package com.twitter.product\_mixer.core.functional\_component.candidate\_source.product\_pipeline

import com.google.inject.Provider

import com.twitter.product\_mixer.core.functional\_component.candidate\_source.CandidateSource

import com.twitter.product\_mixer.core.functional\_component.configapi.ParamsBuilder

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

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

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

import com.twitter.product\_mixer.core.product.registry.ProductPipelineRegistry

import com.twitter.stitch.Stitch

import scala.reflect.runtime.universe.\_

/\*\*

\* A [[CandidateSource]] for getting candidates from a different

\* [[com.twitter.product\_mixer.core.model.marshalling.request.Product]] within the same Product

\* Mixer-based service. This is useful when calling a RecommendationPipeline-based Product from a

\* MixerPipeline-based Product. In this scenario, the two Products can remain

\* independent and encapsulated within the Product Mixer service, which provides future optionality

\* for migrating one of the two products into a new Product Mixer-based service based on the

\* scaling needs.

\*

\* @tparam Query [[PipelineQuery]] from the originating Product

\* @tparam MixerRequest the [[Request]] domain model for the Product Mixer service. Adds a Context

\* bound (syntactic sugar) to add TypeTag to implicit scope for

\* [[ProductPipelineRegistry.getProductPipeline()]]. Note that `trait` does not

\* support context bounds, so this abstraction is expressed as an

\* `abstract class`.

\* @tparam ProductPipelineResult the return type of the candidate source Product. Adds a Context

\* bound (syntactic sugar) to add TypeTag to implicit scope for

\* [[ProductPipelineRegistry.getProductPipeline()]]

\* @tparam Candidate the type of candidate returned by this candidate source, which is typically

\* extracted from within the ProductPipelineResult type

\*/

abstract class ProductPipelineCandidateSource[

-Query <: PipelineQuery,

MixerRequest <: Request: TypeTag,

ProductPipelineResult: TypeTag,

+Candidate]

extends CandidateSource[Query, Candidate] {

/\*\*

\* @note Define as a Guice [[Provider]] in order to break the circular injection dependency

\*/

val productPipelineRegistry: Provider[ProductPipelineRegistry]

/\*\*

\* @note Define as a Guice [[Provider]] in order to break the circular injection dependency

\*/

val paramsBuilder: Provider[ParamsBuilder]

def pipelineRequestTransformer(currentPipelineQuery: Query): MixerRequest

def productPipelineResultTransformer(productPipelineResult: ProductPipelineResult): Seq[Candidate]

override def apply(query: Query): Stitch[Seq[Candidate]] = {

val request = pipelineRequestTransformer(query)

val params = paramsBuilder

.get().build(

clientContext = request.clientContext,

product = request.product,

featureOverrides = request.debugParams.flatMap(\_.featureOverrides).getOrElse(Map.empty)

)

productPipelineRegistry

.get()

.getProductPipeline[MixerRequest, ProductPipelineResult](request.product)

.process(ProductPipelineRequest(request, params))

.map(productPipelineResultTransformer)

}

}