

): LogEvent =

mkLogEvent(

eventName = "action\_toward\_retweet\_event",

eventNamespace = eventNamespace,

eventDetails = Some(

EventDetails(

items = Some(

Seq(LogEventItem(

id = Some(itemTweetId),

itemType = Some(ItemType.Tweet),

impressionDetails = impressionDetails,

tweetDetails = Some(LogEventTweetDetails(

retweetingTweetId = Some(retweetingTweetId),

inReplyToTweetId = inReplyToTweetId))

)))

)

),

logBase = Some(logBase)

)

def actionTowardQuoteEvent(

eventNamespace: Option[EventNamespace],

inReplyToTweetId: Option[Long] = None,

quotedAuthorId: Option[Long] = None,

impressionDetails: Option[ImpressionDetails] = None

): LogEvent =

mkLogEvent(

eventName = "action\_toward\_quote\_event",

eventNamespace = eventNamespace,

eventDetails = Some(

EventDetails(

items = Some(

Seq(

LogEventItem(

id = Some(itemTweetId),

itemType = Some(ItemType.Tweet),

impressionDetails = impressionDetails,

tweetDetails = Some(

LogEventTweetDetails(

quotedTweetId = Some(quotedTweetId),

inReplyToTweetId = inReplyToTweetId,

quotedAuthorId = quotedAuthorId))

))

)

)

),

logBase = Some(logBase)

)

def actionTowardRetweetEventWithReplyAndQuote(

eventNamespace: Option[EventNamespace],

inReplyToTweetId: Long = inReplyToTweetId,

impressionDetails: Option[ImpressionDetails] = None

): LogEvent = mkLogEvent(

eventName = "action\_toward\_retweet\_event\_with\_reply\_and\_quote",

eventNamespace = eventNamespace,

eventDetails = Some(

EventDetails(

items = Some(

Seq(LogEventItem(

id = Some(itemTweetId),

itemType = Some(ItemType.Tweet),

impressionDetails = impressionDetails,

tweetDetails = Some(

LogEventTweetDetails(

retweetingTweetId = Some(retweetingTweetId),

quotedTweetId = Some(quotedTweetId),

inReplyToTweetId = Some(inReplyToTweetId),

))

)))

)

),

logBase = Some(logBase)

)

def pushNotificationEvent(

eventNamespace: Option[EventNamespace],

itemId: Option[Long] = Some(itemTweetId),

itemTypeOpt: Option[ItemType] = Some(ItemType.Tweet),

notificationDetails: Option[NotificationDetails],

): LogEvent =

mkLogEvent(

eventName = "push\_notification\_open",

eventNamespace = eventNamespace,

eventDetails = Some(

EventDetails(

items = Some(

Seq(

LogEventItem(

id = itemId,

itemType = itemTypeOpt,

))))

),

logBase = Some(logBase),

pushNotificationDetails = notificationDetails

)

def actionTowardNotificationEvent(

eventNamespace: Option[EventNamespace],

notificationTabDetails: Option[NotificationTabDetails],

): LogEvent =

mkLogEvent(

eventName = "notification\_event",

eventNamespace = eventNamespace,

eventDetails = Some(

EventDetails(items =

Some(Seq(LogEventItem(notificationTabDetails = notificationTabDetails))))),

logBase = Some(logBase)

)

def profileClickEvent(eventNamespace: Option[EventNamespace]): LogEvent =

mkLogEvent(

eventName = "profile\_click",

eventNamespace = eventNamespace,

eventDetails = Some(

EventDetails(items = Some(Seq(

LogEventItem(id = Some(userId), itemType = Some(ItemType.User)),

LogEventItem(

id = Some(itemTweetId),

itemType = Some(ItemType.Tweet),

tweetDetails = Some(LogEventTweetDetails(authorId = Some(authorId))))

)))),

logBase = Some(logBase)

)

def actionTowardProfileEvent(

eventName: String,

eventNamespace: Option[EventNamespace]

): LogEvent =

mkLogEvent(

eventName = eventName,

eventNamespace = eventNamespace,

eventDetails = Some(

EventDetails(items = Some(

Seq(

LogEventItem(id = Some(itemProfileId), itemType = Some(ItemType.User))

)))),

logBase = Some(logBase)

)

def tweetActionTowardAuthorEvent(

eventName: String,

eventNamespace: Option[EventNamespace]

): LogEvent =

mkLogEvent(

eventName = eventName,

eventNamespace = eventNamespace,

eventDetails = Some(

EventDetails(items = Some(Seq(

LogEventItem(id = Some(userId), itemType = Some(ItemType.User)),

LogEventItem(

id = Some(itemTweetId),

itemType = Some(ItemType.Tweet),

tweetDetails = Some(LogEventTweetDetails(authorId = Some(authorId))))

)))),

logBase = Some(logBase)

)

def actionTowardsTypeaheadEvent(

eventNamespace: Option[EventNamespace],

targets: Option[Seq[LogEventItem]],

searchQuery: String

): LogEvent =

mkLogEvent(

eventNamespace = eventNamespace,

eventDetails = Some(EventDetails(targets = targets)),

logBase = Some(logBase),

searchDetails = Some(SearchDetails(query = Some(searchQuery)))

)

def actionTowardSearchResultPageEvent(

eventNamespace: Option[EventNamespace],

searchDetails: Option[SearchDetails],

items: Option[Seq[LogEventItem]] = None

): LogEvent =

mkLogEvent(

eventNamespace = eventNamespace,

eventDetails = Some(EventDetails(items = items)),

logBase = Some(logBase),

searchDetails = searchDetails

)

def actionTowardsUasEvent(

eventNamespace: Option[EventNamespace],

clientAppId: Option[Long],

duration: Option[Long]

): LogEvent =

mkLogEvent(

eventNamespace = eventNamespace,

logBase = Some(logBase.copy(clientAppId = clientAppId)),

performanceDetails = Some(PerformanceDetails(durationMs = duration))

