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

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.ActionType

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

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

abstract class BaseUASClientEvent(actionType: ActionType)

extends BaseClientEvent(actionType = actionType) {

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

val logBase: Option[LogBase] = logEvent.logBase

val ceItem = LogEventItem.unsafeEmpty

val uuaOpt: Option[UnifiedUserAction] = for {

eventTimestamp <- logBase.flatMap(getSourceTimestamp)

uuaItem <- getUuaItem(ceItem, logEvent)

} 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,

eventMetadata = eventMetaData,

productSurface = productSurface,

productSurfaceInfo =

ProductSurfaceUtils.getProductSurfaceInfo(productSurface, ceItem, logEvent)

)

}

uuaOpt match {

case Some(uua) => Seq(uua)

case \_ => Nil

}

}

override def getUuaItem(

ceItem: LogEventItem,

logEvent: LogEvent

): Option[Item] = for {

performanceDetails <- logEvent.performanceDetails

duration <- performanceDetails.durationMs

} yield {

Item.UasInfo(UASInfo(timeSpentMs = duration))

}

}