package com.twitter.visibility.builder

import com.twitter.spam.rtf.thriftscala.SafetyResult

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

import com.twitter.visibility.rules.ComposableActions.\_

import com.twitter.visibility.features.Feature

import com.twitter.visibility.features.FeatureMap

import com.twitter.visibility.models.ContentId

import com.twitter.visibility.rules.\_

import com.twitter.visibility.{thriftscala => t}

case class VisibilityResult(

contentId: ContentId,

featureMap: FeatureMap = FeatureMap.empty,

ruleResultMap: Map[Rule, RuleResult] = Map.empty,

verdict: Action = Allow,

finished: Boolean = false,

actingRule: Option[Rule] = None,

secondaryActingRules: Seq[Rule] = Seq(),

secondaryVerdicts: Seq[Action] = Seq(),

resolvedFeatureMap: Map[Feature[\_], Any] = Map.empty) {

def getSafetyResult: SafetyResult =

verdict match {

case InterstitialLimitedEngagements(reason: Reason, \_, \_, \_)

if PublicInterest.Reasons

.contains(reason) =>

SafetyResult(

Some(PublicInterest.ReasonToSafetyResultReason(reason)),

verdict.toActionThrift()

)

case ComposableActionsWithInterstitialLimitedEngagements(tweetInterstitial)

if PublicInterest.Reasons.contains(tweetInterstitial.reason) =>

SafetyResult(

Some(PublicInterest.ReasonToSafetyResultReason(tweetInterstitial.reason)),

verdict.toActionThrift()

)

case FreedomOfSpeechNotReachReason(appealableReason) =>

SafetyResult(

Some(FreedomOfSpeechNotReach.reasonToSafetyResultReason(appealableReason)),

verdict.toActionThrift()

)

case \_ => SafetyResult(None, verdict.toActionThrift())

}

def getUserVisibilityResult: Option[t.UserVisibilityResult] =

(verdict match {

case Drop(reason, \_) =>

Some(

t.UserAction.Drop(t.Drop(Reason.toDropReason(reason).map(DropReasonConverter.toThrift))))

case \_ => None

}).map(userAction => t.UserVisibilityResult(Some(userAction)))

}

object VisibilityResult {

class Builder {

var featureMap: FeatureMap = FeatureMap.empty

var ruleResultMap: Map[Rule, RuleResult] = Map.empty

var verdict: Action = Allow

var finished: Boolean = false

var actingRule: Option[Rule] = None

var secondaryActingRules: Seq[Rule] = Seq()

var secondaryVerdicts: Seq[Action] = Seq()

var resolvedFeatureMap: Map[Feature[\_], Any] = Map.empty

def withFeatureMap(featureMapBld: FeatureMap) = {

featureMap = featureMapBld

this

}

def withRuleResultMap(ruleResultMapBld: Map[Rule, RuleResult]) = {

ruleResultMap = ruleResultMapBld

this

}

def withVerdict(verdictBld: Action) = {

verdict = verdictBld

this

}

def withFinished(finishedBld: Boolean) = {

finished = finishedBld

this

}

def withActingRule(actingRuleBld: Option[Rule]) = {

actingRule = actingRuleBld

this

}

def withSecondaryActingRules(secondaryActingRulesBld: Seq[Rule]) = {

secondaryActingRules = secondaryActingRulesBld

this

}

def withSecondaryVerdicts(secondaryVerdictsBld: Seq[Action]) = {

secondaryVerdicts = secondaryVerdictsBld

this

}

def build(contentId: ContentId) = VisibilityResult(

contentId,

featureMap,

ruleResultMap,

verdict,

finished,

actingRule,

secondaryActingRules,

secondaryVerdicts,

resolvedFeatureMap)

}

}