package com.twitter.frigate.pushservice.predicate

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.base.SocialContextActions

import com.twitter.frigate.common.base.SocialContextUserDetails

import com.twitter.frigate.common.base.TargetInfo

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

import com.twitter.frigate.common.base.TweetAuthor

import com.twitter.frigate.common.base.TweetAuthorDetails

import com.twitter.frigate.common.base.TweetCandidate

import com.twitter.frigate.common.base.TweetDetails

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

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

import com.twitter.frigate.common.candidate.TweetImpressionHistory

import com.twitter.frigate.common.predicate.socialcontext.{Predicates => SocialContextPredicates, \_}

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

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

import com.twitter.frigate.pushservice.predicate.ntab\_caret\_fatigue.NtabCaretClickContFnFatiguePredicate

import com.twitter.hermit.predicate.NamedPredicate

class PreRankingPredicatesBuilder(

)(

implicit statsReceiver: StatsReceiver) {

private val SocialProofPredicates = List[NamedPredicate[PushCandidate]](

SocialContextPredicates

.authorInSocialContext()

.applyOnlyToTweetAuthorWithSocialContextActions

.withName("author\_social\_context"),

SocialContextPredicates

.selfInSocialContext[TargetUser, SocialContextActions with TargetInfo[TargetUser]]()

.applyOnlyToSocialContextActionsWithTargetUser

.withName("self\_social\_context"),

SocialContextPredicates

.duplicateSocialContext[SocialContextActions]()

.applyOnlyToSocialContextActions

.withName("duplicate\_social\_context"),

SocialContextPredicates

.socialContextProtected[SocialContextUserDetails]()

.applyOnlyToSocialContextUserDetails

.withName("social\_context\_protected"),

SocialContextPredicates

.socialContextUnsuitable[SocialContextUserDetails]()

.applyOnlyToSocialContextUserDetails

.withName("social\_context\_unsuitable"),

SocialContextPredicates

.socialContextBlink[SocialContextUserDetails]()

.applyOnlyToSocialContextUserDetails

.withName("social\_context\_blink")

)

private val CommonPredicates = List[NamedPredicate[PushCandidate]](

PredicatesForCandidate.candidateEnabledForEmailPredicate(),

PredicatesForCandidate.openAppExperimentUserCandidateAllowList(statsReceiver)

)

private val TweetPredicates = List[NamedPredicate[PushCandidate]](

PredicatesForCandidate.tweetCandidateWithLessThan2SocialContextsIsAReply.applyOnlyToTweetCandidatesWithSocialContextActions

.withName("tweet\_candidate\_with\_less\_than\_2\_social\_contexts\_is\_not\_a\_reply"),

PredicatesForCandidate.filterOONCandidatePredicate(),

PredicatesForCandidate.oldTweetRecsPredicate.applyOnlyToTweetCandidateWithTargetAndABDeciderAndMaxTweetAge

.withName("old\_tweet"),

DuplicatePushTweetPredicate

.apply[

TargetUser with FrigateHistory,

TweetCandidate with TargetInfo[TargetUser with FrigateHistory]

]

.applyOnlyToTweetCandidateWithTargetAndFrigateHistory

.withName("duplicate\_push\_tweet"),

DuplicateEmailTweetPredicate

.apply[

TargetUser with FrigateHistory,

TweetCandidate with TargetInfo[TargetUser with FrigateHistory]

]

.applyOnlyToTweetCandidateWithTargetAndFrigateHistory

.withName("duplicate\_email\_tweet"),

TweetAuthorPredicates

.recTweetAuthorUnsuitable[TweetCandidate with TweetAuthorDetails]

.applyOnlyToTweetCandidateWithTweetAuthorDetails

.withName("tweet\_author\_unsuitable"),

TweetObjectExistsPredicate[

TweetCandidate with TweetDetails

].applyOnlyToTweetCandidatesWithTweetDetails

.withName("tweet\_object\_exists"),

TweetImpressionPredicate[

TargetUser with TweetImpressionHistory,

TweetCandidate with TargetInfo[TargetUser with TweetImpressionHistory]

