package com.twitter.home\_mixer.functional\_component.selector

import com.twitter.home\_mixer.functional\_component.selector.DebunchCandidates.TrailingTweetsMinSize

import com.twitter.home\_mixer.functional\_component.selector.DebunchCandidates.TrailingTweetsPortionToKeep

import com.twitter.home\_mixer.model.HomeFeatures.GetNewerFeature

import com.twitter.product\_mixer.core.functional\_component.common.CandidateScope

import com.twitter.product\_mixer.core.functional\_component.common.CandidateScope.PartitionedCandidates

import com.twitter.product\_mixer.core.functional\_component.selector.Selector

import com.twitter.product\_mixer.core.functional\_component.selector.SelectorResult

import com.twitter.product\_mixer.core.model.common.presentation.CandidateWithDetails

import com.twitter.product\_mixer.core.pipeline.PipelineQuery

trait MustDebunch {

def apply(candidate: CandidateWithDetails): Boolean

}

object DebunchCandidates {

val TrailingTweetsMinSize = 5

val TrailingTweetsPortionToKeep = 0.1

}

/\*\*

\* This selector rearranges the candidates to only allow bunches of size [[maxBunchSize]], where a

\* bunch is a consecutive sequence of candidates that meet [[mustDebunch]].

\*/

case class DebunchCandidates(

override val pipelineScope: CandidateScope,

mustDebunch: MustDebunch,

maxBunchSize: Int)

extends Selector[PipelineQuery] {

override def apply(

query: PipelineQuery,

remainingCandidates: Seq[CandidateWithDetails],

result: Seq[CandidateWithDetails]

): SelectorResult = {

val PartitionedCandidates(selectedCandidates, otherCandidates) =

pipelineScope.partition(remainingCandidates)

val mutableCandidates = collection.mutable.ListBuffer(selectedCandidates: \_\*)

var candidatePointer = 0

var nonDebunchPointer = 0

var bunchSize = 0

var finalNonDebunch = -1

while (candidatePointer < mutableCandidates.size) {

if (mustDebunch(mutableCandidates(candidatePointer))) bunchSize += 1

else {

bunchSize = 0

finalNonDebunch = candidatePointer

}

if (bunchSize > maxBunchSize) {

nonDebunchPointer = Math.max(candidatePointer, nonDebunchPointer)

while (nonDebunchPointer < mutableCandidates.size &&

mustDebunch(mutableCandidates(nonDebunchPointer))) {

nonDebunchPointer += 1

}

if (nonDebunchPointer == mutableCandidates.size)

candidatePointer = mutableCandidates.size

else {

val nextNonDebunch = mutableCandidates(nonDebunchPointer)

mutableCandidates.remove(nonDebunchPointer)

mutableCandidates.insert(candidatePointer, nextNonDebunch)

bunchSize = 0

finalNonDebunch = candidatePointer

}

}

candidatePointer += 1

}

val debunchedCandidates = if (query.features.exists(\_.getOrElse(GetNewerFeature, false))) {

val trailingTweetsSize = mutableCandidates.size - finalNonDebunch - 1

val keepCandidates = finalNonDebunch + 1 +

Math.max(TrailingTweetsMinSize, TrailingTweetsPortionToKeep \* trailingTweetsSize).toInt

mutableCandidates.toList.take(keepCandidates)

} else mutableCandidates.toList

val updatedCandidates = otherCandidates ++ debunchedCandidates

SelectorResult(remainingCandidates = updatedCandidates, result = result)

}

}