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

import com.twitter.clientapp.thriftscala.EventNamespace

import com.twitter.clientapp.thriftscala.LogEvent

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

import com.twitter.logbase.thriftscala.LogBase

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

import com.twitter.unified\_user\_actions.thriftscala.Item.TweetInfo

object ClientEventImpression {

object TweetLingerImpression extends BaseClientEvent(ActionType.ClientTweetLingerImpression) {

override def getUuaItem(

ceItem: LogEventItem,

logEvent: LogEvent

): Option[Item] = {

for {

actionTweetId <- ceItem.id

impressionDetails <- ceItem.impressionDetails

lingerStartTimestampMs <- impressionDetails.visibilityStart

lingerEndTimestampMs <- impressionDetails.visibilityEnd

} yield {

Item.TweetInfo(

ClientEventCommonUtils

.getBasicTweetInfo(actionTweetId, ceItem, logEvent.eventNamespace)

.copy(tweetActionInfo = Some(

TweetActionInfo.ClientTweetLingerImpression(

ClientTweetLingerImpression(

lingerStartTimestampMs = lingerStartTimestampMs,

lingerEndTimestampMs = lingerEndTimestampMs

)

))))

}

}

}

/\*\*

\* To make parity with iesource's definition, render impression for quoted Tweets would emit

\* 2 events: 1 for the quoting Tweet and 1 for the original Tweet!!!

\*/

object TweetRenderImpression extends BaseClientEvent(ActionType.ClientTweetRenderImpression) {

override def toUnifiedUserAction(logEvent: LogEvent): Seq[UnifiedUserAction] = {

val logBase: Option[LogBase] = logEvent.logBase

val raw = for {

ed <- logEvent.eventDetails.toSeq

items <- ed.items.toSeq

ceItem <- items

eventTimestamp <- logBase.flatMap(getSourceTimestamp)

uuaItem <- getUuaItem(ceItem, logEvent)

if isItemTypeValid(ceItem.itemType)

} yield {

val userIdentifier: UserIdentifier = UserIdentifier(

userId = logBase.flatMap(\_.userId),

guestIdMarketing = logBase.flatMap(\_.guestIdMarketing))

val productSurface: Option[ProductSurface] = ProductSurfaceUtils

.getProductSurface(logEvent.eventNamespace)

val eventMetaData: EventMetadata = ClientEventCommonUtils

.getEventMetadata(

eventTimestamp = eventTimestamp,

logEvent = logEvent,

ceItem = ceItem,

productSurface = productSurface

)

UnifiedUserAction(

userIdentifier = userIdentifier,

item = uuaItem,

actionType = ActionType.ClientTweetRenderImpression,

eventMetadata = eventMetaData,

productSurface = productSurface,

productSurfaceInfo =

ProductSurfaceUtils.getProductSurfaceInfo(productSurface, ceItem, logEvent)

)

}

raw.flatMap { e =>

e.item match {

case TweetInfo(t) =>

// If it is an impression toward quoted Tweet we emit 2 impressions, 1 for quoting Tweet

// and 1 for the original Tweet.

if (t.quotedTweetId.isDefined) {

val originalItem = t.copy(

actionTweetId = t.quotedTweetId.get,

actionTweetAuthorInfo = t.quotedAuthorId.map(id => AuthorInfo(authorId = Some(id))),

quotingTweetId = Some(t.actionTweetId),

quotedTweetId = None,

inReplyToTweetId = None,

replyingTweetId = None,

retweetingTweetId = None,

retweetedTweetId = None,

quotedAuthorId = None,

retweetingAuthorId = None,

inReplyToAuthorId = None

)

val original = e.copy(item = TweetInfo(originalItem))

Seq(original, e)

