package com.twitter.home\_mixer.functional\_component.feature\_hydrator

import com.twitter.gizmoduck.{thriftscala => gt}

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

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

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

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

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

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

import com.twitter.home\_mixer.model.request.FollowingProduct

import com.twitter.home\_mixer.param.HomeGlobalParams.EnableNahFeedbackInfoParam

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.Conditionally

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

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

import com.twitter.stitch.Stitch

import com.twitter.stitch.gizmoduck.Gizmoduck

import com.twitter.util.Return

import javax.inject.Inject

import javax.inject.Singleton

protected case class ProfileNames(screenName: String, realName: String)

@Singleton

class NamesFeatureHydrator @Inject() (gizmoduck: Gizmoduck)

extends BulkCandidateFeatureHydrator[PipelineQuery, TweetCandidate]

with Conditionally[PipelineQuery] {

override val identifier: FeatureHydratorIdentifier = FeatureHydratorIdentifier("Names")

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

override def onlyIf(query: PipelineQuery): Boolean = query.product match {

case FollowingProduct => query.params(EnableNahFeedbackInfoParam)

case \_ => true

}

private val queryFields: Set[gt.QueryFields] = Set(gt.QueryFields.Profile)

/\*\*

\* The UI currently only ever displays the first 2 names in social context lines

\* E.g. "User and 3 others like" or "UserA and UserB liked"

\*/

private val MaxCountUsers = 2

override def apply(

query: PipelineQuery,

candidates: Seq[CandidateWithFeatures[TweetCandidate]]

): Stitch[Seq[FeatureMap]] = {

val candidateUserIdsMap = candidates.map { candidate =>

candidate.candidate.id ->

(candidate.features.getOrElse(FavoritedByUserIdsFeature, Nil).take(MaxCountUsers) ++

candidate.features.getOrElse(FollowedByUserIdsFeature, Nil).take(MaxCountUsers) ++

candidate.features.getOrElse(AuthorIdFeature, None) ++

candidate.features.getOrElse(SourceUserIdFeature, None)).distinct

}.toMap

val distinctUserIds = candidateUserIdsMap.values.flatten.toSeq.distinct

Stitch

.collectToTry(distinctUserIds.map(userId => gizmoduck.getUserById(userId, queryFields)))

.map { allUsers =>

val idToProfileNamesMap = allUsers.flatMap {

case Return(allUser) =>

allUser.profile

.map(profile => allUser.id -> ProfileNames(profile.screenName, profile.name))

case \_ => None

}.toMap

val validUserIds = idToProfileNamesMap.keySet

candidates.map { candidate =>

val combinedMap = candidateUserIdsMap

.getOrElse(candidate.candidate.id, Nil)

.flatMap {

case userId if validUserIds.contains(userId) =>

idToProfileNamesMap.get(userId).map(profileNames => userId -> profileNames)

case \_ => None

}

val perCandidateRealNameMap = combinedMap.map { case (k, v) => k -> v.realName }.toMap

val perCandidateScreenNameMap = combinedMap.map { case (k, v) => k -> v.screenName }.toMap

FeatureMapBuilder()

.add(ScreenNamesFeature, perCandidateScreenNameMap)

.add(RealNamesFeature, perCandidateRealNameMap)

.build()

}

}

}

}