|  |
| --- |
| Báo cáo IOT\_BE |
| Hà Công Hậu |

# báo cáo

## Thông tin chung

IOT\_BE

Tech:Typescript,NestJS,TypeORM,…

Database:mssql

## Phân tích main.ts

Sử dụng mấy cái khá lạ như helmet và useLogger

\*\*\* Helmet : lớp bảo mật HTTP bằng cách set các security header (lần đầu thấy)

\*\*\* useLogger: custom lại logger thay vì sử dụng logger của NestJS , dung để debug production

\*\*\* Phân tích về Logger:

Sử dụng Winston + DailyRotateFile:

Log ra file,có backup xoay vòng theo ngày vơi DailyRotateFile

*const* dailyRotateFileConfig = (filename: string, level: string) => ({

      filename,

      datePattern: "DD-MM-YYYY",

      level,

      zippedArchive: true,

      maxSize: "20m",

      maxFiles: "14d"

    });

Config của dailyRotateFilleConfig:

zippedArchive:true => nén lại thành .gz để tiết kiệm dung lượng

maxSize:”20mb” => dung lượng tối đa cho 1 file log 20mb

maxFiles:”14d” => lưu file log 14 ngày gần nhất nếu cũ hơn thì xóa

logFormat : Format lại log theo tùy chỉnh

Ghi đè các phương thức log

**Tổng kết :** Sử dụng Winston giúp ghi log ra file config DailyRotatFile đê xoay vòng file config , do class này kế thừa ConsoleLog của NestJS do vậy sẽ có 2 kênh log 1 là this.log sẽ ghi ra file log cón super.xxx kế thừa từ ConsoleLog của NestJS được override dể hiện thị chi tiết hơn , còn theo Winston thì có format tự định nghĩa

**(Phải sử dụng cái này để làm quen vs log)**

**\*\*\*** Sử dụng Interceptor(lần đầu nghe)

Theo docs là có thể thêm logic sau hoặc trước sau khi phương thức chạy, thay đồi kết quả được trả về từ controller theo 1 format nhất định, thay đôi format ngoại lệ ,…có thể hiểu là middleware nâng cao ko chỉ riêng cho request mà còn cho controller

\*\*\* Phân tích file stack.interceptor.ts

Phương thức intercept():

Nhận 2 tham số ExecutionContext (cái này hình như là cho phép đọc req,res,class,..của endpoint đó ) next:CallHandler

Flow:intercept() được gọi trước và sau controller tham số next có

thêm phần log request ra nữa nếu đang ở môi trương development

next.handle()=>Tiếp tục gọi controller

Next.pipe(tap(…),catchError())=>gọi sau khi controller trả response , ghi log response nếu ở môi trường development , phần cleanup chưa hiểu lắm

**(Phải sử dụng cái này)**

## Phân tích app.module.ts

Import nhưng repo cần thiết vào , đẩy hêt service vào 1 file config duy nhất khá mới r import hay phết vào provider,sử dụng middleware để verifyLogin áp dụng cho endpoint verify và mọi method ,mình nghĩ sử dụng middleware để khi token đc giải thì lấy được payload và query tiếp để lấy them nhiều data nữa để gẵn vào req.userData thay vì sử dụng passport chắc là để tối ưu hiệu suất

## Phân tích thư mục middleware

crypto.middleware.ts:

* Có 2 phần quan trọng la decrypt va encrypt theo mã nhị phân Buffer qua trình giải mã theo mình hiểu là giải mã rồi được gắn tiếp phần giãi mã còn lại cho đến khi kết thúc
* Flow của middleware này như sau:  
  + Nếu body của request có phần dữ liệu được mã hóa (req.body.encrypted) thì sử dụng phương thức decrypt để giải mã và nếu ko thì sẽ ghi đè res để mã hóa
  + Mỗi lần return từ controller thì NestJS tự động biến thành json qua res.json() hoặc rõ hơn là res.send(Json.stringify()) vi thế mới cần ghi đè res.send qua call
  + Mỗi phương thức đểu có this riêng , ở đây gán originalSend = res.send lúc này bị mất context (this===res =🡺 this===undefinded) nến sẽ bị lỗi nên mới phải sử dụng call để ép lại context đồng thời trả them encrypted (quả logic lú vãi)

## Phân tích thư mục pipe

rate\_limit.interceptor.ts:

* *private* *readonly* logger = new Logger(RateLimitInterceptor.name);
* *private* *readonly* requests = new Map<string, RateLimitInfo>();
* *private* *readonly* defaultLimit = 100; *// requests per window*
* *private* *readonly* defaultWindow = 15 \* 60 \* 1000; *// 15 minutes*
* *private* *readonly* endpointLimits = new Map([
* ['/auth/login', { limit: 5, window: 15 \* 60 \* 1000 }], *// 5 attempts per 15 min*
* ['/auth/register', { limit: 3, window: 60 \* 60 \* 1000 }], *// 3 attempts per hour*
* ['/v1/management/devices', { limit: 50, window: 60 \* 1000 }], *// 50 per minute*
* ]);
* Property :
* request: Một map có key với type là string và value là RateLimitInfo
* defaultLimit : request mỗi cửa số (dich theo comment)
* defaultWindow: thơi lượng tính rateLimit của 1 cửa sổ
* endpointLimits:những endpoint quan trọng cần tình rateLimit
* Flow của intercept():
* *const* key = this.generateKey(request);
* *const* endpoint = this.getEndpointFromUrl(request.url);
* *const* { limit, window } = this.endpointLimits.get(endpoint) ||
* { limit: this.defaultLimit, window: this.defaultWindow };
* *// Check rate limit*
* *const* rateLimitInfo = this.checkRateLimit(key, limit, window);
* key: lấy userID + userIp hoạc chỉ ip
* endpoint : cái nay hình như là loại bỏ uuid hoac id đê lây enpoint thuần (api/:id)
* Kiểm tra xem endpoint có tồn tại trong Map nhưng endpoint nhạy cảm không nếu không tra về default
* Logic của checkRateLimit:
* Kiểm tra xem user hay ip có đang truy cập không
* Nếu không tạo mới RateLimitInfo vs count =1 và thời gian tồn là dựa theo window
* Nếu có tang số lượng count lên 1 để tính toán limit tiếp
* Gắn những header để bên FE lấy dữ liệu để làm việc với người dung làm việc tiếp
* Nếu count lớn hơn limit thì trả về TooManyRequest(theo mình tra là 429)

## Cấu trúc thư mục của một module

Đánh giá khác biệt với mình :

MÌnh hay xài nest g resource module mình chưa chia cấu trục thư mục cụ thể hehe h mình lấy cái này làm chuẩ

1.config

Có 2 file chính là import và provider , theo mình đọc thì thấy file import sẽ export các Repo để TypeORM có thể sử dụng tiếp tục là các module khác chắc để sử dụng service của module đó còn them nữa là export toàn bộ entity không hiểu lám tại sao lại export toàn bộ entity ta

2.controller

- Anh đặt tên controller theo dạng v1/ten\_api , khác với mình mình sử dụng @Controller(“name”:..,”path”:…) với đặt prefix ở main

- cách đặt tên endpoint theo mình tìm hiểu thì là RPC, có thể là đặt tên để bên FE có thể hiểu được api dung để làm j  
 - Anh lấy dữ liệu từ dto r mới truyền vào service mình toàn gói 1 cục vào dto r mới lấy dữ liệu ở service

- theo tìm hiểu thì service nên chỉ là data thuần việc bóc dto thanh dữ liệu nhỏ hơn rồi mới truyền vào thì sẽ không thay đổi gi trong service nếu dto có thay đổi field(oh h ms bt)

- cách return cũng theo 1 form duy nhất với những method là Get thì sẽ đi kèm theo data còn những method khác chỉ có mỗi code và message , cách return này gọn mình toàn return mỗi message hoặc là return thằng theo service luôn

- Cách xư lĩ lỗi khá la ghê phương thức saveAuditLog() sẽ cho biết là lỗi ở endpoint nào , ở req nào (chắc là có luôn thông tin của user hoặc ip đó), lỗi ở controller nào (chi tiết phết) , còn handleError sẽ tra lỗi chi tiết hơn cho FE do theo response nên có thê tự định nghĩa

**Phải áp dụng cái này**

**3.**service

**-** Sử dụng các phương thức có sẵn của TypeORM để làm việc với database , điều hay là với những exception có thêm phần định nghĩa như code hay message để dễ làm việc hơn , sử dụng query builder cho các truy vấn phực tạp , nhiều logic quá mìn không nắm được nhưng mình thây ở tầng này thì sẽ quăng exception thông thường , còn về controller sẽ trả code + message chi tiết hơn , best practice theo tìm hiểu có thê trả them path và timestamp nhưng mà mình nghĩ saveAuditLog() đã thực hiện log ở trc đó nên là chắc có cách giải quyết nào khác khi co lỗi, còn lại là nghiệp vụ của mỗi module nhưng mà nhìn rối phết

**4.**dtos

bằng tiếng anh minh hay tra = tiếng việt ở phần docs Swagger mới trả tiếng viết chắc là dễ làm việc với FE , anh sử dụng Length còn r có min max ở trong luôn còn mình hay xài Min,Max , định nghĩa type cho những filed phức tạp còn lại có thể để TypeORM tự định nghĩa

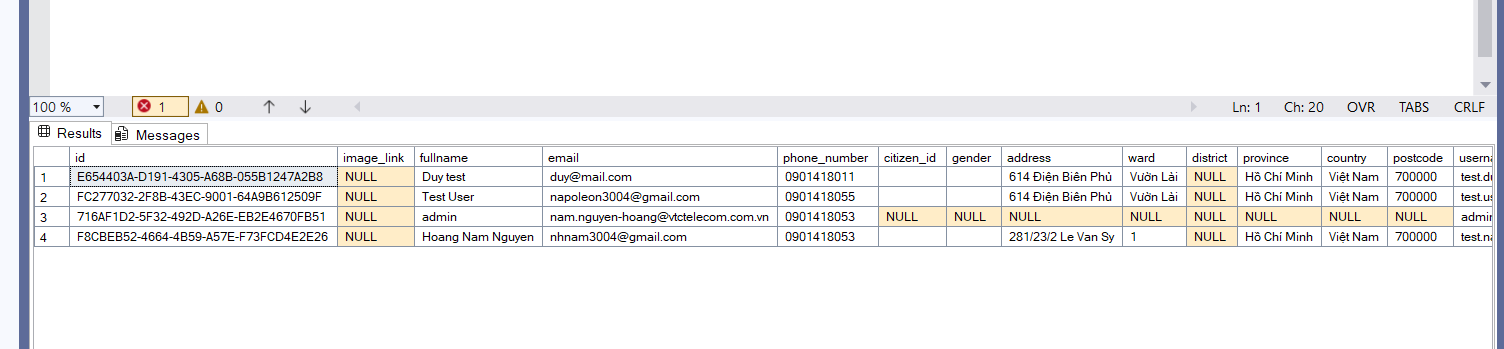
@@@ Phần này em chỉ phân tích mỗi module user tại em chưa năm được nghiệp vụ lắm

