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

import com.twitter.product\_mixer.component\_library.selector.InsertAppendResults

import com.twitter.product\_mixer.core.functional\_component.common.AllPipelines

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.BaseCandidateFeatureHydrator

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

import com.twitter.product\_mixer.core.functional\_component.filter.Filter

import com.twitter.product\_mixer.core.functional\_component.gate.Gate

import com.twitter.product\_mixer.core.functional\_component.premarshaller.DomainMarshaller

import com.twitter.product\_mixer.core.functional\_component.selector.Selector

import com.twitter.product\_mixer.core.functional\_component.side\_effect.PipelineResultSideEffect

import com.twitter.product\_mixer.core.functional\_component.marshaller.TransportMarshaller

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.ComponentIdentifier

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

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

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

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

import com.twitter.product\_mixer.core.model.marshalling.HasMarshalling

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

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

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

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

import com.twitter.product\_mixer.core.pipeline.candidate.CandidatePipelineConfig

import com.twitter.product\_mixer.core.pipeline.candidate.DependentCandidatePipelineConfig

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

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

import com.twitter.product\_mixer.core.pipeline.scoring.ScoringPipelineConfig

import com.twitter.product\_mixer.core.quality\_factor.QualityFactorConfig

/\*\*

\* This is the configuration necessary to generate a Recommendation Pipeline. Product code should create a

\* RecommendationPipelineConfig, and then use a RecommendationPipelineBuilder to get the final RecommendationPipeline which can

\* process requests.

\*

\* @tparam Query - The domain model for the query or request

\* @tparam Candidate - The type of the candidates that the Candidate Pipelines are generating

\* @tparam UnmarshalledResultType - The result type of the pipeline, but before marshalling to a wire protocol like URT

\* @tparam Result - The final result that will be served to users

\*/

trait RecommendationPipelineConfig[

Query <: PipelineQuery,

Candidate <: UniversalNoun[Any],

UnmarshalledResultType <: HasMarshalling,

Result]

extends PipelineConfig {

override val identifier: RecommendationPipelineIdentifier

/\*\*

\* Recommendation Pipeline Gates will be executed before any other step (including retrieval from candidate

\* pipelines). They're executed sequentially, and any "Stop" result will prevent pipeline execution.

\*/

def gates: Seq[Gate[Query]] = Seq.empty

/\*\*

\* A recommendation pipeline can fetch query-level features before candidate pipelines are executed.

\*/

def fetchQueryFeatures: Seq[BaseQueryFeatureHydrator[Query, \_]] = Seq.empty

/\*\*

\* Candidate pipelines retrieve candidates for possible inclusion in the result

\*/

def fetchQueryFeaturesPhase2: Seq[BaseQueryFeatureHydrator[Query, \_]] = Seq.empty

/\*\*

\* What candidate pipelines should this Recommendations Pipeline get candidate from?

\*/

def candidatePipelines: Seq[CandidatePipelineConfig[Query, \_, \_, \_]]

/\*\*

\* Dependent candidate pipelines to retrieve candidates that depend on the result of [[candidatePipelines]]

\* [[DependentCandidatePipelineConfig]] have access to the list of previously retrieved & decorated

\* candidates for use in constructing the query object.

\*/

def dependentCandidatePipelines: Seq[DependentCandidatePipelineConfig[Query, \_, \_, \_]] = Seq.empty

/\*\*

\* Takes final ranked list of candidates & apply any business logic (e.g, deduplicating and merging

\* candidates before scoring).

\*/

def postCandidatePipelinesSelectors: Seq[Selector[Query]] = Seq(InsertAppendResults(AllPipelines))

/\*\*

\* After selectors are run, you can fetch features for each candidate.

\* The existing features from previous hydrations are passed in as inputs. You are not expected to

\* put them into the resulting feature map yourself - they will be merged for you by the platform.

\*/

def postCandidatePipelinesFeatureHydration: Seq[

BaseCandidateFeatureHydrator[Query, Candidate, \_]

] =

Seq.empty

/\*\*

\* Global filters to run on all candidates.

