package com.twitter.home\_mixer.product.list\_recommended\_users.feature\_hydrator

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

import com.twitter.home\_mixer.product.list\_recommended\_users.model.ListRecommendedUsersFeatures.IsGizmoduckValidUserFeature

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

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.spam.rtf.{thriftscala => rtf}

import com.twitter.stitch.Stitch

import com.twitter.stitch.gizmoduck.Gizmoduck

import com.twitter.util.Return

import javax.inject.Inject

import javax.inject.Singleton

@Singleton

class IsGizmoduckValidUserFeatureHydrator @Inject() (gizmoduck: Gizmoduck)

extends BulkCandidateFeatureHydrator[PipelineQuery, UserCandidate] {

override val identifier: FeatureHydratorIdentifier =

FeatureHydratorIdentifier("IsGizmoduckValidUser")

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

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

override def apply(

query: PipelineQuery,

candidates: Seq[CandidateWithFeatures[UserCandidate]]

): Stitch[Seq[FeatureMap]] = {

val context = gt.LookupContext(

forUserId = query.getOptionalUserId,

includeProtected = true,

safetyLevel = Some(rtf.SafetyLevel.Recommendations)

)

val userIds = candidates.map(\_.candidate.id)

Stitch

.collectToTry(

userIds.map(userId => gizmoduck.getUserById(userId, queryFields, context))).map {

userResults =>

val idToUserSafetyMap = userResults

.collect {

case Return(user) => user

}.map(user => user.id -> user.safety).toMap

candidates.map { candidate =>

val safety = idToUserSafetyMap.getOrElse(candidate.candidate.id, None)

val isValidUser = safety.isDefined &&

!safety.exists(\_.deactivated) &&

!safety.exists(\_.suspended) &&

!safety.exists(\_.isProtected) &&

!safety.flatMap(\_.offboarded).getOrElse(false)

FeatureMapBuilder()

.add(IsGizmoduckValidUserFeature, isValidUser)

.build()

}

}

}

}