package com.twitter.simclustersann.controllers

import com.twitter.conversions.DurationOps.\_

import com.twitter.finatra.thrift.Controller

import com.twitter.simclustersann.thriftscala.SimClustersANNService.GetTweetCandidates

import com.twitter.simclustersann.thriftscala.SimClustersANNService

import com.twitter.simclustersann.thriftscala.Query

import com.twitter.simclustersann.thriftscala.SimClustersANNTweetCandidate

import com.twitter.scrooge.Request

import com.twitter.scrooge.Response

import javax.inject.Inject

import com.twitter.finagle.Service

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.inject.annotations.Flag

import com.twitter.simclustersann.candidate\_source.{

SimClustersANNCandidateSource => SANNSimClustersANNCandidateSource

}

import com.twitter.simclustersann.common.FlagNames

import com.twitter.simclustersann.filters.GetTweetCandidatesResponseStatsFilter

import com.twitter.simclustersann.filters.SimClustersAnnVariantFilter

import com.twitter.util.Future

import com.twitter.util.JavaTimer

import com.twitter.util.Timer

class SimClustersANNController @Inject() (

@Flag(FlagNames.ServiceTimeout) serviceTimeout: Int,

variantFilter: SimClustersAnnVariantFilter,

getTweetCandidatesResponseStatsFilter: GetTweetCandidatesResponseStatsFilter,

sannCandidateSource: SANNSimClustersANNCandidateSource,

globalStats: StatsReceiver)

extends Controller(SimClustersANNService) {

import SimClustersANNController.\_

private val stats: StatsReceiver = globalStats.scope(this.getClass.getCanonicalName)

private val timer: Timer = new JavaTimer(true)

val filteredService: Service[Request[GetTweetCandidates.Args], Response[

Seq[SimClustersANNTweetCandidate]

]] = {

variantFilter

.andThen(getTweetCandidatesResponseStatsFilter)

.andThen(Service.mk(handler))

}

handle(GetTweetCandidates).withService(filteredService)

private def handler(

request: Request[GetTweetCandidates.Args]

): Future[Response[Seq[SimClustersANNTweetCandidate]]] = {

val query: Query = request.args.query

val simClustersANNCandidateSourceQuery = SANNSimClustersANNCandidateSource.Query(

sourceEmbeddingId = query.sourceEmbeddingId,

config = query.config

)

val result = sannCandidateSource

.get(simClustersANNCandidateSourceQuery).map {

case Some(tweetCandidatesSeq) =>

Response(tweetCandidatesSeq.map { tweetCandidate =>

SimClustersANNTweetCandidate(

tweetId = tweetCandidate.tweetId,

score = tweetCandidate.score

)

})

case None =>

DefaultResponse

}

result.raiseWithin(serviceTimeout.milliseconds)(timer).rescue {

case e: Throwable =>

stats.scope("failures").counter(e.getClass.getCanonicalName).incr()

Future.value(DefaultResponse)

}

}

}

object SimClustersANNController {

val DefaultResponse: Response[Seq[SimClustersANNTweetCandidate]] = Response(Seq.empty)

}