\*/

def globalFilters: Seq[Filter[Query, Candidate]] = Seq.empty

/\*\*

\* By default, a Recommendation Pipeline will fail closed - if any candidate or scoring

\* pipeline fails to return a result, then the Recommendation Pipeline will not return a result.

\* You can adjust this default policy, or provide specific policies to specific pipelines.

\* Those specific policies will take priority.

\*

\* FailOpenPolicy.All will always fail open (the RecommendationPipeline will continue without that pipeline)

\* FailOpenPolicy.Never will always fail closed (the RecommendationPipeline will fail if that pipeline fails)

\*

\* There's a default policy, and a specific Map of policies that takes precedence.

\*/

def defaultFailOpenPolicy: FailOpenPolicy = FailOpenPolicy(Set(ClosedGate))

def candidatePipelineFailOpenPolicies: Map[CandidatePipelineIdentifier, FailOpenPolicy] =

Map.empty

def scoringPipelineFailOpenPolicies: Map[ScoringPipelineIdentifier, FailOpenPolicy] = Map.empty

/\*\*

\*\* [[qualityFactorConfigs]] associates [[QualityFactorConfig]]s to specific candidate pipelines

\* using [[ComponentIdentifier]].

\*/

def qualityFactorConfigs: Map[ComponentIdentifier, QualityFactorConfig] =

Map.empty

/\*\*

\* Scoring pipelines for scoring candidates.

\* @note These do not drop or re-order candidates, you should do those in the sub-sequent selectors

\* step based off of the scores on candidates set in those [[ScoringPipeline]]s.

\*/

def scoringPipelines: Seq[ScoringPipelineConfig[Query, Candidate]]

/\*\*

\* Takes final ranked list of candidates & apply any business logic (e.g, capping number

\* of ad accounts or pacing ad accounts).

\*/

def resultSelectors: Seq[Selector[Query]]

/\*\*

\* Takes the final selected list of candidates and applies a final list of filters.

\* Useful for doing very expensive filtering at the end of your pipeline.

\*/

def postSelectionFilters: Seq[Filter[Query, Candidate]] = Seq.empty

/\*\*

\* Decorators allow for adding Presentations to candidates. While the Presentation can contain any

\* arbitrary data, Decorators are often used to add a UrtItemPresentation for URT item support. Most

\* customers will prefer to set a decorator in their respective candidate pipeline, however, a final

\* global one is available for those that do global decoration as late possible to avoid unnecessary hydrations.

\* @note This decorator can only return an ItemPresentation.

\* @note This decorator cannot decorate an already decorated candidate from the prior decorator

\* step in candidate pipelines.

\*/

def decorator: Option[CandidateDecorator[Query, Candidate]] = None

/\*\*

\* Domain marshaller transforms the selections into the model expected by the marshaller

\*/

def domainMarshaller: DomainMarshaller[Query, UnmarshalledResultType]

/\*\*

\* Mixer result side effects that are executed after selection and domain marshalling

\*/

def resultSideEffects: Seq[PipelineResultSideEffect[Query, UnmarshalledResultType]] = Seq()

/\*\*

\* Transport marshaller transforms the model into our line-level API like URT or JSON

\*/

def transportMarshaller: TransportMarshaller[UnmarshalledResultType, Result]

/\*\*

\* A pipeline can define a partial function to rescue failures here. They will be treated as failures

\* from a monitoring standpoint, and cancellation exceptions will always be propagated (they cannot be caught here).

\*/

def failureClassifier: PartialFunction[Throwable, PipelineFailure] = PartialFunction.empty

/\*\*

\* Alerts can be used to indicate the pipeline's service level objectives. Alerts and

\* dashboards will be automatically created based on this information.

\*/

val alerts: Seq[Alert] = Seq.empty

/\*\*

\* This method is used by the product mixer framework to build the pipeline.

