package com.twitter.frigate.pushservice.adaptor

import com.twitter.content\_mixer.thriftscala.ContentMixerProductResponse

import com.twitter.content\_mixer.thriftscala.ContentMixerRequest

import com.twitter.content\_mixer.thriftscala.ContentMixerResponse

import com.twitter.content\_mixer.thriftscala.NotificationsTripTweetsProductContext

import com.twitter.content\_mixer.thriftscala.Product

import com.twitter.content\_mixer.thriftscala.ProductContext

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.predicate.CommonOutNetworkTweetCandidatesSourcePredicates.filterOutReplyTweet

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

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

import com.twitter.frigate.pushservice.params.PushFeatureSwitchParams

import com.twitter.frigate.pushservice.params.PushParams

import com.twitter.frigate.pushservice.util.MediaCRT

import com.twitter.frigate.pushservice.util.PushAdaptorUtil

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

import com.twitter.frigate.thriftscala.CommonRecommendationType

import com.twitter.geoduck.util.country.CountryInfo

import com.twitter.product\_mixer.core.thriftscala.ClientContext

import com.twitter.stitch.tweetypie.TweetyPie.TweetyPieResult

import com.twitter.storehaus.ReadableStore

import com.twitter.trends.trip\_v1.trip\_tweets.thriftscala.TripDomain

import com.twitter.trends.trip\_v1.trip\_tweets.thriftscala.TripTweets

import com.twitter.util.Future

case class TripGeoCandidatesAdaptor(

tripTweetCandidateStore: ReadableStore[TripDomain, TripTweets],

contentMixerStore: ReadableStore[ContentMixerRequest, ContentMixerResponse],

tweetyPieStore: ReadableStore[Long, TweetyPieResult],

tweetyPieStoreNoVF: ReadableStore[Long, TweetyPieResult],

statsReceiver: StatsReceiver)

extends CandidateSource[Target, RawCandidate]

with CandidateSourceEligible[Target, RawCandidate] {

override def name: String = this.getClass.getSimpleName

private val stats = statsReceiver.scope(name.stripSuffix("$"))

private val contentMixerRequests = stats.counter("getTripCandidatesContentMixerRequests")

private val loggedOutTripTweetIds = stats.counter("logged\_out\_trip\_tweet\_ids\_count")

private val loggedOutRawCandidates = stats.counter("logged\_out\_raw\_candidates\_count")

private val rawCandidates = stats.counter("raw\_candidates\_count")

private val loggedOutEmptyplaceId = stats.counter("logged\_out\_empty\_place\_id\_count")

private val loggedOutPlaceId = stats.counter("logged\_out\_place\_id\_count")

private val nonReplyTweetsCounter = stats.counter("non\_reply\_tweets")

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

if (target.isLoggedOutUser) {

Future.True

} else {

for {

isRecommendationsSettingEnabled <- PushDeviceUtil.isRecommendationsEligible(target)

inferredLanguage <- target.inferredUserDeviceLanguage

} yield {

isRecommendationsSettingEnabled &&

inferredLanguage.nonEmpty &&

target.params(PushParams.TripGeoTweetCandidatesDecider)

}

}

}

private def buildRawCandidate(target: Target, tweetyPieResult: TweetyPieResult): RawCandidate = {

PushAdaptorUtil.generateOutOfNetworkTweetCandidates(

inputTarget = target,

id = tweetyPieResult.tweet.id,

mediaCRT = MediaCRT(

CommonRecommendationType.TripGeoTweet,

CommonRecommendationType.TripGeoTweet,

CommonRecommendationType.TripGeoTweet

),

result = Some(tweetyPieResult),

localizedEntity = None

)

}

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

if (target.isLoggedOutUser) {

for {

tripTweetIds <- getTripCandidatesForLoggedOutTarget(target)

tweetyPieResults <- Future.collect(tweetyPieStoreNoVF.multiGet(tripTweetIds))

} yield {

val candidates = tweetyPieResults.values.flatten.map(buildRawCandidate(target, \_))

if (candidates.nonEmpty) {

loggedOutRawCandidates.incr(candidates.size)

Some(candidates.toSeq)

} else None

}

} else {

for {

tripTweetIds <- getTripCandidatesContentMixer(target)

tweetyPieResults <-

Future.collect((target.params(PushFeatureSwitchParams.EnableVFInTweetypie) match {

case true => tweetyPieStore

case false => tweetyPieStoreNoVF

}).multiGet(tripTweetIds))

} yield {

val nonReplyTweets = filterOutReplyTweet(tweetyPieResults, nonReplyTweetsCounter)

val candidates = nonReplyTweets.values.flatten.map(buildRawCandidate(target, \_))

if (candidates.nonEmpty && target.params(

PushFeatureSwitchParams.TripTweetCandidateReturnEnable)) {

rawCandidates.incr(candidates.size)

Some(candidates.toSeq)

} else None

}

}

}

private def getTripCandidatesContentMixer(

target: Target

): Future[Set[Long]] = {

contentMixerRequests.incr()

Future

.join(

target.inferredUserDeviceLanguage,

target.deviceInfo

)

.flatMap {

case (languageOpt, deviceInfoOpt) =>

contentMixerStore

.get(

ContentMixerRequest(

clientContext = ClientContext(

userId = Some(target.targetId),

languageCode = languageOpt,

userAgent = deviceInfoOpt.flatMap(\_.guessedPrimaryDeviceUserAgent.map(\_.toString))

),

product = Product.NotificationsTripTweets,

productContext = Some(

ProductContext.NotificationsTripTweetsProductContext(

NotificationsTripTweetsProductContext()

)),

cursor = None,

maxResults =

Some(target.params(PushFeatureSwitchParams.TripTweetMaxTotalCandidates))

)

).map {

\_.map { rawResponse =>

val tripResponse =

rawResponse.contentMixerProductResponse

.asInstanceOf[

ContentMixerProductResponse.NotificationsTripTweetsProductResponse]

.notificationsTripTweetsProductResponse

tripResponse.results.map(\_.tweetResult.tweetId).toSet

}.getOrElse(Set.empty)

}

}

}

private def getTripCandidatesForLoggedOutTarget(

target: Target

): Future[Set[Long]] = {

Future.join(target.targetLanguage, target.countryCode).flatMap {

case (Some(lang), Some(country)) =>

val placeId = CountryInfo.lookupByCode(country).map(\_.placeIdLong)

if (placeId.nonEmpty) {

loggedOutPlaceId.incr()

} else {

loggedOutEmptyplaceId.incr()

}

val tripSource = "TOP\_GEO\_V3\_LR"

val tripQuery = TripDomain(

sourceId = tripSource,

language = Some(lang),

placeId = placeId,

topicId = None

)

val response = tripTweetCandidateStore.get(tripQuery)

val tripTweetIds =

response.map { res =>

if (res.isDefined) {

res.get.tweets

.sortBy(\_.score)(Ordering[Double].reverse).map(\_.tweetId).toSet

} else {

Set.empty[Long]

}

}

tripTweetIds.map { ids => loggedOutTripTweetIds.incr(ids.size) }

tripTweetIds

case (\_, \_) => Future.value(Set.empty)

}

}

}