package com.twitter.cr\_mixer.similarity\_engine

import com.twitter.cr\_mixer.config.TimeoutConfig

import com.twitter.cr\_mixer.similarity\_engine.EarlybirdModelBasedSimilarityEngine.EarlybirdModelBasedSearchQuery

import com.twitter.cr\_mixer.similarity\_engine.EarlybirdSimilarityEngineBase.\_

import com.twitter.cr\_mixer.util.EarlybirdSearchUtil.EarlybirdClientId

import com.twitter.cr\_mixer.util.EarlybirdSearchUtil.FacetsToFetch

import com.twitter.cr\_mixer.util.EarlybirdSearchUtil.MetadataOptions

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.finagle.tracing.Trace

import com.twitter.search.common.ranking.thriftscala.ThriftRankingParams

import com.twitter.search.common.ranking.thriftscala.ThriftScoringFunctionType

import com.twitter.search.earlybird.thriftscala.EarlybirdRequest

import com.twitter.search.earlybird.thriftscala.EarlybirdService

import com.twitter.search.earlybird.thriftscala.ThriftSearchQuery

import com.twitter.search.earlybird.thriftscala.ThriftSearchRankingMode

import com.twitter.search.earlybird.thriftscala.ThriftSearchRelevanceOptions

import com.twitter.simclusters\_v2.common.UserId

import javax.inject.Inject

import javax.inject.Singleton

@Singleton

case class EarlybirdModelBasedSimilarityEngine @Inject() (

earlybirdSearchClient: EarlybirdService.MethodPerEndpoint,

timeoutConfig: TimeoutConfig,

stats: StatsReceiver)

extends EarlybirdSimilarityEngineBase[EarlybirdModelBasedSearchQuery] {

import EarlybirdModelBasedSimilarityEngine.\_

override val statsReceiver: StatsReceiver = stats.scope(this.getClass.getSimpleName)

override def getEarlybirdRequest(

query: EarlybirdModelBasedSearchQuery

): Option[EarlybirdRequest] =

if (query.seedUserIds.nonEmpty)

Some(

EarlybirdRequest(

searchQuery = getThriftSearchQuery(query),

clientId = Some(EarlybirdClientId),

timeoutMs = timeoutConfig.earlybirdServerTimeout.inMilliseconds.intValue(),

clientRequestID = Some(s"${Trace.id.traceId}"),

))

else None

}

object EarlybirdModelBasedSimilarityEngine {

case class EarlybirdModelBasedSearchQuery(

seedUserIds: Seq[UserId],

maxNumTweets: Int,

oldestTweetTimestampInSec: Option[UserId],

frsUserToScoresForScoreAdjustment: Option[Map[UserId, Double]])

extends EarlybirdSearchQuery

/\*\*

\* Used by Push Service

\*/

val RealGraphScoringModel = "frigate\_unified\_engagement\_rg"

val MaxHitsToProcess = 1000

val MaxConsecutiveSameUser = 1

private def getModelBasedRankingParams(

authorSpecificScoreAdjustments: Map[Long, Double]

): ThriftRankingParams = ThriftRankingParams(

`type` = Some(ThriftScoringFunctionType.ModelBased),

selectedModels = Some(Map(RealGraphScoringModel -> 1.0)),

applyBoosts = false,

authorSpecificScoreAdjustments = Some(authorSpecificScoreAdjustments)

)

private def getRelevanceOptions(

authorSpecificScoreAdjustments: Map[Long, Double],

): ThriftSearchRelevanceOptions = {

ThriftSearchRelevanceOptions(

maxConsecutiveSameUser = Some(MaxConsecutiveSameUser),

rankingParams = Some(getModelBasedRankingParams(authorSpecificScoreAdjustments)),

maxHitsToProcess = Some(MaxHitsToProcess),

orderByRelevance = true

)

}

private def getThriftSearchQuery(query: EarlybirdModelBasedSearchQuery): ThriftSearchQuery =

ThriftSearchQuery(

serializedQuery = Some(f"(\* [since\_time ${query.oldestTweetTimestampInSec.getOrElse(0)}])"),

fromUserIDFilter64 = Some(query.seedUserIds),

numResults = query.maxNumTweets,

maxHitsToProcess = MaxHitsToProcess,

rankingMode = ThriftSearchRankingMode.Relevance,

relevanceOptions =

Some(getRelevanceOptions(query.frsUserToScoresForScoreAdjustment.getOrElse(Map.empty))),

facetFieldNames = Some(FacetsToFetch),

resultMetadataOptions = Some(MetadataOptions),

searcherId = None

)

}