package com.twitter.product\_mixer.core.pipeline.step.filter

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

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

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.filter\_executor.FilterExecutor

import com.twitter.product\_mixer.core.service.filter\_executor.FilterExecutorResult

import com.twitter.stitch.Arrow

import javax.inject.Inject

/\*\*

\* A candidate filter step, it takes the input list of candidates and the given filter and applies

\* the filters on the candidates in sequence, returning the final kept candidates list to State.

\*

\* @param filterExecutor Filter Executor

\* @tparam Query Type of PipelineQuery domain model

\* @tparam Candidate Type of Candidates to filter

\* @tparam State The pipeline state domain model.

\*/

case class FilterStep[

Query <: PipelineQuery,

Candidate <: UniversalNoun[Any],

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

Candidate,

State

]] @Inject() (filterExecutor: FilterExecutor)

extends Step[State, Seq[

Filter[Query, Candidate]

], (Query, Seq[CandidateWithFeatures[Candidate]]), FilterExecutorResult[Candidate]] {

override def isEmpty(config: Seq[Filter[Query, Candidate]]): Boolean = config.isEmpty

override def adaptInput(

state: State,

config: Seq[Filter[Query, Candidate]]

): (Query, Seq[CandidateWithFeatures[Candidate]]) =

(state.query, state.candidatesWithFeatures)

override def arrow(

config: Seq[Filter[Query, Candidate]],

context: Executor.Context

): Arrow[(Query, Seq[CandidateWithFeatures[Candidate]]), FilterExecutorResult[Candidate]] =

filterExecutor.arrow(config, context)

override def updateState(

state: State,

executorResult: FilterExecutorResult[Candidate],

config: Seq[Filter[Query, Candidate]]

): State = {

val keptCandidates = executorResult.result

val candidatesMap = state.candidatesWithFeatures.map { candidatesWithFeatures =>

candidatesWithFeatures.candidate -> candidatesWithFeatures

}.toMap

val newCandidates = keptCandidates.flatMap { candidate =>

candidatesMap.get(candidate)

}

state.updateCandidatesWithFeatures(newCandidates)

}

}