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

import com.twitter.product\_mixer.component\_library.selector.InsertIntoModule.ModuleAndIndex

import com.twitter.product\_mixer.component\_library.selector.InsertIntoModule.ModuleWithItemsToAddAndOtherCandidates

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

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.pipeline.PipelineQuery

/\*\*

\* Append all candidates from [[candidatePipeline]] into a module from [[targetModuleCandidatePipeline]].

\* If the results contain multiple modules from the target candidate pipeline,

\* then the candidates will be inserted into the first module.

\*

\* @note this will throw an [[UnsupportedOperationException]] if the [[candidatePipeline]] contains any modules.

\*

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

\*/

case class InsertAppendIntoModuleCandidates(

candidatePipeline: CandidatePipelineIdentifier,

targetModuleCandidatePipeline: CandidatePipelineIdentifier)

extends Selector[PipelineQuery] {

override val pipelineScope: CandidateScope =

SpecificPipelines(candidatePipeline, targetModuleCandidatePipeline)

override def apply(

query: PipelineQuery,

remainingCandidates: Seq[CandidateWithDetails],

result: Seq[CandidateWithDetails]

): SelectorResult = {

val ModuleWithItemsToAddAndOtherCandidates(

moduleToUpdateAndIndex,

itemsToInsertIntoModule,

otherCandidates) =

InsertIntoModule.moduleToUpdate(

candidatePipeline,

targetModuleCandidatePipeline,

remainingCandidates)

val updatedRemainingCandidates = moduleToUpdateAndIndex match {

case None => remainingCandidates

case \_ if itemsToInsertIntoModule.isEmpty => remainingCandidates

case Some(ModuleAndIndex(moduleToUpdate, indexOfModuleInOtherCandidates)) =>

val updatedModuleItems = moduleToUpdate.candidates ++ itemsToInsertIntoModule

val updatedModule = moduleToUpdate.copy(candidates = updatedModuleItems)

otherCandidates.updated(indexOfModuleInOtherCandidates, updatedModule)

}

SelectorResult(remainingCandidates = updatedRemainingCandidates, result = result)

}

}