package com.twitter.cr\_mixer.blender

import com.twitter.core\_workflows.user\_model.thriftscala.UserState

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

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

import com.twitter.cr\_mixer.param.BlenderParams

import com.twitter.cr\_mixer.param.BlenderParams.BlendingAlgorithmEnum

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.timelines.configapi.Params

import com.twitter.util.Future

import com.twitter.util.Time

import javax.inject.Inject

import javax.inject.Singleton

@Singleton

case class SwitchBlender @Inject() (

defaultBlender: InterleaveBlender,

sourceTypeBackFillBlender: SourceTypeBackFillBlender,

adsBlender: AdsBlender,

contentSignalBlender: ContentSignalBlender,

globalStats: StatsReceiver) {

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

def blend(

params: Params,

userState: UserState,

inputCandidates: Seq[Seq[InitialCandidate]],

): Future[Seq[BlendedCandidate]] = {

// Take out empty seq

val nonEmptyCandidates = inputCandidates.collect {

case candidates if candidates.nonEmpty =>

candidates

}

stats.stat("num\_of\_sequences").add(inputCandidates.size)

// Sort the seqs in an order

val innerSignalSorting = params(BlenderParams.SignalTypeSortingAlgorithmParam) match {

case BlenderParams.ContentBasedSortingAlgorithmEnum.SourceSignalRecency =>

SwitchBlender.TimestampOrder

case BlenderParams.ContentBasedSortingAlgorithmEnum.RandomSorting => SwitchBlender.RandomOrder

case \_ => SwitchBlender.TimestampOrder

}

val candidatesToBlend = nonEmptyCandidates.sortBy(\_.head)(innerSignalSorting)

// Blend based on specified blender rules

params(BlenderParams.BlendingAlgorithmParam) match {

case BlendingAlgorithmEnum.RoundRobin =>

defaultBlender.blend(candidatesToBlend)

case BlendingAlgorithmEnum.SourceTypeBackFill =>

sourceTypeBackFillBlender.blend(params, candidatesToBlend)

case BlendingAlgorithmEnum.SourceSignalSorting =>

contentSignalBlender.blend(params, candidatesToBlend)

case \_ => defaultBlender.blend(candidatesToBlend)

}

}

}

object SwitchBlender {

/\*\*

\* Prefers candidates generated from sources with the latest timestamps.

\* The newer the source signal, the higher a candidate ranks.

\* This ordering biases against consumer-based candidates because their timestamp defaults to 0

\*

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

\* are guaranteed to have the same sourceInfo because they are grouped by (sourceInfo, SE model).

\* Hence, we can pick .headOption to represent the whole list when filtering by the internalId of the sourceInfoOpt.

\* But of course the similarityEngine score in a CGInfo could be different.

\*/

val TimestampOrder: Ordering[InitialCandidate] =

math.Ordering

.by[InitialCandidate, Time](

\_.candidateGenerationInfo.sourceInfoOpt

.flatMap(\_.sourceEventTime)

.getOrElse(Time.fromMilliseconds(0L)))

.reverse

private val RandomOrder: Ordering[InitialCandidate] =

Ordering.by[InitialCandidate, Double](\_ => scala.util.Random.nextDouble())

}