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

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

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

import com.twitter.product\_mixer.core.functional\_component.scorer.ScoredCandidateResult

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

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

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.ComponentIdentifierStack

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

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

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.pipeline.NewPipelineBuilder

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

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

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.pipeline.state.HasCandidatesWithDetails

import com.twitter.product\_mixer.core.pipeline.state.HasCandidatesWithFeatures

import com.twitter.product\_mixer.core.pipeline.state.HasExecutorResults

import com.twitter.product\_mixer.core.pipeline.state.HasQuery

import com.twitter.product\_mixer.core.pipeline.state.HasResult

import com.twitter.product\_mixer.core.pipeline.step.candidate\_feature\_hydrator.CandidateFeatureHydratorStep

import com.twitter.product\_mixer.core.pipeline.step.gate.GateStep

import com.twitter.product\_mixer.core.pipeline.step.scorer.ScorerStep

import com.twitter.product\_mixer.core.pipeline.step.selector.SelectorStep

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

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

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

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.selector\_executor.SelectorExecutorResult

import com.twitter.stitch.Arrow

import javax.inject.Inject

import scala.collection.immutable.ListMap

/\*\*

\* NewScoringPipelineBuilder builds [[ScoringPipeline]]s from [[ScoringPipelineConfig]]s.

\* New because it's meant to eventually replace [[ScoringPipelineBuilder]]

\* You should inject a [[ScoringPipelineBuilderFactory]] and call `.get` to build these.

\*

\* @see [[ScoringPipelineConfig]] for the description of the type parameters

\* @tparam Query the type of query these accept.

\* @tparam Candidate the domain model for the candidate being scored

\*/

class NewScoringPipelineBuilder[Query <: PipelineQuery, Candidate <: UniversalNoun[Any]] @Inject() (

selectionStep: SelectorStep[Query, ScoringPipelineState[Query, Candidate]],

gateStep: GateStep[Query, ScoringPipelineState[Query, Candidate]],

candidateFeatureHydrationStep: CandidateFeatureHydratorStep[

Query,

Candidate,

ScoringPipelineState[Query, Candidate]

],

scorerStep: ScorerStep[Query, Candidate, ScoringPipelineState[Query, Candidate]])

extends NewPipelineBuilder[ScoringPipelineConfig[Query, Candidate], Seq[

CandidateWithFeatures[Candidate]

], ScoringPipelineState[Query, Candidate], ScoringPipeline[Query, Candidate]] {

override def build(

parentComponentIdentifierStack: ComponentIdentifierStack,

arrowBuilder: NewPipelineArrowBuilder[ArrowResult, ArrowState],

scoringPipelineConfig: ScoringPipelineConfig[Query, Candidate]

): ScoringPipeline[Query, Candidate] = {

val pipelineIdentifier = scoringPipelineConfig.identifier

val context = Executor.Context(

PipelineFailureClassifier(

scoringPipelineConfig.failureClassifier.orElse(

StoppedGateException.classifier(ClosedGate))),

parentComponentIdentifierStack.push(pipelineIdentifier)

)

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

ParamGate(pipelineIdentifier + EnabledGateSuffix, deciderParam)

}

val supportedClientGateOpt = scoringPipelineConfig.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 ++ scoringPipelineConfig.gates

val underlyingArrow = arrowBuilder

.add(ScoringPipelineConfig.gatesStep, gateStep, allGates)

.add(ScoringPipelineConfig.selectorsStep, selectionStep, scoringPipelineConfig.selectors)

.add(

ScoringPipelineConfig.preScoringFeatureHydrationPhase1Step,

candidateFeatureHydrationStep,

scoringPipelineConfig.preScoringFeatureHydrationPhase1)

.add(

ScoringPipelineConfig.preScoringFeatureHydrationPhase2Step,

candidateFeatureHydrationStep,

scoringPipelineConfig.preScoringFeatureHydrationPhase2)

.add(ScoringPipelineConfig.scorersStep, scorerStep, scoringPipelineConfig.scorers).buildArrow(

context)

val finalArrow = Arrow

.map { inputs: ScoringPipeline.Inputs[Query] =>

ScoringPipelineState[Query, Candidate](inputs.query, inputs.candidates, ListMap.empty)

}.andThen(underlyingArrow).map { pipelineResult =>

ScoringPipelineResult(

gateResults = pipelineResult.executorResultsByPipelineStep

.get(ScoringPipelineConfig.gatesStep)

