package com.twitter.frigate.pushservice.refresh\_handler

import com.twitter.finagle.stats.Counter

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.base.\_

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

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

import com.twitter.frigate.pushservice.predicate.PreRankingPredicates

import com.twitter.hermit.predicate.NamedPredicate

import com.twitter.hermit.predicate.SequentialPredicate

import com.twitter.util.\_

class RFPHPrerankFilter(

)(

globalStats: StatsReceiver) {

def filter(

target: Target,

hydratedCandidates: Seq[CandidateDetails[PushCandidate]]

): Future[

(Seq[CandidateDetails[PushCandidate]], Seq[CandidateResult[PushCandidate, Result]])

] = {

lazy val filterStats: StatsReceiver = globalStats.scope("RefreshForPushHandler/filter")

lazy val okFilterCounter: Counter = filterStats.counter("ok")

lazy val invalidFilterCounter: Counter = filterStats.counter("invalid")

lazy val invalidFilterStat: StatsReceiver = filterStats.scope("invalid")

lazy val invalidFilterReasonStat: StatsReceiver = invalidFilterStat.scope("reason")

val allCandidatesFilteredPreRank = filterStats.counter("all\_candidates\_filtered")

lazy val preRankingPredicates = PreRankingPredicates(

filterStats.scope("predicates")

)

lazy val preRankingPredicateChain =

new SequentialPredicate[PushCandidate](preRankingPredicates)

val predicateChain = if (target.pushContext.exists(\_.predicatesToEnable.exists(\_.nonEmpty))) {

val predicatesToEnable = target.pushContext.flatMap(\_.predicatesToEnable).getOrElse(Nil)

new SequentialPredicate[PushCandidate](preRankingPredicates.filter { pred =>

predicatesToEnable.contains(pred.name)

})

} else preRankingPredicateChain

predicateChain

.track(hydratedCandidates.map(\_.candidate))

.map { results =>

val resultForPreRankFiltering = results

.zip(hydratedCandidates)

.foldLeft(

(

Seq.empty[CandidateDetails[PushCandidate]],

Seq.empty[CandidateResult[PushCandidate, Result]]

)

) {

case ((goodCandidates, filteredCandidates), (result, candidateDetails)) =>

result match {

case None =>

okFilterCounter.incr()

(goodCandidates :+ candidateDetails, filteredCandidates)

case Some(pred: NamedPredicate[\_]) =>

invalidFilterCounter.incr()

invalidFilterReasonStat.counter(pred.name).incr()

invalidFilterReasonStat

.scope(candidateDetails.candidate.commonRecType.toString).counter(

pred.name).incr()

val r = Invalid(Some(pred.name))

(

goodCandidates,

filteredCandidates :+ CandidateResult[PushCandidate, Result](

candidateDetails.candidate,

candidateDetails.source,

r

)

)

case Some(\_) =>

invalidFilterCounter.incr()

invalidFilterReasonStat.counter("unknown").incr()

invalidFilterReasonStat

.scope(candidateDetails.candidate.commonRecType.toString).counter(

"unknown").incr()

val r = Invalid(Some("Filtered by un-named predicate"))

(

goodCandidates,

filteredCandidates :+ CandidateResult[PushCandidate, Result](

candidateDetails.candidate,

candidateDetails.source,

r

)

)

}

}

resultForPreRankFiltering match {

case (validCandidates, \_) if validCandidates.isEmpty && hydratedCandidates.nonEmpty =>

allCandidatesFilteredPreRank.incr()

case \_ => ()

}

resultForPreRankFiltering

}

}

}