\*/

private[core] final def build(

parentComponentIdentifierStack: ComponentIdentifierStack,

builder: RecommendationPipelineBuilderFactory

): RecommendationPipeline[Query, Candidate, Result] =

builder.get.build(parentComponentIdentifierStack, this)

}

object RecommendationPipelineConfig extends PipelineConfigCompanion {

val qualityFactorStep: PipelineStepIdentifier = PipelineStepIdentifier("QualityFactor")

val gatesStep: PipelineStepIdentifier = PipelineStepIdentifier("Gates")

val fetchQueryFeaturesStep: PipelineStepIdentifier = PipelineStepIdentifier("FetchQueryFeatures")

val fetchQueryFeaturesPhase2Step: PipelineStepIdentifier = PipelineStepIdentifier(

"FetchQueryFeaturesPhase2")

val candidatePipelinesStep: PipelineStepIdentifier = PipelineStepIdentifier("CandidatePipelines")

val dependentCandidatePipelinesStep: PipelineStepIdentifier =

PipelineStepIdentifier("DependentCandidatePipelines")

val postCandidatePipelinesSelectorsStep: PipelineStepIdentifier =

PipelineStepIdentifier("PostCandidatePipelinesSelectors")

val postCandidatePipelinesFeatureHydrationStep: PipelineStepIdentifier =

PipelineStepIdentifier("PostCandidatePipelinesFeatureHydration")

val globalFiltersStep: PipelineStepIdentifier = PipelineStepIdentifier("GlobalFilters")

val scoringPipelinesStep: PipelineStepIdentifier = PipelineStepIdentifier("ScoringPipelines")

val resultSelectorsStep: PipelineStepIdentifier = PipelineStepIdentifier("ResultSelectors")

val postSelectionFiltersStep: PipelineStepIdentifier = PipelineStepIdentifier(

"PostSelectionFilters")

val decoratorStep: PipelineStepIdentifier = PipelineStepIdentifier("Decorator")

val domainMarshallerStep: PipelineStepIdentifier = PipelineStepIdentifier("DomainMarshaller")

val resultSideEffectsStep: PipelineStepIdentifier = PipelineStepIdentifier("ResultSideEffects")

val transportMarshallerStep: PipelineStepIdentifier = PipelineStepIdentifier(

"TransportMarshaller")

/\*\* All the Steps which are executed by a [[RecommendationPipeline]] in the order in which they are run \*/

override val stepsInOrder: Seq[PipelineStepIdentifier] = Seq(

qualityFactorStep,

gatesStep,

fetchQueryFeaturesStep,

fetchQueryFeaturesPhase2Step,

asyncFeaturesStep(candidatePipelinesStep),

candidatePipelinesStep,

asyncFeaturesStep(dependentCandidatePipelinesStep),

dependentCandidatePipelinesStep,

asyncFeaturesStep(postCandidatePipelinesSelectorsStep),

postCandidatePipelinesSelectorsStep,

asyncFeaturesStep(postCandidatePipelinesFeatureHydrationStep),

postCandidatePipelinesFeatureHydrationStep,

asyncFeaturesStep(globalFiltersStep),

globalFiltersStep,

asyncFeaturesStep(scoringPipelinesStep),

scoringPipelinesStep,

asyncFeaturesStep(resultSelectorsStep),

resultSelectorsStep,

asyncFeaturesStep(postSelectionFiltersStep),

postSelectionFiltersStep,

asyncFeaturesStep(decoratorStep),

decoratorStep,

domainMarshallerStep,

asyncFeaturesStep(resultSideEffectsStep),

resultSideEffectsStep,

transportMarshallerStep

)

/\*\*

\* All the Steps which an [[com.twitter.product\_mixer.core.functional\_component.feature\_hydrator.AsyncHydrator AsyncHydrator]]

\* can be configured to [[com.twitter.product\_mixer.core.functional\_component.feature\_hydrator.AsyncHydrator.hydrateBefore hydrateBefore]]

\*/

override val stepsAsyncFeatureHydrationCanBeCompletedBy: Set[PipelineStepIdentifier] = Set(

candidatePipelinesStep,

dependentCandidatePipelinesStep,

postCandidatePipelinesSelectorsStep,

postCandidatePipelinesFeatureHydrationStep,

globalFiltersStep,

scoringPipelinesStep,

resultSelectorsStep,

postSelectionFiltersStep,

decoratorStep,

resultSideEffectsStep,

)

}