package com.twitter.product\_mixer.component\_library.scorer.qualityfactor\_gated

import com.twitter.product\_mixer.component\_library.scorer.qualityfactor\_gated.QualityFactorGatedScorer.IdentifierPrefix

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.scorer.Scorer

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

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.ScorerIdentifier

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

/\*\*

\* A [[scorer]] with [[Conditionally]] based on quality factor value and threshold

\*

\* @param qualityFactorThreshold quliaty factor threshold that turn off the scorer

\* @param pipelineIdentifier identifier of the pipeline that quality factor is based on

\* @param scorer the underlying [[scorer]] to run when `enabledParam` is true

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

\* @tparam Result The type of the candidates

\*/

case class QualityFactorGatedScorer[

-Query <: PipelineQuery with HasQualityFactorStatus,

Result <: UniversalNoun[Any]

](

pipelineIdentifier: ComponentIdentifier,

qualityFactorThresholdParam: Param[Double],

scorer: Scorer[Query, Result])

extends Scorer[Query, Result]

with Conditionally[Query] {

override val identifier: ScorerIdentifier = ScorerIdentifier(

IdentifierPrefix + scorer.identifier.name)

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

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

override def onlyIf(query: Query): Boolean =

Conditionally.and(

query,

scorer,

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

qualityFactorThresholdParam))

override def apply(

query: Query,

candidates: Seq[CandidateWithFeatures[Result]]

): Stitch[Seq[FeatureMap]] = scorer(query, candidates)

}

object QualityFactorGatedScorer {

val IdentifierPrefix = "QualityFactorGated"

}