package com.twitter.follow\_recommendations.logging

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.follow\_recommendations.common.constants.GuiceNamedConstants

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

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

import com.twitter.follow\_recommendations.logging.thriftscala.RecommendationLog

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

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

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

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

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

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

import com.twitter.inject.annotations.Flag

import com.twitter.logging.LoggerFactory

import com.twitter.product\_mixer.core.model.marshalling.request.ClientContext

import com.twitter.product\_mixer.core.model.marshalling.request.HasClientContext

import com.twitter.scribelib.marshallers.ClientDataProvider

import com.twitter.scribelib.marshallers.ExternalRefererDataProvider

import com.twitter.scribelib.marshallers.ScribeSerialization

import com.twitter.timelines.configapi.HasParams

import com.twitter.util.Time

import javax.inject.Inject

import javax.inject.Named

import javax.inject.Singleton

/\*\*

\* This is the standard logging class we use to log data into:

\* 1) logs.follow\_recommendations\_logs

\*

\* This logger logs data for 2 endpoints: getRecommendations, scoreUserCandidates

\* All data scribed via this logger have to be converted into the same thrift type: RecommendationLog

\*

\* 2) logs.frs\_recommendation\_flow\_logs

\*

\* This logger logs recommendation flow data for getRecommendations requests

\* All data scribed via this logger have to be converted into the same thrift type: FrsRecommendationFlowLog

\*/

@Singleton

class FrsLogger @Inject() (

@Named(GuiceNamedConstants.REQUEST\_LOGGER) loggerFactory: LoggerFactory,

@Named(GuiceNamedConstants.FLOW\_LOGGER) flowLoggerFactory: LoggerFactory,

stats: StatsReceiver,

@Flag("log\_results") serviceShouldLogResults: Boolean)

extends ScribeSerialization {

private val logger = loggerFactory.apply()

private val flowLogger = flowLoggerFactory.apply()

private val logRecommendationCounter = stats.counter("scribe\_recommendation")

private val logScoringCounter = stats.counter("scribe\_scoring")

private val logRecommendationFlowCounter = stats.counter("scribe\_recommendation\_flow")

def logRecommendationResult(

request: RecommendationRequest,

response: RecommendationResponse

): Unit = {

if (!request.isSoftUser) {

val log =

RecommendationLog(request.toOfflineThrift, response.toOfflineThrift, Time.now.inMillis)

logRecommendationCounter.incr()

logger.info(

serializeThrift(

log,

FrsLogger.LogCategory,

FrsLogger.mkProvider(request.clientContext)

))

}

}

def logScoringResult(request: ScoringUserRequest, response: ScoringUserResponse): Unit = {

if (!request.isSoftUser) {

val log =

RecommendationLog(

request.toRecommendationRequest.toOfflineThrift,

response.toRecommendationResponse.toOfflineThrift,

Time.now.inMillis)

logScoringCounter.incr()

logger.info(

serializeThrift(

log,

FrsLogger.LogCategory,

FrsLogger.mkProvider(request.toRecommendationRequest.clientContext)

))

}

}

def logRecommendationFlowData[Target <: HasClientContext with HasIsSoftUser with HasParams](

request: Target,

flowData: RecommendationFlowData[Target]

): Unit = {

if (!request.isSoftUser && request.params(GlobalParams.EnableRecommendationFlowLogs)) {

val log = flowData.toRecommendationFlowLogOfflineThrift

logRecommendationFlowCounter.incr()

flowLogger.info(

serializeThrift(

log,

FrsLogger.FlowLogCategory,

FrsLogger.mkProvider(request.clientContext)

))

}

}

// We prefer the settings given in the user request, and if none provided we default to the

// aurora service configuration.

def shouldLog(debugParamsOpt: Option[DebugParams]): Boolean =

debugParamsOpt match {

case Some(debugParams) =>

debugParams.debugOptions match {

case Some(debugOptions) =>

!debugOptions.doNotLog

case None =>

serviceShouldLogResults

}

case None =>

serviceShouldLogResults

}

}

object FrsLogger {

val LogCategory = "follow\_recommendations\_logs"

val FlowLogCategory = "frs\_recommendation\_flow\_logs"

def mkProvider(clientContext: ClientContext) = new ClientDataProvider {

/\*\* The id of the current user. When the user is logged out, this method should return None. \*/

override val userId: Option[Long] = clientContext.userId

/\*\* The id of the guest, which is present in logged-in or loged-out states \*/

override val guestId: Option[Long] = clientContext.guestId

/\*\* The personalization id (pid) of the user, used to personalize Twitter services \*/

override val personalizationId: Option[String] = None

/\*\* The id of the individual device the user is currently using. This id will be unique for different users' devices. \*/

override val deviceId: Option[String] = clientContext.deviceId

/\*\* The OAuth application id of the application the user is currently using \*/

override val clientApplicationId: Option[Long] = clientContext.appId

/\*\* The OAuth parent application id of the application the user is currently using \*/

override val parentApplicationId: Option[Long] = None

/\*\* The two-letter, upper-case country code used to designate the country from which the scribe event occurred \*/

override val countryCode: Option[String] = clientContext.countryCode

/\*\* The two-letter, lower-case language code used to designate the probably language spoken by the scribe event initiator \*/

override val languageCode: Option[String] = clientContext.languageCode

/\*\* The user-agent header used to identify the client browser or device that the user is currently active on \*/

override val userAgent: Option[String] = clientContext.userAgent

/\*\* Whether the user is accessing Twitter via a secured connection \*/

override val isSsl: Option[Boolean] = Some(true)

/\*\* The referring URL to the current page for web-based clients, if applicable \*/

override val referer: Option[String] = None

/\*\*

\* The external site, partner, or email that lead to the current Twitter application. Returned value consists of a

\* tuple including the encrypted referral data and the type of referral

\*/

override val externalReferer: Option[ExternalRefererDataProvider] = None

}

}