package com.twitter.frigate.pushservice.util

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.base.CandidateDetails

import com.twitter.frigate.common.base.CandidateResult

import com.twitter.frigate.common.base.Invalid

import com.twitter.frigate.common.base.OK

import com.twitter.frigate.common.base.Result

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

import com.twitter.frigate.pushservice.refresh\_handler.ResultWithDebugInfo

import com.twitter.frigate.pushservice.take.candidate\_validator.SendHandlerPostCandidateValidator

import com.twitter.frigate.pushservice.take.candidate\_validator.SendHandlerPreCandidateValidator

import com.twitter.frigate.pushservice.thriftscala.PushStatus

import com.twitter.hermit.predicate.NamedPredicate

import com.twitter.util.Future

class SendHandlerPredicateUtil()(globalStats: StatsReceiver) {

implicit val statsReceiver: StatsReceiver =

globalStats.scope("SendHandler")

private val validateStats: StatsReceiver = statsReceiver.scope("validate")

private def updateFilteredStatusExptStats(candidate: PushCandidate, predName: String): Unit = {

val recTypeStat = globalStats.scope(

candidate.commonRecType.toString

)

recTypeStat.counter(PushStatus.Filtered.toString).incr()

recTypeStat

.scope(PushStatus.Filtered.toString)

.counter(predName)

.incr()

}

/\*\*

\* Parsing the candidateValidtor result into desired format for preValidation before ml filtering

\* @param hydratedCandidates

\* @param candidateValidator

\* @return

\*/

def preValidationForCandidate(

hydratedCandidates: Seq[CandidateDetails[PushCandidate]],

candidateValidator: SendHandlerPreCandidateValidator

): Future[

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

] = {

val predResultFuture =

Future.collect(

hydratedCandidates.map(hydratedCandidate =>

candidateValidator.validateCandidate(hydratedCandidate.candidate))

)

predResultFuture.map { results =>

results

.zip(hydratedCandidates)

.foldLeft(

(

Seq.empty[CandidateDetails[PushCandidate]],

Seq.empty[CandidateResult[PushCandidate, Result]]

)

) {

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

result match {

case None =>

(goodCandidates :+ candidateDetails, filteredCandidates)

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

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

(

goodCandidates,

filteredCandidates :+ CandidateResult[PushCandidate, Result](

candidateDetails.candidate,

candidateDetails.source,

r

)

)

case Some(\_) =>

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

(

goodCandidates,

filteredCandidates :+ CandidateResult[PushCandidate, Result](

candidateDetails.candidate,

candidateDetails.source,

r

)

)

}

}

}

}

/\*\*

\* Parsing the candidateValidtor result into desired format for postValidation including and after ml filtering

\* @param candidate

\* @param candidateValidator

\* @return

\*/

def postValidationForCandidate(

candidate: PushCandidate,

candidateValidator: SendHandlerPostCandidateValidator

): Future[ResultWithDebugInfo] = {

val predResultFuture =

candidateValidator.validateCandidate(candidate)

predResultFuture.map {

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

validateStats.counter("filtered\_by\_named\_general\_predicate").incr()

updateFilteredStatusExptStats(candidate, pred.name)

ResultWithDebugInfo(

Invalid(Some(pred.name)),

Nil

)

case Some(\_) =>

validateStats.counter("filtered\_by\_unnamed\_general\_predicate").incr()

updateFilteredStatusExptStats(candidate, predName = "unk")

ResultWithDebugInfo(

Invalid(Some("unnamed\_candidate\_predicate")),

Nil

)

case \_ =>

validateStats.counter("accepted\_push\_ok").incr()

ResultWithDebugInfo(

OK,

Nil

)

}

}

}