



RESTful API

ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

CYBERSOFT.EDU.VN



Nội dung

- ☐ @PathVariable, @RequestParam.
- ☐ @RequestBody, @ResponseBody
- ☐ @RestController.
- ☐ JSON là gì?
- ☐ Thư viện Jackson.
- ☐ API là gì?
- ☐ RESTfull API là gì?
- ☐ HttpStatus trong Restful Api.
- ☐ Cấu hình Swagger hỗ trợ API.

CYBERSOFT

ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

@PathVariable

- ❑ Annotation **@PathVariable** được sử dụng để xử lý những URI động có một hoặc nhiều parameter bên trong URI.

localhost:8080/BaiTap01/account/test2/acc001/Nguyen-Van-Teo

```
@Controller
@RequestMapping("/account")
public class AccountController {

    @RequestMapping("/test1/{id}")
    public String test1(@PathVariable("id") int id, ModelMap model) {
        model.addAttribute("id", id);
        return "test1";
    }

    @RequestMapping("/test2/{id}/{name}")
    public String test2(@PathVariable("id") int id,
        @PathVariable("name") String name, ModelMap model) {
        model.addAttribute("id", id);
        model.addAttribute("name", name);
        return "test2";
    }
}
```

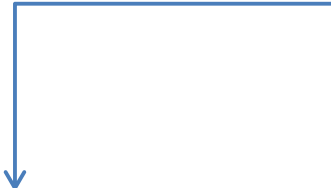
@RequestParam

- ❑ Annotation **@RequestParam** giúp lấy giá trị khi submit dữ liệu từ form hoặc theo cả hai kiểu **GET** và **POST**.
- ❑ **@RequestParam** cũng có thể được sử dụng để lấy giá trị từ url.

localhost:8080/BaiTap01/account?username=admin&password=123456

```
@Controller
@RequestMapping("/account")
public class AccountController {

    @RequestMapping(value = "login")
    public String test1(
        @RequestParam("username") String username,
        @RequestParam("password") String password) {
        //...
        return "test1";
    }
}
```



@RequestParam

- ❑ **@RequestParam**(value, defaultValue, required) là dạng đầy đủ với ý nghĩa của các tham số:
 - ✓ **value**: chỉ ra tên tham số muốn nhận
 - ✓ **defaultValue**: là giá trị mặc định của tham số khi tham số không tồn tại
 - ✓ **required**: tham số có bắt buộc hay không.
- ❖ **Ví dụ:**
@RequestParam(value="tuoi", defaultValue="20", required=false) **Integer age**
 - Tên tham số là **tuoi** sẽ được nhận vào đối số là **age**
 - Nếu không có tham số thì giá trị của **age** là 20
 - Tham số **tuoi** là không bắt buộc

Produces & Consumes

- ❑ Chỉ định kiểu dữ liệu request và response.
- ✓ **consumes**: chỉ chấp nhận các **request** gửi lên có kiểu dữ liệu giống với giá trị khai báo bên trong **consumes**.
- ✓ **produces**: Chỉ định kiểu dữ liệu server trả về cho client.

```
@Controller
@RequestMapping("/account")
public class AccountController {

    @RequestMapping(value= "/login",
        method = RequestMethod.GET,
        consumes = "application/json",
        produces = "application/json")
    public String index() {
        return "home";
    }
}
```

ResponseBody

- ❑ **@ResponseBody** được thêm vào trước các method của các controller để **chỉ dẫn rằng method này sẽ trả về text thay vì trả về view.**

```
@Controller
public class StudentController {

    @GetMapping("/")
    @ResponseBody
    public String index() {

        return "Xin chào!";
    }
}
```


ResponseBody

- ❑ Nếu muốn trả về kiểu Json thì cần thêm thư viện Jackson vào.

Thư viện

```
<!-- jackson-core -->
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-core</artifactId>
  <version>2.9.8</version>
</dependency>
<!-- jackson-databind -->
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>2.9.8</version>
</dependency>
```

Thư viện Jackson tự động chuyển dữ liệu về dạng JSON.

