package com.twitter.follow\_recommendations.common.feature\_hydration.sources

import com.google.inject.Inject

import com.google.inject.Provides

import com.google.inject.Singleton

import com.twitter.follow\_recommendations.common.feature\_hydration.common.FeatureSource

import com.twitter.follow\_recommendations.common.feature\_hydration.common.FeatureSourceId

import com.twitter.follow\_recommendations.common.feature\_hydration.common.HasPreFetchedFeature

import com.twitter.follow\_recommendations.common.models.CandidateUser

import com.twitter.follow\_recommendations.common.models.HasDisplayLocation

import com.twitter.follow\_recommendations.common.models.HasSimilarToContext

import com.twitter.ml.api.DataRecord

import com.twitter.ml.api.DataRecordMerger

import com.twitter.ml.api.FeatureContext

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

import com.twitter.stitch.Stitch

import com.twitter.timelines.configapi.HasParams

/\*\*

\* This source wraps around the separate sources that we hydrate features from

\* @param featureStoreSource gets features that require a RPC call to feature store

\* @param stratoFeatureHydrationSource gets features that require a RPC call to strato columns

\* @param clientContextSource gets features that are already present in the request context

\* @param candidateAlgorithmSource gets features that are already present from candidate generation

\* @param preFetchedFeatureSource gets features that were prehydrated (shared in request lifecycle)

\*/

@Provides

@Singleton

class UserScoringFeatureSource @Inject() (

featureStoreSource: FeatureStoreSource,

featureStoreGizmoduckSource: FeatureStoreGizmoduckSource,

featureStorePostNuxAlgorithmSource: FeatureStorePostNuxAlgorithmSource,

featureStoreTimelinesAuthorSource: FeatureStoreTimelinesAuthorSource,

featureStoreUserMetricCountsSource: FeatureStoreUserMetricCountsSource,

clientContextSource: ClientContextSource,

candidateAlgorithmSource: CandidateAlgorithmSource,

preFetchedFeatureSource: PreFetchedFeatureSource)

extends FeatureSource {

override val id: FeatureSourceId = FeatureSourceId.UserScoringFeatureSourceId

override val featureContext: FeatureContext = FeatureContext.merge(

featureStoreSource.featureContext,

featureStoreGizmoduckSource.featureContext,

featureStorePostNuxAlgorithmSource.featureContext,

featureStoreTimelinesAuthorSource.featureContext,

featureStoreUserMetricCountsSource.featureContext,

clientContextSource.featureContext,

candidateAlgorithmSource.featureContext,

preFetchedFeatureSource.featureContext,

)

val sources =

Seq(

featureStoreSource,

featureStorePostNuxAlgorithmSource,

featureStoreTimelinesAuthorSource,

featureStoreUserMetricCountsSource,

featureStoreGizmoduckSource,

clientContextSource,

candidateAlgorithmSource,

preFetchedFeatureSource

)

val dataRecordMerger = new DataRecordMerger

def hydrateFeatures(

target: HasClientContext

with HasPreFetchedFeature

with HasParams

with HasSimilarToContext

with HasDisplayLocation,

candidates: Seq[CandidateUser]

): Stitch[Map[CandidateUser, DataRecord]] = {

Stitch.collect(sources.map(\_.hydrateFeatures(target, candidates))).map { featureMaps =>

(for {

candidate <- candidates

} yield {

val combinedDataRecord = new DataRecord

featureMaps

.flatMap(\_.get(candidate).toSeq).foreach(dataRecordMerger.merge(combinedDataRecord, \_))

candidate -> combinedDataRecord

}).toMap

}

}

}