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

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

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

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

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

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

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

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

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

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

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

import com.twitter.product\_mixer.core.pipeline.step.pipeline\_selector.PipelineSelectorResult

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

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

import com.twitter.product\_mixer.core.service.pipeline\_executor.PipelineExecutor

import com.twitter.product\_mixer.core.service.pipeline\_executor.PipelineExecutorRequest

import com.twitter.product\_mixer.core.service.pipeline\_executor.PipelineExecutorResult

import com.twitter.stitch.Arrow

import javax.inject.Inject

/\*\*

\* Pipeline Execution step that takes a selected pipeline and executes it.

\*

\* @param pipelineExecutor Pipeline executor that executes the selected pipeline

\*

\* @tparam Query Pipeline query model with quality factor status

\* @tparam Result The expected result type

\* @tparam State The pipeline state domain model.

\*/

case class PipelineExecutorStep[

Query <: PipelineQuery,

Result,

State <: HasQuery[Query, State] with HasExecutorResults[State] with HasResult[Result]] @Inject() (

pipelineExecutor: PipelineExecutor)

extends Step[

State,

PipelineExecutorStepConfig[Query, Result],

PipelineExecutorRequest[Query],

PipelineExecutorResult[Result]

] {

override def isEmpty(config: PipelineExecutorStepConfig[Query, Result]): Boolean =

false

override def adaptInput(

state: State,

config: PipelineExecutorStepConfig[Query, Result]

): PipelineExecutorRequest[Query] = {

val pipelineSelectorResult = state.executorResultsByPipelineStep

.getOrElse(

config.selectedPipelineResultIdentifier,

throw PipelineFailure(

IllegalStateFailure,

"Missing Selected Pipeline in Pipeline Executor Step")).asInstanceOf[

PipelineSelectorResult]

PipelineExecutorRequest(state.query, pipelineSelectorResult.pipelineIdentifier)

}

override def arrow(

config: PipelineExecutorStepConfig[Query, Result],

context: Executor.Context

): Arrow[PipelineExecutorRequest[Query], PipelineExecutorResult[Result]] = pipelineExecutor.arrow(

config.pipelinesByIdentifier,

config.qualityFactorObserversByIdentifier,

context

)

// Noop since the platform will add the final result to the executor result map then state

// is responsible for reading it in [[WithResult]]

override def updateState(

state: State,

executorResult: PipelineExecutorResult[Result],

config: PipelineExecutorStepConfig[Query, Result]

): State = state

}

case class PipelineExecutorStepConfig[Query <: PipelineQuery, Result](

pipelinesByIdentifier: Map[ComponentIdentifier, Pipeline[Query, Result]],

selectedPipelineResultIdentifier: PipelineStepIdentifier,

qualityFactorObserversByIdentifier: Map[ComponentIdentifier, QualityFactorObserver])