package com.twitter.product\_mixer.core.pipeline.candidate

import com.twitter.finagle.stats.StatsReceiver

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

import com.twitter.product\_mixer.core.feature.featuremap.asyncfeaturemap.AsyncFeatureMap

import com.twitter.product\_mixer.core.functional\_component.common.alert.Alert

import com.twitter.product\_mixer.core.functional\_component.decorator.CandidateDecorator

import com.twitter.product\_mixer.core.functional\_component.feature\_hydrator.BaseQueryFeatureHydrator

import com.twitter.product\_mixer.core.functional\_component.transformer.BaseCandidatePipelineQueryTransformer

import com.twitter.product\_mixer.core.functional\_component.transformer.CandidatePipelineResultsTransformer

import com.twitter.product\_mixer.core.gate.ParamGate

import com.twitter.product\_mixer.core.gate.ParamGate.EnabledGateSuffix

import com.twitter.product\_mixer.core.gate.ParamGate.SupportedClientGateSuffix

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

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

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.ComponentIdentifierStack

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

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

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

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

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

import com.twitter.product\_mixer.core.pipeline.pipeline\_failure.ClosedGate

import com.twitter.product\_mixer.core.pipeline.pipeline\_failure.PipelineFailureClassifier

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

import com.twitter.product\_mixer.core.service.async\_feature\_map\_executor.AsyncFeatureMapExecutor

import com.twitter.product\_mixer.core.service.async\_feature\_map\_executor.AsyncFeatureMapExecutorResults

import com.twitter.product\_mixer.core.service.candidate\_decorator\_executor.CandidateDecoratorExecutor

import com.twitter.product\_mixer.core.service.candidate\_decorator\_executor.CandidateDecoratorExecutorResult

import com.twitter.product\_mixer.core.service.candidate\_feature\_hydrator\_executor.CandidateFeatureHydratorExecutor

import com.twitter.product\_mixer.core.service.candidate\_feature\_hydrator\_executor.CandidateFeatureHydratorExecutorResult

import com.twitter.product\_mixer.core.service.candidate\_source\_executor.CandidateSourceExecutor

import com.twitter.product\_mixer.core.service.candidate\_source\_executor.CandidateSourceExecutorResult

import com.twitter.product\_mixer.core.service.candidate\_source\_executor.FetchedCandidateWithFeatures

import com.twitter.product\_mixer.core.service.filter\_executor.FilterExecutor

import com.twitter.product\_mixer.core.service.filter\_executor.FilterExecutorResult

import com.twitter.product\_mixer.core.service.gate\_executor.GateExecutor

import com.twitter.product\_mixer.core.service.gate\_executor.GateExecutorResult

import com.twitter.product\_mixer.core.service.gate\_executor.StoppedGateException

import com.twitter.product\_mixer.core.service.group\_results\_executor.GroupResultsExecutor

import com.twitter.product\_mixer.core.service.group\_results\_executor.GroupResultsExecutorInput

import com.twitter.product\_mixer.core.service.group\_results\_executor.GroupResultsExecutorResult

import com.twitter.product\_mixer.core.service.query\_feature\_hydrator\_executor.QueryFeatureHydratorExecutor

import com.twitter.stitch.Arrow

import com.twitter.util.logging.Logging

import javax.inject.Inject

class CandidatePipelineBuilder[

Query <: PipelineQuery,

CandidateSourceQuery,

CandidateSourceResult,

Result <: UniversalNoun[Any]] @Inject() (

queryFeatureHydratorExecutor: QueryFeatureHydratorExecutor,

asyncFeatureMapExecutor: AsyncFeatureMapExecutor,

candidateDecoratorExecutor: CandidateDecoratorExecutor,

candidateFeatureHydratorExecutor: CandidateFeatureHydratorExecutor,

candidateSourceExecutor: CandidateSourceExecutor,

groupResultsExecutor: GroupResultsExecutor,

filterExecutor: FilterExecutor,

gateExecutor: GateExecutor,

override val statsReceiver: StatsReceiver)

extends PipelineBuilder[CandidatePipeline.Inputs[Query]]

with Logging {

override type UnderlyingResultType = Seq[CandidateWithDetails]

override type PipelineResultType = IntermediateCandidatePipelineResult[Result]

def build(

parentComponentIdentifierStack: ComponentIdentifierStack,

config: BaseCandidatePipelineConfig[

Query,

CandidateSourceQuery,

CandidateSourceResult,

Result

]

