Java Spring RESTful APIs – Xây dựng Backend với Spring Boot

Mục lục

DANH SÁCH TOÀN BỘ API (44)

Phân loại	Chức năng	Phương thức	API
Auth	Lấy thông tin tài khoản	GET	/api/v1/auth/account
	Đăng nhập	POST	/api/v1/auth/login
	Lấy refresh_token	GET	/api/v1/auth/refresh
	Đăng xuất	POST	/api/v1/auth/logout
	Đăng ký	POST	/api/v1/auth/register
Company	Tạo mới công ty	POST	/api/v1/companies
	Lấy thông tin công ty	GET	/api/v1/companies
	Cập nhật công ty	PUT	/api/v1/companies
	Xóa công ty theo ID	DELETE	/api/v1/companies/{id}
	Lấy thông tin công ty (ID)	GET	/api/v1/companies/{id}
Email	Gửi email	GET	/api/v1/email
File	Đăng tải file	POST	/api/v1/files
Job	Tạo mới công việc	POST	/api/v1/jobs
	Cập nhật công việc	PUT	/api/v1/jobs
	Xóa công việc theo ID	DELETE	/api/v1/jobs/{id}
	Lấy thông tin công việc theo ID	GET	/api/v1/jobs/{id}
	Lấy thông tin công việc (phân trang)	GET	/api/v1/jobs
Permission	Tạo mới 1 quyền hạn	POST	/api/v1/permission
	Cập nhật 1 quyền hạn	PUT	/api/v1/permission
	Xóa 1 quyền hạn	DELETE	/api/v1/permission/{id}
	Lấy thông tin quyền hạn	GET	/api/v1/permission
Resume	Tạo 1 hồ sơ	POST	/api/v1/resumes
	Cập nhật 1 hồ sơ	PUT	/api/v1/resumes
	Xóa 1 hồ sơ	DELETE	/api/v1/resumes/{id}
	Lấy thông tin hồ sơ theo ID	GET	/api/v1/resumes/{id}
	Lấy toàn bộ thông tin hồ sơ	GET	/api/v1/resumes
	Lấy danh sách hồ sơ theo người dùng	POST	/api/v1/resumes/by-user
Role	Tạo mới 1 vai trò	POST	/api/v1/roles
	Cập nhật vai trò	PUT	/api/v1/roles
	Xóa 1 vai trò theo id	DELETE	/api/v1/roles/{id}
	Lấy thông tin vai trò	GET	/api/v1/roles
	Lấy thông tin vai trò theo ID	GET	/api/v1/roles/{id}
Skill	Tạo 1 kỹ năng	POST	/api/v1/skills
	Cập nhật 1 kỹ năng	PUT	/api/v1/skills
	Xóa 1 kỹ năng	DELETE	/api/v1/skills/{id}
	Lấy thông tin toàn bộ kỹ năng	GET	/api/v1/skills
Subscriber	Tạo 1 người đăng ký	POST	/api/v1/subscribers
	Cập nhật 1 người đăng ký	PUT	/api/v1/subscribers
	Lấy thông tin người đăng ký theo kỹ năng	POST	/api/v1/subscribers/skills
User	Tạo mới 1 người dùng	POST	/api/v1/users
	Xóa 1 người dùng	DELETE	/api/v1/users/{id}
	Lấy thông tin người dùng theo ID	GET	/api/v1/users/{id}
	Lấy thông tin toàn bộ người dùng	GET	/api/v1/users
	Cập nhật thông tin người dùng	PUT	/api/v1/users

Các kiến thức cần nắm

JSON

JSON : Javascript Object Notation. Là một định dạng dữ liệu để lữu trữ và trao đổi dữ liệu được sử dụng ở nhiều ngôn ngữ khác nhau: Java, C#,…

Ví dụ về JSON:

{

“id”: 1

“name”: “Trinh Quang Lam”

}

Sử dụng cặp dấu {} để định nghĩa JSON

Các thuộc tính được định nghĩa theo quy luật name:value(ngăn cách nhau bởi dấu :). Thuộc tính name luôn được bọc bởi “ “

Chi tiết: https://www.w3schools.com/js/js_json_intro.asp

API

API (Application Programing Interface) hiểu đơn giản là 1 đường link URL tại phía backend, frontend sẽ gọi tới đường link URL này để lấy/sử dụng dữ liệu

HTTP (HyperText Transfer Protocol) là giao thức truyền tải siêu văn bản, được dùng để giao tiếp giữa trình duyệt và máy chủ web. Nó xác định cách gửi và nhận dữ liệu qua internet, chẳng hạn như tải trang web, hình ảnh, hoặc gửi biểu mẫu.

