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

import com.twitter.product\_mixer.core.functional\_component.gate.BaseGate

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

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.gate\_executor.GateExecutor

import com.twitter.product\_mixer.core.service.gate\_executor.GateExecutorResult

import com.twitter.stitch.Arrow

import javax.inject.Inject

/\*\*

\* A gate step, it takes the query and the given gates and executes them. Gates do not update state

\* if they return continue, and throw an exception if any gate says stopped, thus no state changes

\* are expected in this step. The [[NewPipelineArrowBuilder]] and [[PipelineStep]] handle short

\* circuiting the pipeline's execution if this throws.

\*

\* @param gateExecutor Gate Executor for executing the gates

\* @tparam Query Type of PipelineQuery domain model

\* @tparam State The pipeline state domain model.

\*/

case class GateStep[Query <: PipelineQuery, State <: HasQuery[Query, State]] @Inject() (

gateExecutor: GateExecutor)

extends Step[State, Seq[BaseGate[Query]], Query, GateExecutorResult] {

override def adaptInput(state: State, config: Seq[BaseGate[Query]]): Query = state.query

override def arrow(

config: Seq[BaseGate[Query]],

context: Executor.Context

): Arrow[Query, GateExecutorResult] = gateExecutor.arrow(config, context)

// Gate Executor is a noop, if it continues, the state isn't changed. If it stops the world,

// an exception gets thrown.

override def updateState(

input: State,

executorResult: GateExecutorResult,

config: Seq[BaseGate[Query]]

): State = input

override def isEmpty(config: Seq[BaseGate[Query]]): Boolean = config.isEmpty

}