package com.twitter.product\_mixer.core.pipeline.step.side\_effect

import com.twitter.product\_mixer.core.functional\_component.side\_effect.PipelineResultSideEffect

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

import com.twitter.product\_mixer.core.model.marshalling.HasMarshalling

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.step.Step

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

import com.twitter.product\_mixer.core.service.domain\_marshaller\_executor.DomainMarshallerExecutor

import com.twitter.product\_mixer.core.service.pipeline\_result\_side\_effect\_executor.PipelineResultSideEffectExecutor

import com.twitter.product\_mixer.core.service.selector\_executor.SelectorExecutorResult

import com.twitter.stitch.Arrow

import javax.inject.Inject

/\*\*

\* A side effect step, it takes the input list of side effects and and executes them.

\*

\* @param sideEffectExecutor Side Effect Executor

\*

\* @tparam Query Type of PipelineQuery domain model

\* @tparam DomainResultType Domain Marshaller result type

\* @tparam State The pipeline state domain model.

\*/

case class SideEffectStep[

Query <: PipelineQuery,

DomainResultType <: HasMarshalling,

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

sideEffectExecutor: PipelineResultSideEffectExecutor)

extends Step[

State,

PipelineStepConfig[Query, DomainResultType],

PipelineResultSideEffect.Inputs[

Query,

DomainResultType

],

PipelineResultSideEffectExecutor.Result

] {

override def isEmpty(config: PipelineStepConfig[Query, DomainResultType]): Boolean =

config.sideEffects.isEmpty

override def adaptInput(

state: State,

config: PipelineStepConfig[Query, DomainResultType]

): PipelineResultSideEffect.Inputs[Query, DomainResultType] = {

val selectorResults = state.executorResultsByPipelineStep

.getOrElse(

config.selectorStepIdentifier,

throw PipelineFailure(

IllegalStateFailure,

"Missing Selector Result in Side Effect Step")).asInstanceOf[SelectorExecutorResult]

val domainMarshallerResult = state.executorResultsByPipelineStep

.getOrElse(

config.domainMarshallerStepIdentifier,

throw PipelineFailure(

IllegalStateFailure,

"Missing Domain Marshaller Result in Side Effect Step")).asInstanceOf[

DomainMarshallerExecutor.Result[DomainResultType]]

PipelineResultSideEffect.Inputs(

query = state.query,

selectedCandidates = selectorResults.selectedCandidates,

remainingCandidates = selectorResults.remainingCandidates,

droppedCandidates = selectorResults.droppedCandidates,

response = domainMarshallerResult.result

)

}

override def arrow(

config: PipelineStepConfig[Query, DomainResultType],

context: Executor.Context

): Arrow[

PipelineResultSideEffect.Inputs[Query, DomainResultType],

PipelineResultSideEffectExecutor.Result

] = sideEffectExecutor.arrow(config.sideEffects, context)

override def updateState(

state: State,

executorResult: PipelineResultSideEffectExecutor.Result,

config: PipelineStepConfig[Query, DomainResultType]

): State = state

}

/\*\*

\* Wrapper case class containing side effects to be executed and other information needed to execute

\* @param sideEffects The side effects to execute.

\* @param selectorStepIdentifier The identifier of the selector step in the parent

\* pipeline to get selection results from.

\* @param domainMarshallerStepIdentifier The identifier of the domain marshaller step in the parent

\* pipeline to get domain marshalled results from.

\*

\* @tparam Query Type of PipelineQuery domain model

\* @tparam DomainResultType Domain Marshaller result type

\*/

case class PipelineStepConfig[Query <: PipelineQuery, DomainResultType <: HasMarshalling](

sideEffects: Seq[PipelineResultSideEffect[Query, DomainResultType]],

selectorStepIdentifier: PipelineStepIdentifier,

domainMarshallerStepIdentifier: PipelineStepIdentifier)