)

def mkUUAEventMetadata(

clientEventNamespace: Option[ClientEventNamespace],

traceId: Option[Long] = None,

requestJoinId: Option[Long] = None,

clientAppId: Option[Long] = None

): EventMetadata = EventMetadata(

sourceTimestampMs = 1001L,

receivedTimestampMs = frozenTime.inMilliseconds,

sourceLineage = SourceLineage.ClientEvents,

clientEventNamespace = clientEventNamespace,

traceId = traceId,

requestJoinId = requestJoinId,

clientAppId = clientAppId

)

def mkExpectedUUAForActionTowardDefaultTweetEvent(

clientEventNamespace: Option[ClientEventNamespace],

actionType: ActionType,

inReplyToTweetId: Option[Long] = None,

tweetActionInfo: Option[TweetActionInfo] = None,

topicId: Option[Long] = None,

authorInfo: Option[AuthorInfo] = None,

productSurface: Option[ProductSurface] = None,

productSurfaceInfo: Option[ProductSurfaceInfo] = None,

tweetPosition: Option[Int] = None,

promotedId: Option[String] = None,

traceIdOpt: Option[Long] = None,

requestJoinIdOpt: Option[Long] = None,

guestIdMarketingOpt: Option[Long] = None

): UnifiedUserAction = UnifiedUserAction(

userIdentifier =

UserIdentifier(userId = Some(userId), guestIdMarketing = guestIdMarketingOpt),

item = Item.TweetInfo(

TweetInfo(

actionTweetId = itemTweetId,

inReplyToTweetId = inReplyToTweetId,

tweetActionInfo = tweetActionInfo,

actionTweetTopicSocialProofId = topicId,

actionTweetAuthorInfo = authorInfo,

tweetPosition = tweetPosition,

promotedId = promotedId

)

),

actionType = actionType,

eventMetadata = mkUUAEventMetadata(

clientEventNamespace = clientEventNamespace,

traceId = traceIdOpt,

requestJoinId = requestJoinIdOpt

),

productSurface = productSurface,

productSurfaceInfo = productSurfaceInfo

)

def mkExpectedUUAForActionTowardReplyEvent(

clientEventNamespace: Option[ClientEventNamespace],

actionType: ActionType,

tweetActionInfo: Option[TweetActionInfo] = None,

authorInfo: Option[AuthorInfo] = None,

): UnifiedUserAction = UnifiedUserAction(

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

item = Item.TweetInfo(

TweetInfo(

actionTweetId = itemTweetId,

inReplyToTweetId = Some(inReplyToTweetId),

tweetActionInfo = tweetActionInfo,

actionTweetAuthorInfo = authorInfo

)

),

actionType = actionType,

eventMetadata = mkUUAEventMetadata(clientEventNamespace = clientEventNamespace)

)

def mkExpectedUUAForActionTowardRetweetEvent(

clientEventNamespace: Option[ClientEventNamespace],

actionType: ActionType,

inReplyToTweetId: Option[Long] = None,

tweetActionInfo: Option[TweetActionInfo] = None,

authorInfo: Option[AuthorInfo] = None,

): UnifiedUserAction = UnifiedUserAction(

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

item = Item.TweetInfo(

TweetInfo(

actionTweetId = itemTweetId,

retweetingTweetId = Some(retweetingTweetId),

inReplyToTweetId = inReplyToTweetId,

tweetActionInfo = tweetActionInfo,

actionTweetAuthorInfo = authorInfo

)

),

actionType = actionType,

eventMetadata = mkUUAEventMetadata(clientEventNamespace = clientEventNamespace)

)

def mkExpectedUUAForActionTowardQuoteEvent(

clientEventNamespace: Option[ClientEventNamespace],

actionType: ActionType,

inReplyToTweetId: Option[Long] = None,

quotedAuthorId: Option[Long] = None,

tweetActionInfo: Option[TweetActionInfo] = None,

authorInfo: Option[AuthorInfo] = None,

): UnifiedUserAction = UnifiedUserAction(

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

item = Item.TweetInfo(

TweetInfo(

actionTweetId = itemTweetId,

quotedTweetId = Some(quotedTweetId),

quotedAuthorId = quotedAuthorId,

inReplyToTweetId = inReplyToTweetId,

tweetActionInfo = tweetActionInfo,

actionTweetAuthorInfo = authorInfo

)

),

actionType = actionType,

eventMetadata = mkUUAEventMetadata(clientEventNamespace = clientEventNamespace)

)

def mkExpectedUUAForActionTowardQuotingEvent(

clientEventNamespace: Option[ClientEventNamespace],

actionType: ActionType,

inReplyToTweetId: Option[Long] = None,

tweetActionInfo: Option[TweetActionInfo] = None,

authorInfo: Option[AuthorInfo] = None,

): UnifiedUserAction = UnifiedUserAction(

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

item = Item.TweetInfo(

TweetInfo(

actionTweetId = quotedTweetId,

quotingTweetId = Some(itemTweetId),

inReplyToTweetId = inReplyToTweetId,

tweetActionInfo = tweetActionInfo,

actionTweetAuthorInfo = authorInfo

)

),

actionType = actionType,

eventMetadata = mkUUAEventMetadata(clientEventNamespace = clientEventNamespace)

)

def mkExpectedUUAForActionTowardRetweetEventWithReplyAndQuoted(

clientEventNamespace: Option[ClientEventNamespace],

actionType: ActionType,

inReplyToTweetId: Long = inReplyToTweetId,

tweetActionInfo: Option[TweetActionInfo] = None,

authorInfo: Option[AuthorInfo] = None,

): UnifiedUserAction = UnifiedUserAction(

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

item = Item.TweetInfo(

TweetInfo(

actionTweetId = itemTweetId,

retweetingTweetId = Some(retweetingTweetId),

quotedTweetId = Some(quotedTweetId),

inReplyToTweetId = Some(inReplyToTweetId),

tweetActionInfo = tweetActionInfo,

actionTweetAuthorInfo = authorInfo

)

),

actionType = actionType,

eventMetadata = mkUUAEventMetadata(clientEventNamespace = clientEventNamespace)

)

def mkExpectedUUAForActionTowardRetweetEventWithReplyAndQuoting(

clientEventNamespace: Option[ClientEventNamespace],

actionType: ActionType,

inReplyToTweetId: Long = inReplyToTweetId,

tweetActionInfo: Option[TweetActionInfo] = None,

authorInfo: Option[AuthorInfo] = None,

): UnifiedUserAction = UnifiedUserAction(

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

item = Item.TweetInfo(

TweetInfo(

actionTweetId = quotedTweetId,

quotingTweetId = Some(itemTweetId),

tweetActionInfo = tweetActionInfo,

actionTweetAuthorInfo = authorInfo

)

),

actionType = actionType,

eventMetadata = mkUUAEventMetadata(clientEventNamespace = clientEventNamespace)

)

def mkExpectedUUAForActionTowardTopicEvent(

clientEventNamespace: Option[ClientEventNamespace],

actionType: ActionType,

topicId: Long,

traceId: Option[Long] = None,

): UnifiedUserAction = UnifiedUserAction(

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

item = Item.TopicInfo(

TopicInfo(

actionTopicId = topicId,

)

),

actionType = actionType,

eventMetadata =

mkUUAEventMetadata(clientEventNamespace = clientEventNamespace, traceId = traceId)

