package com.twitter.product\_mixer.core.functional\_component.gate

import com.twitter.product\_mixer.core.functional\_component.gate.Gate.SkippedResult

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

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

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

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

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.stitch.Arrow

import com.twitter.stitch.Stitch

/\*\*

\* A gate controls if a pipeline or other component is executed

\*

\* A gate is mostly controlled by it's `shouldContinue` function - when this function

\* returns true, execution Continues.

\*

\* Gates also have a optional `shouldSkip`- When it returns

\* true, then we Continue without executing `main`.

\*

\* @tparam Query The query type that the gate will receive as input

\*

\* @return A GateResult includes both the boolean `continue` and a specific reason. See [[GateResult]] for more

\* information.

\*/

sealed trait BaseGate[-Query <: PipelineQuery] extends Component {

override val identifier: GateIdentifier

/\*\*

\* If a shouldSkip returns true, the gate returns a Skip(continue=true) without executing

\* the main predicate. We expect this to be useful for debugging, dogfooding, etc.

\*/

def shouldSkip(query: Query): Stitch[Boolean] = Stitch.False

/\*\*

\* The main predicate that controls this gate. If this predicate returns true, the gate returns Continue.

\*/

def shouldContinue(query: Query): Stitch[Boolean]

/\*\* returns a [[GateResult]] to determine whether a pipeline should be executed based on `t` \*/

final def apply(t: Query): Stitch[GateResult] = {

shouldSkip(t).flatMap { skipResult =>

if (skipResult) {

SkippedResult

} else {

shouldContinue(t).map { mainResult =>

if (mainResult) GateResult.Continue else GateResult.Stop

}

}

}

}

/\*\* Arrow representation of `this` [[Gate]] \*/

final def arrow: Arrow[Query, GateResult] = Arrow(apply)

}

/\*\*

\* A regular Gate which only has access to the Query typed PipelineQuery. This can be used anywhere

\* Gates are available.

\*

\* A gate is mostly controlled by it's `shouldContinue` function - when this function

\* returns true, execution Continues.

\*

\* Gates also have a optional `shouldSkip`- When it returns

\* true, then we Continue without executing `main`.

\* @tparam Query The query type that the gate will receive as input

\*

\* @return A GateResult includes both the boolean `continue` and a specific reason. See [[GateResult]] for more

\* information.

\*/

trait Gate[-Query <: PipelineQuery] extends BaseGate[Query]

/\*\*

\* A Query And Candidate Gate which only has access both to the Query typed PipelineQuery and the

\* list of previously fetched candidates. This can be used on dependent candidate pipelines to

\* make a decision on whether to enable/disable them based on previous candidates.

\*

\* A gate is mostly controlled by it's `shouldContinue` function - when this function

\* returns true, execution Continues.

\*

\* Gates also have a optional `shouldSkip`- When it returns

\* true, then we Continue without executing `main`.

\*

\* @tparam Query The query type that the gate will receive as input

\*

\* @return A GateResult includes both the boolean `continue` and a specific reason. See [[GateResult]] for more

\* information.

\*/

trait QueryAndCandidateGate[-Query <: PipelineQuery] extends BaseGate[Query] {

/\*\*

\* If a shouldSkip returns true, the gate returns a Skip(continue=true) without executing

\* the main predicate. We expect this to be useful for debugging, dogfooding, etc.

\*/

def shouldSkip(query: Query, candidates: Seq[CandidateWithDetails]): Stitch[Boolean] =

Stitch.False

/\*\*

\* The main predicate that controls this gate. If this predicate returns true, the gate returns Continue.

\*/

def shouldContinue(query: Query, candidates: Seq[CandidateWithDetails]): Stitch[Boolean]

final override def shouldSkip(query: Query): Stitch[Boolean] = {

val candidates = query.features

.map(\_.get(CandidatePipelineResults)).getOrElse(

throw PipelineFailure(

IllegalStateFailure,

"Candidate Pipeline Results Feature missing from query features"))

shouldSkip(query, candidates)

}

final override def shouldContinue(query: Query): Stitch[Boolean] = {

val candidates = query.features

.map(\_.get(CandidatePipelineResults)).getOrElse(

throw PipelineFailure(

IllegalStateFailure,

"Candidate Pipeline Results Feature missing from query features"))

shouldContinue(query, candidates)

}

}

object Gate {

val SkippedResult: Stitch[GateResult] = Stitch.value(GateResult.Skipped)

}