Controller

```
@Controller
public class StudentController {

    @GetMapping("/")
    @ResponseBody
    public Object index() {
        List<Student> students = new ArrayList<Student>();
        students.add(new Student("sv001", "Trần Văn Tâm", 22));
        students.add(new Student("sv002", "Lê Mạnh Cường", 23));
        students.add(new Student("sv003", "Nguyễn Hà My", 20));

        return students;
    }
}
```


@RequestBody

- ❑ **@RequestBody** sử dụng để **lấy các thông tin nằm trong request body** được gửi từ Client gửi lên Server.
- ❑ **@RequestBody** tự động **chuyển chuỗi JSON trong request thành một Object Java**.

```
List<Student> list = new ArrayList<Student>();  
  
@PostMapping("/student")  
public Object addStudent(@RequestBody Student student) {  
    list.add(student);  
    // Body sẽ chứa thông tin về danh đối tượng student vừa được tạo.  
    return student;  
}
```

@RestController

- ❑ @RestController tương đương với @Controller + @ResponseBody
- ❑ @RestController được dùng trước các class, các method trong class này sẽ trả về text thay vì trả về view.

```
@RestController
public class StudentController {

    @GetMapping("/")
    public String index() {

        return "Xin chào!";
    }
}
```

- ❑ **JSON** (JavaScript Object Notation) là một **định dạng để lưu trữ và vận chuyển dữ liệu**.
- ❑ Định dạng JSON có nguồn gốc từ cú pháp đối tượng của Javascript vì vậy nó thừa kế sự **đơn giản và hoàn toàn dựa trên văn bản**.
- ❑ **Ưu điểm của JSON:**
 - ✓ JSON là một định dạng để trao đổi dữ liệu gọn nhẹ (Lightweight).
 - ✓ Dữ liệu JSON tự mô tả chính nó, vì vậy nó dễ hiểu cho tất cả mọi người.
 - ✓ JSON là một ngôn ngữ độc lập, và là một văn bản. Bạn có thể sử dụng một ngôn ngữ bất kỳ để đọc hoặc tạo ra dữ liệu JSON.
 - ✓ Hầu hết các ngôn ngữ lập trình đều có thư viện đọc và ghi dữ liệu JSON.

Cú pháp JSON

- ❑ Cú pháp của JSON rất đơn giản là mỗi thông tin dữ liệu sẽ có 2 phần đó là key và value.
- ✓ Chuỗi JSON được bao lại bởi dấu ngoặc nhọn {}.
- ✓ Các key, value của JSON bắt buộc phải đặt trong dấu nháy kép "".
- ✓ Nếu có nhiều dữ liệu (nhiều cặp key, value) thì ta dùng dấu phẩy (,) để ngăn cách.
- ✓ Các key của JSON bạn nên đặt chữ cái không dấu hoặc số, dấu _ và không có khoảng trắng., ký tự đầu tiên không nên đặt là số.

```
{  
  "name": "Amazon",  
  "ceo": "Jeff Bezos",  
  "employees": [  
    {"firstName": "John", "lastName": "Doe"},  
    {"firstName": "Anna", "lastName": "Smith"},  
    {"firstName": "Peter", "lastName": "Jones"}  
  ]  
}
```

Thư viện Jackson

- ❑ **Spring** cung cấp thư viện **JACKSON** hỗ trợ chuyển đổi các kiểu dữ liệu khác thành kiểu dữ liệu JSON.

```
<!-- jackson-core -->
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-core</artifactId>
  <version>2.9.8</version>
</dependency>
<!-- jackson-databind -->
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>2.9.8</version>
</dependency>
```

OFT
LẬP TRÌNH

Thư viện Jackson

```
Student student = new Student();  
student.setName("Cybersoft");  
student.setAge(30);
```

Chuyển Object thành JSON

```
ObjectMapper om = new ObjectMapper();  
String json = om.writeValueAsString(student);  
System.out.println(json);
```

Chuyển JSON thành Object

```
ObjectMapper om = new ObjectMapper();  
Student obj = om.readValue(json, Student.class);  
System.out.println(json);
```

- ❑ Xây dựng Student Controller với các yêu cầu sau:
 - ✓ Tạo đối tượng Student với 2 thuộc tính **name** và **age**.
 - ✓ Tạo list để chứa danh sách Student.
 - ✓ Viết chức phương thức thêm mới student sử dụng @RequestParam.
 - ✓ Viết chức phương thức thêm mới student sử dụng @PathVariable.
 - ✓ Viết chức phương thức thêm mới student sử dụng @RequestBody.
 - ✓ Tất cả 3 phương thức trên đều trả về danh sách Student dạng JSON.