): CandidatePipeline[Query] = {

val pipelineIdentifier = config.identifier

val candidateSourceIdentifier = config.candidateSource.identifier

val context = Executor.Context(

PipelineFailureClassifier(

config.failureClassifier.orElse(StoppedGateException.classifier(ClosedGate))),

parentComponentIdentifierStack.push(pipelineIdentifier)

)

val enabledGateOpt = config.enabledDeciderParam.map { deciderParam =>

ParamGate(pipelineIdentifier + EnabledGateSuffix, deciderParam)

}

val supportedClientGateOpt = config.supportedClientParam.map { param =>

ParamGate(pipelineIdentifier + SupportedClientGateSuffix, param)

}

/\*\*

\* Evaluate enabled decider gate first since if it's off, there is no reason to proceed

\* Next evaluate supported client feature switch gate, followed by customer configured gates

\*/

val allGates = enabledGateOpt.toSeq ++ supportedClientGateOpt.toSeq ++ config.gates

// Dynamically replace the identifier of both transformers if config used the inline constructor

// which sets a default identifier. We need to do this to ensure uniqueness of identifiers.

val queryTransformer = BaseCandidatePipelineQueryTransformer.copyWithUpdatedIdentifier(

config.queryTransformer,

pipelineIdentifier)

val resultsTransformer = CandidatePipelineResultsTransformer.copyWithUpdatedIdentifier(

config.resultTransformer,

pipelineIdentifier)

val decorator = config.decorator.map(decorator =>

CandidateDecorator.copyWithUpdatedIdentifier(decorator, pipelineIdentifier))

val GatesStep = new Step[Query, GateExecutorResult] {

override def identifier: PipelineStepIdentifier = CandidatePipelineConfig.gatesStep

override def executorArrow: Arrow[Query, GateExecutorResult] = {

gateExecutor.arrow(allGates, context)

}

override def inputAdaptor(

query: CandidatePipeline.Inputs[Query],

previousResult: IntermediateCandidatePipelineResult[Result]

): Query =

query.query

override def resultUpdater(

previousPipelineResult: IntermediateCandidatePipelineResult[Result],

executorResult: GateExecutorResult

): IntermediateCandidatePipelineResult[Result] =

previousPipelineResult.copy(underlyingResult =

previousPipelineResult.underlyingResult.copy(gateResult = Some(executorResult)))

}

def queryFeatureHydrationStep(

queryFeatureHydrators: Seq[BaseQueryFeatureHydrator[Query, \_]],

stepIdentifier: PipelineStepIdentifier,

updater: ResultUpdater[CandidatePipelineResult, QueryFeatureHydratorExecutor.Result]

): Step[Query, QueryFeatureHydratorExecutor.Result] =

new Step[Query, QueryFeatureHydratorExecutor.Result] {

override def identifier: PipelineStepIdentifier = stepIdentifier

override def executorArrow: Arrow[Query, QueryFeatureHydratorExecutor.Result] =

queryFeatureHydratorExecutor.arrow(

queryFeatureHydrators,

CandidatePipelineConfig.stepsAsyncFeatureHydrationCanBeCompletedBy,

context)

override def inputAdaptor(

query: CandidatePipeline.Inputs[Query],

previousResult: IntermediateCandidatePipelineResult[Result]

): Query = query.query

override def resultUpdater(

previousPipelineResult: IntermediateCandidatePipelineResult[Result],

executorResult: QueryFeatureHydratorExecutor.Result

): IntermediateCandidatePipelineResult[Result] =

previousPipelineResult.copy(

underlyingResult = updater(previousPipelineResult.underlyingResult, executorResult))

override def queryUpdater(

query: CandidatePipeline.Inputs[Query],

executorResult: QueryFeatureHydratorExecutor.Result

): CandidatePipeline.Inputs[Query] =

CandidatePipeline.Inputs(

query.query

.withFeatureMap(

query.query.features.getOrElse(

FeatureMap.empty) ++ executorResult.featureMap).asInstanceOf[Query],

query.existingCandidates)

