package com.twitter.product\_mixer.core.service.group\_results\_executor

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.product\_mixer.core.feature.featuremap.FeatureMapBuilder

import com.twitter.product\_mixer.core.model.common.CandidateWithFeatures

import com.twitter.product\_mixer.core.model.common.UniversalNoun

import com.twitter.product\_mixer.core.model.common.identifier.CandidatePipelineIdentifier

import com.twitter.product\_mixer.core.model.common.identifier.CandidateSourceIdentifier

import com.twitter.product\_mixer.core.model.common.identifier.ComponentIdentifier

import com.twitter.product\_mixer.core.model.common.identifier.PlatformIdentifier

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

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

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

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.ItemPresentation

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

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

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

import com.twitter.product\_mixer.core.service.Executor

import com.twitter.product\_mixer.core.service.ExecutorResult

import com.twitter.stitch.Arrow

import javax.inject.Inject

import javax.inject.Singleton

import scala.collection.immutable.ListSet

// Most executors are in the core.service package, but this one is pipeline specific

@Singleton

class GroupResultsExecutor @Inject() (override val statsReceiver: StatsReceiver) extends Executor {

val identifier: ComponentIdentifier = PlatformIdentifier("GroupResults")

def arrow[Candidate <: UniversalNoun[Any]](

pipelineIdentifier: CandidatePipelineIdentifier,

sourceIdentifier: CandidateSourceIdentifier,

context: Executor.Context

): Arrow[GroupResultsExecutorInput[Candidate], GroupResultsExecutorResult] = {

val groupArrow = Arrow.map { input: GroupResultsExecutorInput[Candidate] =>

val modules: Map[Option[ModulePresentation], Seq[CandidateWithFeatures[Candidate]]] =

input.candidates

.map { candidate: CandidateWithFeatures[Candidate] =>

val modulePresentationOpt: Option[ModulePresentation] =

input.decorations.get(candidate.candidate).collect {

case itemPresentation: ItemPresentation

if itemPresentation.modulePresentation.isDefined =>

itemPresentation.modulePresentation.get

}

(candidate, modulePresentationOpt)

}.groupBy {

case (\_, modulePresentationOpt) => modulePresentationOpt

}.mapValues {

resultModuleOptTuples: Seq[

(CandidateWithFeatures[Candidate], Option[ModulePresentation])

] =>

resultModuleOptTuples.map {

case (result, \_) => result

}

}

// Modules should be in their original order, but the groupBy above isn't stable.

// Sort them by the sourcePosition, using the sourcePosition of their first contained

// candidate.

val sortedModules: Seq[(Option[ModulePresentation], Seq[CandidateWithFeatures[Candidate]])] =

modules.toSeq

.sortBy {

case (\_, results) =>

results.headOption.map(\_.features.get(CandidateSourcePosition))

}

val candidatesWithDetails: Seq[CandidateWithDetails] = sortedModules.flatMap {

case (modulePresentationOpt, resultsSeq) =>

val itemsWithDetails = resultsSeq.map { result =>

val presentationOpt = input.decorations.get(result.candidate) match {

case itemPresentation @ Some(\_: ItemPresentation) => itemPresentation

case \_ => None

}

val baseFeatureMap = FeatureMapBuilder()

.add(CandidatePipelines, ListSet.empty + pipelineIdentifier)

.build()

ItemCandidateWithDetails(

candidate = result.candidate,

presentation = presentationOpt,

features = baseFeatureMap ++ result.features

)

}

modulePresentationOpt

.map { modulePresentation =>

val moduleSourcePosition =

resultsSeq.head.features.get(CandidateSourcePosition)

val baseFeatureMap = FeatureMapBuilder()

.add(CandidatePipelines, ListSet.empty + pipelineIdentifier)

.add(CandidateSourcePosition, moduleSourcePosition)

.add(CandidateSources, ListSet.empty + sourceIdentifier)

.build()

Seq(

ModuleCandidateWithDetails(

candidates = itemsWithDetails,

presentation = Some(modulePresentation),

features = baseFeatureMap

))

}.getOrElse(itemsWithDetails)

}

GroupResultsExecutorResult(candidatesWithDetails)

}

wrapWithErrorHandling(context, identifier)(groupArrow)

}

}

case class GroupResultsExecutorInput[Candidate <: UniversalNoun[Any]](

candidates: Seq[CandidateWithFeatures[Candidate]],

decorations: Map[UniversalNoun[Any], UniversalPresentation])

case class GroupResultsExecutorResult(candidatesWithDetails: Seq[CandidateWithDetails])

extends ExecutorResult