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

import com.twitter.product\_mixer.component\_library.selector.sorter.SorterFromOrdering

import com.twitter.product\_mixer.component\_library.selector.sorter.SorterProvider

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

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

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

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.identifier.CandidatePipelineIdentifier

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

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

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

object UpdateSortModuleItemCandidates {

def apply(

candidatePipeline: CandidatePipelineIdentifier,

ordering: Ordering[CandidateWithDetails]

): UpdateSortModuleItemCandidates =

UpdateSortModuleItemCandidates(

SpecificPipeline(candidatePipeline),

SorterFromOrdering(ordering))

def apply(

candidatePipeline: CandidatePipelineIdentifier,

sorterProvider: SorterProvider

): UpdateSortModuleItemCandidates =

UpdateSortModuleItemCandidates(SpecificPipeline(candidatePipeline), sorterProvider)

def apply(

candidatePipelines: Set[CandidatePipelineIdentifier],

ordering: Ordering[CandidateWithDetails]

): UpdateSortModuleItemCandidates =

UpdateSortModuleItemCandidates(

SpecificPipelines(candidatePipelines),

SorterFromOrdering(ordering))

def apply(

candidatePipelines: Set[CandidatePipelineIdentifier],

sorterProvider: SorterProvider

): UpdateSortModuleItemCandidates =

UpdateSortModuleItemCandidates(SpecificPipelines(candidatePipelines), sorterProvider)

}

/\*\*

\* Sort items inside a module from a candidate source and update the remainingCandidates.

\*

\* For example, we could specify the following ordering to sort by score descending:

\*

\* {{{

\* Ordering

\* .by[CandidateWithDetails, Double](\_.features.get(ScoreFeature) match {

\* case Scored(score) => score

\* case \_ => Double.MinValue

\* }).reverse

\*

\* // Before sorting:

\* ModuleCandidateWithDetails(

\* Seq(

\* ItemCandidateWithLowScore,

\* ItemCandidateWithMidScore,

\* ItemCandidateWithHighScore),

\* ... other params

\* )

\*

\* // After sorting:

\* ModuleCandidateWithDetails(

\* Seq(

\* ItemCandidateWithHighScore,

\* ItemCandidateWithMidScore,

\* ItemCandidateWithLowScore),

\* ... other params

\* )

\* }}}

\*

\* @note this updates the modules in the `remainingCandidates`

\*/

case class UpdateSortModuleItemCandidates(

override val pipelineScope: CandidateScope,

sorterProvider: SorterProvider)

extends Selector[PipelineQuery] {

override def apply(

query: PipelineQuery,

remainingCandidates: Seq[CandidateWithDetails],

result: Seq[CandidateWithDetails]

): SelectorResult = {

val updatedCandidates = remainingCandidates.map {

case module: ModuleCandidateWithDetails if pipelineScope.contains(module) =>

module.copy(candidates =

sorterProvider.sorter(query, remainingCandidates, result).sort(module.candidates))

case candidate => candidate

}

SelectorResult(remainingCandidates = updatedCandidates, result = result)

}

}