}

def asyncFeaturesStep(

stepToHydrateFor: PipelineStepIdentifier,

context: Executor.Context

): Step[AsyncFeatureMap, AsyncFeatureMapExecutorResults] =

new Step[AsyncFeatureMap, AsyncFeatureMapExecutorResults] {

override def identifier: PipelineStepIdentifier =

CandidatePipelineConfig.asyncFeaturesStep(stepToHydrateFor)

override def executorArrow: Arrow[AsyncFeatureMap, AsyncFeatureMapExecutorResults] =

asyncFeatureMapExecutor.arrow(stepToHydrateFor, identifier, context)

override def inputAdaptor(

query: CandidatePipeline.Inputs[Query],

previousResult: IntermediateCandidatePipelineResult[Result]

): AsyncFeatureMap =

previousResult.underlyingResult.mergedAsyncQueryFeatures

.getOrElse(

throw InvalidStepStateException(identifier, "MergedAsyncQueryFeatures")

)

override def resultUpdater(

previousPipelineResult: IntermediateCandidatePipelineResult[Result],

executorResult: AsyncFeatureMapExecutorResults

): IntermediateCandidatePipelineResult[Result] =

previousPipelineResult.copy(

underlyingResult =

previousPipelineResult.underlyingResult.copy(asyncFeatureHydrationResults =

previousPipelineResult.underlyingResult.asyncFeatureHydrationResults match {

case Some(existingResults) => Some(existingResults ++ executorResult)

case None => Some(executorResult)

}))

override def queryUpdater(

query: CandidatePipeline.Inputs[Query],

executorResult: AsyncFeatureMapExecutorResults

): CandidatePipeline.Inputs[Query] =

if (executorResult.featureMapsByStep

.getOrElse(stepToHydrateFor, FeatureMap.empty).isEmpty) {

query

} else {

val updatedQuery = query.query

.withFeatureMap(

query.query.features

.getOrElse(FeatureMap.empty) ++ executorResult.featureMapsByStep(

stepToHydrateFor)).asInstanceOf[Query]

CandidatePipeline.Inputs(updatedQuery, query.existingCandidates)

}

}

val CandidateSourceStep =

new Step[Query, CandidateSourceExecutorResult[Result]] {

override def identifier: PipelineStepIdentifier =

CandidatePipelineConfig.candidateSourceStep

override def executorArrow: Arrow[

Query,

CandidateSourceExecutorResult[Result]

] =

candidateSourceExecutor

.arrow(

config.candidateSource,

queryTransformer,

resultsTransformer,

config.featuresFromCandidateSourceTransformers,

context

)

override def inputAdaptor(

query: CandidatePipeline.Inputs[Query],

previousResult: IntermediateCandidatePipelineResult[Result]

): Query =

query.query

override def resultUpdater(

previousPipelineResult: IntermediateCandidatePipelineResult[Result],

executorResult: CandidateSourceExecutorResult[Result]

): IntermediateCandidatePipelineResult[Result] =

previousPipelineResult.copy(underlyingResult =

previousPipelineResult.underlyingResult.copy(

candidateSourceResult =

Some(executorResult.asInstanceOf[CandidateSourceExecutorResult[UniversalNoun[Any]]])

))

override def queryUpdater(

query: CandidatePipeline.Inputs[Query],

executorResult: CandidateSourceExecutorResult[Result]

): CandidatePipeline.Inputs[Query] = {

val updatedFeatureMap =

query.query.features

.getOrElse(FeatureMap.empty) ++ executorResult.candidateSourceFeatureMap

val updatedQuery = query.query

.withFeatureMap(updatedFeatureMap).asInstanceOf[Query]

CandidatePipeline.Inputs(updatedQuery, query.existingCandidates)

}

}

val PreFilterFeatureHydrationPhase1Step =

new Step[

CandidateFeatureHydratorExecutor.Inputs[Query, Result],

CandidateFeatureHydratorExecutorResult[Result]

] {

override def identifier: PipelineStepIdentifier =

CandidatePipelineConfig.preFilterFeatureHydrationPhase1Step

override def executorArrow: Arrow[

CandidateFeatureHydratorExecutor.Inputs[Query, Result],

CandidateFeatureHydratorExecutorResult[Result]

] =

candidateFeatureHydratorExecutor.arrow(config.preFilterFeatureHydrationPhase1, context)

override def inputAdaptor(

query: CandidatePipeline.Inputs[Query],

previousResult: IntermediateCandidatePipelineResult[Result]

): CandidateFeatureHydratorExecutor.Inputs[Query, Result] = {

val candidateSourceExecutorResult =

previousResult.underlyingResult.candidateSourceResult.getOrElse {

throw InvalidStepStateException(identifier, "CandidateSourceResult")