HTTP Method → thao tác CRUD

POST → Tạo mới 1 thực thể

GET → lấy thông tin từ Server

PUT/PATCH → Cập nhật thông tin

DELETE → Xóa một thực thể

Cấu trúc HTTP Request:

Request Line: method + URL

Header Variabled

Message body : Json

Chi tiết: https://jsonplaceholder.typicode.com/

Response Entity

Response Entity trong HTTP là phần dữ liệu chính được trả về từ máy chủ trong một **HTTP Response**.

Để các hệ thống nói chuyện với nhau 1 cách đầy đủ nhất, 1 lời phản hồi response sẽ gồm:

Thông tin header

Thông tin status

Thông tin body

Ví dụ: **ResponseEntity.status(HttpStatus.Ok).headers(Instance_of_HttpHeaders)**

.body(Instance_of_object_send_back_to_client);

Một số HTTP Status hay dùng:

Mã lỗi ám chỉ request thành công:

200 – request succeeded (hay dùng cho method GET/PUT/DELETE)

201 – request created a resource (hay dùng cho method POST)

204 – no content to return (dùng khi muốn có thông báo không có data ở phản hồi)

Mã lỗi ám chỉ request thất bại (lỗi do client)

400 – bad request(lỗi exception, validate)

401 – unauthorized : thường dùng khi client chưa đăng nhập

403 – Forbidden : đã đăng nhập thành công, nhưng không có quyền hạn để thực hiện tác vụ này

404 – Resource not found : không tìm thấy tài nguyên mà client yêu cầu

405 – Method not supported : check cho đúng method khi sử dụng với endpoint

Mã lỗi ám chỉ request thất bại (lỗi do server):

500 – Internal Server error : lỗi xảy ra bên trong Server

503 – Service Unavailable : server không hoạt động nên không có sẵn dịch vụ để sử dụng

Xử lý Exception

@ExceptionHandler: lắng nghe các Exception xảy ra trong controller (phạm vi hẹp)

@ControllerAdvice: xử lý @ExceptionHandler được chia sẻ tại tất cả controller trong ứng dụng MVC

@RestControllerAdvice : sử dụng với RESTFul (= @ControllerAdvice + @ResponseBody)

JSON Web Token

Trước khi đi vào tìm hiểu JSON Web Token, ta cần hiểu về hai khái niệm Stateful và Stateless

Stateful = state application + full : chứa đầy đủ state của application (thường sử dụng trong mô hình MVC)

Lưu trữ thông tin bên trong ứng dụng, ví dụ như thông tin người dùng đăng nhập

Session: phiên đăng nhập. Được dùng trong mô hình Stateful, cách mà ứng dụng lưu trữ data giữa các lời gọi Request

Stateless: state application + less: không chứa state của application (thường sử dụng trong mô hình REST – dùng để chia tách code Frontend + Backend)

Không lưu trữ thông tin trong ứng dụng

Trong mô hình Stateless, không tồn tại khái niệm “Session”, thay vào đấy là : “Token”

Cơ chế xác thực dựa vào Session(Stateful):

Bước 1: login với username/password

Nếu login thành công, server sẽ tạo và lưu thông tin tại:

Client thông qua cookies(lưu SESSION_ID)

Server trong memory (RAM) hoặc database

Bước 2: Mỗi lần người dùng F5(refresh) website gửi 1 request từ client lên server, các bước làm tại Server:

Client sẽ gửi kèm SESSION_ID (thông qua cookies)

Server sẽ kiểm tra SESSION_ID có đang tồn tại không? Nếu có tiếp tục truy cập bình thường, không thì sẽ cho out

Mô hình Stateful với Session chỉ áp dụng hiệu quả khi người lập trình viên muốn kiểm soát cả Frontend và Backend

Cơ chế xác thực dựa vào Token (Stateless)

Server với Server, mobile app, desktop app,…

Token: là một chuỗi ký tự đã được mã hóa (chỉ Server mới có thể hiểu)

Bước 1: login với username/password

Nếu login thành công, server sẽ tạo token, lưu tại đâu, client sẽ tự quyết định

Server không lưu bất cứ thông tin về việc user login

Bước 2: Mỗi lần người dùng F5(refresh) website gửi 1 request từ client lên server, sẽ cần gửi kèm token đã có tại bước 1. Server sẽ giải mã token để biết được user có hợp lệ hay không

