package com.twitter.product\_mixer.component\_library.feature\_hydrator.candidate.param\_gated.featurestorev1

import com.twitter.ml.featurestore.lib.EntityId

import com.twitter.product\_mixer.component\_library.feature\_hydrator.candidate.param\_gated.featurestorev1.ParamGatedFeatureStoreV1CandidateFeatureHydrator.IdentifierPrefix

import com.twitter.product\_mixer.core.feature.featuremap.FeatureMap

import com.twitter.product\_mixer.core.feature.featurestorev1.BaseFeatureStoreV1CandidateFeature

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

import com.twitter.product\_mixer.core.functional\_component.feature\_hydrator.featurestorev1.FeatureStoreV1CandidateFeatureHydrator

import com.twitter.product\_mixer.core.functional\_component.feature\_hydrator.featurestorev1.FeatureStoreV1DynamicClientBuilder

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

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

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

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

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

import com.twitter.stitch.Stitch

import com.twitter.timelines.configapi.Param

/\*\*

\* A [[FeatureStoreV1CandidateFeatureHydrator]] with [[Conditionally]] based on a [[Param]]

\*

\* @param enabledParam the param to turn this [[FeatureStoreV1CandidateFeatureHydrator]] on and off

\* @param candidateFeatureHydrator the underlying [[FeatureStoreV1CandidateFeatureHydrator]] to run when `enabledParam` is true

\* @tparam Query The domain model for the query or request

\* @tparam Candidate The type of the candidates

\*/

case class ParamGatedFeatureStoreV1CandidateFeatureHydrator[

Query <: PipelineQuery,

Candidate <: UniversalNoun[Any]

](

enabledParam: Param[Boolean],

candidateFeatureHydrator: FeatureStoreV1CandidateFeatureHydrator[Query, Candidate])

extends FeatureStoreV1CandidateFeatureHydrator[Query, Candidate]

with Conditionally[Query] {

override val identifier: FeatureHydratorIdentifier = FeatureHydratorIdentifier(

IdentifierPrefix + candidateFeatureHydrator.identifier.name)

override val alerts: Seq[Alert] = candidateFeatureHydrator.alerts

override val features: Set[

BaseFeatureStoreV1CandidateFeature[Query, Candidate, \_ <: EntityId, \_]

] = candidateFeatureHydrator.features

override val clientBuilder: FeatureStoreV1DynamicClientBuilder =

candidateFeatureHydrator.clientBuilder

override def onlyIf(query: Query): Boolean =

Conditionally.and(query, candidateFeatureHydrator, query.params(enabledParam))

override def apply(

query: Query,

candidates: Seq[CandidateWithFeatures[Candidate]]

): Stitch[Seq[FeatureMap]] = candidateFeatureHydrator(query, candidates)

}

object ParamGatedFeatureStoreV1CandidateFeatureHydrator {

val IdentifierPrefix = "ParamGated"

}