package com.twitter.follow\_recommendations.flows.content\_recommender\_flow

import com.twitter.conversions.DurationOps.\_

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.follow\_recommendations.common.clients.geoduck.UserLocationFetcher

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

import com.twitter.follow\_recommendations.common.clients.user\_state.UserStateClient

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

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

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

import com.twitter.follow\_recommendations.products.common.ProductRequest

import com.twitter.stitch.Stitch

import javax.inject.Inject

import javax.inject.Singleton

@Singleton

class ContentRecommenderRequestBuilder @Inject() (

socialGraph: SocialGraphClient,

userLocationFetcher: UserLocationFetcher,

userStateClient: UserStateClient,

statsReceiver: StatsReceiver) {

val stats: StatsReceiver = statsReceiver.scope("content\_recommender\_request\_builder")

val invalidRelationshipUsersStats: StatsReceiver = stats.scope("invalidRelationshipUserIds")

private val invalidRelationshipUsersMaxSizeCounter =

invalidRelationshipUsersStats.counter("maxSize")

private val invalidRelationshipUsersNotMaxSizeCounter =

invalidRelationshipUsersStats.counter("notMaxSize")

def build(req: ProductRequest): Stitch[ContentRecommenderRequest] = {

val userStateStitch = Stitch

.collect(req.recommendationRequest.clientContext.userId.map(userId =>

userStateClient.getUserState(userId))).map(\_.flatten)

val recentFollowedUserIdsStitch =

Stitch

.collect(req.recommendationRequest.clientContext.userId.map { userId =>

rescueWithStatsWithin(

socialGraph.getRecentFollowedUserIds(userId),

stats,

"recentFollowedUserIds",

req

.params(

ContentRecommenderParams.RecentFollowingPredicateBudgetInMillisecond).millisecond

)

})

val recentFollowedByUserIdsStitch =

if (req.params(ContentRecommenderParams.GetFollowersFromSgs)) {

Stitch

.collect(

req.recommendationRequest.clientContext.userId.map(userId =>

rescueWithStatsWithin(

socialGraph.getRecentFollowedByUserIdsFromCachedColumn(userId),

stats,

"recentFollowedByUserIds",

req

.params(ContentRecommenderParams.RecentFollowingPredicateBudgetInMillisecond)

.millisecond

)))

} else Stitch.None

val invalidRelationshipUserIdsStitch: Stitch[Option[Seq[Long]]] =

if (req.params(ContentRecommenderParams.EnableInvalidRelationshipPredicate)) {

Stitch

.collect(

req.recommendationRequest.clientContext.userId.map { userId =>

rescueWithStats(

socialGraph

.getInvalidRelationshipUserIdsFromCachedColumn(userId)

.onSuccess(ids =>

if (ids.size >= SocialGraphClient.MaxNumInvalidRelationship) {

invalidRelationshipUsersMaxSizeCounter.incr()

} else {

invalidRelationshipUsersNotMaxSizeCounter.incr()

}),

stats,

"invalidRelationshipUserIds"

)

}

)

} else {

Stitch.None

}

val locationStitch =

rescueOptionalWithStats(

userLocationFetcher.getGeohashAndCountryCode(

req.recommendationRequest.clientContext.userId,

req.recommendationRequest.clientContext.ipAddress

),

stats,

"userLocation"

)

Stitch

.join(

recentFollowedUserIdsStitch,

recentFollowedByUserIdsStitch,

invalidRelationshipUserIdsStitch,

locationStitch,

userStateStitch)

.map {

case (

recentFollowedUserIds,

recentFollowedByUserIds,

invalidRelationshipUserIds,

location,

userState) =>

ContentRecommenderRequest(

req.params,

req.recommendationRequest.clientContext,

req.recommendationRequest.excludedIds.getOrElse(Nil),

recentFollowedUserIds,

recentFollowedByUserIds,

invalidRelationshipUserIds.map(\_.toSet),

req.recommendationRequest.displayLocation,

req.recommendationRequest.maxResults,

req.recommendationRequest.debugParams.flatMap(\_.debugOptions),

location,

userState

)

}

}

}