)

def mkExpectedUUAForNotificationEvent(

clientEventNamespace: Option[ClientEventNamespace],

actionType: ActionType,

notificationContent: NotificationContent,

productSurface: Option[ProductSurface],

productSurfaceInfo: Option[ProductSurfaceInfo],

): UnifiedUserAction = UnifiedUserAction(

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

item = Item.NotificationInfo(

NotificationInfo(

actionNotificationId = notificationId,

content = notificationContent

)

),

actionType = actionType,

eventMetadata = mkUUAEventMetadata(clientEventNamespace = clientEventNamespace),

productSurface = productSurface,

productSurfaceInfo = productSurfaceInfo

)

def mkExpectedUUAForProfileClick(

clientEventNamespace: Option[ClientEventNamespace],

actionType: ActionType,

authorInfo: Option[AuthorInfo] = None

): UnifiedUserAction = UnifiedUserAction(

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

item = Item.TweetInfo(

TweetInfo(

actionTweetId = itemTweetId,

actionTweetAuthorInfo = authorInfo

)

),

actionType = actionType,

eventMetadata = mkUUAEventMetadata(clientEventNamespace = clientEventNamespace)

)

def mkExpectedUUAForTweetActionTowardAuthor(

clientEventNamespace: Option[ClientEventNamespace],

actionType: ActionType,

authorInfo: Option[AuthorInfo] = None,

tweetActionInfo: Option[TweetActionInfo] = None

): UnifiedUserAction = UnifiedUserAction(

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

item = Item.TweetInfo(

TweetInfo(

actionTweetId = itemTweetId,

actionTweetAuthorInfo = authorInfo,

tweetActionInfo = tweetActionInfo

)

),

actionType = actionType,

eventMetadata = mkUUAEventMetadata(clientEventNamespace = clientEventNamespace)

)

def mkExpectedUUAForProfileAction(

clientEventNamespace: Option[ClientEventNamespace],

actionType: ActionType,

actionProfileId: Long

): UnifiedUserAction = UnifiedUserAction(

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

item = Item.ProfileInfo(

ProfileInfo(

actionProfileId = actionProfileId

)

),

actionType = actionType,

eventMetadata = mkUUAEventMetadata(clientEventNamespace = clientEventNamespace)

)

def mkExpectedUUAForTypeaheadAction(

clientEventNamespace: Option[ClientEventNamespace],

actionType: ActionType,

typeaheadActionInfo: TypeaheadActionInfo,

searchQuery: String,

): UnifiedUserAction = UnifiedUserAction(

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

item = Item.TypeaheadInfo(

TypeaheadInfo(actionQuery = searchQuery, typeaheadActionInfo = typeaheadActionInfo)

),

actionType = actionType,

eventMetadata = mkUUAEventMetadata(clientEventNamespace = clientEventNamespace),

productSurface = Some(ProductSurface.SearchTypeahead),

productSurfaceInfo =

Some(ProductSurfaceInfo.SearchTypeaheadInfo(SearchTypeaheadInfo(query = searchQuery)))

)

def mkExpectedUUAForFeedbackSubmitAction(

clientEventNamespace: Option[ClientEventNamespace],

actionType: ActionType,

feedbackPromptInfo: FeedbackPromptInfo,

searchQuery: String

): UnifiedUserAction = UnifiedUserAction(

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

item = Item.FeedbackPromptInfo(feedbackPromptInfo),

actionType = actionType,

eventMetadata = mkUUAEventMetadata(clientEventNamespace = clientEventNamespace),

productSurface = Some(ProductSurface.SearchResultsPage),

productSurfaceInfo =

Some(ProductSurfaceInfo.SearchResultsPageInfo(SearchResultsPageInfo(query = searchQuery)))

)

def mkExpectedUUAForActionTowardCTAEvent(

clientEventNamespace: Option[ClientEventNamespace],

actionType: ActionType,

guestIdMarketingOpt: Option[Long]

): UnifiedUserAction = UnifiedUserAction(

userIdentifier =

UserIdentifier(userId = Some(userId), guestIdMarketing = guestIdMarketingOpt),

item = Item.CtaInfo(CTAInfo()),

actionType = actionType,

eventMetadata = mkUUAEventMetadata(clientEventNamespace = clientEventNamespace)

)

def mkExpectedUUAForUasEvent(

clientEventNamespace: Option[ClientEventNamespace],

actionType: ActionType,

clientAppId: Option[Long],

duration: Option[Long]

): UnifiedUserAction = UnifiedUserAction(

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

item = Item.UasInfo(UASInfo(timeSpentMs = duration.get)),

actionType = actionType,

eventMetadata =

mkUUAEventMetadata(clientEventNamespace = clientEventNamespace, clientAppId = clientAppId)

)

def mkExpectedUUAForCardEvent(

id: Option[Long],

clientEventNamespace: Option[ClientEventNamespace],

actionType: ActionType,

itemType: Option[ItemType],

authorId: Option[Long],

): UnifiedUserAction = UnifiedUserAction(

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

item = Item.CardInfo(

CardInfo(

id = id,

itemType = itemType,

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

actionType = actionType,

eventMetadata = mkUUAEventMetadata(clientEventNamespace = clientEventNamespace)

)

def timelineTopicControllerData(topicId: Long = topicId): ControllerData.V2 =

ControllerData.V2(

ControllerDataV2.TimelinesTopic(

TimelinesTopicControllerData.V1(

TimelinesTopicControllerDataV1(

topicId = topicId,

topicTypesBitmap = 1

)

)))

def homeTweetControllerData(

topicId: Long = topicId,

traceId: Long = traceId

): ControllerData.V2 =

ControllerData.V2(

ControllerDataV2.HomeTweets(

HomeTweetsControllerData.V1(

HomeTweetsControllerDataV1(

topicId = Some(topicId),

traceId = Some(traceId)

))))

def homeTweetControllerDataV2(

injectedPosition: Option[Int] = None,

requestJoinId: Option[Long] = None,

traceId: Option[Long] = None

): ControllerData.V2 =

ControllerData.V2(

ControllerDataV2.HomeTweets(

HomeTweetsControllerData.V1(

HomeTweetsControllerDataV1(

injectedPosition = injectedPosition,

traceId = traceId,

requestJoinId = requestJoinId

))))

// mock client-events

val ddgEvent: LogEvent = mkLogEvent(

eventName = "ddg",

eventNamespace = Some(

EventNamespace(

page = Some("ddg"),

action = Some("experiment")

)

)

)

val qigRankerEvent: LogEvent = mkLogEvent(

eventName = "qig\_ranker",

eventNamespace = Some(

EventNamespace(

page = Some("qig\_ranker"),

)

)