Bây giờ ta sẽ đi tìm hiểu về JSON Web Token. JSON Web Token được viết tắt là JWT là một chuỗi ký tự được mã hóa thông qua thuật toán và có tính bảo mật cao. Được sử dụng để trao đổi thông tin giữa các hệ thống với nhau (server-server,client-server)

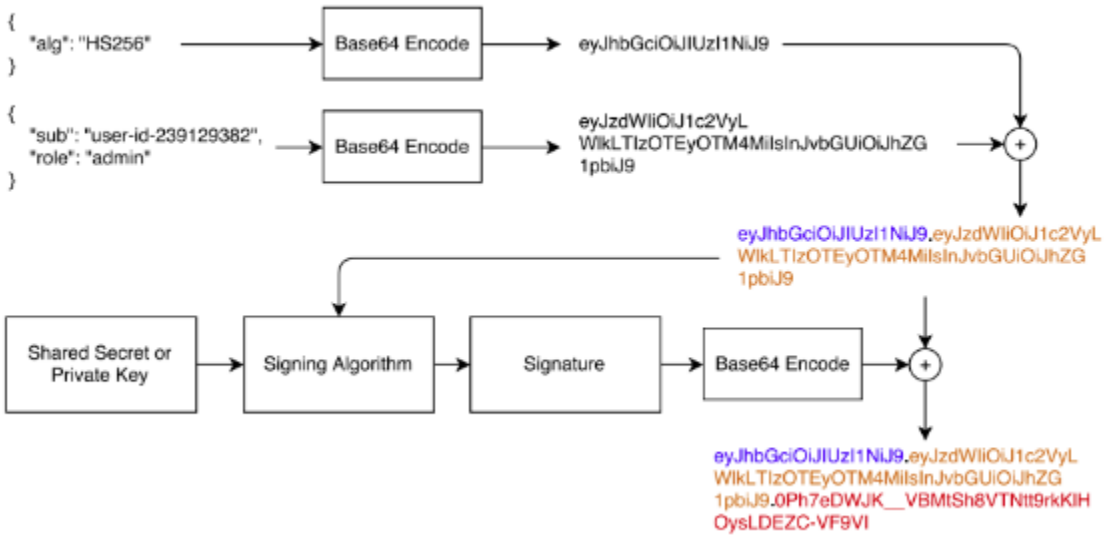
Cấu trúc của JWT

JWT cấu trúc gồm 3 phần chính: **header**.**payload**.**signature**

Header (chứa thông tin về thuật toán mã hóa): được encode dưới dạng base64

Payload (chứa data client): encode dưới dạng base64

Signature : được tạo nên từ thuật toán mã hóa +(header+payload)+key dưới dạng base64. Như vậy signature gồm 3 thành phần: thuật toán mã hóa, data của (header+payload) và key(có thể là mật khẩu/ hoặc sử dụng private/public key)



Quá trình tạo ra 1 JWT

Key được tạo ra với một mục đích, cho dù JWT bị lộ ra, nếu không có key → không thể giải mã token được. Có 2 hình thức tạo key phổ biến: một là dùng mật khẩu(hash) và hai là dùng public/private key

OAuth là một chuẩn dùng để xác thực người dùng thông qua token

Công cụ

Ngôn ngữ : Java

Backend: Java Spring

Spring Boot: cấu hình và chạy dự án một cách nhanh chóng

Spring Security: xác thực và phân quyền người dùng với JWT sử dụng cơ chế oauth2

Spring JPA: xử lý và thao tác với cơ sở dữ liệu database với ORM

Frontend: React Vite (Typescript)

