package com.twitter.frigate.pushservice.adaptor

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.base.CandidateSource

import com.twitter.frigate.common.base.CandidateSourceEligible

import com.twitter.frigate.common.base.DiscoverTwitterCandidate

import com.twitter.frigate.pushservice.model.PushTypes.RawCandidate

import com.twitter.frigate.pushservice.model.PushTypes.Target

import com.twitter.frigate.pushservice.params.{PushFeatureSwitchParams => FS}

import com.twitter.frigate.pushservice.predicate.DiscoverTwitterPredicate

import com.twitter.frigate.pushservice.predicate.TargetPredicates

import com.twitter.frigate.pushservice.util.PushAppPermissionUtil

import com.twitter.frigate.pushservice.util.PushDeviceUtil

import com.twitter.frigate.thriftscala.{CommonRecommendationType => CRT}

import com.twitter.util.Future

class OnboardingPushCandidateAdaptor(

globalStats: StatsReceiver)

extends CandidateSource[Target, RawCandidate]

with CandidateSourceEligible[Target, RawCandidate] {

override val name: String = this.getClass.getSimpleName

private[this] val stats = globalStats.scope(name)

private[this] val requestNum = stats.counter("request\_num")

private[this] val addressBookCandNum = stats.counter("address\_book\_cand\_num")

private[this] val completeOnboardingCandNum = stats.counter("complete\_onboarding\_cand\_num")

private def generateOnboardingPushRawCandidate(

\_target: Target,

\_commonRecType: CRT

): RawCandidate = {

new RawCandidate with DiscoverTwitterCandidate {

override val target = \_target

override val commonRecType = \_commonRecType

}

}

private def getEligibleCandsForTarget(

target: Target

): Future[Option[Seq[RawCandidate]]] = {

val addressBookFatigue =

TargetPredicates

.pushRecTypeFatiguePredicate(

CRT.AddressBookUploadPush,

FS.FatigueForOnboardingPushes,

FS.MaxOnboardingPushInInterval,

stats)(Seq(target)).map(\_.head)

val completeOnboardingFatigue =

TargetPredicates

.pushRecTypeFatiguePredicate(

CRT.CompleteOnboardingPush,

FS.FatigueForOnboardingPushes,

FS.MaxOnboardingPushInInterval,

stats)(Seq(target)).map(\_.head)

Future

.join(

target.appPermissions,

addressBookFatigue,

completeOnboardingFatigue

).map {

case (appPermissionOpt, addressBookPredicate, completeOnboardingPredicate) =>

val addressBookUploaded =

PushAppPermissionUtil.hasTargetUploadedAddressBook(appPermissionOpt)

val abUploadCandidate =

if (!addressBookUploaded && addressBookPredicate && target.params(

FS.EnableAddressBookPush)) {

addressBookCandNum.incr()

Some(generateOnboardingPushRawCandidate(target, CRT.AddressBookUploadPush))

} else if (!addressBookUploaded && (completeOnboardingPredicate ||

target.params(FS.DisableOnboardingPushFatigue)) && target.params(

FS.EnableCompleteOnboardingPush)) {

completeOnboardingCandNum.incr()

Some(generateOnboardingPushRawCandidate(target, CRT.CompleteOnboardingPush))

} else None

val allCandidates =

Seq(abUploadCandidate).filter(\_.isDefined).flatten

if (allCandidates.nonEmpty) Some(allCandidates) else None

}

}

override def get(inputTarget: Target): Future[Option[Seq[RawCandidate]]] = {

requestNum.incr()

val minDurationForMRElapsed =

DiscoverTwitterPredicate

.minDurationElapsedSinceLastMrPushPredicate(

name,

FS.MrMinDurationSincePushForOnboardingPushes,

stats)(Seq(inputTarget)).map(\_.head)

minDurationForMRElapsed.flatMap { minDurationElapsed =>

if (minDurationElapsed) getEligibleCandsForTarget(inputTarget) else Future.None

}

}

override def isCandidateSourceAvailable(target: Target): Future[Boolean] = {

PushDeviceUtil

.isRecommendationsEligible(target).map(\_ && target.params(FS.EnableOnboardingPushes))

}

}