)

val timelineMixerEvent: LogEvent = mkLogEvent(

eventName = "timelinemixer",

eventNamespace = Some(

EventNamespace(

page = Some("timelinemixer"),

)

)

)

val timelineServiceEvent: LogEvent = mkLogEvent(

eventName = "timelineservice",

eventNamespace = Some(

EventNamespace(

page = Some("timelineservice"),

)

)

)

val tweetConcServiceEvent: LogEvent = mkLogEvent(

eventName = "tweetconvosvc",

eventNamespace = Some(

EventNamespace(

page = Some("tweetconvosvc"),

)

)

)

val renderNonTweetItemTypeEvent: LogEvent = mkLogEvent(

eventName = "render non-tweet item-type",

eventNamespace = Some(ceRenderEventNamespace),

eventDetails = Some(

EventDetails(

items = Some(

Seq(LogEventItem(itemType = Some(ItemType.Event)))

)

)

)

)

val renderDefaultTweetWithTopicIdEvent: LogEvent = actionTowardDefaultTweetEvent(

eventNamespace = Some(ceRenderEventNamespace),

suggestionDetails =

Some(SuggestionDetails(decodedControllerData = Some(timelineTopicControllerData())))

)

def renderDefaultTweetUserFollowStatusEvent(

authorId: Option[Long],

isFollowedByActingUser: Boolean = false,

isFollowingActingUser: Boolean = false

): LogEvent = actionTowardDefaultTweetEvent(

eventNamespace = Some(ceRenderEventNamespace),

authorId = authorId,

isFollowedByActingUser = Some(isFollowedByActingUser),

isFollowingActingUser = Some(isFollowingActingUser)

)

val lingerDefaultTweetEvent: LogEvent = actionTowardDefaultTweetEvent(

eventNamespace = Some(ceLingerEventNamespace),

impressionDetails = Some(

ImpressionDetails(

visibilityStart = Some(100L),

visibilityEnd = Some(105L)

))

)

val lingerReplyEvent: LogEvent = actionTowardReplyEvent(

eventNamespace = Some(ceLingerEventNamespace),

impressionDetails = Some(

ImpressionDetails(

visibilityStart = Some(100L),

visibilityEnd = Some(105L)

))

)

val lingerRetweetEvent: LogEvent = actionTowardRetweetEvent(

eventNamespace = Some(ceLingerEventNamespace),

impressionDetails = Some(

ImpressionDetails(

visibilityStart = Some(100L),

visibilityEnd = Some(105L)

))

)

val lingerQuoteEvent: LogEvent = actionTowardQuoteEvent(

eventNamespace = Some(ceLingerEventNamespace),

impressionDetails = Some(

ImpressionDetails(

visibilityStart = Some(100L),

visibilityEnd = Some(105L)

))

)

val lingerRetweetWithReplyAndQuoteEvent: LogEvent = actionTowardRetweetEventWithReplyAndQuote(

eventNamespace = Some(ceLingerEventNamespace),

impressionDetails = Some(

ImpressionDetails(

visibilityStart = Some(100L),

visibilityEnd = Some(105L)

))

)

val replyToDefaultTweetOrReplyEvent: LogEvent = actionTowardReplyEvent(

eventNamespace = Some(ceReplyEventNamespace),

// since the action is reply, item.id = inReplyToTweetId

inReplyToTweetId = itemTweetId,

)

val replyToRetweetEvent: LogEvent = actionTowardRetweetEvent(

eventNamespace = Some(ceReplyEventNamespace),

// since the action is reply, item.id = inReplyToTweetId

inReplyToTweetId = Some(itemTweetId),

)

val replyToQuoteEvent: LogEvent = actionTowardQuoteEvent(

eventNamespace = Some(ceReplyEventNamespace),

// since the action is reply, item.id = inReplyToTweetId

inReplyToTweetId = Some(itemTweetId),

)

val replyToRetweetWithReplyAndQuoteEvent: LogEvent = actionTowardRetweetEventWithReplyAndQuote(

eventNamespace = Some(ceReplyEventNamespace),

// since the action is reply, item.id = inReplyToTweetId

inReplyToTweetId = itemTweetId,

)

// expected UUA corresponding to mock client-events

val expectedTweetRenderDefaultTweetUUA: UnifiedUserAction =

mkExpectedUUAForActionTowardDefaultTweetEvent(

clientEventNamespace = Some(uuaRenderClientEventNamespace),

actionType = ActionType.ClientTweetRenderImpression

)

val expectedTweetRenderReplyUUA: UnifiedUserAction = mkExpectedUUAForActionTowardReplyEvent(

clientEventNamespace = Some(uuaRenderClientEventNamespace),

actionType = ActionType.ClientTweetRenderImpression

)

val expectedTweetRenderRetweetUUA: UnifiedUserAction = mkExpectedUUAForActionTowardRetweetEvent(

clientEventNamespace = Some(uuaRenderClientEventNamespace),

actionType = ActionType.ClientTweetRenderImpression

)

val expectedTweetRenderQuoteUUA1: UnifiedUserAction = mkExpectedUUAForActionTowardQuoteEvent(

clientEventNamespace = Some(uuaRenderClientEventNamespace),

actionType = ActionType.ClientTweetRenderImpression,

quotedAuthorId = Some(quotedAuthorId),

)

val expectedTweetRenderQuoteUUA2: UnifiedUserAction = mkExpectedUUAForActionTowardQuotingEvent(

clientEventNamespace = Some(uuaRenderClientEventNamespace),

actionType = ActionType.ClientTweetRenderImpression,

authorInfo = Some(AuthorInfo(authorId = Some(quotedAuthorId)))

)

val expectedTweetRenderRetweetWithReplyAndQuoteUUA1: UnifiedUserAction =

mkExpectedUUAForActionTowardRetweetEventWithReplyAndQuoted(

clientEventNamespace = Some(uuaRenderClientEventNamespace),

actionType = ActionType.ClientTweetRenderImpression

)

val expectedTweetRenderRetweetWithReplyAndQuoteUUA2: UnifiedUserAction =

mkExpectedUUAForActionTowardRetweetEventWithReplyAndQuoting(

clientEventNamespace = Some(uuaRenderClientEventNamespace),

actionType = ActionType.ClientTweetRenderImpression

)

val expectedTweetRenderDefaultTweetWithTopicIdUUA: UnifiedUserAction =

mkExpectedUUAForActionTowardDefaultTweetEvent(

clientEventNamespace = Some(uuaRenderClientEventNamespace),

actionType = ActionType.ClientTweetRenderImpression,

topicId = Some(topicId)