Database: MySQL

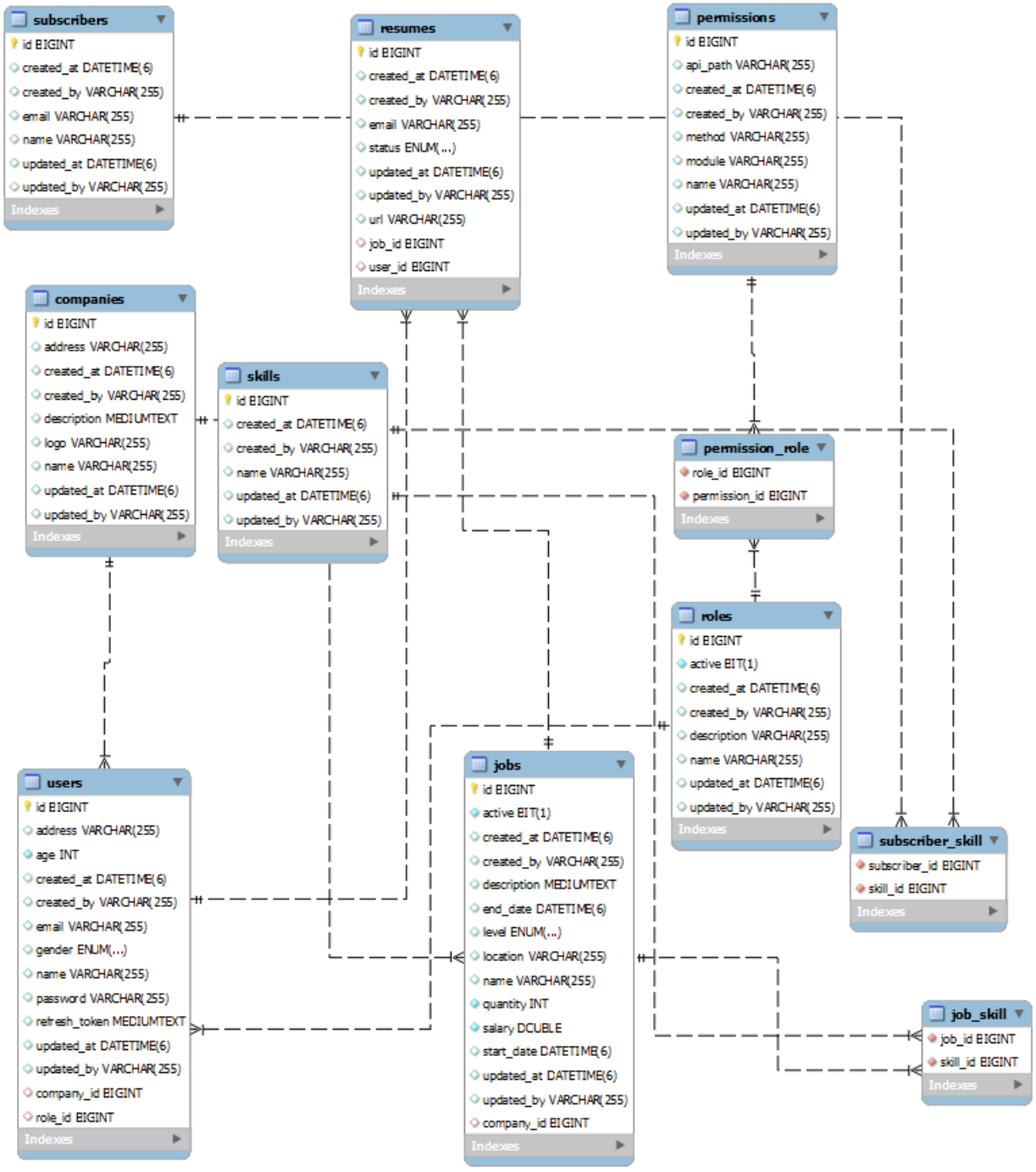
Build Tool: Gradle - Kotlin

Công cụ khác: Postman, Swagger, Lombok, Git

Triển khai

Phân tích thiết kế

ádasd



Các quy tắc viết API

Cần thêm tiền tố để biết được version API hiện tại. Ví dụ: /api/v1/…

Đối với những Entity có quan hệ cha con với nhau, khi viết cần tuân theo thứ tự thực thể cha rồi mới đến thực thể con. Ví dụ: /api/v1/PTIT/B22DCCN482

Tuân thủ RESTful Standards:

Sử dụng các phương thức HTTP chuẩn

Sử dụng đường dẫn URL rõ ràng

Hỗ trợ các trạng thái mã HTTP

Module chính

Modules Company

Model company gồm các trường như hình bên dưới

Id	Long
Name	String
Description	String
Address	String
Logo	String
createdAt	Date
updatedAt	Date
createdBy	String
updatedBy	String

Các API liên quan:

Chức năng	Phương thức	API
Tạo mới 1 công ty	POST	/api/v1/companies
Lấy toàn bộ thông tin công ty	GET	/api/v1/companies
Cập nhật thông tin công ty	PUT	/api/v1/companies
Xóa 1 công ty	DELETE	/api/v1/companies/{id}
Lấy thông tin 1 công ty theo id	GET	/api/v1/companied/{id}

