package com.twitter.home\_mixer.functional\_component.filter

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.Conditionally

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

trait FilterPredicate[-Query <: PipelineQuery] {

def apply(query: Query): Boolean

}

/\*\*

\* A [[Filter]] with [[Conditionally]] based on a [[FilterPredicate]]

\*

\* @param predicate the predicate to turn this filter on and off

\* @param filter the underlying filter to run when `predicate` is true

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

\* @tparam Candidate The type of the candidates

\*/

case class PredicateGatedFilter[-Query <: PipelineQuery, Candidate <: UniversalNoun[Any]](

predicate: FilterPredicate[Query],

filter: Filter[Query, Candidate])

extends Filter[Query, Candidate]

with Filter.Conditionally[Query, Candidate] {

override val identifier: FilterIdentifier = FilterIdentifier(

PredicateGatedFilter.IdentifierPrefix + filter.identifier.name)

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

override def onlyIf(query: Query, candidates: Seq[CandidateWithFeatures[Candidate]]): Boolean =

Conditionally.and(Filter.Input(query, candidates), filter, predicate(query))

override def apply(

query: Query,

candidates: Seq[CandidateWithFeatures[Candidate]]

): Stitch[FilterResult[Candidate]] = filter.apply(query, candidates)

}

object PredicateGatedFilter {

val IdentifierPrefix = "PredicateGated"

}