package com.twitter.product\_mixer.component\_library.pipeline.candidate.ads

import com.twitter.adserver.{thriftscala => ads}

import com.twitter.product\_mixer.component\_library.model.query.ads.AdsQuery

import com.twitter.product\_mixer.component\_library.pipeline.candidate.ads.AdsCandidatePipelineQueryTransformer.buildAdRequestParams

import com.twitter.product\_mixer.core.functional\_component.transformer.CandidatePipelineQueryTransformer

import com.twitter.product\_mixer.core.pipeline.PipelineQuery

/\*\*

\* Transform a PipelineQuery with AdsQuery into an AdsRequestParams

\*

\* @param adsDisplayLocationBuilder Builder that determines the display location for the ads

\* @param estimatedNumOrganicItems Estimate for the number of organic items that will be served

\* alongside inorganic items such as ads.

\*/

case class AdsCandidatePipelineQueryTransformer[Query <: PipelineQuery with AdsQuery](

adsDisplayLocationBuilder: AdsDisplayLocationBuilder[Query],

estimatedNumOrganicItems: EstimateNumOrganicItems[Query],

urtRequest: Option[Boolean],

) extends CandidatePipelineQueryTransformer[Query, ads.AdRequestParams] {

override def transform(query: Query): ads.AdRequestParams =

buildAdRequestParams(

query = query,

adsDisplayLocation = adsDisplayLocationBuilder(query),

organicItemIds = None,

numOrganicItems = Some(estimatedNumOrganicItems(query)),

urtRequest = urtRequest

)

}

object AdsCandidatePipelineQueryTransformer {

def buildAdRequestParams(

query: PipelineQuery with AdsQuery,

adsDisplayLocation: ads.DisplayLocation,

organicItemIds: Option[Seq[Long]],

numOrganicItems: Option[Short],

urtRequest: Option[Boolean],

): ads.AdRequestParams = {

val searchRequestContext = query.searchRequestContext

val queryString = query.searchRequestContext.flatMap(\_.queryString)

val adRequest = ads.AdRequest(

queryString = queryString,

displayLocation = adsDisplayLocation,

searchRequestContext = searchRequestContext,

organicItemIds = organicItemIds,

numOrganicItems = numOrganicItems,

profileUserId = query.userProfileViewedUserId,

isDebug = Some(false),

isTest = Some(false),

requestTriggerType = query.requestTriggerType,

disableNsfwAvoidance = query.disableNsfwAvoidance,

timelineRequestParams = query.timelineRequestParams,

)

val context = query.clientContext

val clientInfo = ads.ClientInfo(

clientId = context.appId.map(\_.toInt),

userId64 = context.userId,

userIp = context.ipAddress,

guestId = context.guestIdAds,

userAgent = context.userAgent,

deviceId = context.deviceId,

languageCode = context.languageCode,

countryCode = context.countryCode,

mobileDeviceId = context.mobileDeviceId,

mobileDeviceAdId = context.mobileDeviceAdId,

limitAdTracking = context.limitAdTracking,

autoplayEnabled = query.autoplayEnabled,

urtRequest = urtRequest,

dspClientContext = query.dspClientContext

)

ads.AdRequestParams(adRequest, clientInfo)

}

}