## Phân tích database

Trước khi phân tích src code thì em tìm hiểu mối quan hệ của users , usergroup,grouprole,user\_group\_lv1,user\_group\_lv2,user\_group\_lv3

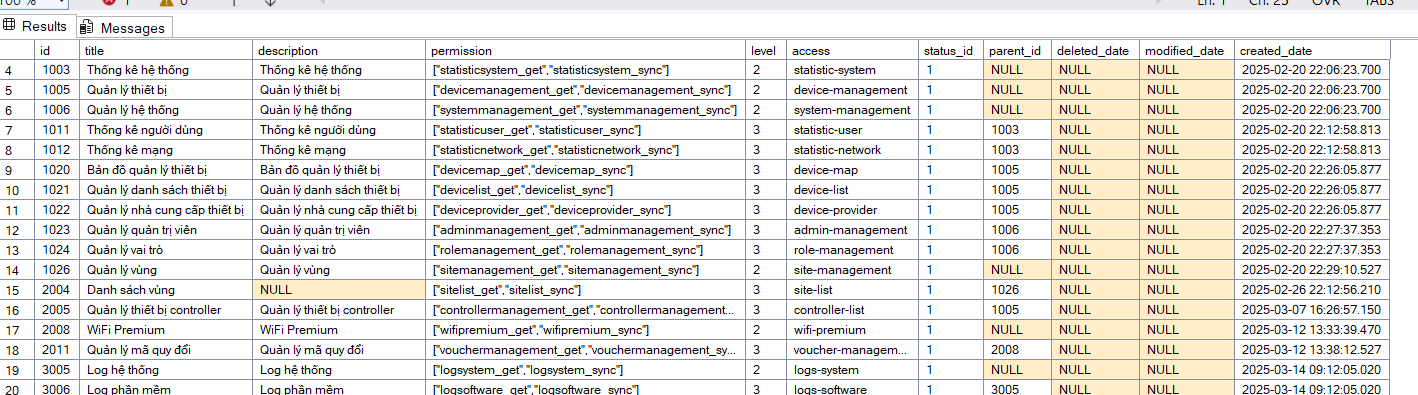
Theo tìm hiểu thì đây là phân quyên theo RBAC / Hierachial RBAC

**users**



Ở đây user có id là uuid chư ko phải id tang dần như bình thường ok chỉ chứa thông tin của user chư không chứa gì them

**group\_role**



Bảng này hình như thê hiện quyên truy cập của user ở đây theo tìm hiểu có parent\_id (gọi là hierarchical RBAC) có nghĩa là con sẽ có luôn cả quyền của cha nhưng không phải ngược lại

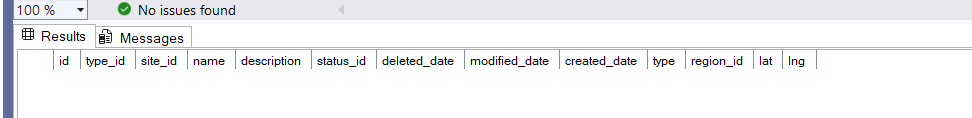
\*\*\* Mối quan hệ giữa bảng users và group\_role được thể hiện qua user\_group mối quan hệ nhiều nhiều

Mấy cái user\_group\_lvX là kiểu chia nhóm user theo nhiều tầng mình vẫn chưa hình dung được cái này lắm

## Phân tích các module (theo thứ tự trong src code)

Hướng phân tích của em sẽ là show database và xem module này có chức năng chính là gi bằng cách xem endpoint của controller dẫn đễn service nào r sẽ phân tích chức năng

**1.devices**

****

Devices theo mình nghĩ có thể là thông tin chung của 1 thiết bị

=>V thì module này sẽ quản lí thiết bị

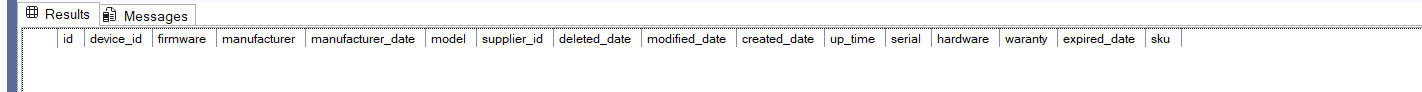
\*\*\* Controller:

 @Post("/add\_device")

  @ApiOperation({ summary: "Thêm cấu hình đồng hồ trên trang quản trị" })