.map(\_.asInstanceOf[GateExecutorResult]),

selectorResults = pipelineResult.executorResultsByPipelineStep

.get(ScoringPipelineConfig.selectorsStep)

.map(\_.asInstanceOf[SelectorExecutorResult]),

preScoringHydrationPhase1Result = pipelineResult.executorResultsByPipelineStep

.get(ScoringPipelineConfig.preScoringFeatureHydrationPhase1Step)

.map(\_.asInstanceOf[CandidateFeatureHydratorExecutorResult[Candidate]]),

preScoringHydrationPhase2Result = pipelineResult.executorResultsByPipelineStep

.get(ScoringPipelineConfig.preScoringFeatureHydrationPhase2Step)

.map(\_.asInstanceOf[CandidateFeatureHydratorExecutorResult[Candidate]]),

scorerResults = pipelineResult.executorResultsByPipelineStep

.get(ScoringPipelineConfig.scorersStep)

.map(\_.asInstanceOf[CandidateFeatureHydratorExecutorResult[Candidate]]),

failure = pipelineResult match {

case failure: NewPipelineResult.Failure =>

Some(failure.failure)

case \_ => None

},

result = pipelineResult match {

case result: NewPipelineResult.Success[Seq[CandidateWithFeatures[Candidate]]] =>

Some(result.result.map { candidateWithFeatures =>

ScoredCandidateResult(

candidateWithFeatures.candidate,

candidateWithFeatures.features)

})

case \_ => None

}

)

}

new ScoringPipeline[Query, Candidate] {

override val arrow: Arrow[ScoringPipeline.Inputs[Query], ScoringPipelineResult[Candidate]] =

finalArrow

override val identifier: ScoringPipelineIdentifier = scoringPipelineConfig.identifier

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

override val children: Seq[Component] =

allGates ++ scoringPipelineConfig.preScoringFeatureHydrationPhase1 ++ scoringPipelineConfig.preScoringFeatureHydrationPhase2 ++ scoringPipelineConfig.scorers

override private[core] val config = scoringPipelineConfig

}

}

}

case class ScoringPipelineState[Query <: PipelineQuery, Candidate <: UniversalNoun[Any]](

override val query: Query,

candidates: Seq[ItemCandidateWithDetails],

override val executorResultsByPipelineStep: ListMap[PipelineStepIdentifier, ExecutorResult])

extends HasQuery[Query, ScoringPipelineState[Query, Candidate]]

with HasCandidatesWithDetails[ScoringPipelineState[Query, Candidate]]

with HasCandidatesWithFeatures[Candidate, ScoringPipelineState[Query, Candidate]]

with HasExecutorResults[ScoringPipelineState[Query, Candidate]]

with HasResult[Seq[CandidateWithFeatures[Candidate]]] {

override val candidatesWithDetails: Seq[CandidateWithDetails] = candidates

override val candidatesWithFeatures: Seq[CandidateWithFeatures[Candidate]] =

candidates.asInstanceOf[Seq[CandidateWithFeatures[Candidate]]]

override val buildResult: Seq[CandidateWithFeatures[Candidate]] = candidatesWithFeatures

override def updateCandidatesWithDetails(

newCandidates: Seq[CandidateWithDetails]

): ScoringPipelineState[Query, Candidate] = {

this.copy(candidates = newCandidates.asInstanceOf[Seq[ItemCandidateWithDetails]])

}

override def updateQuery(newQuery: Query): ScoringPipelineState[Query, Candidate] =

this.copy(query = newQuery)

override def updateDecorations(

decoration: Seq[Decoration]

): ScoringPipelineState[Query, Candidate] = ???

override def updateCandidatesWithFeatures(

newCandidates: Seq[CandidateWithFeatures[Candidate]]

): ScoringPipelineState[Query, Candidate] = {

val updatedCandidates = candidates.zip(newCandidates).map {

case (itemCandidateWithDetails, newCandidate) =>

itemCandidateWithDetails.copy(features =

itemCandidateWithDetails.features ++ newCandidate.features)

}

this.copy(query, updatedCandidates)

}

override private[pipeline] def setExecutorResults(

newMap: ListMap[PipelineStepIdentifier, ExecutorResult]

) = this.copy(executorResultsByPipelineStep = newMap)

}