package com.twitter.home\_mixer.product.scored\_tweets.feature\_hydrator.offline\_aggregates

import com.twitter.product\_mixer.core.feature.Feature

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

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

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

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

import com.twitter.servo.repository.Repository

import com.twitter.stitch.Stitch

import com.twitter.timelines.aggregate\_interactions.thriftjava.UserAggregateInteractions

import com.twitter.timelines.data\_processing.ml\_util.aggregation\_framework.AggregateType.AggregateType

import com.twitter.timelines.data\_processing.ml\_util.aggregation\_framework.StoreConfig

import com.twitter.timelines.suggests.common.dense\_data\_record.thriftscala.DenseFeatureMetadata

import com.twitter.user\_session\_store.thriftjava.UserSession

import com.twitter.util.Future

abstract class BaseAggregateQueryFeatureHydrator(

featureRepository: Repository[Long, Option[UserSession]],

metadataRepository: Repository[Int, Option[DenseFeatureMetadata]],

feature: Feature[PipelineQuery, Option[AggregateFeaturesToDecodeWithMetadata]])

extends QueryFeatureHydrator[PipelineQuery] {

override def hydrate(query: PipelineQuery): Stitch[FeatureMap] = {

val viewerId = query.getRequiredUserId

Stitch.callFuture(

featureRepository(viewerId)

.flatMap { userSession: Option[UserSession] =>

val featuresWithMetadata: Option[Future[AggregateFeaturesToDecodeWithMetadata]] =

userSession

.flatMap(decodeUserSession(\_))

featuresWithMetadata

.map { fu: Future[AggregateFeaturesToDecodeWithMetadata] => fu.map(Some(\_)) }

.getOrElse(Future.None)

.map { value =>

FeatureMapBuilder()

.add(feature, value)

.build()

}

}

)

}

private def decodeUserSession(

session: UserSession

): Option[Future[AggregateFeaturesToDecodeWithMetadata]] = {

Option(session.user\_aggregate\_interactions).flatMap { aggregates =>

aggregates.getSetField match {

case UserAggregateInteractions.\_Fields.V17 =>

Some(

getAggregateFeaturesWithMetadata(

aggregates.getV17.user\_aggregates.versionId,

UserAggregateInteractions.v17(aggregates.getV17))

)

case \_ =>

None

}

}

}

private def getAggregateFeaturesWithMetadata(

versionId: Int,

userAggregateInteractions: UserAggregateInteractions,

): Future[AggregateFeaturesToDecodeWithMetadata] = {

metadataRepository(versionId)

.map(AggregateFeaturesToDecodeWithMetadata(\_, userAggregateInteractions))

}

}

trait BaseAggregateRootFeature

extends Feature[PipelineQuery, Option[AggregateFeaturesToDecodeWithMetadata]] {

def aggregateStores: Set[StoreConfig[\_]]

lazy val aggregateTypes: Set[AggregateType] = aggregateStores.map(\_.aggregateType)

}