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

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

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

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

import com.twitter.ml.featurestore.lib.dataset.DatasetId

import com.twitter.product\_mixer.core.feature.featurestorev1.BaseFeatureStoreV1Feature

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

object FeatureStoreDatasetErrorHandler {

/\*\*

\* This function takes a set of feature store features and constructs a mapping from the underlying

\* feature store dataset back to the features. This is useful for looking up what ProMix features

\* failed based off of a failed feature store dataset at request time. A ProMix feature can be

\* powered by multiple feature store datasets, and conversely, a dataset can be used by many features.

\*/

def datasetToFeaturesMapping[

Query <: PipelineQuery,

Input,

FeatureType <: BaseFeatureStoreV1Feature[Query, Input, \_ <: EntityId, \_]

](

features: Set[FeatureType]

): Map[DatasetId, Set[FeatureType]] = {

val datasetsAndFeatures: Set[(DatasetId, FeatureType)] = features

.flatMap { feature: FeatureType =>

feature.boundFeatureSet.sourceDatasets.map(\_.id).map { datasetId: DatasetId =>

datasetId -> feature

}

}

datasetsAndFeatures

.groupBy { case (datasetId, \_) => datasetId }.mapValues(\_.map {

case (\_, feature) => feature

})

}

/\*\*

\* This takes a mapping of Feature Store Dataset => ProMix Features, as well as the dataset errors

\* from PredictionRecord and computing a final, deduped mapping from ProMix Feature to Exceptions.

\*/

def featureToHydrationErrors[

Query <: PipelineQuery,

Input,

FeatureType <: BaseFeatureStoreV1Feature[Query, Input, \_ <: EntityId, \_]

](

datasetToFeatures: Map[DatasetId, Set[

FeatureType

]],

errorsByDatasetId: DatasetErrorsById

): Map[FeatureType, Set[HydrationError]] = {

val hasError = errorsByDatasetId.datasets.nonEmpty

if (hasError) {

val featuresAndErrors: Set[(FeatureType, Set[HydrationError])] = errorsByDatasetId.datasets

.flatMap { id: DatasetId =>

val errors: Set[HydrationError] = errorsByDatasetId.get(id).values.toSet

if (errors.nonEmpty) {

val datasetFeatures: Set[FeatureType] = datasetToFeatures.getOrElse(id, Set.empty)

datasetFeatures.map { feature =>

feature -> errors

}.toSeq

} else {

Seq.empty

}

}

featuresAndErrors

.groupBy { case (feature, \_) => feature }.mapValues(\_.flatMap {

case (\_, errors: Set[HydrationError]) => errors

})

} else {

Map.empty

}

}

}