package com.twitter.product\_mixer.component\_library.feature\_hydrator.candidate.qualityfactor\_gated

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

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

import com.twitter.product\_mixer.core.functional\_component.common.alert.Alert

import com.twitter.product\_mixer.core.functional\_component.feature\_hydrator.CandidateFeatureHydrator

import com.twitter.product\_mixer.core.model.common.Conditionally

import com.twitter.product\_mixer.core.model.common.UniversalNoun

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

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

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

import com.twitter.product\_mixer.core.quality\_factor.HasQualityFactorStatus

import com.twitter.stitch.Stitch

import com.twitter.timelines.configapi.Param

object QualityFactorGatedCandidateFeatureHydrator {

val IdentifierPrefix = "QfGated"

}

/\*\*

\* A [[CandidateFeatureHydrator]] with [[Conditionally]] based on a qualityFactor threshold.

\* @param pipelineIdentifier identifier of the pipeline that associated with observed quality factor

\* @param qualityFactorInclusiveThreshold the inclusive threshold of quality factor that value below it results in

\* the underlying hydrator being turned off

\* @param candidateFeatureHydrator the underlying [[CandidateFeatureHydrator]] to run when quality factor value

\* is above the given inclusive threshold

\* @tparam Query The domain model for the query or request

\* @tparam Result The type of the candidates

\*/

case class QualityFactorGatedCandidateFeatureHydrator[

-Query <: PipelineQuery with HasQualityFactorStatus,

Result <: UniversalNoun[Any]

](

pipelineIdentifier: ComponentIdentifier,

qualityFactorInclusiveThreshold: Param[Double],

candidateFeatureHydrator: CandidateFeatureHydrator[Query, Result])

extends CandidateFeatureHydrator[Query, Result]

with Conditionally[Query] {

import QualityFactorGatedCandidateFeatureHydrator.\_

override val identifier: FeatureHydratorIdentifier = FeatureHydratorIdentifier(

IdentifierPrefix + candidateFeatureHydrator.identifier.name)

override val alerts: Seq[Alert] = candidateFeatureHydrator.alerts

override val features: Set[Feature[\_, \_]] = candidateFeatureHydrator.features

override def onlyIf(query: Query): Boolean = Conditionally.and(

query,

candidateFeatureHydrator,

query.getQualityFactorCurrentValue(pipelineIdentifier) >= query.params(

qualityFactorInclusiveThreshold))

override def apply(

query: Query,

candidate: Result,

existingFeatures: FeatureMap

): Stitch[FeatureMap] = candidateFeatureHydrator.apply(query, candidate, existingFeatures)

}