}

CandidateFeatureHydratorExecutor.Inputs(

query.query,

candidateSourceExecutorResult.candidates

.asInstanceOf[Seq[CandidateWithFeatures[Result]]])

}

override def resultUpdater(

previousPipelineResult: IntermediateCandidatePipelineResult[Result],

executorResult: CandidateFeatureHydratorExecutorResult[Result]

): IntermediateCandidatePipelineResult[Result] = {

val candidateSourceExecutorResult =

previousPipelineResult.underlyingResult.candidateSourceResult.getOrElse {

throw InvalidStepStateException(identifier, "CandidateSourceResult")

}

val featureMapsFromPreFilter = executorResult.results.map { result =>

result.candidate -> result.features

}.toMap

val mergedFeatureMaps = candidateSourceExecutorResult.candidates.map { candidate =>

val candidateFeatureMap = candidate.features

val preFilterFeatureMap =

featureMapsFromPreFilter.getOrElse(

candidate.candidate.asInstanceOf[Result],

FeatureMap.empty)

candidate.candidate.asInstanceOf[Result] -> (candidateFeatureMap ++ preFilterFeatureMap)

}.toMap

previousPipelineResult.copy(

underlyingResult = previousPipelineResult.underlyingResult.copy(

preFilterHydrationResult = Some(

executorResult

.asInstanceOf[CandidateFeatureHydratorExecutorResult[UniversalNoun[Any]]])

),

featureMaps = Some(mergedFeatureMaps)

)

}

}

val PreFilterFeatureHydrationPhase2Step =

new Step[

CandidateFeatureHydratorExecutor.Inputs[Query, Result],

CandidateFeatureHydratorExecutorResult[Result]

] {

override def identifier: PipelineStepIdentifier =

CandidatePipelineConfig.preFilterFeatureHydrationPhase2Step

override def executorArrow: Arrow[

CandidateFeatureHydratorExecutor.Inputs[Query, Result],

CandidateFeatureHydratorExecutorResult[Result]

] =

candidateFeatureHydratorExecutor.arrow(config.preFilterFeatureHydrationPhase2, context)

override def inputAdaptor(

query: CandidatePipeline.Inputs[Query],

previousResult: IntermediateCandidatePipelineResult[Result]

): CandidateFeatureHydratorExecutor.Inputs[Query, Result] = {

val candidates = previousResult.underlyingResult.preFilterHydrationResult.getOrElse {

throw InvalidStepStateException(identifier, "PreFilterHydrationResult")

}.results

CandidateFeatureHydratorExecutor.Inputs(

query.query,

candidates.asInstanceOf[Seq[CandidateWithFeatures[Result]]]

)

}

override def resultUpdater(

previousPipelineResult: IntermediateCandidatePipelineResult[Result],

executorResult: CandidateFeatureHydratorExecutorResult[Result]

): IntermediateCandidatePipelineResult[Result] = {

val featureMapsFromPreFilterPhase2 = executorResult.results.map { result =>

result.candidate -> result.features

}.toMap

val mergedFeatureMaps = previousPipelineResult.featureMaps

.getOrElse(throw InvalidStepStateException(identifier, "FeatureMaps"))

.map {

case (candidate, featureMap) =>

val preFilterPhase2FeatureMap =

featureMapsFromPreFilterPhase2.getOrElse(candidate, FeatureMap.empty)

candidate -> (featureMap ++ preFilterPhase2FeatureMap)

}

previousPipelineResult.copy(

underlyingResult = previousPipelineResult.underlyingResult.copy(

preFilterHydrationResultPhase2 = Some(

executorResult

.asInstanceOf[CandidateFeatureHydratorExecutorResult[UniversalNoun[Any]]])

),

featureMaps = Some(mergedFeatureMaps)

)

}

}

val FiltersStep =

new Step[(Query, Seq[CandidateWithFeatures[Result]]), FilterExecutorResult[Result]] {

override def identifier: PipelineStepIdentifier = CandidatePipelineConfig.filtersStep

override def executorArrow: Arrow[

(Query, Seq[CandidateWithFeatures[Result]]),

FilterExecutorResult[

Result

]

] =

filterExecutor.arrow(config.filters, context)

override def inputAdaptor(

query: CandidatePipeline.Inputs[Query],

previousResult: IntermediateCandidatePipelineResult[Result]

