package com.twitter.product\_mixer.component\_library.side\_effect

import com.twitter.product\_mixer.component\_library.side\_effect.ParamGatedPipelineResultSideEffect.IdentifierPrefix

import com.twitter.product\_mixer.core.functional\_component.common.alert.Alert

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

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

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

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

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

import com.twitter.product\_mixer.core.model.common.presentation.CandidateWithDetails

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

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

import com.twitter.stitch.Stitch

import com.twitter.timelines.configapi.Param

/\*\*

\* A [[PipelineResultSideEffect]] with [[Conditionally]] based on a [[Param]]

\*

\* @param enabledParam the param to turn this filter on and off

\* @param sideEffect the underlying side effect to run when `enabledParam` is true

\* @tparam Query The domain model for the query or request

\*/

sealed case class ParamGatedPipelineResultSideEffect[

-Query <: PipelineQuery,

ResultType <: HasMarshalling

] private (

enabledParam: Param[Boolean],

sideEffect: PipelineResultSideEffect[Query, ResultType])

extends PipelineResultSideEffect[Query, ResultType]

with PipelineResultSideEffect.Conditionally[Query, ResultType] {

override val identifier: SideEffectIdentifier = SideEffectIdentifier(

IdentifierPrefix + sideEffect.identifier.name)

override val alerts: Seq[Alert] = sideEffect.alerts

override def onlyIf(

query: Query,

selectedCandidates: Seq[CandidateWithDetails],

remainingCandidates: Seq[CandidateWithDetails],

droppedCandidates: Seq[CandidateWithDetails],

response: ResultType

): Boolean =

Conditionally.and(

PipelineResultSideEffect

.Inputs(query, selectedCandidates, remainingCandidates, droppedCandidates, response),

sideEffect,

query.params(enabledParam))

override def apply(inputs: PipelineResultSideEffect.Inputs[Query, ResultType]): Stitch[Unit] =

sideEffect.apply(inputs)

}

object ParamGatedPipelineResultSideEffect {

val IdentifierPrefix = "ParamGated"

/\*\*

\* A [[PipelineResultSideEffect]] with [[Conditionally]] based on a [[Param]]

\*

\* @param enabledParam the param to turn this filter on and off

\* @param sideEffect the underlying side effect to run when `enabledParam` is true

\* @tparam Query The domain model for the query or request

\*/

def apply[Query <: PipelineQuery, ResultType <: HasMarshalling](

enabledParam: Param[Boolean],

sideEffect: PipelineResultSideEffect[Query, ResultType]

): ParamGatedPipelineResultSideEffect[Query, ResultType] = {

sideEffect match {

case \_: FailOpen =>

new ParamGatedPipelineResultSideEffect(enabledParam, sideEffect)

with ExecuteSynchronously

with FailOpen

case \_: ExecuteSynchronously =>

new ParamGatedPipelineResultSideEffect(enabledParam, sideEffect) with ExecuteSynchronously

case \_ =>

new ParamGatedPipelineResultSideEffect(enabledParam, sideEffect)

}

}

}