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

import com.twitter.conversions.DurationOps.\_

import com.twitter.ads.entities.db.{thriftscala => ae}

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

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

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

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

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

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

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

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

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.stitch.Stitch

import com.twitter.snowflake.id.SnowflakeId

import javax.inject.Inject

import javax.inject.Singleton

@Singleton

class GizmoduckAuthorFeatureHydrator @Inject() (gizmoduck: gt.UserService.MethodPerEndpoint)

extends BulkCandidateFeatureHydrator[PipelineQuery, TweetCandidate] {

override val identifier: FeatureHydratorIdentifier =

FeatureHydratorIdentifier("GizmoduckAuthor")

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

Set(AuthorIsBlueVerifiedFeature, AuthorIsProtectedFeature, IsSupportAccountReplyFeature)

private val queryFields: Set[gt.QueryFields] =

Set(gt.QueryFields.AdvertiserAccount, gt.QueryFields.Profile, gt.QueryFields.Safety)

override def apply(

query: PipelineQuery,

candidates: Seq[CandidateWithFeatures[TweetCandidate]]

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

val authorIds = candidates.flatMap(\_.features.getOrElse(AuthorIdFeature, None))

val response = gizmoduck.get(

userIds = authorIds.distinct,

queryFields = queryFields,

context = gt.LookupContext()

)

response.map { hydratedAuthors =>

val userMetadataMap = hydratedAuthors

.collect {

case userResult if userResult.user.isDefined =>

val user = userResult.user.get

val blueVerified = user.safety.flatMap(\_.isBlueVerified).getOrElse(false)

val isProtected = user.safety.exists(\_.isProtected)

(user.id, (blueVerified, isProtected))

}.toMap.withDefaultValue((false, false))

candidates.map { candidate =>

val authorId = candidate.features.get(AuthorIdFeature).get

val (isBlueVerified, isProtected) = userMetadataMap(authorId)

// Some accounts run promotions on Twitter and send replies automatically.

// We assume that a reply that took more than one minute is not an auto-reply.

// If time difference doesn't exist, this means that one of the tweets was

// not snowflake and therefore much older, and therefore OK as an extended reply.

val timeDifference = candidate.features.getOrElse(InReplyToTweetIdFeature, None).map {

SnowflakeId.timeFromId(candidate.candidate.id) - SnowflakeId.timeFromId(\_)

}

val isAutoReply = timeDifference.exists(\_ < 1.minute)

FeatureMapBuilder()

.add(AuthorIsBlueVerifiedFeature, isBlueVerified)

.add(AuthorIsProtectedFeature, isProtected)

.add(IsSupportAccountReplyFeature, isAutoReply)

.build()

}

}

}

}