package com.twitter.frigate.pushservice.controller

import com.google.inject.Inject

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.finagle.thrift.ClientId

import com.twitter.finatra.thrift.Controller

import com.twitter.frigate.pushservice.exception.DisplayLocationNotSupportedException

import com.twitter.frigate.pushservice.refresh\_handler.RefreshForPushHandler

import com.twitter.frigate.pushservice.send\_handler.SendHandler

import com.twitter.frigate.pushservice.refresh\_handler.LoggedOutRefreshForPushHandler

import com.twitter.frigate.pushservice.thriftscala.PushService.Loggedout

import com.twitter.frigate.pushservice.thriftscala.PushService.Refresh

import com.twitter.frigate.pushservice.thriftscala.PushService.Send

import com.twitter.frigate.pushservice.{thriftscala => t}

import com.twitter.frigate.thriftscala.NotificationDisplayLocation

import com.twitter.util.logging.Logging

import com.twitter.util.Future

class PushServiceController @Inject() (

sendHandler: SendHandler,

refreshForPushHandler: RefreshForPushHandler,

loggedOutRefreshForPushHandler: LoggedOutRefreshForPushHandler,

statsReceiver: StatsReceiver)

extends Controller(t.PushService)

with Logging {

private val stats: StatsReceiver = statsReceiver.scope(s"${this.getClass.getSimpleName}")

private val failureCount = stats.counter("failures")

private val failureStatsScope = stats.scope("failures")

private val uncaughtErrorCount = failureStatsScope.counter("uncaught")

private val uncaughtErrorScope = failureStatsScope.scope("uncaught")

private val clientIdScope = stats.scope("client\_id")

handle(t.PushService.Send) { request: Send.Args =>

send(request)

}

handle(t.PushService.Refresh) { args: Refresh.Args =>

refresh(args)

}

handle(t.PushService.Loggedout) { request: Loggedout.Args =>

loggedOutRefresh(request)

}

private def loggedOutRefresh(

request: t.PushService.Loggedout.Args

): Future[t.PushService.Loggedout.SuccessType] = {

val fut = request.request.notificationDisplayLocation match {

case NotificationDisplayLocation.PushToMobileDevice =>

loggedOutRefreshForPushHandler.refreshAndSend(request.request)

case \_ =>

Future.exception(

new DisplayLocationNotSupportedException(

"Specified notification display location is not supported"))

}

fut.onFailure { ex =>

logger.error(

s"Failure in push service for logged out refresh request: $request - ${ex.getMessage} - ${ex.getStackTrace

.mkString(", \n\t")}",

ex)

failureCount.incr()

uncaughtErrorCount.incr()

uncaughtErrorScope.counter(ex.getClass.getCanonicalName).incr()

}

}

private def refresh(

request: t.PushService.Refresh.Args

): Future[t.PushService.Refresh.SuccessType] = {

val fut = request.request.notificationDisplayLocation match {

case NotificationDisplayLocation.PushToMobileDevice =>

val clientId: String =

ClientId.current

.flatMap { cid => Option(cid.name) }

.getOrElse("none")

clientIdScope.counter(clientId).incr()

refreshForPushHandler.refreshAndSend(request.request)

case \_ =>

Future.exception(

new DisplayLocationNotSupportedException(

"Specified notification display location is not supported"))

}

fut.onFailure { ex =>

logger.error(

s"Failure in push service for refresh request: $request - ${ex.getMessage} - ${ex.getStackTrace

.mkString(", \n\t")}",

ex

)

failureCount.incr()

uncaughtErrorCount.incr()

uncaughtErrorScope.counter(ex.getClass.getCanonicalName).incr()

}

}

private def send(

request: t.PushService.Send.Args

): Future[t.PushService.Send.SuccessType] = {

sendHandler(request.request).onFailure { ex =>

logger.error(

s"Failure in push service for send request: $request - ${ex.getMessage} - ${ex.getStackTrace

.mkString(", \n\t")}",

ex

)

failureCount.incr()

uncaughtErrorCount.incr()

uncaughtErrorScope.counter(ex.getClass.getCanonicalName).incr()

}

}

}