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

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.finagle.util.DefaultTimer

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

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

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

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

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

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

import com.twitter.hss.api.thriftscala.SignalValue

import com.twitter.hss.api.thriftscala.UserHealthSignal.AgathaCseDouble

import com.twitter.hss.api.thriftscala.UserHealthSignal.NsfwAgathaUserScoreDouble

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

import com.twitter.stitch.Stitch

import com.twitter.strato.generated.client.hss.user\_signals.api.HealthSignalsOnUserClientColumn

import com.twitter.timelines.configapi.HasParams

import com.twitter.util.logging.Logging

import com.twitter.util.Duration

import javax.inject.Inject

import javax.inject.Singleton

/\*\*

\* Filter out candidates based on Health Signal Store (HSS) health signals

\*/

@Singleton

case class HssPredicate @Inject() (

healthSignalsOnUserClientColumn: HealthSignalsOnUserClientColumn,

statsReceiver: StatsReceiver)

extends Predicate[(HasClientContext with HasParams, CandidateUser)]

with Logging {

private val stats: StatsReceiver = statsReceiver.scope(this.getClass.getName)

override def apply(

pair: (HasClientContext with HasParams, CandidateUser)

): Stitch[PredicateResult] = {

val (request, candidate) = pair

StatsUtil.profileStitch(

getHssPredicateResult(request, candidate),

stats.scope("getHssPredicateResult")

)

}

private def getHssPredicateResult(

request: HasClientContext with HasParams,

candidate: CandidateUser

): Stitch[PredicateResult] = {

val hssCseScoreThreshold: Double = request.params(HssPredicateParams.HssCseScoreThreshold)

val hssNsfwScoreThreshold: Double = request.params(HssPredicateParams.HssNsfwScoreThreshold)

val timeout: Duration = request.params(HssPredicateParams.HssApiTimeout)

healthSignalsOnUserClientColumn.fetcher

.fetch(candidate.id, Seq(AgathaCseDouble, NsfwAgathaUserScoreDouble))

.map { fetchResult =>

fetchResult.v match {

case Some(response) =>

val agathaCseScoreDouble: Double = userHealthSignalValueToDoubleOpt(

response.signalValues.get(AgathaCseDouble)).getOrElse(0d)

val agathaNsfwScoreDouble: Double = userHealthSignalValueToDoubleOpt(

response.signalValues.get(NsfwAgathaUserScoreDouble)).getOrElse(0d)

stats.stat("agathaCseScoreDistribution").add(agathaCseScoreDouble.toFloat)

stats.stat("agathaNsfwScoreDistribution").add(agathaNsfwScoreDouble.toFloat)

/\*\*

\* Only filter out the candidate when it has both high Agatha CSE score and NSFW score, as the Agatha CSE

\* model is an old one that may not be precise or have high recall.

\*/

if (agathaCseScoreDouble >= hssCseScoreThreshold && agathaNsfwScoreDouble >= hssNsfwScoreThreshold) {

PredicateResult.Invalid(Set(FilterReason.HssSignal))

} else {

PredicateResult.Valid

}

case None =>

PredicateResult.Valid

}

}

.within(timeout)(DefaultTimer)

.rescue {

case e: Exception =>

stats.scope("rescued").counter(e.getClass.getSimpleName).incr()

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

}

}

private def userHealthSignalValueToDoubleOpt(signalValue: Option[SignalValue]): Option[Double] = {

signalValue match {

case Some(SignalValue.DoubleValue(value)) => Some(value)

case \_ => None

}

}

}