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

import com.twitter.product\_mixer.core.functional\_component.candidate\_source.BaseCandidateSource

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

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

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

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

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

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

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

import com.twitter.product\_mixer.core.pipeline.step.Step

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

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.stitch.Arrow

import javax.inject.Inject

/\*\*

\* A candidate source step, which takes the query and gets csandidates from the candidate source.

\*

\* @param candidateSourceExecutor Candidate Source Executor

\* @tparam Query Type of PipelineQuery domain model

\* @tparam Candidate Type of Candidates to filter

\* @tparam State The pipeline state domain model.

\*/

case class CandidateSourceStep[

Query <: PipelineQuery,

CandidateSourceQuery,

CandidateSourceResult,

Candidate <: UniversalNoun[Any],

State <: HasQuery[Query, State] with HasCandidatesWithFeatures[Candidate, State]] @Inject() (

candidateSourceExecutor: CandidateSourceExecutor)

extends Step[

State,

CandidateSourceConfig[Query, CandidateSourceQuery, CandidateSourceResult, Candidate],

Query,

CandidateSourceExecutorResult[

Candidate

]

] {

override def isEmpty(

config: CandidateSourceConfig[Query, CandidateSourceQuery, CandidateSourceResult, Candidate]

): Boolean = false

override def adaptInput(

state: State,

config: CandidateSourceConfig[Query, CandidateSourceQuery, CandidateSourceResult, Candidate]

): Query = state.query

override def arrow(

config: CandidateSourceConfig[Query, CandidateSourceQuery, CandidateSourceResult, Candidate],

context: Executor.Context

): Arrow[Query, CandidateSourceExecutorResult[Candidate]] = candidateSourceExecutor.arrow(

config.candidateSource,

config.queryTransformer,

config.resultTransformer,

config.resultFeaturesTransformers,

context

)

override def updateState(

state: State,

executorResult: CandidateSourceExecutorResult[Candidate],

config: CandidateSourceConfig[Query, CandidateSourceQuery, CandidateSourceResult, Candidate]

): State = state

.updateQuery(

state.query

.withFeatureMap(executorResult.candidateSourceFeatureMap).asInstanceOf[

Query]).updateCandidatesWithFeatures(executorResult.candidates)

}

case class CandidateSourceConfig[

Query <: PipelineQuery,

CandidateSourceQuery,

CandidateSourceResult,

Candidate <: UniversalNoun[Any]

](

candidateSource: BaseCandidateSource[CandidateSourceQuery, CandidateSourceResult],

queryTransformer: BaseCandidatePipelineQueryTransformer[

Query,

CandidateSourceQuery

],

resultTransformer: CandidatePipelineResultsTransformer[CandidateSourceResult, Candidate],

resultFeaturesTransformers: Seq[CandidateFeatureTransformer[CandidateSourceResult]])