package com.twitter.product\_mixer.component\_library.gate

import com.twitter.product\_mixer.core.feature.Feature

import com.twitter.product\_mixer.core.feature.featuremap.MissingFeatureException

import com.twitter.product\_mixer.core.functional\_component.gate.Gate

import com.twitter.product\_mixer.core.functional\_component.gate.GateResult

import com.twitter.product\_mixer.core.model.common.identifier.GateIdentifier

import com.twitter.product\_mixer.core.pipeline.PipelineQuery

import com.twitter.product\_mixer.core.pipeline.pipeline\_failure.MisconfiguredFeatureMapFailure

import com.twitter.product\_mixer.core.pipeline.pipeline\_failure.PipelineFailure

import com.twitter.stitch.Stitch

import com.twitter.util.Return

import com.twitter.util.Throw

trait ShouldContinue[Value] {

/\*\* Given the [[Feature]] value, returns whether the execution should continue \*/

def apply(featureValue: Value): Boolean

/\*\* If the [[Feature]] is a failure, use this value \*/

def onFailedFeature(t: Throwable): GateResult = GateResult.Stop

/\*\*

\* If the [[Feature]], or [[com.twitter.product\_mixer.core.feature.featuremap.FeatureMap]],

\* is missing use this value

\*/

def onMissingFeature: GateResult = GateResult.Stop

}

object FeatureGate {

def fromFeature(

feature: Feature[\_, Boolean]

): FeatureGate[Boolean] =

FeatureGate.fromFeature(GateIdentifier(feature.toString), feature)

def fromNegatedFeature(

feature: Feature[\_, Boolean]

): FeatureGate[Boolean] =

FeatureGate.fromNegatedFeature(GateIdentifier(feature.toString), feature)

def fromFeature(

gateIdentifier: GateIdentifier,

feature: Feature[\_, Boolean]

): FeatureGate[Boolean] =

FeatureGate[Boolean](gateIdentifier, feature, identity)

def fromNegatedFeature(

gateIdentifier: GateIdentifier,

feature: Feature[\_, Boolean]

): FeatureGate[Boolean] =

FeatureGate[Boolean](gateIdentifier, feature, !identity(\_))

}

/\*\*

\* A [[Gate]] that is actuated based upon the value of the provided feature

\*/

case class FeatureGate[Value](

gateIdentifier: GateIdentifier,

feature: Feature[\_, Value],

continue: ShouldContinue[Value])

extends Gate[PipelineQuery] {

override val identifier: GateIdentifier = gateIdentifier

override def shouldContinue(query: PipelineQuery): Stitch[Boolean] = {

Stitch

.value(

query.features.map(\_.getTry(feature)) match {

case Some(Return(value)) => continue(value)

case Some(Throw(\_: MissingFeatureException)) => continue.onMissingFeature.continue

case Some(Throw(t)) => continue.onFailedFeature(t).continue

case None =>

throw PipelineFailure(

MisconfiguredFeatureMapFailure,

"Expected a FeatureMap to be present but none was found, ensure that your" +

"PipelineQuery has a FeatureMap configured before gating on Feature values"

)

}

)

}

}