)

val expectedTweetDetailImpressionUUA1: UnifiedUserAction =

mkExpectedUUAForActionTowardDefaultTweetEvent(

clientEventNamespace = Some(ceTweetDetailsClientEventNamespace1),

actionType = ActionType.ClientTweetDetailsImpression

)

val expectedTweetGalleryImpressionUUA: UnifiedUserAction =

mkExpectedUUAForActionTowardDefaultTweetEvent(

clientEventNamespace = Some(ceGalleryClientEventNamespace),

actionType = ActionType.ClientTweetGalleryImpression

)

def expectedTweetRenderDefaultTweetWithAuthorInfoUUA(

authorInfo: Option[AuthorInfo] = None

): UnifiedUserAction =

mkExpectedUUAForActionTowardDefaultTweetEvent(

clientEventNamespace = Some(uuaRenderClientEventNamespace),

actionType = ActionType.ClientTweetRenderImpression,

authorInfo = authorInfo

)

val expectedTweetLingerDefaultTweetUUA: UnifiedUserAction =

mkExpectedUUAForActionTowardDefaultTweetEvent(

clientEventNamespace = Some(uuaLingerClientEventNamespace),

actionType = ActionType.ClientTweetLingerImpression,

tweetActionInfo = Some(

TweetActionInfo.ClientTweetLingerImpression(

ClientTweetLingerImpression(

lingerStartTimestampMs = 100L,

lingerEndTimestampMs = 105L

))

)

)

val expectedTweetLingerReplyUUA: UnifiedUserAction = mkExpectedUUAForActionTowardReplyEvent(

clientEventNamespace = Some(uuaLingerClientEventNamespace),

actionType = ActionType.ClientTweetLingerImpression,

tweetActionInfo = Some(

TweetActionInfo.ClientTweetLingerImpression(

ClientTweetLingerImpression(

lingerStartTimestampMs = 100L,

lingerEndTimestampMs = 105L

))

)

)

val expectedTweetLingerRetweetUUA: UnifiedUserAction = mkExpectedUUAForActionTowardRetweetEvent(

clientEventNamespace = Some(uuaLingerClientEventNamespace),

actionType = ActionType.ClientTweetLingerImpression,

tweetActionInfo = Some(

TweetActionInfo.ClientTweetLingerImpression(

ClientTweetLingerImpression(

lingerStartTimestampMs = 100L,

lingerEndTimestampMs = 105L

))

)

)

val expectedTweetLingerQuoteUUA: UnifiedUserAction = mkExpectedUUAForActionTowardQuoteEvent(

clientEventNamespace = Some(uuaLingerClientEventNamespace),

actionType = ActionType.ClientTweetLingerImpression,

tweetActionInfo = Some(

TweetActionInfo.ClientTweetLingerImpression(

ClientTweetLingerImpression(

lingerStartTimestampMs = 100L,

lingerEndTimestampMs = 105L

))

)

)

val expectedTweetLingerRetweetWithReplyAndQuoteUUA: UnifiedUserAction =

mkExpectedUUAForActionTowardRetweetEventWithReplyAndQuoted(

clientEventNamespace = Some(uuaLingerClientEventNamespace),

actionType = ActionType.ClientTweetLingerImpression,

tweetActionInfo = Some(

TweetActionInfo.ClientTweetLingerImpression(

ClientTweetLingerImpression(

lingerStartTimestampMs = 100L,

lingerEndTimestampMs = 105L

))

)

)

val expectedTweetClickQuoteUUA: UnifiedUserAction =

mkExpectedUUAForActionTowardRetweetEventWithReplyAndQuoted(

clientEventNamespace = Some(

ClientEventNamespace(

action = Some("quote")

)),

actionType = ActionType.ClientTweetClickQuote

)

def expectedTweetQuoteUUA(action: String): UnifiedUserAction =

mkExpectedUUAForActionTowardRetweetEventWithReplyAndQuoted(

clientEventNamespace = Some(

ClientEventNamespace(

action = Some(action)

)),

actionType = ActionType.ClientTweetQuote

)

val expectedTweetFavoriteDefaultTweetUUA: UnifiedUserAction =

mkExpectedUUAForActionTowardDefaultTweetEvent(

clientEventNamespace = Some(uuaFavoriteClientEventNamespace),

actionType = ActionType.ClientTweetFav

)

val expectedHomeTweetEventWithControllerDataSuggestType: UnifiedUserAction =

mkExpectedUUAForActionTowardDefaultTweetEvent(

clientEventNamespace = Some(uuaHomeFavoriteClientEventNamespace),

actionType = ActionType.ClientTweetFav,

productSurface = Some(ProductSurface.HomeTimeline),

productSurfaceInfo = Some(

ProductSurfaceInfo.HomeTimelineInfo(

HomeTimelineInfo(suggestionType = Some("Test\_type"), injectedPosition = Some(1)))),

traceIdOpt = Some(traceId),

requestJoinIdOpt = Some(requestJoinId)

)

val expectedHomeTweetEventWithControllerData: UnifiedUserAction =

mkExpectedUUAForActionTowardDefaultTweetEvent(

clientEventNamespace = Some(uuaHomeFavoriteClientEventNamespace),

actionType = ActionType.ClientTweetFav,

productSurface = Some(ProductSurface.HomeTimeline),

productSurfaceInfo =

Some(ProductSurfaceInfo.HomeTimelineInfo(HomeTimelineInfo(injectedPosition = Some(1)))),

traceIdOpt = Some(traceId),

requestJoinIdOpt = Some(requestJoinId)

)

val expectedSearchTweetEventWithControllerData: UnifiedUserAction =

mkExpectedUUAForActionTowardDefaultTweetEvent(

clientEventNamespace = Some(uuaSearchFavoriteClientEventNamespace),

actionType = ActionType.ClientTweetFav,

productSurface = Some(ProductSurface.SearchResultsPage),

productSurfaceInfo =

Some(ProductSurfaceInfo.SearchResultsPageInfo(SearchResultsPageInfo(query = "twitter"))),

traceIdOpt = Some(traceId),

requestJoinIdOpt = Some(requestJoinId)

)

val expectedHomeTweetEventWithSuggestType: UnifiedUserAction =

mkExpectedUUAForActionTowardDefaultTweetEvent(

clientEventNamespace = Some(uuaHomeFavoriteClientEventNamespace),

actionType = ActionType.ClientTweetFav,

productSurface = Some(ProductSurface.HomeTimeline),

productSurfaceInfo = Some(

ProductSurfaceInfo.HomeTimelineInfo(HomeTimelineInfo(suggestionType = Some("Test\_type"))))

