package com.twitter.product\_mixer.component\_library.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.model.common.presentation.ItemCandidateWithDetails

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

import scala.collection.immutable.Queue

private[selector] object InsertIntoModule {

case class ModuleAndIndex(

moduleToInsertInto: ModuleCandidateWithDetails,

indexOfModuleInOtherCandidates: Int)

case class ModuleWithItemsToAddAndOtherCandidates(

moduleToUpdateAndIndex: Option[ModuleAndIndex],

itemsToInsertIntoModule: Queue[ItemCandidateWithDetails],

otherCandidates: Queue[CandidateWithDetails])

/\*\*

\* Given a Seq of `candidates`, returns the first module with it's index that matches the

\* `targetModuleCandidatePipeline` with all the [[ItemCandidateWithDetails]] that match the

\* `candidatePipeline` added to the `itemsToInsert` and the remaining candidates, including the

\* module, in the `otherCandidates`

\*/

def moduleToUpdate(

candidatePipeline: CandidatePipelineIdentifier,

targetModuleCandidatePipeline: CandidatePipelineIdentifier,

candidates: Seq[CandidateWithDetails]

): ModuleWithItemsToAddAndOtherCandidates = {

candidates.foldLeft[ModuleWithItemsToAddAndOtherCandidates](

ModuleWithItemsToAddAndOtherCandidates(None, Queue.empty, Queue.empty)) {

case (

state @ ModuleWithItemsToAddAndOtherCandidates(\_, itemsToInsertIntoModule, \_),

selectedItem: ItemCandidateWithDetails) if selectedItem.source == candidatePipeline =>

state.copy(itemsToInsertIntoModule = itemsToInsertIntoModule :+ selectedItem)

case (

state @ ModuleWithItemsToAddAndOtherCandidates(None, \_, otherCandidates),

module: ModuleCandidateWithDetails) if module.source == targetModuleCandidatePipeline =>

val insertionIndex = otherCandidates.length

val moduleAndIndex = Some(

ModuleAndIndex(

moduleToInsertInto = module,

indexOfModuleInOtherCandidates = insertionIndex))

val otherCandidatesWithModuleAppended = otherCandidates :+ module

state.copy(

moduleToUpdateAndIndex = moduleAndIndex,

otherCandidates = otherCandidatesWithModuleAppended)

case (\_, invalidModule: ModuleCandidateWithDetails)

if invalidModule.source == candidatePipeline =>

/\*\*

\* while not exactly an illegal state, its most likely an incorrectly configured candidate pipeline

\* that returned a module instead of returning the candidates the module contains. Since you can't

\* nest a module inside of a module, we can either throw or ignore it and we choose to ignore it

\* to catch a potential bug a customer may do accidentally.

\*/

throw new UnsupportedOperationException(

s"Expected the candidatePipeline $candidatePipeline to contain items to put into the module from the targetModuleCandidatePipeline $targetModuleCandidatePipeline, but not contain modules itself. " +

s"This can occur if your $candidatePipeline was incorrectly configured and returns a module when you intended to return the candidates the module contained."

)

case (

state @ ModuleWithItemsToAddAndOtherCandidates(\_, \_, otherCandidates),

unselectedCandidate) =>

state.copy(otherCandidates = otherCandidates :+ unselectedCandidate)

}

}

}