package com.twitter.follow\_recommendations.common.predicates

import com.google.inject.name.Named

import com.twitter.core\_workflows.user\_model.thriftscala.UserState

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.follow\_recommendations.common.base.Predicate

import com.twitter.follow\_recommendations.common.base.PredicateResult

import com.twitter.follow\_recommendations.common.constants.GuiceNamedConstants

import com.twitter.follow\_recommendations.common.models.CandidateUser

import com.twitter.follow\_recommendations.common.models.FilterReason

import com.twitter.follow\_recommendations.common.predicates.InactivePredicateParams.\_

import com.twitter.service.metastore.gen.thriftscala.UserRecommendabilityFeatures

import com.twitter.stitch.Stitch

import com.twitter.strato.client.Fetcher

import com.twitter.timelines.configapi.HasParams

import com.twitter.util.Duration

import com.twitter.util.Time

import javax.inject.Inject

import javax.inject.Singleton

import com.twitter.conversions.DurationOps.\_

import com.twitter.escherbird.util.stitchcache.StitchCache

import com.twitter.follow\_recommendations.common.models.HasUserState

import com.twitter.follow\_recommendations.common.predicates.InactivePredicateParams.DefaultInactivityThreshold

import com.twitter.product\_mixer.core.model.marshalling.request.HasClientContext

import java.lang.{Long => JLong}

@Singleton

case class InactivePredicate @Inject() (

statsReceiver: StatsReceiver,

@Named(GuiceNamedConstants.USER\_RECOMMENDABILITY\_FETCHER) userRecommendabilityFetcher: Fetcher[

Long,

Unit,

UserRecommendabilityFeatures

]) extends Predicate[(HasParams with HasClientContext with HasUserState, CandidateUser)] {

private val stats: StatsReceiver = statsReceiver.scope("InactivePredicate")

private val cacheStats = stats.scope("cache")

private def queryUserRecommendable(userId: Long): Stitch[Option[UserRecommendabilityFeatures]] =

userRecommendabilityFetcher.fetch(userId).map(\_.v)

private val userRecommendableCache =

StitchCache[JLong, Option[UserRecommendabilityFeatures]](

maxCacheSize = 100000,

ttl = 12.hours,

statsReceiver = cacheStats.scope("UserRecommendable"),

underlyingCall = (userId: JLong) => queryUserRecommendable(userId)

)

override def apply(

targetAndCandidate: (HasParams with HasClientContext with HasUserState, CandidateUser)

): Stitch[PredicateResult] = {

val (target, candidate) = targetAndCandidate

userRecommendableCache

.readThrough(candidate.id).map {

case recFeaturesFetchResult =>

recFeaturesFetchResult match {

case None =>

PredicateResult.Invalid(Set(FilterReason.MissingRecommendabilityData))

case Some(recFeatures) =>

if (disableInactivityPredicate(target, target.userState, recFeatures.userState)) {

PredicateResult.Valid

} else {

val defaultInactivityThreshold = target.params(DefaultInactivityThreshold).days

val hasBeenActiveRecently = recFeatures.lastStatusUpdateMs

.map(Time.now - Time.fromMilliseconds(\_)).getOrElse(

Duration.Top) < defaultInactivityThreshold

stats

.scope(defaultInactivityThreshold.toString).counter(

if (hasBeenActiveRecently)

"active"

else

"inactive"

).incr()

if (hasBeenActiveRecently && (!target

.params(UseEggFilter) || recFeatures.isNotEgg.contains(1))) {

PredicateResult.Valid

} else {

PredicateResult.Invalid(Set(FilterReason.Inactive))

}

}

}

}.rescue {

case e: Exception =>

stats.counter(e.getClass.getSimpleName).incr()

Stitch(PredicateResult.Invalid(Set(FilterReason.FailOpen)))

}

}

private[this] def disableInactivityPredicate(

target: HasParams,

consumerState: Option[UserState],

candidateState: Option[UserState]

): Boolean = {

target.params(MightBeDisabled) &&

consumerState.exists(InactivePredicate.ValidConsumerStates.contains) &&

(

(

candidateState.exists(InactivePredicate.ValidCandidateStates.contains) &&

!target.params(OnlyDisableForNewUserStateCandidates)

) ||

(

candidateState.contains(UserState.New) &&

target.params(OnlyDisableForNewUserStateCandidates)

)

)

}

}

object InactivePredicate {

val ValidConsumerStates: Set[UserState] = Set(

UserState.HeavyNonTweeter,

UserState.MediumNonTweeter,

UserState.HeavyTweeter,

UserState.MediumTweeter

)

val ValidCandidateStates: Set[UserState] =

Set(UserState.New, UserState.VeryLight, UserState.Light, UserState.NearZero)

}