)

val expectedHomeLatestTweetEventWithControllerDataSuggestType: UnifiedUserAction =

mkExpectedUUAForActionTowardDefaultTweetEvent(

clientEventNamespace = Some(uuaHomeLatestFavoriteClientEventNamespace),

actionType = ActionType.ClientTweetFav,

productSurface = Some(ProductSurface.HomeTimeline),

productSurfaceInfo = Some(

ProductSurfaceInfo.HomeTimelineInfo(

HomeTimelineInfo(suggestionType = Some("Test\_type"), injectedPosition = Some(1)))),

traceIdOpt = Some(traceId),

requestJoinIdOpt = Some(requestJoinId)

)

val expectedHomeLatestTweetEventWithControllerData: UnifiedUserAction =

mkExpectedUUAForActionTowardDefaultTweetEvent(

clientEventNamespace = Some(uuaHomeLatestFavoriteClientEventNamespace),

actionType = ActionType.ClientTweetFav,

productSurface = Some(ProductSurface.HomeTimeline),

productSurfaceInfo =

Some(ProductSurfaceInfo.HomeTimelineInfo(HomeTimelineInfo(injectedPosition = Some(1)))),

traceIdOpt = Some(traceId),

requestJoinIdOpt = Some(requestJoinId)

)

val expectedHomeLatestTweetEventWithSuggestType: UnifiedUserAction =

mkExpectedUUAForActionTowardDefaultTweetEvent(

clientEventNamespace = Some(uuaHomeLatestFavoriteClientEventNamespace),

actionType = ActionType.ClientTweetFav,

productSurface = Some(ProductSurface.HomeTimeline),

productSurfaceInfo = Some(

ProductSurfaceInfo.HomeTimelineInfo(HomeTimelineInfo(suggestionType = Some("Test\_type"))))

)

val expectedTweetFavoriteReplyUUA: UnifiedUserAction = mkExpectedUUAForActionTowardReplyEvent(

clientEventNamespace = Some(uuaFavoriteClientEventNamespace),

actionType = ActionType.ClientTweetFav

)

val expectedTweetFavoriteRetweetUUA: UnifiedUserAction =

mkExpectedUUAForActionTowardRetweetEvent(

clientEventNamespace = Some(uuaFavoriteClientEventNamespace),

actionType = ActionType.ClientTweetFav

)

val expectedTweetFavoriteQuoteUUA: UnifiedUserAction = mkExpectedUUAForActionTowardQuoteEvent(

clientEventNamespace = Some(uuaFavoriteClientEventNamespace),

actionType = ActionType.ClientTweetFav)

val expectedTweetFavoriteRetweetWithReplyAndQuoteUUA: UnifiedUserAction =

mkExpectedUUAForActionTowardRetweetEventWithReplyAndQuoted(

clientEventNamespace = Some(uuaFavoriteClientEventNamespace),

actionType = ActionType.ClientTweetFav

)

val expectedTweetClickReplyDefaultTweetUUA: UnifiedUserAction =

mkExpectedUUAForActionTowardDefaultTweetEvent(

clientEventNamespace = Some(uuaClickReplyClientEventNamespace),

actionType = ActionType.ClientTweetClickReply

)

val expectedTweetClickReplyReplyUUA: UnifiedUserAction =

mkExpectedUUAForActionTowardReplyEvent(

clientEventNamespace = Some(uuaClickReplyClientEventNamespace),

actionType = ActionType.ClientTweetClickReply

)

val expectedTweetClickReplyRetweetUUA: UnifiedUserAction =

mkExpectedUUAForActionTowardRetweetEvent(

clientEventNamespace = Some(uuaClickReplyClientEventNamespace),

actionType = ActionType.ClientTweetClickReply

)

val expectedTweetClickReplyQuoteUUA: UnifiedUserAction =

mkExpectedUUAForActionTowardQuoteEvent(

clientEventNamespace = Some(uuaClickReplyClientEventNamespace),

actionType = ActionType.ClientTweetClickReply

)

val expectedTweetClickReplyRetweetWithReplyAndQuoteUUA: UnifiedUserAction =

mkExpectedUUAForActionTowardRetweetEventWithReplyAndQuoted(

clientEventNamespace = Some(uuaClickReplyClientEventNamespace),

actionType = ActionType.ClientTweetClickReply

)

val expectedTweetReplyDefaultTweetUUA: UnifiedUserAction =

mkExpectedUUAForActionTowardDefaultTweetEvent(

clientEventNamespace = Some(uuaReplyClientEventNamespace),

actionType = ActionType.ClientTweetReply,

inReplyToTweetId = Some(itemTweetId)

)

val expectedTweetReplyRetweetUUA: UnifiedUserAction =

mkExpectedUUAForActionTowardRetweetEvent(

clientEventNamespace = Some(uuaReplyClientEventNamespace),

actionType = ActionType.ClientTweetReply,

inReplyToTweetId = Some(itemTweetId)

)

val expectedTweetReplyQuoteUUA: UnifiedUserAction =

mkExpectedUUAForActionTowardQuoteEvent(

clientEventNamespace = Some(uuaReplyClientEventNamespace),

actionType = ActionType.ClientTweetReply,

inReplyToTweetId = Some(itemTweetId)

)

val expectedTweetReplyRetweetWithReplyAndQuoteUUA: UnifiedUserAction =

mkExpectedUUAForActionTowardRetweetEventWithReplyAndQuoted(

clientEventNamespace = Some(uuaReplyClientEventNamespace),

actionType = ActionType.ClientTweetReply,

inReplyToTweetId = itemTweetId

)

val expectedTweetRetweetDefaultTweetUUA: UnifiedUserAction =

mkExpectedUUAForActionTowardDefaultTweetEvent(

clientEventNamespace = Some(uuaRetweetClientEventNamespace),

actionType = ActionType.ClientTweetRetweet

)

val expectedTweetRetweetReplyUUA: UnifiedUserAction = mkExpectedUUAForActionTowardReplyEvent(

clientEventNamespace = Some(uuaRetweetClientEventNamespace),

actionType = ActionType.ClientTweetRetweet

)

val expectedTweetRetweetRetweetUUA: UnifiedUserAction =

mkExpectedUUAForActionTowardRetweetEvent(

clientEventNamespace = Some(uuaRetweetClientEventNamespace),

actionType = ActionType.ClientTweetRetweet

)

val expectedTweetRetweetQuoteUUA: UnifiedUserAction = mkExpectedUUAForActionTowardQuoteEvent(

clientEventNamespace = Some(uuaRetweetClientEventNamespace),

actionType = ActionType.ClientTweetRetweet

)

val expectedTweetRetweetRetweetWithReplyAndQuoteUUA: UnifiedUserAction =

mkExpectedUUAForActionTowardRetweetEventWithReplyAndQuoted(

clientEventNamespace = Some(uuaRetweetClientEventNamespace),

actionType = ActionType.ClientTweetRetweet

)