Modules User

Model User gồm các trường thông tin như bên dưới:

Id	Long
Name	String
Email	String
Password	String
Age	Int
Gender	String
Address	String
refreshToken	String
createdAt	Date
updatedAt	Date
createdBy	String
updatedBy	String

Các API liên quan:

Chức năng	Phương thức	API
Tạo mới 1 User	POST	/api/v1/users
Xóa 1 User	DELETE	/api/v1/users/{id}
Lấy thông tin User	GET	/api/v1/users/{id}
Lấy thông tin toàn bộ User	GET	/api/v1/users
Cập nhật User	PUT	/api/v1/users
Lấy account user	GET	/api/v1/auth/account
Lấy được access_token	POST	/api/v1/auth/login
Lấy access_token mới	GET	/api/v1/auth/refresh
Logout	POST	/api/v1/auth/logout

Đối với API tạo mới 1 user, cần kiểm tra email đã tồn tại trong hệ thống chưa. Nếu đã tồn tại thông báo lỗi

```
@PostMapping("/users")
@ApiMessage("Create a new user")
public ResponseEntity<ResCreateUserDTO> createNewUser(@Valid @RequestBody User user) throws IdInvalidException {

    boolean isEmailExist = this.userService.isEmailExist(user.getEmail());
    if (isEmailExist) {
        throw new IdInvalidException("Email " + user.getEmail() + " đã tồn tại trong hệ thống.");
    }
    String hashPassword = this.passwordEncoder.encode(user.getPassword());
```

Đối với API login: Nếu người dùng đăng nhập thành công, backend sẽ làm 2 việc:

Trả về phản hồi cho API, bao gồm access_token và thông tin của user, theo format sau:

{

Access_token: “JSON WEB TOKEN”

User: {

Email:trinhquanglam2k4@gmail.com,

Name:”trinhquanglam”,

Id:”1”}

}

Ngoài ra, refresh_token sẽ lưu vào cookies

Lưu trữ data:

Để lưu trữ Data người dùng có 3 cách phổ biến nhất

Local Storage: thông tin được lưu trữ mãi mãi

Session Storage: khi đóng browser là thông tin lưu trữ sẽ clear

Cookies: chỉ mất khi và chỉ khi “bị hết hạn” (không liên quan gì tới việc đóng browser)

Cơ chế thường dùng trong mô hình stateless :

Người dùng sau khi login thành công, sẽ được server trả về:

Access_token: token để định danh người dùng (thời gian sống ngắn)

Refresh_token: nếu access_token bị hết hạn, sử dụng refresh token để renew (thời gian sống lâu hơn)

Access_token được lưu tại local storage (FE dễ dàng truy cập và sử dụng). Đặt thời gian sống ngắn để giảm thiểu rủi ro

Refresh_token được lưu lại cookies với mục đích là server sử dụng (vì cookies luôn được gửi kèm với mỗi lời gọi request) → lưu ở cookies sẽ an toàn hơn

Giải thích cơ chế JWT:

Do sử dụng mô hình stateless, nên không thể sử dụng session, chúng ta sử dụng token để định danh người dùng

Để truy cập endpoint (APIs) tạo backend, đối với mỗi lời gọi, frontend cần truyền thêm Token (gọi là access_token). Token này sẽ được gán ở header mỗi request (bearer token)

Access_token này có thời gian sống ngắn để đảm bảo an toàn, đồng thời được viết dưới dạng JWT

Ví dụ về access_token:
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJob2lkYW5pdEBnbWFpbC5jb20iLCJleHAiOjE3MjQwNTkzNTEsImlhdCI6MTcxNTQxOTM1MSwidXNlciI6eyJpZCI6MSwiZW1haWwiOiJob2lkYW5pdEBnbWFpbC5jb20iLCJuYW1lIjoiSOG7j2kgRMOibiBJVCJ9fQ.0V3Ss4AWnHf8f6HtSEWKlchHSJAmHW0ef3WUtDj2fn0vGYEjbyqIO2nor0Iq06DTcAOAthT9BBvQ3gD9M0jThw

Modules Job/Resum

Job entity	
Id	**Long**
Name	String
Location	String
Salary	Int
Level	String
Description	String
startDate	Date
endDate	Date
isActive	Boolean
createdAt	Date
updatedAt	Date
createdBy	String
updatedBy	String
Company_id	Long
Skills	Array

