package com.twitter.product\_mixer.core.pipeline.step.group\_results

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.CandidateSourceIdentifier

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.step.Step

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

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.stitch.Arrow

import javax.inject.Inject

/\*\*

\* A group results step, it takes the input list of candidates and decorations, and assembles

\* properly decorated candidates with details.

\*

\* @param groupResultsExecutor Group results executor

\* @tparam Candidate Type of candidates

\* @tparam State The pipeline state domain model.

\*/

case class GroupResultsStep[

Candidate <: UniversalNoun[Any],

State <: HasCandidatesWithDetails[State] with HasCandidatesWithFeatures[

Candidate,

State

]] @Inject() (

groupResultsExecutor: GroupResultsExecutor)

extends Step[State, CandidatePipelineContext, GroupResultsExecutorInput[

Candidate

], GroupResultsExecutorResult] {

override def isEmpty(config: CandidatePipelineContext): Boolean = false

override def adaptInput(

state: State,

config: CandidatePipelineContext

): GroupResultsExecutorInput[Candidate] = {

val presentationMap = state.candidatesWithDetails.flatMap { candidateWithDetails =>

candidateWithDetails.presentation

.map { presentation =>

candidateWithDetails.getCandidate[UniversalNoun[Any]] -> presentation

}

}.toMap

GroupResultsExecutorInput(state.candidatesWithFeatures, presentationMap)

}

override def arrow(

config: CandidatePipelineContext,

context: Executor.Context

): Arrow[GroupResultsExecutorInput[Candidate], GroupResultsExecutorResult] =

groupResultsExecutor.arrow(

config.candidatePipelineIdentifier,

config.candidateSourceIdentifier,

context)

override def updateState(

state: State,

executorResult: GroupResultsExecutorResult,

config: CandidatePipelineContext

): State = state.updateCandidatesWithDetails(executorResult.candidatesWithDetails)

}

case class CandidatePipelineContext(

candidatePipelineIdentifier: CandidatePipelineIdentifier,

candidateSourceIdentifier: CandidateSourceIdentifier)