package com.twitter.visibility.interfaces.tweets.enrichments

import com.twitter.featureswitches.FSRecipient

import com.twitter.featureswitches.v2.FeatureSwitches

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.visibility.builder.VisibilityResult

import com.twitter.visibility.common.LocalizedLimitedActionsSource

import com.twitter.visibility.common.actions.converter.scala.LimitedActionTypeConverter

import com.twitter.visibility.common.actions.LimitedActionsPolicy

import com.twitter.visibility.common.actions.LimitedActionType

import com.twitter.visibility.common.actions.LimitedEngagementReason

import com.twitter.visibility.rules.Action

import com.twitter.visibility.rules.EmergencyDynamicInterstitial

import com.twitter.visibility.rules.InterstitialLimitedEngagements

import com.twitter.visibility.rules.LimitedEngagements

case class PolicyFeatureSwitchResults(

limitedActionTypes: Option[Seq[LimitedActionType]],

copyNamespace: String,

promptType: String,

learnMoreUrl: Option[String])

object LimitedActionsPolicyEnrichment {

object FeatureSwitchKeys {

val LimitedActionTypes = "limited\_actions\_policy\_limited\_actions"

val CopyNamespace = "limited\_actions\_policy\_copy\_namespace"

val PromptType = "limited\_actions\_policy\_prompt\_type"

val LearnMoreUrl = "limited\_actions\_policy\_prompt\_learn\_more\_url"

}

val DefaultCopyNameSpace = "Default"

val DefaultPromptType = "basic"

val LimitedActionsPolicyEnrichmentScope = "limited\_actions\_policy\_enrichment"

val MissingLimitedActionTypesScope = "missing\_limited\_action\_types"

val ExecutedScope = "executed"

def apply(

result: VisibilityResult,

localizedLimitedActionSource: LocalizedLimitedActionsSource,

languageCode: String,

countryCode: Option[String],

featureSwitches: FeatureSwitches,

statsReceiver: StatsReceiver

): VisibilityResult = {

val scopedStatsReceiver = statsReceiver.scope(LimitedActionsPolicyEnrichmentScope)

val enrichVerdict\_ = enrichVerdict(

\_: Action,

localizedLimitedActionSource,

languageCode,

countryCode,

featureSwitches,

scopedStatsReceiver

)

result.copy(

verdict = enrichVerdict\_(result.verdict),

secondaryVerdicts = result.secondaryVerdicts.map(enrichVerdict\_)

)

}

private def enrichVerdict(

verdict: Action,

localizedLimitedActionsSource: LocalizedLimitedActionsSource,

languageCode: String,

countryCode: Option[String],

featureSwitches: FeatureSwitches,

statsReceiver: StatsReceiver

): Action = {

val limitedActionsPolicyForReason\_ = limitedActionsPolicyForReason(

\_: LimitedEngagementReason,

localizedLimitedActionsSource,

languageCode,

countryCode,

featureSwitches,

statsReceiver

)

val executedCounter = statsReceiver.scope(ExecutedScope)

verdict match {

case le: LimitedEngagements => {

executedCounter.counter("").incr()

executedCounter.counter(le.name).incr()

le.copy(

policy = limitedActionsPolicyForReason\_(le.getLimitedEngagementReason)

)

}

case ile: InterstitialLimitedEngagements => {

executedCounter.counter("").incr()

executedCounter.counter(ile.name).incr()

ile.copy(

policy = limitedActionsPolicyForReason\_(

ile.getLimitedEngagementReason

)

)

}

case edi: EmergencyDynamicInterstitial => {

executedCounter.counter("").incr()

executedCounter.counter(edi.name).incr()

EmergencyDynamicInterstitial(

copy = edi.copy,

linkOpt = edi.linkOpt,

localizedMessage = edi.localizedMessage,

policy = limitedActionsPolicyForReason\_(edi.getLimitedEngagementReason)

)

}

case \_ => verdict

}

}

private def limitedActionsPolicyForReason(

reason: LimitedEngagementReason,

localizedLimitedActionsSource: LocalizedLimitedActionsSource,

languageCode: String,

countryCode: Option[String],

featureSwitches: FeatureSwitches,

statsReceiver: StatsReceiver

): Option[LimitedActionsPolicy] = {

val policyConfig = getPolicyFeatureSwitchResults(featureSwitches, reason)

policyConfig.limitedActionTypes match {

case Some(limitedActionTypes) if limitedActionTypes.nonEmpty =>

Some(

LimitedActionsPolicy(

limitedActionTypes.map(

localizedLimitedActionsSource.fetch(

\_,

languageCode,

countryCode,

policyConfig.promptType,

policyConfig.copyNamespace,

policyConfig.learnMoreUrl

)

)

)

)

case \_ => {

statsReceiver

.scope(MissingLimitedActionTypesScope).counter(reason.toLimitedActionsString).incr()

None

}

}

}

private def getPolicyFeatureSwitchResults(

featureSwitches: FeatureSwitches,

reason: LimitedEngagementReason

): PolicyFeatureSwitchResults = {

val recipient = FSRecipient().withCustomFields(

("LimitedEngagementReason", reason.toLimitedActionsString)

)

val featureSwitchesResults = featureSwitches

.matchRecipient(recipient)

val limitedActionTypes = featureSwitchesResults

.getStringArray(FeatureSwitchKeys.LimitedActionTypes)

.map(\_.map(LimitedActionTypeConverter.fromString).flatten)

val copyNamespace = featureSwitchesResults

.getString(FeatureSwitchKeys.CopyNamespace)

.getOrElse(DefaultCopyNameSpace)

val promptType = featureSwitchesResults

.getString(FeatureSwitchKeys.PromptType)

.getOrElse(DefaultPromptType)

val learnMoreUrl = featureSwitchesResults

.getString(FeatureSwitchKeys.LearnMoreUrl)

.filter(\_.nonEmpty)

PolicyFeatureSwitchResults(limitedActionTypes, copyNamespace, promptType, learnMoreUrl)

}

}