Skill entity	
Id	**Long**
Name	String
createdAt	Date
updatedAt	Date
createdBy	String
updatedBy	String

Các API liên quan:

Chức năng	Phương thức	API
Tạo mới 1 skill	POST	/api/v1/skills
Cập nhật 1 skill	PUT	/api/v1/skills
Delete a skill	DELETE	/api/v1/skills/{id}
Lấy thông tin về skill	GET	/api/v1/skills

Chức năng	Phương thức	API
Tạo mới job	POST	/api/v1/jobs
Cập nhật job	PUT	/api/v1/jobs
Xóa job	DELETE	/api/v1/jobs/{id}
Lấy job theo id	GET	/api/v1/jobs/{id}

Lấy thông tin job(phân trang)	GET	/api/v1/jobs

Chức năng upload file:

Có 3 cách phổ biến nhất dùng để upload File:

Sử dụng dịch vụ: AWS S3

Lưu vào database (Blob)

Lưu file vào server

Trong bài này sẽ sử dụng cách thứ 3. Trước tiên ta cần tìm hiểu các khái niệm hay dùng khi làm việc với chức năng này:

MultipartFile: tượng trưng cho file gửi từ client lên server, thay vì gửi text chúng ta sử dụng binary

Path: tượng trưng cho địa chỉ lưu trữ file trên máy tính

Tiếp đến cần cấu hình để hệ thống có thể lưu file từ local. Cấu hình base path:

```
# base path
trinhlam.upload-file.base-uri=file:///C:/Users/Lenovo/Documents/upload/
```

Toàn bộ quá trình xảy ra khi người dùng upload file:

Chức năng	Phương thức	API
Upload single file	POST	/api/v1/files

Bước 1: Frontend gửi lên file và tên folder lưu trữ(cần truyền vào file và folder)

File: file cần upload

Folder: tên thư mục upload được phân theo từng tính năng

Lưu ý: Sử dụng form-data thay vì json

Bước 2: Backend lấy thông tin gửi lên

```
@PostMapping("/files")
@ApiMessage("upload single file success")
public ResponseEntity<ResUploadFileDTO> uploadFile(@RequestParam("file") MultipartFile file,
        @RequestParam(name = "folder", required = false) String folder)
        throws URISyntaxException, IOException, StorageException {
```

Bước 3: Tạo mới folder để lưu trữ nếu nó không tồn tại

Nếu folder không tồn tại thì cần tạo để lưu file

```java
public void createUploadFolder(String folder) throws URISyntaxException {
    URI uri = new URI(folder);
    Path path = Paths.get(uri);
    File tmpDir = new File(path.toString());
    if (!tmpDir.isDirectory()) {
        try {
            Files.createDirectory(tmpDir.toPath());
            System.out.println(">>> CREATE NEW DIRECTORY SUCCESSFUL, PATH = " + tmpDir.toPath());
        } catch (IOException e) {
            e.printStackTrace();
        }
    } else {
        System.out.println(x:">>> SKIP MAKING DIRECTORY, ALREADY EXISTS");
    }
}
```

Bước 4: Lưu file vào folder

Lưu file vào folder

```java
public String store(MultipartFile file, String folder) throws URISyntaxException, IOException {
    // create unique filename
    String finalName = System.currentTimeMillis() + "-" + file.getOriginalFilename();

    URI uri = new URI(basePath + folder + "/" + finalName);
    Path path = Paths.get(uri);
    try (InputStream inputStream = file.getInputStream()) {
        Files.copy(inputStream, path,
                StandardCopyOption.REPLACE_EXISTING);

    }
    return finalName;
}
```

Bước 5: Trả về phản hồi với file name

Bước 6: validate file upload

File trống, File vượt quá dung lượng, File không đúng định dạng

```java
@PostMapping("/files")
@ApiMessage("upload single file success")
public ResponseEntity<ResUploadFileDTO> uploadFile(@RequestParam("file") MultipartFile file,
        @RequestParam(name = "folder", required = false) String folder)
        throws URISyntaxException, IOException, StorageException {

    // validate
    if (file == null || file.isEmpty()) {
        throw new StorageException(message:"file is empty. Please upload file");
    }
    String fileName = file.getOriginalFilename();
    List<String> allowedExtensions = Arrays.asList(...a:"pdf", "jpg", "jpeg", "png", "doc", "docx"
    boolean isValid = allowedExtensions.stream().anyMatch(item -> fileName.toLowerCase().endsWith(
    if (isValid == false) {
        throw new StorageException("Invalid file extension only about: " + allowedExtensions.toStr
    }
    // create a dir if not exist
    this.fileService.createUploadFolder(baseUri + folder);
    // save file
    String uploadedFile = this.fileService.store(file, folder);

    return ResponseEntity.ok().body(new ResUploadFileDTO(uploadedFile, Instant.now()));
}
```

