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

import com.twitter.clientapp.thriftscala.ItemType

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.adapter.client\_event.ClientEventCommonUtils.getProfileIdFromUserItem

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

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

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

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

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

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

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

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

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

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

abstract class BaseSearchTypeaheadEvent(actionType: ActionType)

extends BaseClientEvent(actionType = actionType) {

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

val logBase: Option[LogBase] = logEvent.logBase

for {

ed <- logEvent.eventDetails.toSeq

targets <- ed.targets.toSeq

ceTarget <- targets

eventTimestamp <- logBase.flatMap(getSourceTimestamp)

uuaItem <- getUuaItem(ceTarget, logEvent)

if isItemTypeValid(ceTarget.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 = ceTarget,

productSurface = productSurface

)

UnifiedUserAction(

userIdentifier = userIdentifier,

item = uuaItem,

actionType = actionType,

eventMetadata = eventMetaData,

productSurface = productSurface,

productSurfaceInfo =

ProductSurfaceUtils.getProductSurfaceInfo(productSurface, ceTarget, logEvent)

)

}

}

override def isItemTypeValid(itemTypeOpt: Option[ItemType]): Boolean =

ItemTypeFilterPredicates.isItemTypeTypeaheadResult(itemTypeOpt)

override def getUuaItem(

ceTarget: LogEventItem,

logEvent: LogEvent

): Option[Item] =

logEvent.searchDetails.flatMap(\_.query).flatMap { query =>

ceTarget.itemType match {

case Some(ItemType.User) =>

getProfileIdFromUserItem(ceTarget).map { profileId =>

Item.TypeaheadInfo(

TypeaheadInfo(

actionQuery = query,

typeaheadActionInfo =

TypeaheadActionInfo.UserResult(UserResult(profileId = profileId))))

}

case Some(ItemType.Search) =>

ceTarget.name.map { name =>

Item.TypeaheadInfo(

TypeaheadInfo(

actionQuery = query,

typeaheadActionInfo = TypeaheadActionInfo.TopicQueryResult(

TopicQueryResult(suggestedTopicQuery = name))))

}

case \_ => None

}

}

}