package com.twitter.frigate.pushservice.predicate

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.predicate.FatiguePredicate.\_

import com.twitter.frigate.pushservice.model.PushTypes.PushCandidate

import com.twitter.frigate.thriftscala.CommonRecommendationType

import com.twitter.frigate.thriftscala.{NotificationDisplayLocation => DisplayLocation}

import com.twitter.hermit.predicate.NamedPredicate

import com.twitter.util.Duration

object FatiguePredicate {

/\*\*

\* Predicate that operates on a candidate, and applies custom fatigue rules for the slice of history only

\* corresponding to a given rec type.

\*

\* @param interval

\* @param maxInInterval

\* @param minInterval

\* @param recommendationType

\* @param statsReceiver

\* @return

\*/

def recTypeOnly(

interval: Duration,

maxInInterval: Int,

minInterval: Duration,

recommendationType: CommonRecommendationType,

notificationDisplayLocation: DisplayLocation = DisplayLocation.PushToMobileDevice

)(

implicit statsReceiver: StatsReceiver

): NamedPredicate[PushCandidate] = {

build(

interval = interval,

maxInInterval = maxInInterval,

minInterval = minInterval,

filterHistory = recOnlyFilter(recommendationType),

notificationDisplayLocation = notificationDisplayLocation

).flatContraMap { candidate: PushCandidate => candidate.target.history }

.withStats(statsReceiver.scope(s"predicate\_${recTypeOnlyFatigue}"))

.withName(recTypeOnlyFatigue)

}

/\*\*

\* Predicate that operates on a candidate, and applies custom fatigue rules for the slice of history only

\* corresponding to specified rec types

\*

\* @param interval

\* @param maxInInterval

\* @param minInterval

\* @param statsReceiver

\* @return

\*/

def recTypeSetOnly(

interval: Duration,

maxInInterval: Int,

minInterval: Duration,

recTypes: Set[CommonRecommendationType],

notificationDisplayLocation: DisplayLocation = DisplayLocation.PushToMobileDevice

)(

implicit statsReceiver: StatsReceiver

): NamedPredicate[PushCandidate] = {

val name = "rec\_type\_set\_fatigue"

build(

interval = interval,

maxInInterval = maxInInterval,

minInterval = minInterval,

filterHistory = recTypesOnlyFilter(recTypes),

notificationDisplayLocation = notificationDisplayLocation

).flatContraMap { candidate: PushCandidate => candidate.target.history }

.withStats(statsReceiver.scope(s"${name}\_predicate"))

.withName(name)

}

}