package com.twitter.visibility.rules

import com.twitter.escherbird.thriftscala.TweetEntityAnnotation

import com.twitter.gizmoduck.thriftscala.Label

import com.twitter.spam.rtf.thriftscala.BotMakerAction

import com.twitter.spam.rtf.thriftscala.SafetyLabelSource

import com.twitter.spam.rtf.thriftscala.SemanticCoreAction

import com.twitter.visibility.common.actions.EscherbirdAnnotation

import com.twitter.visibility.common.actions.SoftInterventionReason

import com.twitter.visibility.configapi.configs.DeciderKey

import com.twitter.visibility.features.AuthorUserLabels

import com.twitter.visibility.features.Feature

import com.twitter.visibility.features.TweetSafetyLabels

import com.twitter.visibility.logging.thriftscala.ActionSource

import com.twitter.visibility.models.LabelSource.\_

import com.twitter.visibility.models.TweetSafetyLabel

import com.twitter.visibility.models.TweetSafetyLabelType

import com.twitter.visibility.models.UserLabel

import com.twitter.visibility.models.UserLabelValue

sealed trait RuleActionSourceBuilder {

def build(resolvedFeatureMap: Map[Feature[\_], Any], verdict: Action): Option[ActionSource]

}

object RuleActionSourceBuilder {

case class TweetSafetyLabelSourceBuilder(tweetSafetyLabelType: TweetSafetyLabelType)

extends RuleActionSourceBuilder {

override def build(

resolvedFeatureMap: Map[Feature[\_], Any],

verdict: Action

): Option[ActionSource] = {

resolvedFeatureMap

.getOrElse(TweetSafetyLabels, Seq.empty[TweetSafetyLabel])

.asInstanceOf[Seq[TweetSafetyLabel]]

.find(\_.labelType == tweetSafetyLabelType)

.flatMap(\_.safetyLabelSource)

.map(ActionSource.SafetyLabelSource(\_))

}

}

case class UserSafetyLabelSourceBuilder(userLabel: UserLabelValue)

extends RuleActionSourceBuilder {

override def build(

resolvedFeatureMap: Map[Feature[\_], Any],

verdict: Action

): Option[ActionSource] = {

resolvedFeatureMap

.getOrElse(AuthorUserLabels, Seq.empty[Label])

.asInstanceOf[Seq[Label]]

.map(UserLabel.fromThrift)

.find(\_.labelValue == userLabel)

.flatMap(\_.source)

.collect {

case BotMakerRule(ruleId) =>

ActionSource.SafetyLabelSource(SafetyLabelSource.BotMakerAction(BotMakerAction(ruleId)))

}

}

}

case class SemanticCoreActionSourceBuilder() extends RuleActionSourceBuilder {

override def build(

resolvedFeatureMap: Map[Feature[\_], Any],

verdict: Action

): Option[ActionSource] = {

verdict match {

case softIntervention: SoftIntervention =>

getSemanticCoreActionSourceOption(softIntervention)

case tweetInterstitial: TweetInterstitial =>

tweetInterstitial.softIntervention.flatMap(getSemanticCoreActionSourceOption)

case \_ => None

}

}

def getSemanticCoreActionSourceOption(

softIntervention: SoftIntervention

): Option[ActionSource] = {

val siReason = softIntervention.reason

.asInstanceOf[SoftInterventionReason.EscherbirdAnnotations]

val firstAnnotation: Option[EscherbirdAnnotation] =

siReason.escherbirdAnnotations.headOption

firstAnnotation.map { annotation =>

ActionSource.SafetyLabelSource(

SafetyLabelSource.SemanticCoreAction(SemanticCoreAction(

TweetEntityAnnotation(annotation.groupId, annotation.domainId, annotation.entityId))))

}

}

}

}

trait DoesLogVerdict {}

trait DoesLogVerdictDecidered extends DoesLogVerdict {

def verdictLogDeciderKey: DeciderKey.Value

}