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

import com.twitter.product\_mixer.component\_library.filter.FeatureConditionalFilter.IdentifierInfix

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

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

import com.twitter.product\_mixer.core.functional\_component.filter.Filter

import com.twitter.product\_mixer.core.functional\_component.filter.FilterResult

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

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

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

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

import com.twitter.stitch.Stitch

/\*\*

\* Predicate to apply to candidate feature, to determine whether to apply filter.

\* True indicates we will apply the filter. False indicates to keep candidate and not apply filter.

\* @tparam FeatureValue

\*/

trait ShouldApplyFilter[FeatureValue] {

def apply(feature: FeatureValue): Boolean

}

/\*\*

\* A filter that applies the [[filter]] for candidates for which [[shouldApplyFilter]] is true, and keeps the others

\* @param feature feature to determine whether to apply underyling filter

\* @param shouldApplyFilter function to determine whether to apply filter

\* @param filter the actual filter to apply if shouldApplyFilter is True

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

\* @tparam Candidate The type of the candidates

\* @tparam FeatureValueType

\*/

case class FeatureValueConditionalFilter[

-Query <: PipelineQuery,

Candidate <: UniversalNoun[Any],

FeatureValueType

](

feature: Feature[Candidate, FeatureValueType],

shouldApplyFilter: ShouldApplyFilter[FeatureValueType],

filter: Filter[Query, Candidate])

extends Filter[Query, Candidate] {

override val identifier: FilterIdentifier = FilterIdentifier(

feature.toString + IdentifierInfix + filter.identifier.name

)

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

override def apply(

query: Query,

candidates: Seq[CandidateWithFeatures[Candidate]]

): Stitch[FilterResult[Candidate]] = {

val (candidatesToFilter, candidatesToKeep) = candidates.partition { candidate =>

shouldApplyFilter(candidate.features.get(feature))

}

filter.apply(query, candidatesToFilter).map { filterResult =>

FilterResult(

kept = filterResult.kept ++ candidatesToKeep.map(\_.candidate),

removed = filterResult.removed)

}

}

}

object FeatureConditionalFilter {

val IdentifierInfix = "FeatureConditional"

}