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

import com.twitter.finagle.stats.NullStatsReceiver

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.follow\_recommendations.common.candidate\_sources.addressbook.AddressBookParams.ReadFromABV2Only

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.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.ForwardEmailBookClientColumn

import com.twitter.timelines.configapi.HasParams

import javax.inject.Inject

import javax.inject.Singleton

@Singleton

class ForwardEmailBookSource @Inject() (

forwardEmailBookClientColumn: ForwardEmailBookClientColumn,

addressBookClient: AddressbookClient,

statsReceiver: StatsReceiver = NullStatsReceiver)

extends CandidateSource[HasParams with HasClientContext, CandidateUser] {

override val identifier: CandidateSourceIdentifier =

ForwardEmailBookSource.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 candidateUsers: Stitch[Seq[Long]] = target.getOptionalUserId

.map { userId =>

rescueWithStats(

addressBookClient.getUsers(

userId = userId,

identifiers =

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

batchSize = AddressbookClient.AddressBook2BatchSize,

edgeType = ForwardEmailBookSource.DefaultEdgeType,

fetcherOption =

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

else Some(forwardEmailBookClientColumn.fetcher),

queryOption = AddressbookClient

.createQueryOption(

edgeType = ForwardEmailBookSource.DefaultEdgeType,

isPhone = ForwardEmailBookSource.IsPhone)

),

stats,

"AddressBookClient"

)

}.getOrElse(Stitch.Nil)

candidateUsers

.map(

\_.take(ForwardEmailBookSource.NumEmailBookEntries)

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

.withCandidateSource(identifier)))

}

}

object ForwardEmailBookSource {

val Identifier: CandidateSourceIdentifier = CandidateSourceIdentifier(

Algorithm.ForwardEmailBook.toString)

val NumEmailBookEntries: Int = 1000

val IsPhone = false

val DefaultEdgeType: EdgeType = EdgeType.Forward

}