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

import com.twitter.product\_mixer.core.functional\_component.common.CandidateScope

import com.twitter.product\_mixer.core.functional\_component.selector.Selector

import com.twitter.product\_mixer.core.functional\_component.selector.SelectorResult

import com.twitter.product\_mixer.core.model.common.presentation.CandidateWithDetails

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

import com.twitter.timelines.configapi.Param

trait IncludeSelector[-Query <: PipelineQuery] {

def apply(

query: Query,

remainingCandidates: Seq[CandidateWithDetails],

result: Seq[CandidateWithDetails]

): Boolean

}

/\*\*

\* Run [[selector]] if [[includeSelector]] resolves to true, else no-op the selector

\*/

case class SelectConditionally[-Query <: PipelineQuery](

selector: Selector[Query],

includeSelector: IncludeSelector[Query])

extends Selector[Query] {

override val pipelineScope: CandidateScope = selector.pipelineScope

override def apply(

query: Query,

remainingCandidates: Seq[CandidateWithDetails],

result: Seq[CandidateWithDetails]

): SelectorResult = {

if (includeSelector(query, remainingCandidates, result)) {

selector(query, remainingCandidates, result)

} else SelectorResult(remainingCandidates = remainingCandidates, result = result)

}

}

object SelectConditionally {

/\*\*

\* Wrap each [[Selector]] in `selectors` in an [[IncludeSelector]] with `includeSelector` as the [[SelectConditionally.includeSelector]]

\*/

def apply[Query <: PipelineQuery](

selectors: Seq[Selector[Query]],

includeSelector: IncludeSelector[Query]

): Seq[Selector[Query]] =

selectors.map(SelectConditionally(\_, includeSelector))

/\*\*

\* A [[SelectConditionally]] based on a [[Param]]

\*/

def paramGated[Query <: PipelineQuery](

selector: Selector[Query],

enabledParam: Param[Boolean],

): SelectConditionally[Query] =

SelectConditionally(selector, (query, \_, \_) => query.params(enabledParam))

/\*\*

\* Wrap each [[Selector]] in `selectors` in a [[SelectConditionally]] based on a [[Param]]

\*/

def paramGated[Query <: PipelineQuery](

selectors: Seq[Selector[Query]],

enabledParam: Param[Boolean],

): Seq[Selector[Query]] =

selectors.map(SelectConditionally.paramGated(\_, enabledParam))

/\*\*

\* A [[SelectConditionally]] based on an inverted [[Param]]

\*/

def paramNotGated[Query <: PipelineQuery](

selector: Selector[Query],

enabledParamToInvert: Param[Boolean],

): SelectConditionally[Query] =

SelectConditionally(selector, (query, \_, \_) => !query.params(enabledParamToInvert))

/\*\*

\* Wrap each [[Selector]] in `selectors` in a [[SelectConditionally]] based on an inverted [[Param]]

\*/

def paramNotGated[Query <: PipelineQuery](

selectors: Seq[Selector[Query]],

enabledParamToInvert: Param[Boolean],

): Seq[Selector[Query]] =

selectors.map(SelectConditionally.paramNotGated(\_, enabledParamToInvert))

}