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

import com.twitter.product\_mixer.component\_library.model.candidate.CursorCandidate

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

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

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

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

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

import com.twitter.util.Return

import com.twitter.util.Throw

import com.twitter.util.Try

/\*\*

\* Pipelines return a PipelineResult.

\*

\* This allows us to return a single main result (optionally, incase the pipeline didn't execute successfully), but

\* still have a detailed response object to show how that result was produced.

\*/

trait PipelineResult[ResultType] {

val failure: Option[PipelineFailure]

val result: Option[ResultType]

def withFailure(failure: PipelineFailure): PipelineResult[ResultType]

def withResult(result: ResultType): PipelineResult[ResultType]

def resultSize(): Int

private[pipeline] def stopExecuting: Boolean = failure.isDefined || result.isDefined

final def toTry: Try[this.type] = (result, failure) match {

case (\_, Some(failure)) =>

Throw(failure)

case (\_: Some[ResultType], \_) =>

Return(this)

// Pipelines should always finish with either a result or a failure

case \_ => Throw(PipelineFailure(ExecutionFailed, "Pipeline did not execute"))

}

final def toResultTry: Try[ResultType] = {

// `.get` is safe here because `toTry` guarantees a value in the `Return` case

toTry.map(\_.result.get)

}

}

object PipelineResult {

/\*\*

\* Track number of candidates returned by a Pipeline. Cursors are excluded from this

\* count and modules are counted as the sum of their candidates.

\*

\* @note this is a somewhat subjective measure of 'size' and it is spread across pipeline

\* definitions as well as selectors.

\*/

def resultSize(results: Seq[CandidateWithDetails]): Int = results.map {

case module: ModuleCandidateWithDetails => resultSize(module.candidates)

case ItemCandidateWithDetails(\_: CursorCandidate, \_, \_) => 0

case \_ => 1

}.sum

}