): (Query, Seq[CandidateWithFeatures[Result]]) = {

val candidates =

previousResult.underlyingResult.candidateSourceResult

.getOrElse {

throw InvalidStepStateException(identifier, "CandidateSourceResult")

}.candidates.map(\_.candidate).asInstanceOf[Seq[Result]]

val featureMaps = previousResult.featureMaps

.getOrElse(throw InvalidStepStateException(identifier, "FeatureMaps"))

(

query.query,

candidates.map(candidate =>

CandidateWithFeaturesImpl(

candidate,

featureMaps.getOrElse(candidate, FeatureMap.empty))))

}

override def resultUpdater(

previousPipelineResult: IntermediateCandidatePipelineResult[Result],

executorResult: FilterExecutorResult[Result]

): IntermediateCandidatePipelineResult[Result] =

previousPipelineResult.copy(underlyingResult =

previousPipelineResult.underlyingResult.copy(

filterResult =

Some(executorResult.asInstanceOf[FilterExecutorResult[UniversalNoun[Any]]])

))

}

val PostFilterFeatureHydrationStep =

new Step[

CandidateFeatureHydratorExecutor.Inputs[Query, Result],

CandidateFeatureHydratorExecutorResult[Result]

] {

override def identifier: PipelineStepIdentifier =

CandidatePipelineConfig.postFilterFeatureHydrationStep

override def executorArrow: Arrow[

CandidateFeatureHydratorExecutor.Inputs[Query, Result],

CandidateFeatureHydratorExecutorResult[Result]

] =

candidateFeatureHydratorExecutor.arrow(config.postFilterFeatureHydration, context)

override def inputAdaptor(

query: CandidatePipeline.Inputs[Query],

previousResult: IntermediateCandidatePipelineResult[Result]

): CandidateFeatureHydratorExecutor.Inputs[Query, Result] = {

val filterResult = previousResult.underlyingResult.filterResult

.getOrElse(

throw InvalidStepStateException(identifier, "FilterResult")

).result.asInstanceOf[Seq[Result]]

val featureMaps = previousResult.featureMaps.getOrElse(

throw InvalidStepStateException(identifier, "FeatureMaps")

)

val filteredCandidates = filterResult.map { candidate =>

CandidateWithFeaturesImpl(candidate, featureMaps.getOrElse(candidate, FeatureMap.empty))

}

CandidateFeatureHydratorExecutor.Inputs(query.query, filteredCandidates)

}

override def resultUpdater(

previousPipelineResult: IntermediateCandidatePipelineResult[Result],

executorResult: CandidateFeatureHydratorExecutorResult[Result]

): IntermediateCandidatePipelineResult[Result] = {

val filterResult = previousPipelineResult.underlyingResult.filterResult

.getOrElse(

throw InvalidStepStateException(identifier, "FilterResult")

).result.asInstanceOf[Seq[Result]]

val featureMaps = previousPipelineResult.featureMaps.getOrElse(

throw InvalidStepStateException(identifier, "FeatureMaps")

)

val postFilterFeatureMaps = executorResult.results.map { result =>

result.candidate -> result.features

}.toMap

val mergedFeatureMaps = filterResult.map { candidate =>

candidate ->

(featureMaps

.getOrElse(candidate, FeatureMap.empty) ++ postFilterFeatureMaps.getOrElse(

candidate,

FeatureMap.empty))

}.toMap

previousPipelineResult.copy(

underlyingResult = previousPipelineResult.underlyingResult.copy(

postFilterHydrationResult = Some(

executorResult

.asInstanceOf[CandidateFeatureHydratorExecutorResult[UniversalNoun[Any]]])

),

featureMaps = Some(mergedFeatureMaps)

)

}

}

val ScorersStep =

new Step[

CandidateFeatureHydratorExecutor.Inputs[Query, Result],

CandidateFeatureHydratorExecutorResult[Result]

] {

override def identifier: PipelineStepIdentifier = CandidatePipelineConfig.scorersStep

override def executorArrow: Arrow[

CandidateFeatureHydratorExecutor.Inputs[Query, Result],

CandidateFeatureHydratorExecutorResult[Result]

] =

candidateFeatureHydratorExecutor.arrow(config.scorers, context)

override def inputAdaptor(

query: CandidatePipeline.Inputs[Query],

previousResult: IntermediateCandidatePipelineResult[Result]

