package com.twitter.product\_mixer.component\_library.decorator.urt.builder.promoted

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

import com.twitter.ads.common.base.{thriftscala => ac}

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

import com.twitter.product\_mixer.core.feature.Feature

import com.twitter.product\_mixer.core.feature.featuremap.FeatureMap

import com.twitter.product\_mixer.core.functional\_component.decorator.urt.builder.promoted.BasePromotedMetadataBuilder

import com.twitter.product\_mixer.core.model.common.UniversalNoun

import com.twitter.product\_mixer.core.model.marshalling.response.urt.promoted.\_

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

import com.twitter.timelines.util.AdMetadataContainerSerializer

case class FeaturePromotedMetadataBuilder(adImpressionFeature: Feature[\_, Option[ad.AdImpression]])

extends BasePromotedMetadataBuilder[PipelineQuery, UniversalNoun[Any]] {

def apply(

query: PipelineQuery,

candidate: UniversalNoun[Any],

candidateFeatures: FeatureMap

): Option[PromotedMetadata] = {

candidateFeatures.getOrElse(adImpressionFeature, None).map { impression =>

PromotedMetadata(

advertiserId = impression.advertiserId,

disclosureType = impression.disclosureType.map(convertDisclosureType),

experimentValues = impression.experimentValues.map(\_.toMap),

promotedTrendId = impression.promotedTrendId.map(\_.toLong),

promotedTrendName = impression.promotedTrendName,

promotedTrendQueryTerm = impression.promotedTrendQueryTerm,

adMetadataContainer =

impression.serializedAdMetadataContainer.flatMap(convertAdMetadataContainer),

promotedTrendDescription = impression.promotedTrendDescription,

impressionString = impression.impressionString,

clickTrackingInfo = impression.clickTrackingInfo.map(convertClickTrackingInfo),

)

}

}

private def convertAdMetadataContainer(

serializedAdMetadataContainer: ac.SerializedThrift

): Option[AdMetadataContainer] =

AdMetadataContainerSerializer.deserialize(serializedAdMetadataContainer).map { container =>

AdMetadataContainer(

removePromotedAttributionForPreroll = container.removePromotedAttributionForPreroll,

sponsorshipCandidate = container.sponsorshipCandidate,

sponsorshipOrganization = container.sponsorshipOrganization,

sponsorshipOrganizationWebsite = container.sponsorshipOrganizationWebsite,

sponsorshipType = container.sponsorshipType.map(convertSponsorshipType),

disclaimerType = container.disclaimerType.map(convertDisclaimerType),

skAdNetworkDataList = container.skAdNetworkDataList.map(convertSkAdNetworkDataList),

unifiedCardOverride = container.unifiedCardOverride

)

}

private def convertDisclosureType(disclosureType: ad.DisclosureType): DisclosureType =

disclosureType match {

case ad.DisclosureType.None => NoDisclosure

case ad.DisclosureType.Political => Political

case ad.DisclosureType.Earned => Earned

case ad.DisclosureType.Issue => Issue

case \_ => throw new UnsupportedOperationException(s"Unsupported: $disclosureType")

}

private def convertSponsorshipType(sponsorshipType: ads.SponsorshipType): SponsorshipType =

sponsorshipType match {

case ads.SponsorshipType.Direct => DirectSponsorshipType

case ads.SponsorshipType.Indirect => IndirectSponsorshipType

case ads.SponsorshipType.NoSponsorship => NoSponsorshipSponsorshipType

case \_ => throw new UnsupportedOperationException(s"Unsupported: $sponsorshipType")

}

private def convertDisclaimerType(disclaimerType: ads.DisclaimerType): DisclaimerType =

disclaimerType match {

case ads.DisclaimerType.Political => DisclaimerPolitical

case ads.DisclaimerType.Issue => DisclaimerIssue

case \_ => throw new UnsupportedOperationException(s"Unsupported: $disclaimerType")

}

private def convertSkAdNetworkDataList(

skAdNetworkDataList: Seq[ads.SkAdNetworkData]

): Seq[SkAdNetworkData] = skAdNetworkDataList.map { sdAdNetwork =>

SkAdNetworkData(

version = sdAdNetwork.version,

srcAppId = sdAdNetwork.srcAppId,

dstAppId = sdAdNetwork.dstAppId,

adNetworkId = sdAdNetwork.adNetworkId,

campaignId = sdAdNetwork.campaignId,

impressionTimeInMillis = sdAdNetwork.impressionTimeInMillis,

nonce = sdAdNetwork.nonce,

signature = sdAdNetwork.signature,

fidelityType = sdAdNetwork.fidelityType

)

}

private def convertClickTrackingInfo(clickTracking: ad.ClickTrackingInfo): ClickTrackingInfo =

ClickTrackingInfo(

urlParams = clickTracking.urlParams.getOrElse(Map.empty),

urlOverride = clickTracking.urlOverride,

urlOverrideType = clickTracking.urlOverrideType.map {

case ad.UrlOverrideType.Unknown => UnknownUrlOverrideType

case ad.UrlOverrideType.Dcm => DcmUrlOverrideType

case ad.UrlOverrideType.EnumUnknownUrlOverrideType(value) =>

throw new UnsupportedOperationException(s"Unsupported: $value")

}

)

}