].applyOnlyToTweetCandidateWithTargetAndTweetImpressionHistory

.withStats(statsReceiver.scope("tweet\_impression"))

.withName("tweet\_impression"),

SelfTweetPredicate[

TargetUser,

TweetAuthor with TargetInfo[TargetUser]]().applyOnlyToTweetAuthorWithTargetInfo

.withName("self\_author"),

PredicatesForCandidate.tweetIsNotAreply.applyOnlyToTweetCandidateWithoutSocialContextWithTweetDetails

.withName("tweet\_candidate\_not\_a\_reply"),

PredicatesForCandidate.f1CandidateIsNotAReply.applyOnlyToF1CandidateWithTargetAndABDecider

.withName("f1\_candidate\_is\_not\_a\_reply"),

PredicatesForCandidate.outOfNetworkTweetCandidateIsNotAReply.applyOnlyToOutOfNetworkTweetCandidateWithTargetAndABDecider

.withName("out\_of\_network\_tweet\_candidate\_is\_not\_a\_reply"),

PredicatesForCandidate.outOfNetworkTweetCandidateEnabledCrTag.applyOnlyToOutOfNetworkTweetCandidateWithTargetAndABDecider

.withName("out\_of\_network\_tweet\_candidate\_enabled\_crtag"),

PredicatesForCandidate.outOfNetworkTweetCandidateEnabledCrtGroup.applyOnlyToOutOfNetworkTweetCandidateWithTargetAndABDecider

.withName("out\_of\_network\_tweet\_candidate\_enabled\_crt\_group"),

OutOfNetworkCandidatesQualityPredicates

.oonTweetLengthBasedPrerankingPredicate(characterBased = true)

.applyOnlyToOutOfNetworkTweetCandidateWithTargetAndABDecider

.withName("oon\_tweet\_char\_length\_too\_short"),

OutOfNetworkCandidatesQualityPredicates

.oonTweetLengthBasedPrerankingPredicate(characterBased = false)

.applyOnlyToOutOfNetworkTweetCandidateWithTargetAndABDecider

.withName("oon\_tweet\_word\_length\_too\_short"),

PredicatesForCandidate

.protectedTweetF1ExemptPredicate[

TargetUser with TargetABDecider,

TweetCandidate with TweetAuthorDetails with TargetInfo[

TargetUser with TargetABDecider

]

]

.applyOnlyToTweetCandidateWithAuthorDetailsWithTargetABDecider

.withName("f1\_exempt\_tweet\_author\_protected"),

)

private val SgsPreRankingPredicates = List[NamedPredicate[PushCandidate]](

SGSPredicatesForCandidate.authorBeingFollowed.applyOnlyToAuthorBeingFollowPredicates

.withName("author\_not\_being\_followed"),

SGSPredicatesForCandidate.authorNotBeingDeviceFollowed.applyOnlyToBasicTweetPredicates

.withName("author\_being\_device\_followed"),

SGSPredicatesForCandidate.recommendedTweetAuthorAcceptableToTargetUser.applyOnlyToBasicTweetPredicates

.withName("recommended\_tweet\_author\_not\_acceptable\_to\_target\_user"),

SGSPredicatesForCandidate.disableInNetworkTweetPredicate.applyOnlyToBasicTweetPredicates

.withName("enable\_in\_network\_tweet"),

SGSPredicatesForCandidate.disableOutNetworkTweetPredicate.applyOnlyToBasicTweetPredicates

.withName("enable\_out\_network\_tweet")

)

private val SeeLessOftenPredicates = List[NamedPredicate[PushCandidate]](

NtabCaretClickContFnFatiguePredicate

.ntabCaretClickContFnFatiguePredicates(

)

.withName("seelessoften\_cont\_fn\_fatigue")

)

final def build(): List[NamedPredicate[PushCandidate]] = {

TweetPredicates ++

CommonPredicates ++

SocialProofPredicates ++

SgsPreRankingPredicates ++

SeeLessOftenPredicates

}

}

object PreRankingPredicates {

def apply(

statsReceiver: StatsReceiver

): List[NamedPredicate[PushCandidate]] =

new PreRankingPredicatesBuilder()(statsReceiver).build()

}