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

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.home\_mixer.model.HomeFeatures.AuthorIdFeature

import com.twitter.home\_mixer.param.HomeMixerInjectionNames.UserFollowedTopicIdsRepository

import com.twitter.home\_mixer.util.ObservedKeyValueResultHandler

import com.twitter.product\_mixer.component\_library.model.candidate.TweetCandidate

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.BulkCandidateFeatureHydrator

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

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

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

import com.twitter.product\_mixer.core.util.OffloadFuturePools

import com.twitter.servo.keyvalue.KeyValueResult

import com.twitter.servo.repository.KeyValueRepository

import com.twitter.stitch.Stitch

import com.twitter.util.Future

import com.twitter.util.Try

import javax.inject.Inject

import javax.inject.Named

import javax.inject.Singleton

object UserFollowedTopicIdsFeature extends Feature[TweetCandidate, Seq[Long]]

@Singleton

class UserFollowedTopicIdsFeatureHydrator @Inject() (

@Named(UserFollowedTopicIdsRepository)

client: KeyValueRepository[Seq[Long], Long, Seq[Long]],

override val statsReceiver: StatsReceiver)

extends BulkCandidateFeatureHydrator[PipelineQuery, TweetCandidate]

with ObservedKeyValueResultHandler {

override val identifier: FeatureHydratorIdentifier =

FeatureHydratorIdentifier("UserFollowedTopicIds")

override val features: Set[Feature[\_, \_]] = Set(UserFollowedTopicIdsFeature)

override val statScope: String = identifier.toString

override def apply(

query: PipelineQuery,

candidates: Seq[CandidateWithFeatures[TweetCandidate]]

): Stitch[Seq[FeatureMap]] = OffloadFuturePools.offloadFuture {

val possiblyAuthorIds = extractKeys(candidates)

val authorIds = possiblyAuthorIds.flatten

val response: Future[KeyValueResult[Long, Seq[Long]]] =

if (authorIds.isEmpty) Future.value(KeyValueResult.empty) else client(authorIds)

response.map { result =>

possiblyAuthorIds.map { possiblyAuthorId =>

val value = observedGet(key = possiblyAuthorId, keyValueResult = result)

val transformedValue = postTransformer(value)

FeatureMapBuilder().add(UserFollowedTopicIdsFeature, transformedValue).build()

}

}

}

private def postTransformer(input: Try[Option[Seq[Long]]]): Try[Seq[Long]] = {

input.map(\_.getOrElse(Seq.empty[Long]))

}

private def extractKeys(

candidates: Seq[CandidateWithFeatures[TweetCandidate]]

): Seq[Option[Long]] = {

candidates.map { candidate =>

candidate.features

.getTry(AuthorIdFeature)

.toOption

.flatten

}

}

}