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

import com.twitter.product\_mixer.core.functional\_component.common.access\_policy.AccessPolicy

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

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

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

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

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

import com.twitter.product\_mixer.core.model.marshalling.request.Product

import com.twitter.product\_mixer.core.model.marshalling.request.Request

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

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

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

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

import com.twitter.product\_mixer.core.product.ProductParamConfig

import com.twitter.product\_mixer.core.quality\_factor.QualityFactorConfig

import com.twitter.timelines.configapi.Params

trait ProductPipelineConfig[TRequest <: Request, Query <: PipelineQuery, Response]

extends PipelineConfig {

override val identifier: ProductPipelineIdentifier

val product: Product

val paramConfig: ProductParamConfig

/\*\*

\* Product Pipeline Gates will be executed before any other step (including retrieval from mixer

\* pipelines). They're executed sequentially, and any "Stop" result will prevent pipeline execution.

\*/

def gates: Seq[Gate[Query]] = Seq.empty

def pipelineQueryTransformer(request: TRequest, params: Params): Query

/\*\*

\* A list of all pipelines that power this product directly (there is no need to include pipelines

\* called by those pipelines).

\*

\* Only pipeline from this list should referenced from the pipelineSelector

\*/

def pipelines: Seq[PipelineConfig]

/\*\*

\* A pipeline selector selects a pipeline (from the list in `def pipelines`) to handle the

\* current request.

\*/

def pipelineSelector(query: Query): ComponentIdentifier

/\*\*

\*\* [[qualityFactorConfigs]] associates [[QualityFactorConfig]]s to specific pipelines

\* using [[ComponentIdentifier]].

\*/

def qualityFactorConfigs: Map[ComponentIdentifier, QualityFactorConfig] =

Map.empty

/\*\*

\* By default (for safety), product mixer pipelines do not allow logged out requests.

\* A "DenyLoggedOutUsersGate" will be generated and added to the pipeline.

\*

\* You can disable this behavior by overriding `denyLoggedOutUsers` with False.

\*/

val denyLoggedOutUsers: Boolean = true

/\*\*

\* A pipeline can define a partial function to rescue failures here. They will be treated as failures

\* from a monitoring standpoint, and cancellation exceptions will always be propagated (they cannot be caught here).

\*/

def failureClassifier: PartialFunction[Throwable, PipelineFailure] = PartialFunction.empty

/\*\*

\* Alerts can be used to indicate the pipeline's service level objectives. Alerts and

\* dashboards will be automatically created based on this information.

\*/

val alerts: Seq[Alert] = Seq.empty

/\*\*

\* Access Policies can be used to gate who can query a product from Product Mixer's query tool

\* (go/turntable).

\*

\* This will typically be gated by an LDAP group associated with your team. For example:

\*

\* {{{

\* override val debugAccessPolicies: Set[AccessPolicy] = Set(AllowedLdapGroups("NAME"))

\* }}}

\*

\* You can disable all queries by using the [[com.twitter.product\_mixer.core.functional\_component.common.access\_policy.BlockEverything]] policy.

\*/

val debugAccessPolicies: Set[AccessPolicy]

}

object ProductPipelineConfig extends PipelineConfigCompanion {

val pipelineQueryTransformerStep: PipelineStepIdentifier = PipelineStepIdentifier(

"PipelineQueryTransformer")

val qualityFactorStep: PipelineStepIdentifier = PipelineStepIdentifier("QualityFactor")

val gatesStep: PipelineStepIdentifier = PipelineStepIdentifier("Gates")

val pipelineSelectorStep: PipelineStepIdentifier = PipelineStepIdentifier("PipelineSelector")

val pipelineExecutionStep: PipelineStepIdentifier = PipelineStepIdentifier("PipelineExecution")

/\*\* All the Steps which are executed by a [[ProductPipeline]] in the order in which they are run \*/

override val stepsInOrder: Seq[PipelineStepIdentifier] = Seq(

pipelineQueryTransformerStep,

qualityFactorStep,

gatesStep,

pipelineSelectorStep,

pipelineExecutionStep

)

}