package com.twitter.visibility.builder

import com.twitter.visibility.features.Feature

import com.twitter.visibility.features.FeatureMap

import com.twitter.visibility.models.ContentId

import com.twitter.visibility.rules.Action

import com.twitter.visibility.rules.Allow

import com.twitter.visibility.rules.EvaluationContext

import com.twitter.visibility.rules.FailClosedException

import com.twitter.visibility.rules.FeaturesFailedException

import com.twitter.visibility.rules.MissingFeaturesException

import com.twitter.visibility.rules.Rule

import com.twitter.visibility.rules.RuleFailedException

import com.twitter.visibility.rules.RuleResult

import com.twitter.visibility.rules.State.FeatureFailed

import com.twitter.visibility.rules.State.MissingFeature

import com.twitter.visibility.rules.State.RuleFailed

class VisibilityResultBuilder(

val contentId: ContentId,

val featureMap: FeatureMap = FeatureMap.empty,

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

private var mapBuilder = Map.newBuilder[Rule, RuleResult]

mapBuilder ++= ruleResultMap

var verdict: Action = Allow

var finished: Boolean = false

var features: FeatureMap = featureMap

var actingRule: Option[Rule] = None

var secondaryVerdicts: Seq[Action] = Seq()

var secondaryActingRules: Seq[Rule] = Seq()

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

def ruleResults: Map[Rule, RuleResult] = mapBuilder.result()

def withFeatureMap(featureMap: FeatureMap): VisibilityResultBuilder = {

this.features = featureMap

this

}

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

this.ruleResultMap = ruleResultMap

mapBuilder = Map.newBuilder[Rule, RuleResult]

mapBuilder ++= ruleResultMap

this

}

def withRuleResult(rule: Rule, result: RuleResult): VisibilityResultBuilder = {

mapBuilder += ((rule, result))

this

}

def withVerdict(verdict: Action, ruleOpt: Option[Rule] = None): VisibilityResultBuilder = {

this.verdict = verdict

this.actingRule = ruleOpt

this

}

def withSecondaryVerdict(verdict: Action, rule: Rule): VisibilityResultBuilder = {

this.secondaryVerdicts = this.secondaryVerdicts :+ verdict

this.secondaryActingRules = this.secondaryActingRules :+ rule

this

}

def withFinished(finished: Boolean): VisibilityResultBuilder = {

this.finished = finished

this

}

def withResolvedFeatureMap(

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

): VisibilityResultBuilder = {

this.resolvedFeatureMap = resolvedFeatureMap

this

}

def isVerdictComposable(): Boolean = this.verdict.isComposable

def failClosedException(evaluationContext: EvaluationContext): Option[FailClosedException] = {

mapBuilder

.result().collect {

case (r: Rule, RuleResult(\_, MissingFeature(mf)))

if r.shouldFailClosed(evaluationContext.params) =>

Some(MissingFeaturesException(r.name, mf))

case (r: Rule, RuleResult(\_, FeatureFailed(ff)))

if r.shouldFailClosed(evaluationContext.params) =>

Some(FeaturesFailedException(r.name, ff))

case (r: Rule, RuleResult(\_, RuleFailed(t)))

if r.shouldFailClosed(evaluationContext.params) =>

Some(RuleFailedException(r.name, t))

}.toList.foldLeft(None: Option[FailClosedException]) { (acc, arg) =>

(acc, arg) match {

case (None, Some(\_)) => arg

case (Some(FeaturesFailedException(\_, \_)), Some(MissingFeaturesException(\_, \_))) => arg

case (Some(RuleFailedException(\_, \_)), Some(MissingFeaturesException(\_, \_))) => arg

case (Some(RuleFailedException(\_, \_)), Some(FeaturesFailedException(\_, \_))) => arg

case \_ => acc

}

}

}

def build: VisibilityResult = {

VisibilityResult(

contentId = contentId,

featureMap = features,

ruleResultMap = mapBuilder.result(),

verdict = verdict,

finished = finished,

actingRule = actingRule,

secondaryActingRules = secondaryActingRules,

secondaryVerdicts = secondaryVerdicts,

resolvedFeatureMap = resolvedFeatureMap

)

}

}