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.AllPipelines

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

object UpdateSortResults {

def apply(

ordering: Ordering[CandidateWithDetails]

) =

new UpdateSortResults((\_, \_, \_) => SorterFromOrdering(ordering))

}

/\*\*

\* Sort item and module (not items inside modules) results.

\*

\* 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 UpdateSortResults(

sorterProvider: SorterProvider,

override val pipelineScope: CandidateScope = AllPipelines)

extends Selector[PipelineQuery] {

override def apply(

query: PipelineQuery,

remainingCandidates: Seq[CandidateWithDetails],

result: Seq[CandidateWithDetails]

): SelectorResult = {

val updatedResult = sorterProvider.sorter(query, remainingCandidates, result).sort(result)

SelectorResult(remainingCandidates = remainingCandidates, result = updatedResult)

}

}