package com.twitter.frigate.pushservice.predicate

import com.twitter.conversions.DurationOps.\_

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.base.TargetUser

import com.twitter.frigate.common.candidate.CaretFeedbackHistory

import com.twitter.frigate.common.candidate.FrigateHistory

import com.twitter.frigate.common.candidate.HTLVisitHistory

import com.twitter.frigate.common.candidate.TargetABDecider

import com.twitter.frigate.common.history.History

import com.twitter.frigate.common.predicate.FrigateHistoryFatiguePredicate.TimeSeries

import com.twitter.frigate.common.predicate.ntab\_caret\_fatigue.NtabCaretClickFatiguePredicateHelper

import com.twitter.frigate.common.rec\_types.RecTypes

import com.twitter.frigate.common.util.FeatureSwitchParams

import com.twitter.frigate.pushservice.params.PushFeatureSwitchParams

import com.twitter.hermit.predicate.NamedPredicate

import com.twitter.hermit.predicate.Predicate

import com.twitter.notificationservice.thriftscala.CaretFeedbackDetails

import com.twitter.util.Duration

import com.twitter.util.Future

import com.twitter.frigate.common.predicate.{FatiguePredicate => CommonFatiguePredicate}

object TargetNtabCaretClickFatiguePredicate {

import NtabCaretClickFatiguePredicateHelper.\_

private val MagicRecsCategory = "MagicRecs"

def apply[

T <: TargetUser with TargetABDecider with CaretFeedbackHistory with FrigateHistory with HTLVisitHistory

](

filterHistory: TimeSeries => TimeSeries =

CommonFatiguePredicate.recTypesOnlyFilter(RecTypes.sharedNTabCaretFatigueTypes),

filterCaretFeedbackHistory: TargetUser with TargetABDecider with CaretFeedbackHistory => Seq[

CaretFeedbackDetails

] => Seq[CaretFeedbackDetails] =

CaretFeedbackHistoryFilter.caretFeedbackHistoryFilter(Seq(MagicRecsCategory)),

calculateFatiguePeriod: Seq[CaretFeedbackDetails] => Duration = calculateFatiguePeriodMagicRecs,

useMostRecentDislikeTime: Boolean = false,

name: String = "NtabCaretClickFatiguePredicate"

)(

implicit statsReceiver: StatsReceiver

): NamedPredicate[T] = {

val scopedStats = statsReceiver.scope(name)

val crtStats = scopedStats.scope("crt")

Predicate

.fromAsync { target: T =>

Future.join(target.history, target.caretFeedbacks).map {

case (history, Some(feedbackDetails)) => {

val feedbackDetailsDeduped = dedupFeedbackDetails(

filterCaretFeedbackHistory(target)(feedbackDetails),

scopedStats

)

val fatiguePeriod =

if (hasUserDislikeInLast30Days(feedbackDetailsDeduped) && target.params(

PushFeatureSwitchParams.EnableReducedFatigueRulesForSeeLessOften)) {

durationToFilterMRForSeeLessOftenExpt(

feedbackDetailsDeduped,

target.params(FeatureSwitchParams.NumberOfDaysToFilterMRForSeeLessOften),

target.params(FeatureSwitchParams.NumberOfDaysToReducePushCapForSeeLessOften),

scopedStats

)

} else {

calculateFatiguePeriod(feedbackDetailsDeduped)

}

val crtlist = feedbackDetailsDeduped

.flatMap { fd =>

fd.genericNotificationMetadata.map { gm =>

gm.genericType.name

}

}.distinct.sorted.mkString("-")

if (fatiguePeriod > 0.days) {

crtStats.scope(crtlist).counter("fatigued").incr()

} else {

crtStats.scope(crtlist).counter("non\_fatigued").incr()

}

val hasRecentSent =

hasRecentSend(History(filterHistory(history.history.toSeq).toMap), fatiguePeriod)

!hasRecentSent

}

case \_ => true

}

}

.withStats(scopedStats)

.withName(name)

}

}