Resume entity	
Id	**Long**
Email	String
url	String
Static	Enum
createdAt	Date
UpdatedAt	Date
createdBy	String
updatedBy	String
User_id	
Job_id	

Các API liên quan:

Chức năng	Phương thức	API
Tạo mới 1 resumes	POST	/api/v1/resumes
Cập nhật hồ sơ	PUT	/api/v1/resumes
Xóa hồ sơ theo ID	DELETE	/api/v1/resumes/{id}
Lấy thông tin hồ sơ theo ID	GET	/api/v1/resumes/{id}
Lấy toàn bộ hồ sơ(phân trang)	GET	/api/v1/resumes
Lấy hồ sơ theo user	POST	/api/v1/resumes/by-user

Modules Permission & Role

Permissions	
Id	Long
Name	String
apiPath	String
Method	String
Module	String
createdAt	Date
updatedAt	Date
createdBy	String
updatedBy	String

Roles	
Id	Long
Name	String
Description	String
Active	Boolean
createdAt	Date
createdBy	String
updatedAt	Date
updatedBy	String
Permissions	Array

Các API liên quan:

Permissions		
Chức năng	Phương thức	API
Tạo mới permission	POST	/api/v1/permissions
Cập nhật permission	PUT	/api/v1/permissions
Xóa permission	DELETE	/api/v1/permissions/{id}
Lấy thông tin permission	GET	/api/v1/permission

Roles		
Chức năng	Phương thức	API
Tạo mới roles	POST	/api/v1/roles
Cập nhật roles	PUT	/api/v1/roles
Xóa roles theo ID	DELETE	/api/v1/roles/{id}
Lấy thông tin roles	GET	/api/v1/roles
Lấy thông tin roles theo ID	GET	/api/v1/roles/{id}

Cách xử lý phân quyền tại Frontend:

Để có thể hide/show giao diện ứng với phân quyền, frontend sẽ cần biết Role (vai trò) và Permission (quyền hạn) của người dùng đăng nhập

Khi login/refresh token/getAccount(F5), backend cần trả ra role và permission cho frontend

Tại giao diện frontend, thực chất là viết if/else để render giao diện on/off với tham số:
VITE_ACL_ENABLE: true/false

Các keywords để xử lý phân quyền phía frontend:

Lưu trữ quyền hạn mà backend trả về (role/permission) ứng với người dùng đăng nhập. Với source code cung cấp, data được lưu tại redux

Khai báo list permission tại Frontend, file permission.ts

Viết component access.tsx

(component trên là component cha, bọc ngoài các component con “cần check quyền hạn)

Interceptor:

Mặc định, với 1 lời gọi request từ client gửi lên:

Request ➔ Spring Security (Filter chain) ➔ Controller ➔ Service..

Do chúng ta không sửa Spring Security ➔ sẽ can thiệp vào Controller

Can thiệp vào request sau khi đã qua Spring Security và trước khi gọi tới Controller cần sử dụng interceptor

Mô hình sau khi đã cấu hình lại:

Request ➔ Spring Security ➔ Interceptor ➔ Controller ➔ Service..

Ý tưởng:

Mỗi lời gọi request đều kèm theo JWT (access token) ➔ chúng ta biết được ai đang đăng nhập (email) ➔ biết được user đấy có quyền hạn gì

Check target controller và permission user có. Nếu tồn tại, cho request đi tiếp, còn ngược lại, ném ra exception

Các bước cấu hình:

