package com.twitter.follow\_recommendations.common.candidate\_sources.promoted\_accounts

import com.twitter.adserver.thriftscala.AdServerException

import com.twitter.adserver.{thriftscala => adthrift}

import com.twitter.finagle.TimeoutException

import com.twitter.finagle.stats.Counter

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.follow\_recommendations.common.clients.adserver.AdRequest

import com.twitter.follow\_recommendations.common.clients.adserver.AdserverClient

import com.twitter.follow\_recommendations.common.clients.socialgraph.SocialGraphClient

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

import com.twitter.hermit.model.Algorithm

import com.twitter.inject.Logging

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

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

import com.twitter.stitch.Stitch

import javax.inject.Inject

import javax.inject.Singleton

case class PromotedCandidateUser(

id: Long,

position: Int,

adImpression: adthrift.AdImpression,

followProof: FollowProof,

primaryCandidateSource: Option[CandidateSourceIdentifier])

@Singleton

class PromotedAccountsCandidateSource @Inject() (

adserverClient: AdserverClient,

sgsClient: SocialGraphClient,

statsReceiver: StatsReceiver)

extends CandidateSource[AdRequest, PromotedCandidateUser]

with Logging {

override val identifier: CandidateSourceIdentifier =

PromotedAccountsCandidateSource.Identifier

val stats: StatsReceiver = statsReceiver.scope(identifier.name)

val failureStat: StatsReceiver = stats.scope("failures")

val adServerExceptionsCounter: Counter = failureStat.counter("AdServerException")

val timeoutCounter: Counter = failureStat.counter("TimeoutException")

def apply(request: AdRequest): Stitch[Seq[PromotedCandidateUser]] = {

adserverClient

.getAdImpressions(request)

.rescue {

case e: TimeoutException =>

timeoutCounter.incr()

logger.warn("Timeout on Adserver", e)

Stitch.Nil

case e: AdServerException =>

adServerExceptionsCounter.incr()

logger.warn("Failed to fetch ads", e)

Stitch.Nil

}

.flatMap { adImpressions: Seq[adthrift.AdImpression] =>

profileNumResults(adImpressions.size, "results\_from\_ad\_server")

val idToImpMap = (for {

imp <- adImpressions

promotedAccountId <- imp.promotedAccountId

} yield promotedAccountId -> imp).toMap

request.clientContext.userId

.map { userId =>

sgsClient

.getIntersections(

userId,

adImpressions.filter(shouldShowSocialContext).flatMap(\_.promotedAccountId),

PromotedAccountsCandidateSource.NumIntersections

).map { promotedAccountWithIntersections =>

idToImpMap.map {

case (promotedAccountId, imp) =>

PromotedCandidateUser(

promotedAccountId,

imp.insertionPosition

.map(\_.toInt).getOrElse(

getInsertionPositionDefaultValue(request.isTest.getOrElse(false))

),

imp,

promotedAccountWithIntersections

.getOrElse(promotedAccountId, FollowProof(Nil, 0)),

Some(identifier)

)

}.toSeq

}.onSuccess(result => profileNumResults(result.size, "final\_results"))

}.getOrElse(Stitch.Nil)

}

}

private def shouldShowSocialContext(imp: adthrift.AdImpression): Boolean =

imp.experimentValues.exists { expValues =>

expValues.get("display.display\_style").contains("show\_social\_context")

}

private def getInsertionPositionDefaultValue(isTest: Boolean): Int = {

if (isTest) 0 else -1

}

private def profileNumResults(resultsSize: Int, statName: String): Unit = {

if (resultsSize <= 5) {

stats.scope(statName).counter(resultsSize.toString).incr()

} else {

stats.scope(statName).counter("more\_than\_5").incr()

}

}

}

object PromotedAccountsCandidateSource {

val Identifier: CandidateSourceIdentifier = CandidateSourceIdentifier(

Algorithm.PromotedAccount.toString)

val NumIntersections = 3

}