}

trait EmailNotificationEventFixture extends CommonFixture {

val timestamp = 1001L

val pageUrlStatus =

"https://twitter.com/a/status/3?cn=a%3D%3D&refsrc=email"

val tweetIdStatus = 3L

val pageUrlEvent =

"https://twitter.com/i/events/2?cn=a%3D%3D&refsrc=email"

val tweetIdEvent = 2L

val pageUrlNoArgs = "https://twitter.com/i/events/1"

val tweetIdNoArgs = 1L

val logBase1: LogBase = LogBase(

transactionId = "test",

ipAddress = "127.0.0.1",

userId = Some(userId),

guestId = Some(2L),

timestamp = timestamp,

page = Some(pageUrlStatus),

)

val logBase2: LogBase = LogBase(

transactionId = "test",

ipAddress = "127.0.0.1",

userId = Some(userId),

guestId = Some(2L),

timestamp = timestamp

)

val notificationEvent: NotificationScribe = NotificationScribe(

`type` = NotificationScribeType.Click,

impressionId = Some("1234"),

userId = Some(userId),

timestamp = timestamp,

logBase = Some(logBase1)

)

val notificationEventWOTweetId: NotificationScribe = NotificationScribe(

`type` = NotificationScribeType.Click,

impressionId = Some("1234"),

userId = Some(userId),

timestamp = timestamp,

logBase = Some(logBase2)

)

val notificationEventWOImpressionId: NotificationScribe = NotificationScribe(

`type` = NotificationScribeType.Click,

userId = Some(userId),

timestamp = timestamp,

logBase = Some(logBase1)

)

val expectedUua: UnifiedUserAction = UnifiedUserAction(

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

item = Item.TweetInfo(

TweetInfo(

actionTweetId = tweetIdStatus,

)

),

actionType = ActionType.ClientTweetEmailClick,

eventMetadata = EventMetadata(

sourceTimestampMs = timestamp,

receivedTimestampMs = frozenTime.inMilliseconds,

sourceLineage = SourceLineage.EmailNotificationEvents,

traceId = None

),

productSurfaceInfo = Some(

ProductSurfaceInfo.EmailNotificationInfo(EmailNotificationInfo(notificationId = "1234"))),

productSurface = Some(ProductSurface.EmailNotification)

)

}

trait UserModificationEventFixture extends CommonFixture {

val timestamp = 1001L

val userName = "A"

val screenName = "B"

val description = "this is A"

val location = "US"

val url = s"https://www.twitter.com/${userName}"

val baseUserModification = UserModification(

forUserId = Some(userId),

userId = Some(userId),

)

val userCreate = baseUserModification.copy(

create = Some(

User(

id = userId,

createdAtMsec = timestamp,

updatedAtMsec = timestamp,

userType = UserType.Normal,

profile = Some(

Profile(

name = userName,

screenName = screenName,

description = description,

auth = null.asInstanceOf[Auth],

location = location,

url = url

))

)),

)

val updateDiffs = Seq(

UpdateDiffItem(fieldName = "user\_name", before = Some("abc"), after = Some("def")),

UpdateDiffItem(fieldName = "description", before = Some("d1"), after = Some("d2")),

)

val userUpdate = baseUserModification.copy(

updatedAtMsec = Some(timestamp),

update = Some(updateDiffs),

success = Some(true)

)

val expectedUuaUserCreate: UnifiedUserAction = UnifiedUserAction(

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

item = Item.ProfileInfo(

ProfileInfo(

actionProfileId = userId,

name = Some(userName),

handle = Some(screenName),

description = Some(description)

)

),

actionType = ActionType.ServerUserCreate,

eventMetadata = EventMetadata(

sourceTimestampMs = timestamp,

receivedTimestampMs = frozenTime.inMilliseconds,

sourceLineage = SourceLineage.ServerGizmoduckUserModificationEvents,

)

)

val expectedUuaUserUpdate: UnifiedUserAction = UnifiedUserAction(

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

item = Item.ProfileInfo(

ProfileInfo(

actionProfileId = userId,

profileActionInfo = Some(

ProfileActionInfo.ServerUserUpdate(

ServerUserUpdate(updates = updateDiffs, success = Some(true))))

)

),

actionType = ActionType.ServerUserUpdate,

eventMetadata = EventMetadata(

sourceTimestampMs = timestamp,

receivedTimestampMs = frozenTime.inMilliseconds,

sourceLineage = SourceLineage.ServerGizmoduckUserModificationEvents,

)

)

}

trait AdsCallbackEngagementsFixture extends CommonFixture {

val timestamp = 1001L

val engagementId = 123

val accountTimeZone = "PST"

val advertiserId = 2002L

val displayLocation: DisplayLocation = DisplayLocation(value = 1)

val trendId = 1002

val authorInfo: AuthorInfo = AuthorInfo(authorId = Some(advertiserId))

val openLinkWithUrl: TweetActionInfo =

TweetActionInfo.ServerPromotedTweetOpenLink(ServerPromotedTweetOpenLink(url = Some("go/url")))

val openLinkWithoutUrl: TweetActionInfo =

TweetActionInfo.ServerPromotedTweetOpenLink(ServerPromotedTweetOpenLink(url = None))

def createTweetInfoItem(

authorInfo: Option[AuthorInfo] = None,

actionInfo: Option[TweetActionInfo] = None

): Item = {

Item.TweetInfo(

TweetInfo(

actionTweetId = itemTweetId,

actionTweetAuthorInfo = authorInfo,

tweetActionInfo = actionInfo))

}

val trendInfoItem: Item = Item.TrendInfo(TrendInfo(actionTrendId = trendId))

val organicTweetId = Some(100001L)

val promotedTweetId = Some(200002L)

val organicTweetVideoUuid = Some("organic\_video\_1")

val organicTweetVideoOwnerId = Some(123L)

val promotedTweetVideoUuid = Some("promoted\_video\_1")

val promotedTweetVideoOwnerId = Some(345L)

val prerollAdUuid = Some("preroll\_ad\_1")

val prerollAdOwnerId = Some(567L)

val amplifyDetailsPrerollAd = Some(

AmplifyDetails(

videoOwnerId = prerollAdOwnerId,

videoUuid = prerollAdUuid,

prerollOwnerId = prerollAdOwnerId,

prerollUuid = prerollAdUuid

))

val tweetActionInfoPrerollAd = Some(

TweetActionInfo.TweetVideoWatch(

TweetVideoWatch(

isMonetizable = Some(true),

videoOwnerId = prerollAdOwnerId,

videoUuid = prerollAdUuid,

prerollOwnerId = prerollAdOwnerId,

prerollUuid = prerollAdUuid

)