Endpoint này gọi phương thức **saveDevice()** ,

Flow:

* Kiểm tra liệu tồn tại thiết bị chưa , nếu chưa tạo mới và ngược lại
* Có thêm phần chi tiết thiệt bị 
* *return* {
* code: 0,
* message: responseMessage.success,
* data: { **...**savedDevice, **...**savedDeviceDetail }
* };
* Nếu thành công sẽ trả theo format có sẵn + them data về thông tin chung và chi tiết của thiết bị mới tạo
* *return* res.status(HttpStatus.OK).json({
* code: handleAddDevice?.code,
* message: handleAddDevice?.message
* });
* Ở phần controller chỉ tra code và message , ko trả dữ liệu kèm theo chắc là do phương thức POST nến v

---------------------------------@@@@@@@@@@@@@----------------------------

 @Post("/reboot\_device")

  @ApiOperation({ summary: "Reboot thiết bị" })

Theo tìm hiểu thì reboot thiết bị là khởi động lại thiết bị

Endpount này gọi phương thức **rebootDevice()**

Flow:

* Nhận 2 tham số là id , macRequest (siteId là kiểu khu vực , chi nhánh nào còn macRequest giúp định danh phần cứng của thiết bị)
* Dựa vào 2 tham số truyền vào tìm site và thiết bị tương ứng
* Tiếp tục gọi phương thức **rebootDeviceFromHardware()**
* Ở phương thứ này gọi **obtainingAccessToken()**
* Tiếp tục gọi phương thúc **loginAuthorizeHardawareController(),**lúc này gọi Oauth của **Omada(theo tìm hiểu trên mạng)** với clientId có sẵn và thông tin user đăng nhập
* Mới tìm hiểu Omada có cơ chế đăng nhập không như Oauth của Google, bên gg chỉ cần clientID vs clientSecret vs permission của ngưới dung sẽ trả về authorization code còn Omada này trả cả csrf token va sessionId oh r mới dung 2 cái này đế sử dụng cho phương thức obtainAuthorizeCode để xin obtainCode thay vì accessToken của Google
* Đọc tiếp thì thấy lúc này phải có đầy đủ cả obtainCode,csrfToken,sessionId mới có xin được accessToken , vs refreshToken còn typeToken là j mình chưa hiểu lắm
* Có được accessToken r thì gọi endpoint đính kèm token để reboot
* Tiếp theo ở **rebootDevice()** sau khi xong thì nếu như không có lỗi gi thì thông báo **reboot thành công và ngược lại**

---------------------------------@@@@@@@@@@@@@----------------------------

@Post("/refresh\_devices")

@ApiOperation({ summary: "Làm mới thiết bị từ phần cứng" })

|  |  |
| --- | --- |
|  |  |

Endpoint này gọi phương thức r**efreshDataDevices()**

Flow:

* Đọc ko hiểu j hết nên tự đoán v
* Lấy siteID từ user kiểm tra xem có phải systemSite ko nếu có lấy danh sách site có statusId là 1
* Đẩy vào 1 màng là siteDevice những site nào có site\_id\_hardware khác null
* À nếu không thì sẽ tìm những site có trong siteListData , v có nghĩa là systemSite sẽ query ra hết dữ liệt tấc cả site còn ko thì chỉ nhuwwuxng site tương ứng vs user
* Ở đây tiếp tục gọi phương thức **getDeviceListFromHardware()** như hồi này lấy dữ liệu tư Omada chắc v tại thây gọi api có đính kèm accessToken ,logic sẽ là gọi đến khi nào dữ liệu giống nhau sau 2 lần gọi liên tiếp thì dừng đẩy hết vào allDevices rồi return về
* Lúc này lấy thông tin chi tiết của thiết bị theo siteID
* Tạo ra Map mới với key là detai.serial(serial của thiết bị) còn value sẽ là detail(nguyên 1 object lớn)
* OK mơi tìm hiều thì **sn** là số seri của thiết bị , **mac** là địa chỉ của card mạng
* Sẽ tạo ra Set đê lưu sn và mac với điều kiện là giá trị duy nhất (set là lưu giá trị duy nhất chỉ lưu key)
* *const* hardwareDevices = syncUpHardware.result.data.filter(
* (device: any) => {
* *// Kiểm tra SN trước*
* if (device.sn && !seenSn.has(device.sn)) {
* seenSn.add(device.sn);
* seenMac.add(device.sn);
* *return* true;
* }
* *// Nếu SN không hợp lệ, kiểm tra MAC*
* if (device.sn && !seenMac.has(device.sn)) {
* seenMac.add(device.sn);
* *return* true;
* }
* *// console.warn(*
* *//   `Duplicate or invalid device: SN=${device.sn}, MAC=${device.sn}`*
* *// );*
* *return* false;
* }
* );
* Nếu thiết bị có số seri và chưa xuất hiện trong **seesSn** thì add vào 2 set
* Nếu thiết bị có số seri thì mà ko có trong **seenMac** thì add vào seenMac(chưa hiểu logic này lắm)
* Lúc này sẽ kiểm tra xem thiết bị có tồn tại chưa qua số seri nếu chưa tạo mới nếu rồi thì cập nhật thiết bị(mình chưa hiểu logic này lắm)

---------------------------------@@@@@@@@@@@@@----------------------------

  @Post("/edit\_device")

