package com.twitter.follow\_recommendations.common.candidate\_sources.addressbook

import com.twitter.cds.contact\_consent\_state.thriftscala.PurposeOfProcessing

import com.twitter.finagle.stats.NullStatsReceiver

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.follow\_recommendations.common.clients.addressbook.AddressbookClient

import com.twitter.follow\_recommendations.common.clients.addressbook.models.EdgeType

import com.twitter.follow\_recommendations.common.clients.addressbook.models.RecordIdentifier

import com.twitter.follow\_recommendations.common.clients.phone\_storage\_service.PhoneStorageServiceClient

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

import com.twitter.follow\_recommendations.common.utils.RescueWithStatsUtils.rescueWithStats

import com.twitter.hermit.model.Algorithm

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.stitch.Stitch

import com.twitter.strato.generated.client.onboarding.userrecs.ReversePhoneContactsClientColumn

import com.twitter.timelines.configapi.HasParams

import javax.inject.Inject

import javax.inject.Singleton

@Singleton

class ReversePhoneBookSource @Inject() (

reversePhoneContactsClientColumn: ReversePhoneContactsClientColumn,

pssClient: PhoneStorageServiceClient,

addressBookClient: AddressbookClient,

statsReceiver: StatsReceiver = NullStatsReceiver)

extends CandidateSource[HasParams with HasClientContext, CandidateUser] {

override val identifier: CandidateSourceIdentifier = ReversePhoneBookSource.Identifier

private val stats: StatsReceiver = statsReceiver.scope(this.getClass.getSimpleName)

/\*\*

\* Generate a list of candidates for the target

\*/

override def apply(target: HasParams with HasClientContext): Stitch[Seq[CandidateUser]] = {

val reverseCandidatesFromPhones: Stitch[Seq[Long]] = target.getOptionalUserId

.map { userId =>

pssClient

.getPhoneNumbers(userId, PurposeOfProcessing.ContentRecommendations)

.flatMap { phoneNumbers =>

rescueWithStats(

addressBookClient.getUsers(

userId = userId,

identifiers = phoneNumbers.map(phoneNumber =>

RecordIdentifier(userId = None, email = None, phoneNumber = Some(phoneNumber))),

batchSize = ReversePhoneBookSource.NumPhoneBookEntries,

edgeType = ReversePhoneBookSource.DefaultEdgeType,

fetcherOption =

if (target.params(AddressBookParams.ReadFromABV2Only)) None

else Some(reversePhoneContactsClientColumn.fetcher),

queryOption = AddressbookClient.createQueryOption(

edgeType = ReversePhoneBookSource.DefaultEdgeType,

isPhone = ReversePhoneBookSource.IsPhone)

),

stats,

"AddressBookClient"

)

}

}.getOrElse(Stitch.Nil)

reverseCandidatesFromPhones.map(

\_.take(ReversePhoneBookSource.NumPhoneBookEntries)

.map(

CandidateUser(\_, score = Some(CandidateUser.DefaultCandidateScore))

.withCandidateSource(identifier))

)

}

}

object ReversePhoneBookSource {

val Identifier: CandidateSourceIdentifier = CandidateSourceIdentifier(

Algorithm.ReversePhoneBook.toString)

val NumPhoneBookEntries: Int = 500

val IsPhone = true

val DefaultEdgeType: EdgeType = EdgeType.Reverse

}