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

import com.twitter.finagle.http.Request

import com.twitter.inject.Injector

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

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

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

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

import com.twitter.product\_mixer.core.pipeline.mixer.MixerPipelineConfig

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

import com.twitter.product\_mixer.core.pipeline.recommendation.RecommendationPipelineConfig

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

import com.twitter.product\_mixer.core.service.component\_registry

import com.twitter.product\_mixer.core.service.component\_registry.ComponentRegistry

import com.twitter.product\_mixer.core.service.component\_registry.ComponentRegistrySnapshot

import com.twitter.util.Future

case class GetComponentRegistryHandler(injector: Injector) {

lazy val componentRegistry: ComponentRegistry = injector.instance[ComponentRegistry]

def apply(request: Request): Future[ComponentRegistryResponse] = {

componentRegistry.get.map { currentComponentRegistry: ComponentRegistrySnapshot =>

val registeredComponents = currentComponentRegistry.getAllRegisteredComponents.map {

registeredComponent =>

val componentIdentifier = registeredComponent.identifier

val childComponents = currentComponentRegistry

.getChildComponents(componentIdentifier)

.map { childComponent =>

ChildComponent(

componentType = childComponent.componentType,

name = childComponent.name,

relativeScopes = componentIdentifier.toScopes ++ childComponent.toScopes,

qualityFactorMonitoringConfig =

buildQualityFactoringMonitoringConfig(registeredComponent, childComponent)

)

}

RegisteredComponent(

componentType = componentIdentifier.componentType,

name = componentIdentifier.name,

scopes = componentIdentifier.toScopes,

children = childComponents,

alertConfig = Some(registeredComponent.component.alerts.map(AlertConfig.apply)),

sourceFile = Some(registeredComponent.sourceFile),

debugAccessPolicies = Some(registeredComponent.component match {

case withDebugAccessPolicies: WithDebugAccessPolicies =>

withDebugAccessPolicies.debugAccessPolicies

case \_ => Set.empty

})

)

}

ComponentRegistryResponse(registeredComponents)

}

}

private def buildQualityFactoringMonitoringConfig(

parent: component\_registry.RegisteredComponent,

child: ComponentIdentifier

): Option[QualityFactorMonitoringConfig] = {

val qualityFactorConfigs: Option[Map[ComponentIdentifier, QualityFactorConfig]] =

parent.component match {

case pipeline: Pipeline[\_, \_] =>

pipeline.config match {

case config: RecommendationPipelineConfig[\_, \_, \_, \_] =>

Some(config.qualityFactorConfigs)

case config: MixerPipelineConfig[\_, \_, \_] =>

Some(

config.qualityFactorConfigs

.asInstanceOf[Map[ComponentIdentifier, QualityFactorConfig]])

case config: ProductPipelineConfig[\_, \_, \_] =>

Some(config.qualityFactorConfigs)

case \_ => None

}

case \_ => None

}

val qfConfigForChild: Option[QualityFactorConfig] = qualityFactorConfigs.flatMap(\_.get(child))

qfConfigForChild.map { qfConfig =>

QualityFactorMonitoringConfig(

boundMin = qfConfig.qualityFactorBounds.bounds.minInclusive,

boundMax = qfConfig.qualityFactorBounds.bounds.maxInclusive

)

}

}

}

case class RegisteredComponent(

componentType: String,

name: String,

scopes: Seq[String],

children: Seq[ChildComponent],

alertConfig: Option[Seq[AlertConfig]],

sourceFile: Option[String],

debugAccessPolicies: Option[Set[AccessPolicy]])

case class ChildComponent(

componentType: String,

name: String,

relativeScopes: Seq[String],

qualityFactorMonitoringConfig: Option[QualityFactorMonitoringConfig])

/\*\*

\* The shape of the data returned to callers after hitting the `component-registry` endpoint

\*

\* @note changes to [[ComponentRegistryResponse]] or contained types should be reflected

\* in dashboard generation code in the `monitoring-configs/product\_mixer` directory.

\*/

case class ComponentRegistryResponse(

registeredComponents: Seq[RegisteredComponent])

case class ProductPipeline(identifier: String)

case class ProductPipelinesResponse(productPipelines: Seq[ProductPipeline])