  @ApiOperation({ summary: "Thay đổi thông tin thiết bị" })

Endpoint này gọi phương thức **editService() nhận** 2 tham số là id,editRequest(dto)

**Flow:**

* Kiểm tra thiết bị có tồn ko (nếu có và ngược lại)
* Lấy thông tin chi tiết của thiết bị
* Kiểm tra serial trùng (cái này có thê do lí do đặc biệt nào đó)
* Sẽ phải update 2 thứ là thông tin chung và thông tin chi tiết của thiết bị
* Cập xong xong vần return theo format kèm data

---------------------------------@@@@@@@@@@@@@----------------------------

@Get("/check\_connection")

  @ApiOperation({ summary: "Kiểm tra kết nối thiết bị" })

**E**ndpoint này gọi phương thức **checkStatusController() với** tham số truyền vào là deviceId  
Flow:

* Lấy thông tin chi tiết của thiết bị
* Rồi đến bước kiểm tra trạng thái của thiết bị
* Nếu thiết bị khong co số seri thì quăng thông báo
* Rồi sẽ kiểm tra status.id của thiết bị để đưa ra thông báo kèm (0,2,3,4),nếu status\_id là những trường hợp này update status\_id = 7
* Cỏn nếu status\_id = 1 thì update status\_id=8
* Chắc là trạng thái nào dó mình chưa biết

---------------------------------@@@@@@@@@@@@@----------------------------

 @Post("/delete\_device")

  @ApiOperation({ summary: "Xóa Thiết bị trên hệ thống" })

Endpoint này gọi phương thức **deleteDevice() với** tham số truyền vào là dataQuery.id

Flow:

* Tìm thiết bị dựa trên tham số truyền vào là id
* Nếu tìm được thì sẽ thì tìm tiếp các thiết bị con qua **region\_id** của thiết bị được tìm thấy (em không có biết có phải thiết bị con ko em dịch ra)
* Tiếp tục là tìm site cũng theo **region\_id** luôn
* Tóm gọn phần xóa không phải là xóa thẳng mà la update lại status\_id = 6 và cập nhật ngày xóa
* Những cái còn lại update status\_id = 2 và cập nhật lại ngày chỉnh sửa

---------------------------------@@@@@@@@@@@@@----------------------------

@Post("/change\_status")

  @ApiOperation({ summary: "Thay đổi trạng thái on/off của controller" })

Endpoint này gọi phương thức **switchControllerStatus()** truyền 2 tham số id , statusId

Flow:

* Vẫn giống như delete sẽ tìm kiếm thiết bị và thiết bị cùng site theo **region\_id**  của existingDevice
* Ở đây có 2 trường hợp là **case 1**  và **case 13 :**
  + **case 1** thì update devices,sys\_config,siteDevice(site của thiết bị đó) (chắc là sẽ có bảng chung ) thành 1 và cập nhât lại ngày chỉnh sửa
  + **case 13** thì cũng tương tự vs status\_id = 13

---------------------------------@@@@@@@@@@@@@----------------------------

 @Get("/data\_devices")

  @ApiOperation({ summary: "Lấy danh sách thiết bị" })

Endpoint này gọi phương thức **getDataDevices()** với tham số truyền vào là các param lây từ query

Flow:

* Hehe cái này mình có áp dụng khi kết hợp vs reactQuery nhưng mà của anh còn co join vơi bảng khác
* Khi đã có queryBuilder r thì sẽ xem có tồn tại các param ko để kết hợp vs **andWhere** để lấy dữ liệu cuối cùng

---------------------------------@@@@@@@@@@@@@----------------------------

@Post("/update\_location")

  @ApiOperation({ summary: "Cập nhật vị trí địa lý của thiết bị (lat/lng)" })

Endpoint này sử dụng phương thức **updateDeviceLocation()** với 2 tham số truyền vao là id, locationRequest(dto)

Flow:

* Tìm thiết bị dựa trên id
* Cập nhật lại lat,lng (hình như là kinh dộ và vĩ độ) và ngày chình sửa
* Tiếp theo là tra về những trường đã được select

**2.excel\_management**

\*\*\*Controller:

 @Get("/export\_excel")

  @ApiOperation({ summary: "Export Excel" })

Endpoint này gọi phương thức **handleExportExcel()** với thám số truền vào tương ứng

À nếu không thuộc type hoặc không tồn tại endDate hoặc startDate thì trả 400

Flow:

* *const* workbook = XLSX.utils.book\_new();
* Cái này là tạo ra một sheet excel trống
* Tùy theo case sẽ trả về 1 format khác nhau
* Cuối cúng là chỉnh lại theo tùy chỉnh
* *return* new StreamableFile(excelBuffer);
* Theo giai thich của chatGPT , thì làm như v để NestJS có thể lo phần trả về còn có thể setHeader để phối hợp với trình duyệt trình duyệt

---------------------------------@@@@@@@@@@@@@----------------------------

**3.log\_hardware**

\*\*\*Controller:

  @Get("/data\_logs")

  @ApiOperation({ summary: "Lấy thông tin log phần cứng" })

Endpoint này có 2 case nên có 2 phương thức **findAlertLogHardware()** và **findEventLogHardware()** truyền param tương ứng

