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

import com.twitter.follow\_recommendations.{thriftscala => frs}

import com.twitter.home\_mixer.product.scored\_tweets.model.ScoredTweetsQuery

import com.twitter.product\_mixer.component\_library.candidate\_source.recommendations.UserFollowRecommendationsCandidateSource

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.candidate\_source.strato.StratoKeyView

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

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

import com.twitter.stitch.Stitch

import javax.inject.Inject

import javax.inject.Singleton

object FrsSeedUserIdsFeature extends Feature[TweetCandidate, Option[Seq[Long]]]

object FrsUserToFollowedByUserIdsFeature extends Feature[TweetCandidate, Map[Long, Seq[Long]]]

@Singleton

case class FrsSeedUsersQueryFeatureHydrator @Inject() (

userFollowRecommendationsCandidateSource: UserFollowRecommendationsCandidateSource)

extends QueryFeatureHydrator[ScoredTweetsQuery] {

private val maxUsersToFetch = 100

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

override def features: Set[Feature[\_, \_]] = Set(

FrsSeedUserIdsFeature,

FrsUserToFollowedByUserIdsFeature

)

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

val frsRequest = frs.RecommendationRequest(

clientContext = frs.ClientContext(query.getOptionalUserId),

displayLocation = frs.DisplayLocation.HomeTimelineTweetRecs,

maxResults = Some(maxUsersToFetch)

)

userFollowRecommendationsCandidateSource(StratoKeyView(frsRequest, Unit))

.map { userRecommendations: Seq[frs.UserRecommendation] =>

val seedUserIds = userRecommendations.map(\_.userId)

val seedUserIdsSet = seedUserIds.toSet

val userToFollowedByUserIds: Map[Long, Seq[Long]] = userRecommendations.flatMap {

userRecommendation =>

if (seedUserIdsSet.contains(userRecommendation.userId)) {

val followProof =

userRecommendation.reason.flatMap(\_.accountProof).flatMap(\_.followProof)

val followedByUserIds = followProof.map(\_.userIds).getOrElse(Seq.empty)

Some(userRecommendation.userId -> followedByUserIds)

} else {

None

}

}.toMap

FeatureMapBuilder()

.add(FrsSeedUserIdsFeature, Some(seedUserIds))

.add(FrsUserToFollowedByUserIdsFeature, userToFollowedByUserIds)

.build()

}

}

}