package com.twitter.follow\_recommendations.services

import com.twitter.finagle.stats.Counter

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.follow\_recommendations.common.base.StatsUtil.profileStitchSeqResults

import com.twitter.follow\_recommendations.common.clients.impression\_store.WtfImpressionStore

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

import com.twitter.follow\_recommendations.common.rankers.ml\_ranker.ranking.HydrateFeaturesTransform

import com.twitter.follow\_recommendations.common.rankers.ml\_ranker.ranking.MlRanker

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

import com.twitter.follow\_recommendations.configapi.deciders.DeciderParams

import com.twitter.follow\_recommendations.logging.FrsLogger

import com.twitter.follow\_recommendations.models.ScoringUserRequest

import com.twitter.follow\_recommendations.models.ScoringUserResponse

import com.twitter.stitch.Stitch

import javax.inject.Inject

import javax.inject.Singleton

@Singleton

class UserScoringService @Inject() (

socialGraph: SocialGraphClient,

wtfImpressionStore: WtfImpressionStore,

hydrateFeaturesTransform: HydrateFeaturesTransform[ScoringUserRequest],

mlRanker: MlRanker[ScoringUserRequest],

resultLogger: FrsLogger,

stats: StatsReceiver) {

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

private val disabledCounter: Counter = scopedStats.counter("disabled")

def get(request: ScoringUserRequest): Stitch[ScoringUserResponse] = {

if (request.params(DeciderParams.EnableScoreUserCandidates)) {

val hydratedRequest = hydrate(request)

val candidatesStitch = hydratedRequest.flatMap { req =>

hydrateFeaturesTransform.transform(req, request.candidates).flatMap {

candidateWithFeatures =>

mlRanker.rank(req, candidateWithFeatures)

}

}

profileStitchSeqResults(candidatesStitch, scopedStats)

.map(ScoringUserResponse)

.onSuccess { response =>

if (resultLogger.shouldLog(request.debugParams)) {

resultLogger.logScoringResult(request, response)

}

}

} else {

disabledCounter.incr()

Stitch.value(ScoringUserResponse(Nil))

}

}

private def hydrate(request: ScoringUserRequest): Stitch[ScoringUserRequest] = {

val allStitches = Stitch.collect(request.clientContext.userId.map { userId =>

val recentFollowedUserIdsStitch =

rescueWithStats(

socialGraph.getRecentFollowedUserIds(userId),

stats,

"recentFollowedUserIds")

val recentFollowedByUserIdsStitch =

rescueWithStats(

socialGraph.getRecentFollowedByUserIds(userId),

stats,

"recentFollowedByUserIds")

val wtfImpressionsStitch =

rescueWithStats(

wtfImpressionStore.get(userId, request.displayLocation),

stats,

"wtfImpressions")

Stitch.join(recentFollowedUserIdsStitch, recentFollowedByUserIdsStitch, wtfImpressionsStitch)

})

allStitches.map {

case Some((recentFollowedUserIds, recentFollowedByUserIds, wtfImpressions)) =>

request.copy(

recentFollowedUserIds =

if (recentFollowedUserIds.isEmpty) None else Some(recentFollowedUserIds),

recentFollowedByUserIds =

if (recentFollowedByUserIds.isEmpty) None else Some(recentFollowedByUserIds),

wtfImpressions = if (wtfImpressions.isEmpty) None else Some(wtfImpressions)

)

case \_ => request

}

}

}