package com.twitter.cr\_mixer.filter

import com.twitter.cr\_mixer.model.CandidateGeneratorQuery

import com.twitter.cr\_mixer.model.InitialCandidate

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

import com.twitter.simclusters\_v2.thriftscala.InternalId

import com.twitter.util.Future

import javax.inject.Singleton

@Singleton

case class ImpressedTweetlistFilter() extends FilterBase {

import ImpressedTweetlistFilter.\_

override val name: String = this.getClass.getCanonicalName

override type ConfigType = FilterConfig

/\*

Filtering removes some candidates based on configurable criteria.

\*/

override def filter(

candidates: Seq[Seq[InitialCandidate]],

config: FilterConfig

): Future[Seq[Seq[InitialCandidate]]] = {

// Remove candidates which match a source tweet, or which are passed in impressedTweetList

val sourceTweetsMatch = candidates

.flatMap {

/\*\*\*

\* Within a Seq[Seq[InitialCandidate]], all candidates within a inner Seq

\* are guaranteed to have the same sourceInfo. Hence, we can pick .headOption

\* to represent the whole list when filtering by the internalId of the sourceInfoOpt.

\* But of course the similarityEngineInfo could be different.

\*/

\_.headOption.flatMap { candidate =>

candidate.candidateGenerationInfo.sourceInfoOpt.map(\_.internalId)

}

}.collect {

case InternalId.TweetId(id) => id

}

val impressedTweetList: Set[TweetId] =

config.impressedTweetList ++ sourceTweetsMatch

val filteredCandidateMap: Seq[Seq[InitialCandidate]] =

candidates.map {

\_.filterNot { candidate =>

impressedTweetList.contains(candidate.tweetId)

}

}

Future.value(filteredCandidateMap)

}

override def requestToConfig[CGQueryType <: CandidateGeneratorQuery](

request: CGQueryType

): FilterConfig = {

FilterConfig(request.impressedTweetList)

}

}

object ImpressedTweetlistFilter {

case class FilterConfig(impressedTweetList: Set[TweetId])

}