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

import com.twitter.product\_mixer.component\_library.model.candidate.CursorCandidate

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.model.common.presentation.ItemCandidateWithDetails

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

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

/\*\*

\* Keep only the candidates in `remainingCandidates` that appear multiple times.

\* This ignores modules and cursors from being removed.

\*

\* @param duplicationKey how to generate the key used to identify duplicate candidates

\*

\* @note [[com.twitter.product\_mixer.component\_library.model.candidate.CursorCandidate]] are ignored.

\* @note [[com.twitter.product\_mixer.core.model.common.presentation.ModuleCandidateWithDetails]] are ignored.

\*

\* @example if `remainingCandidates`

\* `Seq(sourceA\_Id1, sourceA\_Id1, sourceA\_Id2, sourceB\_id1, sourceB\_id2, sourceB\_id3, sourceC\_id4)`

\* then the output result will be `Seq(sourceA\_Id1, sourceA\_Id2)`

\*/

case class DropNonDuplicateCandidates(

override val pipelineScope: CandidateScope,

duplicationKey: DeduplicationKey[\_] = IdAndClassDuplicationKey)

extends Selector[PipelineQuery] {

override def apply(

query: PipelineQuery,

remainingCandidates: Seq[CandidateWithDetails],

result: Seq[CandidateWithDetails]

): SelectorResult = {

val duplicateCandidates = dropNonDuplicates(

pipelineScope = pipelineScope,

candidates = remainingCandidates,

duplicationKey = duplicationKey)

SelectorResult(remainingCandidates = duplicateCandidates, result = result)

}

/\*\*

\* Identify and keep candidates using the supplied key extraction and merger functions. By default

\* this will keep only candidates that appear multiple times as determined by comparing

\* the contained candidate ID and class type. Candidates appearing only once will be dropped.

\*

\* @note [[com.twitter.product\_mixer.component\_library.model.candidate.CursorCandidate]] are ignored.

\* @note [[com.twitter.product\_mixer.core.model.common.presentation.ModuleCandidateWithDetails]] are ignored.

\*

\* @param candidates which may have elements to drop

\* @param duplicationKey how to generate a key for a candidate for identifying duplicates

\*/

private[this] def dropNonDuplicates[Candidate <: CandidateWithDetails, Key](

pipelineScope: CandidateScope,

candidates: Seq[Candidate],

duplicationKey: DeduplicationKey[Key],

): Seq[Candidate] = {

// Here we are checking if each candidate has multiple appearances or not

val isCandidateADuplicate: Map[Key, Boolean] = candidates

.collect {

case item: ItemCandidateWithDetails

if pipelineScope.contains(item) && !item.candidate.isInstanceOf[CursorCandidate] =>

item

}.groupBy(duplicationKey(\_))

.mapValues(\_.length > 1)

candidates.filter {

case item: ItemCandidateWithDetails =>

isCandidateADuplicate.getOrElse(duplicationKey(item), true)

case \_: ModuleCandidateWithDetails => true

case \_ => false

}

}

}