**Nguyên tắc đặt api khi có quan hệ lồng nhau:**

Nên giữ URI ở mức collection/item/collection và không nên sâu hơn.

VD:

GET /companies (list cty)

POST /companies (tạo mới cty)

GET /companies/{companyId} (chi tiết 1 cty)

GET /companies/{companyId}/departments (list phòng ban của 1 cty cụ thể)

Hoặc có thể dùng query param (FromQuery): GET /departments?companyId=123

POST /companies/{companyId}/departments (tạo 1 phòng ban cho 1 công ty cụ thể)

/departments/{departmentId} (cập nhật 1 phòng ban)

GET /departments/{departmentId} (chi tiết 1 phòng ban)

GET /departments/{departmentId}/employees (list nhân viên của 1 phòng ban)

GET /employees/{employeeId} (chi tiết 1 nhân viên)

🡪  **Sử dụng query parameters khi cần thiết**: Để truy vấn phức tạp hoặc kết hợp, sử dụng query parameters giúp giữ đường dẫn ngắn gọn và dễ hiểu.

 **Hỗ trợ cả hai loại đường dẫn**: Sử dụng đường dẫn lồng nhau cho các thao tác tạo mới và đường dẫn không lồng nhau cho các thao tác truy xuất, cập nhật hoặc xóa.

VD code:

[ApiController]

[Route("companies")]

public class CompaniesController : ControllerBase

{

private readonly CompanyService \_companyService;

private readonly DepartmentService \_departmentService;

public CompaniesController(CompanyService companyService, DepartmentService departmentService)

{

\_companyService = companyService;

\_departmentService = departmentService;

}

[HttpGet]

public IActionResult GetCompanies() => Ok(\_companyService.GetCompanies());

[HttpPost]

public IActionResult CreateCompany([FromBody] Company company)

{

\_companyService.AddCompany(company);

return CreatedAtAction(nameof(GetCompany), new { companyId = company.Id }, company);

}

[HttpGet("{companyId}")]

public IActionResult GetCompany(int companyId)

{

var company = \_companyService.GetCompany(companyId);

if (company == null) return NotFound();

return Ok(company);

}

[HttpGet("{companyId}/departments")]

public IActionResult GetDepartments(int companyId)

{

var company = \_companyService.GetCompany(companyId);

if (company == null) return NotFound();

var departments = \_departmentService.GetDepartments(companyId);

return Ok(departments);

}

[HttpPost("{companyId}/departments")]

public IActionResult CreateDepartment(int companyId, [FromBody] Department department)

{

var company = \_companyService.GetCompany(companyId);

if (company == null) return NotFound();

department.CompanyId = companyId;

\_departmentService.AddDepartment(department);

return CreatedAtAction(nameof(GetDepartment), new { departmentId = department.Id }, department);

}

}

**Middleware là các thành phần trung gian mà các yêu cầu HTTP đều phải đi qua khi muốn gửi request đến ứng dụng**

Middleware có thể xử lý yêu cầu, tạo ra phản hồi, hoặc thậm chí là chuyển tiếp yêu cầu đến các middleware khác

**Redirect và Rewriting Middleware là hai loại middleware thường được sử dụng để điều hướng yêu cầu hoặc thay đổi các địa chỉ URL.**

**Redirect Middleware:** được sử dụng để chuyển hướng các yêu cầu từ một URL đến một URL khác. Điều này thường được sử dụng khi bạn muốn chuyển hướng người dùng từ một địa chỉ URL cũ đến một địa chỉ URL mới.

VD: khi ta thay đổi cấu trúc của trang web và muốn chuyển hướng từ các URL cũ sang các URL mới, bạn có thể sử dụng Redirect Middleware để thực hiện điều này (thực hiện điều hướng url cũ phía client thành url mới)

var options = new RewriteOptions()

.AddRedirect("^product/detail\\?id=(\\d+)&name=(.\*)$", "products/$1/$2", permanent: true)

; app.UseRewriter(options);

**Rewriting Middleware**: Middleware này cho phép bạn thay đổi các yêu cầu và/hoặc phản hồi HTTP dựa trên các quy tắc được cấu hình trước. (tức là sẽ chỉ thay đổi cách mà ứng dụng xử lý yêu cầu nội bộ mà không thay đổi url. Request = url cũ nhưng mình muốn thực hiện request đó dưới dạng url mới nhưng vẫn giữ url cũ trên trình duyệt client)

VD: var options = new RewriteOptions()

.AddRewrite("^products/books/(\\d+)$", "books/$1", true); // khi gọi api bên trái, 🡪 chuyển thành api bên phải để thực hiện yêu cầu

app.UseRewriter(options);

UnitOfWork: Trên tầng service có thể 1 lúc thực hiện nhiều công việc ví dụ gọi phương thức book.create() để lưu book, sau đó lại gọi bookimage.create() để lưu id của book vào book images. Nếu mỗi lần gọi savechange trực tiếp trong create book thì sẽ mở kết nối 1 lần, sau đó ta lại gọi create trong book image sẽ mở kết nối lần nữa 🡪 mở đóng kết nối liên tục dẫn đến ảnh hưởng performance.

🡺 Unit ofword giúp ta group những repository vào 1 file, đồng nhất việc thực thi nhiều việc trên 1 transaction của db. (tức là ta sẽ gọi create của book, sau đó gọi create của bookimage, cuối cùng gọi save change của unitofwork để mở kết nối 1 lần và thực thi những lệnh trên vào db. )