)

)

val amplifyDetailsPromotedTweetWithoutAd = Some(

AmplifyDetails(

videoOwnerId = promotedTweetVideoOwnerId,

videoUuid = promotedTweetVideoUuid

))

val tweetActionInfoPromotedTweetWithoutAd = Some(

TweetActionInfo.TweetVideoWatch(

TweetVideoWatch(

isMonetizable = Some(true),

videoOwnerId = promotedTweetVideoOwnerId,

videoUuid = promotedTweetVideoUuid,

)

)

)

val amplifyDetailsPromotedTweetWithAd = Some(

AmplifyDetails(

videoOwnerId = promotedTweetVideoOwnerId,

videoUuid = promotedTweetVideoUuid,

prerollOwnerId = prerollAdOwnerId,

prerollUuid = prerollAdUuid

))

val tweetActionInfoPromotedTweetWithAd = Some(

TweetActionInfo.TweetVideoWatch(

TweetVideoWatch(

isMonetizable = Some(true),

videoOwnerId = promotedTweetVideoOwnerId,

videoUuid = promotedTweetVideoUuid,

prerollOwnerId = prerollAdOwnerId,

prerollUuid = prerollAdUuid

)

)

)

val amplifyDetailsOrganicTweetWithAd = Some(

AmplifyDetails(

videoOwnerId = organicTweetVideoOwnerId,

videoUuid = organicTweetVideoUuid,

prerollOwnerId = prerollAdOwnerId,

prerollUuid = prerollAdUuid

))

val tweetActionInfoOrganicTweetWithAd = Some(

TweetActionInfo.TweetVideoWatch(

TweetVideoWatch(

isMonetizable = Some(true),

videoOwnerId = organicTweetVideoOwnerId,

videoUuid = organicTweetVideoUuid,

prerollOwnerId = prerollAdOwnerId,

prerollUuid = prerollAdUuid

)

)

)

def createExpectedUua(

actionType: ActionType,

item: Item

): UnifiedUserAction = {

UnifiedUserAction(

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

item = item,

actionType = actionType,

eventMetadata = EventMetadata(

sourceTimestampMs = timestamp,

receivedTimestampMs = frozenTime.inMilliseconds,

sourceLineage = SourceLineage.ServerAdsCallbackEngagements

)

)

}

def createExpectedUuaWithProfileInfo(

actionType: ActionType

): UnifiedUserAction = {

UnifiedUserAction(

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

item = Item.ProfileInfo(ProfileInfo(actionProfileId = advertiserId)),

actionType = actionType,

eventMetadata = EventMetadata(

sourceTimestampMs = timestamp,

receivedTimestampMs = frozenTime.inMilliseconds,

sourceLineage = SourceLineage.ServerAdsCallbackEngagements

)

)

}

def createSpendServerEvent(

engagementType: EngagementType,

url: Option[String] = None

): SpendServerEvent = {

SpendServerEvent(

engagementEvent = Some(

EngagementEvent(

clientInfo = Some(ClientInfo(userId64 = Some(userId))),

engagementId = engagementId,

engagementEpochTimeMilliSec = timestamp,

engagementType = engagementType,

accountTimeZone = accountTimeZone,

url = url,

impressionData = Some(

ImpressionDataNeededAtEngagementTime(

advertiserId = advertiserId,

promotedTweetId = Some(itemTweetId),

displayLocation = displayLocation,

promotedTrendId = Some(trendId)))

)))

}

def createExpectedVideoUua(

actionType: ActionType,

tweetActionInfo: Option[TweetActionInfo],

actionTweetId: Option[Long]

): UnifiedUserAction = {

UnifiedUserAction(

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

item = Item.TweetInfo(

TweetInfo(

actionTweetId = actionTweetId.get,

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

tweetActionInfo = tweetActionInfo

)

),

actionType = actionType,

eventMetadata = EventMetadata(

sourceTimestampMs = timestamp,

receivedTimestampMs = frozenTime.inMilliseconds,

sourceLineage = SourceLineage.ServerAdsCallbackEngagements

)

)

}

def createVideoSpendServerEvent(

engagementType: EngagementType,

amplifyDetails: Option[AmplifyDetails],

promotedTweetId: Option[Long],

organicTweetId: Option[Long]

): SpendServerEvent = {

SpendServerEvent(

engagementEvent = Some(

EngagementEvent(

clientInfo = Some(ClientInfo(userId64 = Some(userId))),

engagementId = engagementId,

engagementEpochTimeMilliSec = timestamp,

engagementType = engagementType,

accountTimeZone = accountTimeZone,

impressionData = Some(

ImpressionDataNeededAtEngagementTime(

advertiserId = advertiserId,

promotedTweetId = promotedTweetId,

displayLocation = displayLocation,

organicTweetId = organicTweetId)),

cardEngagement = Some(

CardEvent(

amplifyDetails = amplifyDetails

)

)

)))

}

}

trait InteractionEventsFixtures extends CommonFixture {

val timestamp = 123456L

val tweetId = 1L

val engagingUserId = 11L

val baseInteractionEvent: InteractionEvent = InteractionEvent(

targetId = tweetId,

targetType = InteractionTargetType.Tweet,

engagingUserId = engagingUserId,

eventSource = EventSource.ClientEvent,

timestampMillis = timestamp,

interactionType = Some(InteractionType.TweetRenderImpression),

details = InteractionDetails.TweetRenderImpression(TweetImpression()),

additionalEngagingUserIdentifiers = UserIdentifierIE(),

engagingContext = EngagingContext.ClientEventContext(

ClientEventContext(

clientEventNamespace = ContextualEventNamespace(),

clientType = ClientType.Iphone,

displayLocation = DisplayLocation(1),

isTweetDetailsImpression = Some(false)))

)

val loggedOutInteractionEvent: InteractionEvent = baseInteractionEvent.copy(engagingUserId = 0L)

val detailImpressionInteractionEvent: InteractionEvent = baseInteractionEvent.copy(

engagingContext = EngagingContext.ClientEventContext(

ClientEventContext(

clientEventNamespace = ContextualEventNamespace(),

clientType = ClientType.Iphone,

displayLocation = DisplayLocation(1),

isTweetDetailsImpression = Some(true)))

)

val expectedBaseKeyedUuaTweet: KeyedUuaTweet = KeyedUuaTweet(

tweetId = tweetId,

actionType = ActionType.ClientTweetRenderImpression,

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

eventMetadata = EventMetadata(

sourceTimestampMs = timestamp,

receivedTimestampMs = frozenTime.inMilliseconds,

sourceLineage = SourceLineage.ClientEvents

)

)

}

}