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

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

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

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.CandidateFeatureHydrator

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.tweetconvosvc.tweet\_ancestor.{thriftscala => ta}

import com.twitter.tweetconvosvc.{thriftscala => tcs}

import com.twitter.util.Future

import javax.inject.Inject

import javax.inject.Singleton

@Singleton

class AncestorFeatureHydrator @Inject() (

conversationServiceClient: tcs.ConversationService.MethodPerEndpoint)

extends CandidateFeatureHydrator[PipelineQuery, TweetCandidate] {

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

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

private val DefaultFeatureMap = FeatureMapBuilder().add(AncestorsFeature, Seq.empty).build()

override def apply(

query: PipelineQuery,

candidate: TweetCandidate,

existingFeatures: FeatureMap

): Stitch[FeatureMap] = OffloadFuturePools.offloadFuture {

if (existingFeatures.getOrElse(InReplyToTweetIdFeature, None).isDefined) {

val ancestorsRequest = tcs.GetAncestorsRequest(Seq(candidate.id))

conversationServiceClient.getAncestors(ancestorsRequest).map { getAncestorsResponse =>

val ancestors = getAncestorsResponse.ancestors.headOption

.collect {

case tcs.TweetAncestorsResult.TweetAncestors(ancestorsResult)

if ancestorsResult.nonEmpty =>

ancestorsResult.head.ancestors ++ getTruncatedRootTweet(ancestorsResult.head)

}.getOrElse(Seq.empty)

FeatureMapBuilder().add(AncestorsFeature, ancestors).build()

}

} else Future.value(DefaultFeatureMap)

}

private def getTruncatedRootTweet(

ancestors: ta.TweetAncestors,

): Option[ta.TweetAncestor] = {

ancestors.conversationRootAuthorId.collect {

case rootAuthorId

if ancestors.state == ta.ReplyState.Partial &&

ancestors.ancestors.last.tweetId != ancestors.conversationId =>

ta.TweetAncestor(ancestors.conversationId, rootAuthorId)

}

}

}