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

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

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\_feature\_hydrator\_executor.CandidateFeatureHydratorExecutor

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

import com.twitter.stitch.Arrow

import javax.inject.Inject

/\*\*

\* A candidate level feature hydration step, it takes the input list of candidates and the given

\* hydrators and executes them. The [[State]] object is responsible for merging the resulting

\* feature maps with the hydrated ones in its updateCandidatesWithFeatures.

\*

\* @param candidateFeatureHydratorExecutor Hydrator Executor

\* @tparam Query Type of PipelineQuery domain model

\* @tparam Candidate Type of Candidates to hydrate features for.

\* @tparam State The pipeline state domain model.

\*/

case class CandidateFeatureHydratorStep[

Query <: PipelineQuery,

Candidate <: UniversalNoun[Any],

State <: HasQuery[Query, State] with HasCandidatesWithFeatures[

Candidate,

State

]] @Inject() (

candidateFeatureHydratorExecutor: CandidateFeatureHydratorExecutor)

extends Step[State, Seq[

BaseCandidateFeatureHydrator[Query, Candidate, \_]

], CandidateFeatureHydratorExecutor.Inputs[

Query,

Candidate

], CandidateFeatureHydratorExecutorResult[Candidate]] {

override def adaptInput(

state: State,

config: Seq[BaseCandidateFeatureHydrator[Query, Candidate, \_]]

): CandidateFeatureHydratorExecutor.Inputs[Query, Candidate] =

CandidateFeatureHydratorExecutor.Inputs(state.query, state.candidatesWithFeatures)

override def arrow(

config: Seq[BaseCandidateFeatureHydrator[Query, Candidate, \_]],

context: Executor.Context

): Arrow[

CandidateFeatureHydratorExecutor.Inputs[Query, Candidate],

CandidateFeatureHydratorExecutorResult[Candidate]

] = candidateFeatureHydratorExecutor.arrow(config, context)

override def updateState(

input: State,

executorResult: CandidateFeatureHydratorExecutorResult[Candidate],

config: Seq[BaseCandidateFeatureHydrator[Query, Candidate, \_]]

): State = {

val candidatesWithHydratedFeatures = executorResult.results

if (candidatesWithHydratedFeatures.isEmpty) {

input

} else {

input.updateCandidatesWithFeatures(candidatesWithHydratedFeatures)

}

}

override def isEmpty(

config: Seq[BaseCandidateFeatureHydrator[Query, Candidate, \_]]

): Boolean =

config.isEmpty

}