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

import com.twitter.ml.api.util.SRichDataRecord

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

import com.twitter.ml.featurestore.lib.data.PredictionRecordAdapter

import com.twitter.ml.featurestore.lib.entity.EntityWithId

import com.twitter.ml.featurestore.lib.online.FeatureStoreRequest

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

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

import com.twitter.product\_mixer.core.feature.featurestorev1.BaseFeatureStoreV1QueryFeature

import com.twitter.product\_mixer.core.feature.featurestorev1.FeatureStoreV1QueryEntity

import com.twitter.product\_mixer.core.feature.featurestorev1.featurevalue.FeatureStoreV1Response

import com.twitter.product\_mixer.core.feature.featurestorev1.featurevalue.FeatureStoreV1ResponseFeature

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

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

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

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

import com.twitter.stitch.Stitch

import com.twitter.util.logging.Logging

trait FeatureStoreV1QueryFeatureHydrator[Query <: PipelineQuery]

extends BaseQueryFeatureHydrator[

Query,

BaseFeatureStoreV1QueryFeature[Query, \_ <: EntityId, \_]

]

with Logging {

def features: Set[BaseFeatureStoreV1QueryFeature[Query, \_ <: EntityId, \_]]

def clientBuilder: FeatureStoreV1DynamicClientBuilder

private lazy val hydrationConfig = FeatureStoreV1QueryFeatureHydrationConfig(features)

private lazy val client = clientBuilder.build(hydrationConfig)

private lazy val datasetToFeatures =

FeatureStoreDatasetErrorHandler.datasetToFeaturesMapping(features)

private lazy val dataRecordAdapter =

PredictionRecordAdapter.oneToOne(hydrationConfig.allBoundFeatures)

private lazy val featureContext = hydrationConfig.allBoundFeatures.toFeatureContext

override def hydrate(

query: Query

): Stitch[FeatureMap] = {

// Duplicate entities are expected across features, so de-dupe via the Set before converting to Seq

val entities: Seq[FeatureStoreV1QueryEntity[Query, \_ <: EntityId]] =

features.map(\_.entity).toSeq

val entityIds: Seq[EntityWithId[\_ <: EntityId]] = entities.map(\_.entityWithId(query))

val featureStoreRequest = Seq(FeatureStoreRequest(entityIds = entityIds))

val featureMap = client(featureStoreRequest, query).map { predictionRecords =>

// Should not happen as FSv1 is guaranteed to return a prediction record per feature store request

val predictionRecord = predictionRecords.headOption.getOrElse {

throw PipelineFailure(

FeatureHydrationFailed,

"Unexpected empty response from Feature Store V1 while hydrating query features")

}

val datasetErrors = predictionRecord.getDatasetHydrationErrors

val errorMap =

FeatureStoreDatasetErrorHandler.featureToHydrationErrors(datasetToFeatures, datasetErrors)

if (errorMap.nonEmpty) {

logger.debug(() => s"$identifier hydration errors for query: $errorMap")

}

val richDataRecord =

SRichDataRecord(dataRecordAdapter.adaptToDataRecord(predictionRecord), featureContext)

val featureStoreResponse =

FeatureStoreV1Response(richDataRecord, errorMap)

FeatureMapBuilder().add(FeatureStoreV1ResponseFeature, featureStoreResponse).build()

}

Stitch.callFuture(featureMap)

}

}