package com.twitter.follow\_recommendations.utils

import com.twitter.follow\_recommendations.common.base.RecommendationFlow

import com.twitter.follow\_recommendations.common.base.SideEffectsUtil

import com.twitter.follow\_recommendations.common.models.CandidateUser

import com.twitter.product\_mixer.core.functional\_component.candidate\_source.CandidateSource

import com.twitter.product\_mixer.core.model.common.identifier.CandidateSourceIdentifier

import com.twitter.product\_mixer.core.model.marshalling.request.HasClientContext

import com.twitter.snowflake.id.SnowflakeId

import com.twitter.stitch.Stitch

trait RecommendationFlowBaseSideEffectsUtil[Target <: HasClientContext, Candidate <: CandidateUser]

extends SideEffectsUtil[Target, Candidate] {

recommendationFlow: RecommendationFlow[Target, Candidate] =>

override def applySideEffects(

target: Target,

candidateSources: Seq[CandidateSource[Target, Candidate]],

candidatesFromCandidateSources: Seq[Candidate],

mergedCandidates: Seq[Candidate],

filteredCandidates: Seq[Candidate],

rankedCandidates: Seq[Candidate],

transformedCandidates: Seq[Candidate],

truncatedCandidates: Seq[Candidate],

results: Seq[Candidate]

): Stitch[Unit] = {

Stitch.async(

Stitch.collect(

Seq(

applySideEffectsCandidateSourceCandidates(

target,

candidateSources,

candidatesFromCandidateSources),

applySideEffectsMergedCandidates(target, mergedCandidates),

applySideEffectsFilteredCandidates(target, filteredCandidates),

applySideEffectsRankedCandidates(target, rankedCandidates),

applySideEffectsTransformedCandidates(target, transformedCandidates),

applySideEffectsTruncatedCandidates(target, truncatedCandidates),

applySideEffectsResults(target, results)

)

))

}

/\*

In subclasses, override functions below to apply custom side effects at each step in pipeline.

Call super.applySideEffectsXYZ to scribe basic scribes implemented in this parent class

\*/

def applySideEffectsCandidateSourceCandidates(

target: Target,

candidateSources: Seq[CandidateSource[Target, Candidate]],

candidatesFromCandidateSources: Seq[Candidate]

): Stitch[Unit] = {

val candidatesGroupedByCandidateSources =

candidatesFromCandidateSources.groupBy(

\_.getPrimaryCandidateSource.getOrElse(CandidateSourceIdentifier("NoCandidateSource")))

target.getOptionalUserId match {

case Some(userId) =>

val userAgeOpt = SnowflakeId.timeFromIdOpt(userId).map(\_.untilNow.inDays)

userAgeOpt match {

case Some(userAge) if userAge <= 30 =>

candidateSources.map { candidateSource =>

{

val candidateSourceStats = statsReceiver.scope(candidateSource.identifier.name)

val isEmpty =

!candidatesGroupedByCandidateSources.keySet.contains(candidateSource.identifier)

if (userAge <= 1)

candidateSourceStats

.scope("user\_age", "1", "empty").counter(isEmpty.toString).incr()

if (userAge <= 7)

candidateSourceStats

.scope("user\_age", "7", "empty").counter(isEmpty.toString).incr()

if (userAge <= 30)

candidateSourceStats

.scope("user\_age", "30", "empty").counter(isEmpty.toString).incr()

}

}

case \_ => Nil

}

case None => Nil

}

Stitch.Unit

}

def applySideEffectsBaseCandidates(

target: Target,

candidates: Seq[Candidate]

): Stitch[Unit] = Stitch.Unit

def applySideEffectsMergedCandidates(

target: Target,

candidates: Seq[Candidate]

): Stitch[Unit] = applySideEffectsBaseCandidates(target, candidates)

def applySideEffectsFilteredCandidates(

target: Target,

candidates: Seq[Candidate]

): Stitch[Unit] = applySideEffectsBaseCandidates(target, candidates)

def applySideEffectsRankedCandidates(

target: Target,

candidates: Seq[Candidate]

): Stitch[Unit] = applySideEffectsBaseCandidates(target, candidates)

def applySideEffectsTransformedCandidates(

target: Target,

candidates: Seq[Candidate]

): Stitch[Unit] = applySideEffectsBaseCandidates(target, candidates)

def applySideEffectsTruncatedCandidates(

target: Target,

candidates: Seq[Candidate]

): Stitch[Unit] = applySideEffectsBaseCandidates(target, candidates)

def applySideEffectsResults(

target: Target,

candidates: Seq[Candidate]

): Stitch[Unit] = applySideEffectsBaseCandidates(target, candidates)

}