Bài tập

❑ Sử dụng @RequestParam

```
@RequestMapping(value = "student", method = RequestMethod.GET)
@ResponseBody
public Object index(@RequestParam String name, @RequestParam int age) {

    list.add(new Student(name, age));

    return list;
}
```

❑ Sử dụng @PathVariable

```
@RequestMapping(value = "student/{name}/{age}", method = RequestMethod.GET)
@ResponseBody
public Object index2(@PathVariable("name") String name, @PathVariable("age") int age) {

    list.add(new Student(name, age));
    return list;
}
```

Bài tập

❑ Sử dụng @RequestBody

```
@RequestMapping(value = "student", method = RequestMethod.POST)
@ResponseBody
public Object index(@RequestBody Student student) {

    Student entity = new Student();
    entity.setName(student.getName());
    entity.setAge(student.getAge());

    list.add(entity);
    return list;
}
```

API là gì?

- ❑ **API** (Application Programming Interface) là một tập các quy tắc, cơ chế để một ứng dụng sẽ tương tác với một ứng dụng khác hay dễ hiểu hơn API là một phần mềm trung gian cho phép 2 ứng dụng có thể giao tiếp với nhau.
- ❑ **API** có thể trả về dữ liệu mà bạn cần cho ứng dụng của mình ở những kiểu dữ liệu phổ biến như JSON hay XML.
- ❑ **API** có khá nhiều mục đích nhưng mục đích chính của API là cung cấp khả năng truy xuất đến một hàm hay tệp hay dùng.
- ❖ **Ví dụ:** Các lập trình viên của facebook sẽ đưa ra các thư viện có chứa các hàm post, like, share... để cho các lập trình viên khác khai thác, đó chính là các **API**.

RESTful là gì?

- ❑ **REST** (Representtational State Transfer) lần đầu tiên được **giới thiệu vào năm 2000** trong luận văn tiến sĩ của Roy Thomas Fielding (đồng sáng lập giao thức HTTP).
- ❑ **REST** là một **bộ quy tắc để tạo ra một ứng dụng Web Service**, chỉ cần đảm bảo những điều đó hệ thống của bạn có thể được gọi là **RESTful**.
- ❑ **REST** tuân thủ **4 nguyên tắc thiết kế cơ bản** sau:
 - ✓ Sử dụng các phương thức HTTP một cách rõ ràng.
 - ✓ Phi trạng thái.
 - ✓ Hiển thị cấu trúc thư mục như các Urls.
 - ✓ Truyền tải JavaScript Object Notation (JSON), XML hoặc cả hai.

❑ Sử dụng các phương thức HTTP một cách rõ ràng

❖ REST đặt ra một **quy tắc đòi hỏi lập trình viên xác định rõ ý định của mình thông qua các phương thức của HTTP.**

- ✓ Để tạo một tài nguyên trên máy chủ, bạn cần sử dụng phương thức POST.
- ✓ Để truy xuất một tài nguyên, sử dụng GET.
- ✓ Để thay đổi trạng thái một tài nguyên hoặc để cập nhật nó, sử dụng PUT.
- ✓ Để huỷ bỏ hoặc xoá một tài nguyên, sử dụng DELETE.

❑ Phi trạng thái (Stateless)

- ✓ REST là phi trạng thái (stateless), nó **không lưu giữ thông tin của client, điều đó có nghĩa là REST không quản lý phiên làm việc (Session).**

❑ Hiện thị cấu trúc thư mục như các Urls

- ✓ Các địa chỉ **REST service** cần phải thật **trực quan đến mức người dùng dễ đoán**.
- ❖ Một vài **nguyên tắc lưu ý khi cấu trúc địa chỉ của RESTful**:
 - ✓ Giấu các đuôi tài liệu mở rộng của bản gốc trong máy chủ (.jsp, .php,...).
 - ✓ Để mọi thứ là chữ thường.
 - ✓ Thay thế các khoảng trống bằng gạch chân hoặc gạch nối (một trong hai loại).
 - ✓ Thay vì sử dụng mã (404 Not Found) khi yêu cầu địa chỉ cho một phần đường dẫn, luôn luôn cung cấp một trang mặc định hoặc tài nguyên như một phản hồi.

