package com.twitter.follow\_recommendations.services

import com.twitter.finagle.stats.StatsReceiver

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

import com.twitter.follow\_recommendations.common.models.Recommendation

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

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

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

import com.twitter.stitch.Stitch

import com.twitter.follow\_recommendations.configapi.params.GlobalParams.EnableWhoToFollowProducts

import com.twitter.timelines.configapi.Params

import javax.inject.Inject

import javax.inject.Singleton

@Singleton

class ProductRecommenderService @Inject() (

productRegistry: ProductRegistry,

statsReceiver: StatsReceiver) {

private val stats = statsReceiver.scope("ProductRecommenderService")

def getRecommendations(

request: RecommendationRequest,

params: Params

): Stitch[Seq[Recommendation]] = {

val displayLocation = request.displayLocation

val displayLocationStatName = displayLocation.toString

val locationStats = stats.scope(displayLocationStatName)

val loggedInOrOutStats = if (request.clientContext.userId.isDefined) {

stats.scope("logged\_in").scope(displayLocationStatName)

} else {

stats.scope("logged\_out").scope(displayLocationStatName)

}

loggedInOrOutStats.counter("requests").incr()

val product = productRegistry.getProductByDisplayLocation(displayLocation)

val productRequest = ProductRequest(request, params)

val productEnabledStitch =

StatsUtil.profileStitch(product.enabled(productRequest), locationStats.scope("enabled"))

productEnabledStitch.flatMap { productEnabled =>

if (productEnabled && params(EnableWhoToFollowProducts)) {

loggedInOrOutStats.counter("enabled").incr()

val stitch = for {

workflows <- StatsUtil.profileStitch(

product.selectWorkflows(productRequest),

locationStats.scope("select\_workflows"))

workflowRecos <- StatsUtil.profileStitch(

Stitch.collect(

workflows.map(\_.process(productRequest).map(\_.result.getOrElse(Seq.empty)))),

locationStats.scope("execute\_workflows")

)

blendedCandidates <- StatsUtil.profileStitch(

product.blender.transform(productRequest, workflowRecos.flatten),

locationStats.scope("blend\_results"))

resultsTransformer <- StatsUtil.profileStitch(

product.resultsTransformer(productRequest),

locationStats.scope("results\_transformer"))

transformedCandidates <- StatsUtil.profileStitch(

resultsTransformer.transform(productRequest, blendedCandidates),

locationStats.scope("execute\_results\_transformer"))

} yield {

transformedCandidates

}

StatsUtil.profileStitchResults[Seq[Recommendation]](stitch, locationStats, \_.size)

} else {

loggedInOrOutStats.counter("disabled").incr()

locationStats.counter("disabled\_product").incr()

Stitch.Nil

}

}

}

}