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.CandidateScope.PartitionedCandidates

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.\_

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.pipeline.PipelineQuery

object UpdateSortCandidates {

def apply(

candidatePipeline: CandidatePipelineIdentifier,

sorterProvider: SorterProvider,

) = new UpdateSortCandidates(SpecificPipeline(candidatePipeline), sorterProvider)

def apply(

candidatePipeline: CandidatePipelineIdentifier,

ordering: Ordering[CandidateWithDetails]

) =

new UpdateSortCandidates(SpecificPipeline(candidatePipeline), SorterFromOrdering(ordering))

def apply(

candidatePipelines: Set[CandidatePipelineIdentifier],

ordering: Ordering[CandidateWithDetails]

) =

new UpdateSortCandidates(SpecificPipelines(candidatePipelines), SorterFromOrdering(ordering))

def apply(

candidatePipelines: Set[CandidatePipelineIdentifier],

sorterProvider: SorterProvider,

) = new UpdateSortCandidates(SpecificPipelines(candidatePipelines), sorterProvider)

def apply(

pipelineScope: CandidateScope,

ordering: Ordering[CandidateWithDetails]

) = new UpdateSortCandidates(pipelineScope, SorterFromOrdering(ordering))

}

/\*\*

\* Sort item and module (not items inside modules) candidates in a pipeline scope.

\* Note that if sorting across multiple candidate sources, the candidates will be grouped together

\* in sorted order, starting from the position of the first candidate.

\*

\* 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

\*/

case class UpdateSortCandidates(

override val pipelineScope: CandidateScope,

sorterProvider: SorterProvider)

extends Selector[PipelineQuery] {

override def apply(

query: PipelineQuery,

remainingCandidates: Seq[CandidateWithDetails],

result: Seq[CandidateWithDetails]

): SelectorResult = {

val PartitionedCandidates(selectedCandidates, otherCandidates) =

pipelineScope.partition(remainingCandidates)

val updatedRemainingCandidates = if (selectedCandidates.nonEmpty) {

// Safe .head due to nonEmpty check

val position = remainingCandidates.indexOf(selectedCandidates.head)

val orderedSelectedCandidates =

sorterProvider.sorter(query, remainingCandidates, result).sort(selectedCandidates)

if (position < otherCandidates.length) {

val (left, right) = otherCandidates.splitAt(position)

left ++ orderedSelectedCandidates ++ right

} else {

otherCandidates ++ orderedSelectedCandidates

}

} else {

remainingCandidates

}

SelectorResult(remainingCandidates = updatedRemainingCandidates, result = result)

}

}