package com.twitter.visibility.rules

import com.twitter.visibility.common.actions.ComplianceTweetNoticeEventType

import com.twitter.visibility.configapi.params.RuleParam

import com.twitter.visibility.configapi.params.RuleParams.EnableSearchIpiSafeSearchWithoutUserInQueryDropRule

import com.twitter.visibility.features.Feature

import com.twitter.visibility.features.TweetSafetyLabels

import com.twitter.visibility.models.LabelSource.StringSource

import com.twitter.visibility.models.LabelSource.parseStringSource

import com.twitter.visibility.models.TweetSafetyLabel

import com.twitter.visibility.models.TweetSafetyLabelType

import com.twitter.visibility.rules.Condition.And

import com.twitter.visibility.rules.Condition.LoggedOutOrViewerOptInFiltering

import com.twitter.visibility.rules.Condition.Not

import com.twitter.visibility.rules.Condition.SearchQueryHasUser

import com.twitter.visibility.rules.Condition.TweetHasLabel

import com.twitter.visibility.rules.Reason.Unspecified

object EmergencyDynamicInterstitialActionBuilder

extends ActionBuilder[EmergencyDynamicInterstitial] {

def actionType: Class[\_] = classOf[EmergencyDynamicInterstitial]

override val actionSeverity = 11

override def build(

evaluationContext: EvaluationContext,

featureMap: Map[Feature[\_], \_]

): RuleResult = {

val label = featureMap(TweetSafetyLabels)

.asInstanceOf[Seq[TweetSafetyLabel]]

.find(slv => slv.labelType == TweetSafetyLabelType.ForEmergencyUseOnly)

label.flatMap(\_.source) match {

case Some(StringSource(name)) =>

val (copy, linkOpt) = parseStringSource(name)

RuleResult(EmergencyDynamicInterstitial(copy, linkOpt), State.Evaluated)

case \_ =>

Rule.EvaluatedRuleResult

}

}

}

object EmergencyDynamicComplianceTweetNoticeActionBuilder

extends ActionBuilder[ComplianceTweetNoticePreEnrichment] {

def actionType: Class[\_] = classOf[ComplianceTweetNoticePreEnrichment]

override val actionSeverity = 2

override def build(

evaluationContext: EvaluationContext,

featureMap: Map[Feature[\_], \_]

): RuleResult = {

val label = featureMap(TweetSafetyLabels)

.asInstanceOf[Seq[TweetSafetyLabel]]

.find(slv => slv.labelType == TweetSafetyLabelType.ForEmergencyUseOnly)

label.flatMap(\_.source) match {

case Some(StringSource(name)) =>

val (copy, linkOpt) = parseStringSource(name)

RuleResult(

ComplianceTweetNoticePreEnrichment(

reason = Unspecified,

complianceTweetNoticeEventType = ComplianceTweetNoticeEventType.PublicInterest,

details = Some(copy),

extendedDetailsUrl = linkOpt

),

State.Evaluated

)

case \_ =>

Rule.EvaluatedRuleResult

}

}

}

object EmergencyDynamicInterstitialRule

extends Rule(

EmergencyDynamicInterstitialActionBuilder,

TweetHasLabel(TweetSafetyLabelType.ForEmergencyUseOnly)

)

object EmergencyDropRule

extends RuleWithConstantAction(

Drop(Unspecified),

TweetHasLabel(TweetSafetyLabelType.ForEmergencyUseOnly)

)

object SearchEdiSafeSearchWithoutUserInQueryDropRule

extends RuleWithConstantAction(

Drop(Unspecified),

And(

TweetHasLabel(TweetSafetyLabelType.ForEmergencyUseOnly),

LoggedOutOrViewerOptInFiltering,

Not(SearchQueryHasUser)

)

) {

override def enabled: Seq[RuleParam[Boolean]] = Seq(

EnableSearchIpiSafeSearchWithoutUserInQueryDropRule)

}