Flow:

* Vẫn là xây dựng queryBuilder r xem có param nào thì gắn vs andwhere tương ứng

---------------------------------@@@@@@@@@@@@@----------------------------

  @Post("/refresh\_log\_alerts")

  @ApiOperation({ summary: "Sync up dữ liệu alert với phần cứng" })

Endpoint này sử dụng phương thức **automaticRefreshAlert()**

Cái này có chức năng đông bộ dữ liệu từ backend lên 1 thiết bị nào đó thiêt bị mạng hả ta

Flow:

* Lấy tất cả site có status\_id = 1
* Sẽ có endDate và startDate vs startDate là trc 15 ngày với endDate rồi truyền vào phương thức **refreshDateAlert()**
* Vẫn giống hồi nãy xem có phải systemSite hay ko nếu có lấy hết tất cả site có status\_id = 1 còn nếu không chỉ lấy trong mảng siteListData
* *const* chunkArray = <T>(array: T[], size: number): T[][] => {
* *const* result: T[][] = [];
* for (*let* i = 0; i < array.length; i += size) {
* result.push(array.slice(i, i + size));
* }
* *return* result;
* };
* Theo tìm hiểu thì chunk là chia nho mảng thành nhưng mảng con ra theo 1 size cố địng , T[][] là mảng trong mảng sau khi chia nho mảng thành nhiểu mảng con thì ta có mảng trong mảng sử dụng generic cho nhiều kiểu dữ liệu khác nhau
* Tiếp tục sử dụng phương thức **getErrorLogSiteNotiFromHardware()** cái này vẫn gọi api kèm token v cái Omada này là lưu log hả ta
* Nếu thành công lấy log\_alert vs log\_alert.site\_id = siteId, cuối cùng là lưu lại dưới dạng map vs key là detail.time\_alert\_hardware và value là giá trị của log
* Kết quả tra về từ phương thức **getErrorLogSiteNotiFromHardware()**  luu vào alertLogActive thông qua hardwareAlertLog
* Đọc code thì mình hiểu là chạy 1 vòng for nếu alert đã tồn tại trong map alertLogMap thì updateLog = alertLog hiện tại oh v cái này đảm bảo là alert sẽ luôn mới nhất hả ta rồi đây vào mang object nếu có tồn tại key trong updateData updateData trong vòng for nên sẽ luôn được tạo mới
* Còn nếu alert chưa tồn tại thì tạo mới theo new LogAlertHardware() r push vào mảng entiesToInsert
* Oh entiesToUpdate là dung để update còn entitiesToInSert dung để insert vào database
* Lúc này se dung chunk để chia nhỏ 2 mảng trên và sẽ có những hành dộng tương ưnsg

---------------------------------@@@@@@@@@@@@@----------------------------

 @Get("/data\_chart\_logs")

  @ApiOperation({ summary: "Lấy thông tin log phần cứng" })

Endpoint này sử dụng phương thức **chartSummaryAlertLog() và chartSummaryEventLog()**

Flow:

* Vẫn như cũ kiểm tra xem có phải systemSite ko nếu không thì lấy log theo những siteId tương ứng
* Lọc theo siteDateInput
* Tiếp theo mình thấy return sẽ lấy từ kết quả query 1 ngày liên tục trong 15 ngày Vơi object có 2 property là timsestamp và count , timestamp sẽ cố định là 7h sang và count là bản ghi trong 1 ngày theo mình dộc là v xong r sẽ trả dateArray và total(tổng số bản ghi được cộng dồn thông qua count )

eventLog là ghi lại mọi sự kiện xảy ra , còn alertlog dung khi có 1 cái j đó vượt mức,cảnh báo

---------------------------------@@@@@@@@@@@@@----------------------------

  @Post("/resolve\_logs")

  @ApiOperation({ summary: "Resolve log alert phần cứng thiết bị" })

Endpoint này sử dụng **resolveAlertLogs()**

**Resovle log có nghìa là thông báo thiết bị đã hoạt dộng lại bình thường**

Khi sinh ra alertLog nên có resolveLog để kết thúc vòng đời h ms bt

Flow**:**

* Lấy những alertLog theo site\_id, những log này cx phải nằm trong danh sách id và chưa bào h xóa và không có resolve\_date
* Nếu alertLog = 0 thì return luôn
* Tiếp theo là gom nhóm sử dụng reduce , mình tìm hiểu thì reduce sẽ trả về acc có dạng giống hashMap kiếm tra xem siteKey đã tồn tại chưa nếu chưa sẽ tạo mới và push vào object đó nếu r thì tương tự cuối cùng acc sẽ là map có dạng key value vs key là siteKey và value là các log
* Gọi API hardware để resolve từng nhóm site
* Sử dụng map hồi này để làm tiếp , lúc này sẽ truyền từng log vào qua vong for vào phương thức **resolveSiteAlertLogs()** đề gọi endpoint resolve của omada
* Cuối cùng là đẩy vào hardwareResult
* Kiểm tra xem có hardwareApi nào thất bại ko , return lỗi
* Phần còn lại là update Database

---------------------------------@@@@@@@@@@@@@----------------------------

  @Post("/delete\_logs")

  @ApiOperation({ summary: "Xóa log phần cứng thiết bị" })