): CandidateFeatureHydratorExecutor.Inputs[Query, Result] = {

val filterResult = previousResult.underlyingResult.filterResult

.getOrElse(

throw InvalidStepStateException(identifier, "FilterResult")

).result.asInstanceOf[Seq[Result]]

val featureMaps = previousResult.featureMaps.getOrElse(

throw InvalidStepStateException(identifier, "FeatureMaps")

)

val filteredCandidates = filterResult.map { candidate =>

CandidateWithFeaturesImpl(candidate, featureMaps.getOrElse(candidate, FeatureMap.empty))

}

CandidateFeatureHydratorExecutor.Inputs(query.query, filteredCandidates)

}

override def resultUpdater(

previousPipelineResult: IntermediateCandidatePipelineResult[Result],

executorResult: CandidateFeatureHydratorExecutorResult[Result]

): IntermediateCandidatePipelineResult[Result] = {

val filterResult = previousPipelineResult.underlyingResult.filterResult

.getOrElse(

throw InvalidStepStateException(identifier, "FilterResult")

).result.asInstanceOf[Seq[Result]]

val featureMaps = previousPipelineResult.featureMaps.getOrElse(

throw InvalidStepStateException(identifier, "FeatureMaps")

)

val scoringFeatureMaps = executorResult.results.map { result =>

result.candidate -> result.features

}.toMap

val mergedFeatureMaps = filterResult.map { candidate =>

candidate ->

(featureMaps

.getOrElse(candidate, FeatureMap.empty) ++ scoringFeatureMaps.getOrElse(

candidate,

FeatureMap.empty))

}.toMap

previousPipelineResult.copy(

underlyingResult = previousPipelineResult.underlyingResult.copy(

scorersResult = Some(

executorResult

.asInstanceOf[CandidateFeatureHydratorExecutorResult[UniversalNoun[Any]]])

),

featureMaps = Some(mergedFeatureMaps)

)

}

}

val DecorationStep =

new Step[(Query, Seq[CandidateWithFeatures[Result]]), CandidateDecoratorExecutorResult] {

override def identifier: PipelineStepIdentifier = CandidatePipelineConfig.decoratorStep

override def executorArrow: Arrow[

(Query, Seq[CandidateWithFeatures[Result]]),

CandidateDecoratorExecutorResult

] =

candidateDecoratorExecutor.arrow(decorator, context)

override def inputAdaptor(

query: CandidatePipeline.Inputs[Query],

previousResult: IntermediateCandidatePipelineResult[Result]

): (Query, Seq[CandidateWithFeatures[Result]]) = {

val keptCandidates = previousResult.underlyingResult.filterResult

.getOrElse {

throw InvalidStepStateException(identifier, "FilterResult")

}.result.asInstanceOf[Seq[Result]]

val featureMaps = previousResult.featureMaps.getOrElse {

throw InvalidStepStateException(identifier, "FeatureMaps")

}

(

query.query,

keptCandidates.map(candidate =>

CandidateWithFeaturesImpl(

candidate,

featureMaps.getOrElse(candidate, FeatureMap.empty))))

}

override def resultUpdater(

previousPipelineResult: IntermediateCandidatePipelineResult[Result],

executorResult: CandidateDecoratorExecutorResult

): IntermediateCandidatePipelineResult[Result] =

previousPipelineResult.copy(underlyingResult =

previousPipelineResult.underlyingResult.copy(

candidateDecoratorResult = Some(executorResult)

))

}

/\*\*

\* ResultStep is a synchronous step that basically takes the outputs from the other steps, groups modules,

\* and puts things into the final result object

\*/

val ResultStep = new Step[GroupResultsExecutorInput[Result], GroupResultsExecutorResult] {

override def identifier: PipelineStepIdentifier = CandidatePipelineConfig.resultStep

override def executorArrow: Arrow[

GroupResultsExecutorInput[Result],

GroupResultsExecutorResult

] = groupResultsExecutor.arrow(pipelineIdentifier, candidateSourceIdentifier, context)

override def inputAdaptor(

query: CandidatePipeline.Inputs[Query],

previousResult: IntermediateCandidatePipelineResult[Result]

): GroupResultsExecutorInput[Result] = {

val underlying = previousResult.underlyingResult

val keptCandidates = underlying.filterResult

.getOrElse(

throw InvalidStepStateException(identifier, "FilterResult")