} else Seq(e)

case \_ => Nil

}

}

}

}

object TweetGalleryImpression extends BaseClientEvent(ActionType.ClientTweetGalleryImpression)

object TweetDetailsImpression extends BaseClientEvent(ActionType.ClientTweetDetailsImpression) {

case class EventNamespaceInternal(

client: String,

page: String,

section: String,

component: String,

element: String,

action: String)

def isTweetDetailsImpression(eventNamespaceOpt: Option[EventNamespace]): Boolean =

eventNamespaceOpt.exists { eventNamespace =>

val eventNamespaceInternal = EventNamespaceInternal(

client = eventNamespace.client.getOrElse(""),

page = eventNamespace.page.getOrElse(""),

section = eventNamespace.section.getOrElse(""),

component = eventNamespace.component.getOrElse(""),

element = eventNamespace.element.getOrElse(""),

action = eventNamespace.action.getOrElse(""),

)

isIphoneAppOrMacAppOrIpadAppClientTweetDetailsImpression(

eventNamespaceInternal) || isAndroidAppClientTweetDetailsImpression(

eventNamespaceInternal) || isWebClientTweetDetailImpression(

eventNamespaceInternal) || isTweetDeckAppClientTweetDetailsImpression(

eventNamespaceInternal) || isOtherAppClientTweetDetailsImpression(eventNamespaceInternal)

}

private def isWebClientTweetDetailImpression(

eventNamespace: EventNamespaceInternal

): Boolean = {

val eventNameSpaceStr =

eventNamespace.client + ":" + eventNamespace.page + ":" + eventNamespace.section + ":" + eventNamespace.component + ":" + eventNamespace.element + ":" + eventNamespace.action

eventNameSpaceStr.equalsIgnoreCase("m5:tweet::::show") || eventNameSpaceStr.equalsIgnoreCase(

"m5:tweet:landing:::show") || eventNameSpaceStr

.equalsIgnoreCase("m2:tweet::::impression") || eventNameSpaceStr.equalsIgnoreCase(

"m2:tweet::tweet::impression") || eventNameSpaceStr

.equalsIgnoreCase("LiteNativeWrapper:tweet::::show") || eventNameSpaceStr.equalsIgnoreCase(

"LiteNativeWrapper:tweet:landing:::show")

}

private def isOtherAppClientTweetDetailsImpression(

eventNamespace: EventNamespaceInternal

): Boolean = {

val excludedClients = Set(

"web",

"m5",

"m2",

"LiteNativeWrapper",

"iphone",

"ipad",

"mac",

"android",

"android\_tablet",

"deck")

(!excludedClients.contains(eventNamespace.client)) && eventNamespace.page

.equalsIgnoreCase("tweet") && eventNamespace.section

.equalsIgnoreCase("") && eventNamespace.component

.equalsIgnoreCase("tweet") && eventNamespace.element

.equalsIgnoreCase("") && eventNamespace.action.equalsIgnoreCase("impression")

}

private def isTweetDeckAppClientTweetDetailsImpression(

eventNamespace: EventNamespaceInternal

): Boolean =

eventNamespace.client

.equalsIgnoreCase("deck") && eventNamespace.page

.equalsIgnoreCase("tweet") && eventNamespace.section

.equalsIgnoreCase("") && eventNamespace.component

.equalsIgnoreCase("tweet") && eventNamespace.element

.equalsIgnoreCase("") && eventNamespace.action.equalsIgnoreCase("impression")

private def isAndroidAppClientTweetDetailsImpression(

eventNamespace: EventNamespaceInternal

): Boolean =

(eventNamespace.client

.equalsIgnoreCase("android") || eventNamespace.client

.equalsIgnoreCase("android\_tablet")) && eventNamespace.page

.equalsIgnoreCase("tweet") && eventNamespace.section.equalsIgnoreCase(

"") && (eventNamespace.component

.equalsIgnoreCase("tweet") || eventNamespace.component

.matches("^suggest.\*\_tweet.\*$") || eventNamespace.component

.equalsIgnoreCase("")) && eventNamespace.element

.equalsIgnoreCase("") && eventNamespace.action.equalsIgnoreCase("impression")

private def isIphoneAppOrMacAppOrIpadAppClientTweetDetailsImpression(

eventNamespace: EventNamespaceInternal

): Boolean =

(eventNamespace.client

.equalsIgnoreCase("iphone") || eventNamespace.client

.equalsIgnoreCase("ipad") || eventNamespace.client

.equalsIgnoreCase("mac")) && eventNamespace.page.equalsIgnoreCase(

"tweet") && eventNamespace.section

.equalsIgnoreCase("") && (eventNamespace.component

.equalsIgnoreCase("tweet") || eventNamespace.component

.matches("^suggest.\*\_tweet.\*$")) && eventNamespace.element

.equalsIgnoreCase("") && eventNamespace.action.equalsIgnoreCase("impression")

}

}