Endpoint này sử dụng phương thức **deleteLogHardware()** nhận 2 tham số là id và type

Flow:

* Flow cũng giống như resolve luôn khác là là gọi endpoint delete từ omada th

---------------------------------@@@@@@@@@@@@@----------------------------

  @Get("/alert-log-chart")

  @ApiOperation({ summary: "Lấy dữ liệu alert log dạng chart (group theo level và resolved/unresolved) từ Omada" })

Endpoint này khá lạ

Flow:

* Tìm siteEntity thông qua query.siteId
* Nếu siteEntity tồn tại thì sẽ có regionId tương ứng
* Khi này gọi phương thức **getAlertLogChartService()**  đê xử lí chart
* Hàm này kiểm tra 2 mảng resolvedLogs , unresolveLog để tìm xem có bao nhiêu log resolve hoặc unresolved và group theo level

**4.orders**

**\*\*\***Controller:

@Get("/data\_billings")

  @ApiOperation({ summary: "Lấy danh sách hóa đơn" })

Endpoint này gọi phương thức **findBilling()**

Flow:

* Lây thông tin của những billing có billing\_id nằm trong orders
* Nếu có them filter thì andwhere
* Không hiểu đoạn billngData lắm

---------------------------------@@@@@@@@@@@@@----------------------------

 @Get("/data\_orders")

  @ApiOperation({ summary: "Lấy danh sách đơn hàng" })

Endpoint này sử dụng phương thức **getDataOrders()**

Flow:

* Query và left join giữa các bảng để lấy dữ liệu
* Nếu không có systemSite thì lọc theo site\_id
* Tiếp tục là andwhere nếu có tồn tại query nào

---------------------------------@@@@@@@@@@@@@----------------------------

@Get("/chart\_orders")

  @ApiOperation({ summary: "Lấy chart trên overview" })

Endpoint này có 3 case khác nhau

Flow:

* **Case “summary”**:
  + Lấy orders trong 1 khoang thời gian nhất định
  + Xong r data trả về là bao gôm total\_orders(Tổng số lượng đơn hang),gross\_profit(tổng tiền tất cả đơn hàng),net\_profit(Tông tiền tất cả đơn hang sau khi trừ thuế và giảm giá)
* **Case “income”:**
  + Lây orders trong 1 khoang thời gian nhất dịnh và được nhóm theo ngày và đếm tổng sổ lượng đơn hang có trong 1 ngày
  + Lọc dữ liệu lấy 1 màng catagories mảng này chứa tất cả ngày có đơn hang còn lại nếu ngày đó ko tồn tại trong record trả về thì gán số lượng đơn hang = 0
* **Case “orders”:**
  + Logic tương tự như income nhưng có them grossRevenue,netRevenue

---------------------------------@@@@@@@@@@@@@----------------------------

  @Post("/add\_order")

  @ApiOperation({ summary: "Thêm đơn hàng thủ công từ trang portal" })

Endpoint này sử dụng phương thức **createManualOrder()**

Flow:

* Tạo mới bill
* Tạo mới orders
* Tìm voucherDetail dựa trên itemList
* Nếu số lượng record trả về ít hơn số lượng yêu cầu thì trả lỗi
* Nếu không thì tạo them thông tin voucher vào orderDetail

---------------------------------@@@@@@@@@@@@@----------------------------

@Get("/billings")

@ApiOperation({ summary: "Lấy danh sách hóa đơn nước" })

Endpoint này sử dụng phương thức **getWaterBilling()**

Flow:

* Lấy dữ liệu billing kết hợp left join vs user,site,water\_usage,sắp xếp theo ngày tạo thứ từ ngày mới nhất đến ngày cũ nhât
* andWhere với những param có tồn tại

---------------------------------@@@@@@@@@@@@@----------------------------

  @Post("/generate\_monthly\_billing")

  @ApiOperation({ summary: "Tạo hóa đơn hàng tháng cho tất cả người dùng" })

Endpoint này sử dụng phương thức **generateMonthBilling()**

Flow:

* query để lấy lượng nước sử dụng hang tháng theo user , site ,device có ca billing\_month,billing\_year
* Nếu có them cả site\_id thì andwhere tiếp
* Kiểm tra xem đã tạo hóa đơn hay chưa nêu chưa tạo mới hóa đơn sau khi đã qua các bước tính toán số liệu

---------------------------------@@@@@@@@@@@@@----------------------------

 @Post("/pay\_billing")

  @ApiOperation({ summary: "Thanh toán hóa đơn nước" })

Endpoint này gọi phương thức **payWaterBilling()**

Flow:

* Tìm hóa đơn theo id
* Nếu không tồn tại hóa đơn trả lỗi
* Nếu hóa đơn đã thanh toán rồi thì xuất thông báo
* Nếu hóa đơn chưa thanh toán thi cập nhât lại trạng thái thành đã thanh tóan
* Cái này không qua payment nào hả ta chắc la thu tiền mặt r cập nhật lên

---------------------------------@@@@@@@@@@@@@----------------------------

  @Get("/billing\_details")

  @ApiOperation({ summary: "Lấy chi tiết hóa đơn nước" })

Endpoint này gọi phương thức **getBillingById()** với tham số truyền vào là billing\_id

Flow:

* Lấy dữ liệu billing kèm user, site, water\_usage tương ứng
* Trả hóa đơn nếu có kêt quả

---------------------------------@@@@@@@@@@@@@----------------------------

  @Get("/statistics")

  @ApiOperation({ summary: "Lấy thống kê hóa đơn nước" })

Endpoint này sử dụng phương thức getBillingStatics() truyền vào 2 tham số là billing\_month,billing\_year

Flow:

* Lấy tât cả billing theo billing\_month,billing\_year
* Đếm tống só lượng bill
* Đêm số lượng bill đã thanh toán
* Đếm số lượng bill chưa thanh toán
* Tính tồng tiền của số bill đã thanh toán
* Tính tổng water.consumption
* Trả hết dữ liệu về

---------------------------------@@@@@@@@@@@@@----------------------------

@Get("/user\_billings")

  @ApiOperation({ summary: "Lấy danh sách hóa đơn theo user\_id" })

Endpoint này gọi phương thức **getUserBillings()**

Flow:

* Lây hết tất cả billing kèm theo user,site,water\_usage sắp xếp theo ngày mới đến cũ nhất của user
* Nếu có them billing\_month,billing\_year thì andwhere vào

**5.pattern**

@Controller("/v1/partner\_management")

@ApiTags("API quản lý đối tác quảng cáo, đối tác thiết bị")

**\*\*\*Controller:**

  @Post("/add\_partner")

  @ApiOperation({ summary: "Thêm cấu hình quảng cáo trên trang quản trị" })

Endpoint này sử dụng phương thức savePartner() với tham số truyền vào là addRequest

Flow:

* Kiêm tra xem partner có tồn tại hay chưa nếu chưa thì tạo mới

---------------------------------@@@@@@@@@@@@@----------------------------

  @Post("/edit\_partner")

  @ApiOperation({ summary: "Thay đổi thông tin đối tác" })

Endpoint này sử dụng phương thức **editPartner()**  với 2 thám số truyền vào là id ,editRequest

Flow:

* Kiểm tra xem partner có tồn tại hay không nếu có thì update dữ liệu

**6.region**

@Controller("/v1/region\_management")

@ApiTags("API quản lý chi nhánh")

**\*\*\***Controller:

  @Get("/data\_regions")

  @ApiOperation({ summary: "Lấy danh sách chi nhánh" })

Endpoint này gọi phương thức **getDataRegions()**

Flow:

* Lấy hết các region có status\_id = 1 nếu không phải specialRegion thì lấy theo các region có trong list
* Tiếp tục query theo regionDataInput
* Kết hợp với filter nếu có

---------------------------------@@@@@@@@@@@@@----------------------------

  @Post("/delete\_regions")

  @ApiOperation({ summary: "Xóa chi nhánh trên hệ thống" })

Endpoint này gọi phương thức deleteDâtaRegions() có tham số truyền vào là dataQuery

Flow:

* Kiểm tra xem region có tồn tại hay ko nếu có update status\_id = 2

---------------------------------@@@@@@@@@@@@@----------------------------

  @Post("/edit\_regions")

  @ApiOperation({ summary: "Thay đổi thông tin chi nhánh" })

Endpoint này gọi phương thức **editDataRegions()**

Flow:

* Kiểm tả xem region có tồn tại hay ko theo id
* Nếu có cập nhật region

---------------------------------@@@@@@@@@@@@@----------------------------

  @Post("/add\_regions")

  @ApiOperation({ summary: "Thêm chi nhánh quản lý trang quản trị" })

Endpoint này gọi phương thức **addDataRegion()** với thám số là addRequest

Flow:

* Tạo mới theo addRequest

**7.site**

@Controller("/v1/site\_management")

@ApiTags("API quản lý vùng")

**\*\*\***Controller:

  @Get("/data\_sites")

  @ApiOperation({ summary: "Lấy danh sách vùng" })

Endpoint này sử dụng phương thức **getDataSite()**

Flow:

* Lấy site\_detail kèm theo site tương ứng và có status\_id = 1
* Nếu có siteListData thì andwhere
* Nếu có them query nào thì andwhere tiếp

---------------------------------@@@@@@@@@@@@@----------------------------

 @Post("/delete\_sites")

  @ApiOperation({ summary: "Xóa vùng trên hệ thống" })

Endpoint này sử dụng phương thức **deleteDataSites()**  với tham số truyền vào là dataQuery

Flow:

* Tìm site tương ứng theo id
* Nếu ko tồn tại tra lỗi không tìm thấy
* Nếu có cập nhật status\_id = 2

---------------------------------@@@@@@@@@@@@@----------------------------

  @Post("/edit\_sites")

  @ApiOperation({ summary: "Thay đổi thông tin vùng" })

Endpoint này gọi phương thức **editDataSites()** với 2 tham số là dataQuery và editRequest

Flow:

* Tìm xem site có tồn tại hay không theo id tương ứng
* Nếu có update theo editRequest

---------------------------------@@@@@@@@@@@@@----------------------------

  @Post("/add\_sites")

  @ApiOperation({ summary: "Thêm vùng quản lý trang quản trị" })

Endpoint này sử dụng phương thức **addDataSite()** với tham số là dataRequest

Flow:

* Tạo mới site theo dataRequest