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.suggests.controller\_data.home\_tweets.thriftscala.HomeTweetsControllerDataAliases.V1Alias

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

object ProductSurfaceUtils {

def getProductSurface(eventNamespace: Option[EventNamespace]): Option[ProductSurface] = {

(

eventNamespace.flatMap(\_.page),

eventNamespace.flatMap(\_.section),

eventNamespace.flatMap(\_.element)) match {

case (Some("home") | Some("home\_latest"), \_, \_) => Some(ProductSurface.HomeTimeline)

case (Some("ntab"), \_, \_) => Some(ProductSurface.NotificationTab)

case (Some(page), Some(section), \_) if isPushNotification(page, section) =>

Some(ProductSurface.PushNotification)

case (Some("search"), \_, \_) => Some(ProductSurface.SearchResultsPage)

case (\_, \_, Some("typeahead")) => Some(ProductSurface.SearchTypeahead)

case \_ => None

}

}

private def isPushNotification(page: String, section: String): Boolean = {

Seq[String]("notification", "toasts").contains(page) ||

(page == "app" && section == "push")

}

def getProductSurfaceInfo(

productSurface: Option[ProductSurface],

ceItem: LogEventItem,

logEvent: LogEvent

): Option[ProductSurfaceInfo] = {

productSurface match {

case Some(ProductSurface.HomeTimeline) => createHomeTimelineInfo(ceItem)

case Some(ProductSurface.NotificationTab) => createNotificationTabInfo(ceItem)

case Some(ProductSurface.PushNotification) => createPushNotificationInfo(logEvent)

case Some(ProductSurface.SearchResultsPage) => createSearchResultPageInfo(ceItem, logEvent)

case Some(ProductSurface.SearchTypeahead) => createSearchTypeaheadInfo(ceItem, logEvent)

case \_ => None

}

}

private def createPushNotificationInfo(logEvent: LogEvent): Option[ProductSurfaceInfo] =

NotificationClientEventUtils.getNotificationIdForPushNotification(logEvent) match {

case Some(notificationId) =>

Some(

ProductSurfaceInfo.PushNotificationInfo(

PushNotificationInfo(notificationId = notificationId)))

case \_ => None

}

private def createNotificationTabInfo(ceItem: LogEventItem): Option[ProductSurfaceInfo] =

NotificationClientEventUtils.getNotificationIdForNotificationTab(ceItem) match {

case Some(notificationId) =>

Some(

ProductSurfaceInfo.NotificationTabInfo(

NotificationTabInfo(notificationId = notificationId)))

case \_ => None

}

private def createHomeTimelineInfo(ceItem: LogEventItem): Option[ProductSurfaceInfo] = {

def suggestType: Option[String] = HomeInfoUtils.getSuggestType(ceItem)

def controllerData: Option[V1Alias] = HomeInfoUtils.getHomeTweetControllerDataV1(ceItem)

if (suggestType.isDefined || controllerData.isDefined) {

Some(

ProductSurfaceInfo.HomeTimelineInfo(

HomeTimelineInfo(

suggestionType = suggestType,

injectedPosition = controllerData.flatMap(\_.injectedPosition)

)))

} else None

}

private def createSearchResultPageInfo(

ceItem: LogEventItem,

logEvent: LogEvent

): Option[ProductSurfaceInfo] = {

val searchInfoUtil = new SearchInfoUtils(ceItem)

searchInfoUtil.getQueryOptFromItem(logEvent).map { query =>

ProductSurfaceInfo.SearchResultsPageInfo(

SearchResultsPageInfo(

query = query,

querySource = searchInfoUtil.getQuerySourceOptFromControllerDataFromItem,

itemPosition = ceItem.position,

tweetResultSources = searchInfoUtil.getTweetResultSources,

userResultSources = searchInfoUtil.getUserResultSources,

queryFilterType = searchInfoUtil.getQueryFilterType(logEvent)

))

}

}

private def createSearchTypeaheadInfo(

ceItem: LogEventItem,

logEvent: LogEvent

): Option[ProductSurfaceInfo] = {

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

ProductSurfaceInfo.SearchTypeaheadInfo(

SearchTypeaheadInfo(

query = query,

itemPosition = ceItem.position

)

)

}

}

}