package com.twitter.frigate.pushservice.predicate.ntab\_caret\_fatigue

import com.twitter.finagle.stats.StatsReceiver

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

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

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

import com.twitter.hermit.predicate.NamedPredicate

import com.twitter.hermit.predicate.Predicate

import com.twitter.util.Future

object NtabCaretClickFatiguePredicate {

val name = "NtabCaretClickFatiguePredicate"

def isSpacesTypeAndTeamMember(candidate: PushCandidate): Future[Boolean] = {

candidate.target.isTeamMember.map { isTeamMember =>

val isSpacesType = RecTypes.isRecommendedSpacesType(candidate.commonRecType)

isTeamMember && isSpacesType

}

}

def apply()(implicit globalStats: StatsReceiver): NamedPredicate[PushCandidate] = {

val scopedStats = globalStats.scope(name)

val genericTypeCategories = Seq("MagicRecs")

val crts = RecTypes.sharedNTabCaretFatigueTypes

val recTypeNtabCaretClickFatiguePredicate =

RecTypeNtabCaretClickFatiguePredicate.apply(

genericTypeCategories,

crts,

NtabCaretClickFatiguePredicateHelper.calculateFatiguePeriodMagicRecs,

useMostRecentDislikeTime = false

)

Predicate

.fromAsync { candidate: PushCandidate =>

isSpacesTypeAndTeamMember(candidate).flatMap { isSpacesTypeAndTeamMember =>

if (RecTypes.sharedNTabCaretFatigueTypes(

candidate.commonRecType) && !isSpacesTypeAndTeamMember) {

recTypeNtabCaretClickFatiguePredicate

.apply(Seq(candidate)).map(\_.headOption.getOrElse(false))

} else {

Future.True

}

}

}

.withStats(scopedStats)

.withName(name)

}

}