).result.asInstanceOf[Seq[Result]]

val decorations = underlying.candidateDecoratorResult

.getOrElse(

throw InvalidStepStateException(identifier, "DecorationResult")

).result.map(decoration => decoration.candidate -> decoration.presentation).toMap

val combinedFeatureMaps: Map[Result, FeatureMap] = previousResult.featureMaps.getOrElse(

throw InvalidStepStateException(identifier, "FeatureMaps"))

val filteredCandidates = keptCandidates.map { candidate =>

val updatedMap = combinedFeatureMaps

.get(candidate).getOrElse(FeatureMap.empty)

FetchedCandidateWithFeatures(candidate, updatedMap)

}

GroupResultsExecutorInput(

candidates = filteredCandidates,

decorations = decorations

)

}

override def resultUpdater(

previousPipelineResult: IntermediateCandidatePipelineResult[Result],

executorResult: GroupResultsExecutorResult

): IntermediateCandidatePipelineResult[Result] =

previousPipelineResult.copy(underlyingResult = previousPipelineResult.underlyingResult

.copy(result = Some(executorResult.candidatesWithDetails)))

}

val builtSteps = Seq(

GatesStep,

queryFeatureHydrationStep(

config.queryFeatureHydration,

CandidatePipelineConfig.fetchQueryFeaturesStep,

(pipelineResult, executorResult) =>

pipelineResult.copy(queryFeatures = Some(executorResult))

),

queryFeatureHydrationStep(

config.queryFeatureHydrationPhase2,

CandidatePipelineConfig.fetchQueryFeaturesPhase2Step,

(pipelineResult, executorResult) =>

pipelineResult.copy(

queryFeaturesPhase2 = Some(executorResult),

mergedAsyncQueryFeatures = Some(

pipelineResult.queryFeatures

.getOrElse(

throw InvalidStepStateException(

CandidatePipelineConfig.fetchQueryFeaturesPhase2Step,

"QueryFeatures")

).asyncFeatureMap ++ executorResult.asyncFeatureMap)

)

),

asyncFeaturesStep(CandidatePipelineConfig.candidateSourceStep, context),

CandidateSourceStep,

asyncFeaturesStep(CandidatePipelineConfig.preFilterFeatureHydrationPhase1Step, context),

PreFilterFeatureHydrationPhase1Step,

asyncFeaturesStep(CandidatePipelineConfig.preFilterFeatureHydrationPhase2Step, context),

PreFilterFeatureHydrationPhase2Step,

asyncFeaturesStep(CandidatePipelineConfig.filtersStep, context),

FiltersStep,

asyncFeaturesStep(CandidatePipelineConfig.postFilterFeatureHydrationStep, context),

PostFilterFeatureHydrationStep,

asyncFeaturesStep(CandidatePipelineConfig.scorersStep, context),

ScorersStep,

asyncFeaturesStep(CandidatePipelineConfig.decoratorStep, context),

DecorationStep,

ResultStep

)

/\*\* The main execution logic for this Candidate Pipeline. \*/

val finalArrow: Arrow[CandidatePipeline.Inputs[Query], CandidatePipelineResult] =

buildCombinedArrowFromSteps(

steps = builtSteps,

context = context,

initialEmptyResult =

IntermediateCandidatePipelineResult.empty[Result](config.candidateSource.identifier),

stepsInOrderFromConfig = CandidatePipelineConfig.stepsInOrder

).map(\_.underlyingResult)

val configFromBuilder = config

new CandidatePipeline[Query] {

override private[core] val config: BaseCandidatePipelineConfig[Query, \_, \_, \_] =

configFromBuilder

override val arrow: Arrow[CandidatePipeline.Inputs[Query], CandidatePipelineResult] =

finalArrow

override val identifier: CandidatePipelineIdentifier = pipelineIdentifier

override val alerts: Seq[Alert] = config.alerts

override val children: Seq[Component] =

allGates ++

config.queryFeatureHydration ++

Seq(queryTransformer, config.candidateSource, resultsTransformer) ++

config.featuresFromCandidateSourceTransformers ++

decorator.toSeq ++

config.preFilterFeatureHydrationPhase1 ++

config.filters ++

config.postFilterFeatureHydration ++

config.scorers

}

}

private case class CandidateWithFeaturesImpl(candidate: Result, features: FeatureMap)

extends CandidateWithFeatures[Result]

}