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

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

/\*\*

\* Limit candidates to the first candidate source in the provided orthogonalCandidatePipelines

\* seq that has candidates in the candidate pool. For the subsequent candidate sources in the seq,

\* remove their candidates from the candidate pool.

\*

\* @example if [[orthogonalCandidatePipelines]] is `Seq(D, A, C)`, and the remaining candidates

\* component identifiers are `Seq(A, A, A, B, B, C, C, D, D, D)`, then `Seq(B, B, D, D, D)` will remain

\* in the candidate pool.

\*

\* @example if [[orthogonalCandidatePipelines]] is `Seq(D, A, C)`, and the remaining candidates

\* component identifiers are `Seq(A, A, A, B, B, C, C)`, then `Seq(A, A, A, B, B)` will remain

\* in the candidate pool.

\*/

case class DropOrthogonalCandidates(

orthogonalCandidatePipelines: Seq[CandidatePipelineIdentifier])

extends Selector[PipelineQuery] {

override val pipelineScope: CandidateScope =

SpecificPipelines(orthogonalCandidatePipelines.toSet)

override def apply(

query: PipelineQuery,

remainingCandidates: Seq[CandidateWithDetails],

result: Seq[CandidateWithDetails]

): SelectorResult = {

val firstMatchingOrthogonalSourceOpt = orthogonalCandidatePipelines

.find { orthogonalCandidatePipeline =>

remainingCandidates.exists(\_.source == orthogonalCandidatePipeline)

}

val remainingCandidatesLimited = firstMatchingOrthogonalSourceOpt match {

case Some(firstMatchingOrthogonalSource) =>

val subsequentOrthogonalSources =

orthogonalCandidatePipelines.toSet - firstMatchingOrthogonalSource

remainingCandidates.filterNot { candidate =>

subsequentOrthogonalSources.contains(candidate.source)

}

case None => remainingCandidates

}

SelectorResult(remainingCandidates = remainingCandidatesLimited, result = result)

}

}