Bước 1: Khai báo interceptor

```java
public class PermissionInterceptor implements HandlerInterceptor {
    @Autowired
    UserService userService;

    @Override
    @Transactional
    public boolean preHandle(
            HttpServletRequest request,
            HttpServletResponse response, Object handler)
            throws Exception, IdInvalidException {

        String path = (String) request.getAttribute(HandlerMapping.BEST_MATCHING_PATTERN_ATTRIBUTE);
        String requestURI = request.getRequestURI();
        String httpMethod = request.getMethod();
        System.out.println(x:">>> RUN preHandle");
        System.out.println(">>> path= " + path);
        System.out.println(">>> httpMethod= " + httpMethod);
        System.out.println(">>> requestURI= " + requestURI);
```

```java
@Configuration
public class PermissionInterceptorConfiguration implements WebMvcConfigurer {
    @Bean
    PermissionInterceptor getPermissionInterceptor() {
        return new PermissionInterceptor();
    }

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        String[] whiteList = {
                "/", "/api/v1/auth/**", "/storage/**",
                "/api/v1/companies/**", "/api/v1/jobs/**", "/api/v1/skills/**", "/api/v1/files",
                "/api/v1/resumes/**", "/api/v1/subscribers/**"
        };
        registry.addInterceptor(getPermissionInterceptor())
                .excludePathPatterns(whiteList);
    }
}
```

Bước 2: Check permission

Request gửi kèm JWT(access_token) ➔ spring giải mã token, và lưu thông tin vào Security Context

Trước khi tới interceptor, chúng ta đã biết được email của User

Query user theo email ➔ lấy role ➔ lấy permission

Modules Subscriber

Subcriber

Long	Id
String	Name
String	Email
Instant	createdAt
Instant	updatedAt
String	createdBy
String	updatedBy

Các API liên quan:

Chức năng	Phương thức	API
Tạo 1 subcriber	POST	/api/v1/subscribers
Cập nhật 1 subcriber	PUT	/api/v1/subcribers
Lấy thông tin kỹ năng của user đã đăng ký	POST	/api/v1/subscribers/skills

Không làm API fetch data và xóa, vì không làm tại admin

Chức năng gửi mail:

Cài đặt thư viện

Tạo app password với Gmail (tài khoản Gmail cần bật xác thực 2 lớp)

Cấu hình tham số môi trường:

spring.mail.host=smtp.gmail.com

spring.mail.port=587

spring.mail.username=your-email

spring.mail.password=your-gmail-app-password

spring.mail.properties.mail.smtp.auth=true

spring.mail.properties.mail.smtp.starttls.enable=true

Mô hình gửi email diễn ra như sau:

Client gửi request lên server Java spring → Gọi server gmail → gửi email tới client

Vấn đề đặt ra bây giờ là: Khi gửi email với text đơn thuần thường có giao diện không đẹp mắt. Chúng ta muốn có màu sắc, và với giao diện web chúng ta cần có CSS. Đồng thời chúng ta muốn tái sử dụng "format gửi email" để gửi cùng lúc cho nhiều khách hàng khác nhau → cần đến template engine. Trong bài này sẽ sử dụng **thymleaf**

Các bước cấu hình :

Gửi email với html:

```
public void sendEmailSync(String to, String subject, String content, boolean isMultipart, boolean isHtml) {
    // Prepare message using a Spring helper
    MimeMessage mimeMessage = this.javaMailSender.createMimeMessage();
    try {
        MimeMessageHelper message = new MimeMessageHelper(mimeMessage, isMultipart, StandardCharsets.UTF_8.name());
        message.setTo(to);
        message.setSubject(subject);
        message.setText(content, isHtml);
        this.javaMailSender.send(mimeMessage);
    } catch (MailException | MessagingException e) {
        System.out.println("ERROR SEND EMAIL: " + e);
    }
}
```

Gửi email với template:

```
@Async
public void sendEmailFromTemplateSync(String to, String subject, String templateName, String username,
        Object value) {

    Context context = new Context();
    context.setVariable(name:"name", username);
    context.setVariable(name:"jobs", value);
    String content = this.templateEngine.process(templateName, context);
    this.sendEmailSync(to, subject, content, isMultipart:false, isHtml:true);

}
```

Gửi mail tự động với Cron Job:

Cron là cách chúng ta đặt lịch để tự động làm một cộng việc gì đấy

Bước 1: enabled:

@EnableScheduling

```
@SpringBootApplication
@EnableAsync
@EnableScheduling
public class JobhunterApplication {

    Run | Debug
    public static void main(String[] args) {
        SpringApplication.run(primarySource:JobhunterApplication.class, args);
    }

}
```

Bước 2: sử dụng schedule với cron

@Scheduled(fixedRate = 5000)

Kết quả

Phụ lục 1: Hướng dẫn cài đặt và chạy ứng dụng

Tài liệu tham khảo