package com.twitter.cr\_mixer.blender

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

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

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

import com.twitter.cr\_mixer.util.InterleaveUtil

import com.twitter.finagle.stats.StatsReceiver

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

import com.twitter.util.Future

import javax.inject.Inject

import javax.inject.Singleton

import scala.collection.mutable

@Singleton

case class AdsBlender @Inject() (globalStats: StatsReceiver) {

private val name: String = this.getClass.getCanonicalName

private val stats: StatsReceiver = globalStats.scope(name)

/\*\*

\* Interleaves candidates by iteratively choosing InterestedIn candidates and TWISTLY candidates

\* in turn. InterestedIn candidates have no source signal, whereas TWISTLY candidates do. TWISTLY

\* candidates themselves are interleaved by source before equal blending with InterestedIn

\* candidates.

\*/

def blend(

inputCandidates: Seq[Seq[InitialAdsCandidate]],

): Future[Seq[BlendedAdsCandidate]] = {

// Filter out empty candidate sequence

val candidates = inputCandidates.filter(\_.nonEmpty)

val (interestedInCandidates, twistlyCandidates) =

candidates.partition(\_.head.candidateGenerationInfo.sourceInfoOpt.isEmpty)

// First interleave twistly candidates

val interleavedTwistlyCandidates = InterleaveUtil.interleave(twistlyCandidates)

val twistlyAndInterestedInCandidates =

Seq(interestedInCandidates.flatten, interleavedTwistlyCandidates)

// then interleave twistly candidates with interested in to make them even

val interleavedCandidates = InterleaveUtil.interleave(twistlyAndInterestedInCandidates)

stats.stat("candidates").add(interleavedCandidates.size)

val blendedCandidates = buildBlendedAdsCandidate(inputCandidates, interleavedCandidates)

Future.value(blendedCandidates)

}

private def buildBlendedAdsCandidate(

inputCandidates: Seq[Seq[InitialAdsCandidate]],

interleavedCandidates: Seq[InitialAdsCandidate]

): Seq[BlendedAdsCandidate] = {

val cgInfoLookupMap = buildCandidateToCGInfosMap(inputCandidates)

interleavedCandidates.map { interleavedCandidate =>

interleavedCandidate.toBlendedAdsCandidate(cgInfoLookupMap(interleavedCandidate.tweetId))

}

}

private def buildCandidateToCGInfosMap(

candidateSeq: Seq[Seq[InitialAdsCandidate]],

): Map[TweetId, Seq[CandidateGenerationInfo]] = {

val tweetIdMap = mutable.HashMap[TweetId, Seq[CandidateGenerationInfo]]()

candidateSeq.foreach { candidates =>

candidates.foreach { candidate =>

val candidateGenerationInfoSeq = {

tweetIdMap.getOrElse(candidate.tweetId, Seq.empty)

}

val candidateGenerationInfo = candidate.candidateGenerationInfo

tweetIdMap.put(

candidate.tweetId,

candidateGenerationInfoSeq ++ Seq(candidateGenerationInfo))

}

}

tweetIdMap.toMap

}

}