❑ Bốn phương thức chính của HTTP:

- ✓ **GET**: Dùng khi lấy dữ liệu – Mã trả về có thể là **200** (OK) + **404** (NOT FOUND) + **400** (BAD REQUEST).
- ✓ **POST**: Dùng khi tạo mới dữ liệu – Mã trả về thường là **201** (CREATED).
- ✓ **PUT**: Dùng khi cập nhật dữ liệu đã tồn tại – Mã code trả về thường là **200** (OK).
- ✓ **DELETE**: Dùng khi xóa dữ liệu đã tồn tại – Mã code trả về thường là **200** (OK).

Http Status

❑ **Http** định nghĩa một số mã phản hồi cơ bản:

- ✓ **200 - OK** : Trả về thành công cho những phương thức GET, PUT hoặc DELETE.
- ✓ **201 - Created** : Trả về khi một Resource vừa được tạo thành công.
- ✓ **204 - No Content** : Trả về khi Resource xóa thành công.
- ✓ **304 - Not Modified** : Client có thể sử dụng dữ liệu cache.
- ✓ **400 - Bad Request** : Request không hợp lệ
- ✓ **401 - Unauthorized** : Request cần có auth.
- ✓ **403 - Forbidden** : bị từ chối không cho phép.

Http Status

- ✓ **404 - Not Found** : Không tìm thấy resource từ URI
- ✓ **405 - Method Not Allowed** : Phương thức không cho phép với user hiện tại.
- ✓ **410** - Resource không còn tồn tại, Version cũ đã không còn hỗ trợ.
- ✓ **415 - Unsupported Media Type** : Không hỗ trợ kiểu Resource này.
- ✓ **422 - Unprocessable Entity** : Dữ liệu không được xác thực
- ✓ **429 - Too Many Requests** : Request bị từ chối do bị giới hạn

ResponseEntity

- ❑ **ResponseEntity** là đối tượng được Spring cung cấp để đóng gói toàn bộ phản hồi HTTP: mã trạng thái, tiêu đề và nội dung. Do đó, chúng ta có thể sử dụng nó để định cấu hình đầy đủ phản hồi HTTP.

```
@RequestMapping(value = "/products", method = RequestMethod.GET)
public ResponseEntity<List<Product>> findAllProduct() {
    List<Product> products = productService.findAllProduct();
    if (products.isEmpty()) {
        return new ResponseEntity<>(HttpStatus.NO_CONTENT);
    }
    return new ResponseEntity<>(products, HttpStatus.OK);
}
```

Api trả về danh sách

```
@GetMapping(value = "/products")
public ResponseEntity<List<Product>> get() {
    List<Product> products = productService.findAll();
    if (products.isEmpty()) {
        return new ResponseEntity<>(HttpStatus.NOT_FOUND);
    }
    return new ResponseEntity<>(products, HttpStatus.OK);
}
```

Api trả về đối tượng

```
@GetMapping(value = "/products/{id}")
public ResponseEntity<Product> get(@PathVariable("id") Integer id) {

    Product product = productService.findById(id);
    if (product == null) {
        return new ResponseEntity<>(HttpStatus.NOT_FOUND);
    }
    return new ResponseEntity<>(product, HttpStatus.OK);
}
```

Api thêm mới

```
@PostMapping(value = "/products/post")
@RequestBody
public ResponseEntity<Product> post(
    @Valid @ModelAttribute("product") Product product,
    BindingResult errors) {
    if(errors.hasError()){
        return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
    }
    productService.save(product);
    return new ResponseEntity<>(product, HttpStatus.CREATED);
}
```

Api cập nhật

```
@PutMapping(value = "/products/put")
@RequestBody
public ResponseEntity<Product> put(
    @Valid @ModelAttribute("product") Product product,
    BindingResult errors) {
    if(errors.hasError()){
        return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
    }
    productService.update(product);
    return new ResponseEntity<>(HttpStatus.OK);
}
```

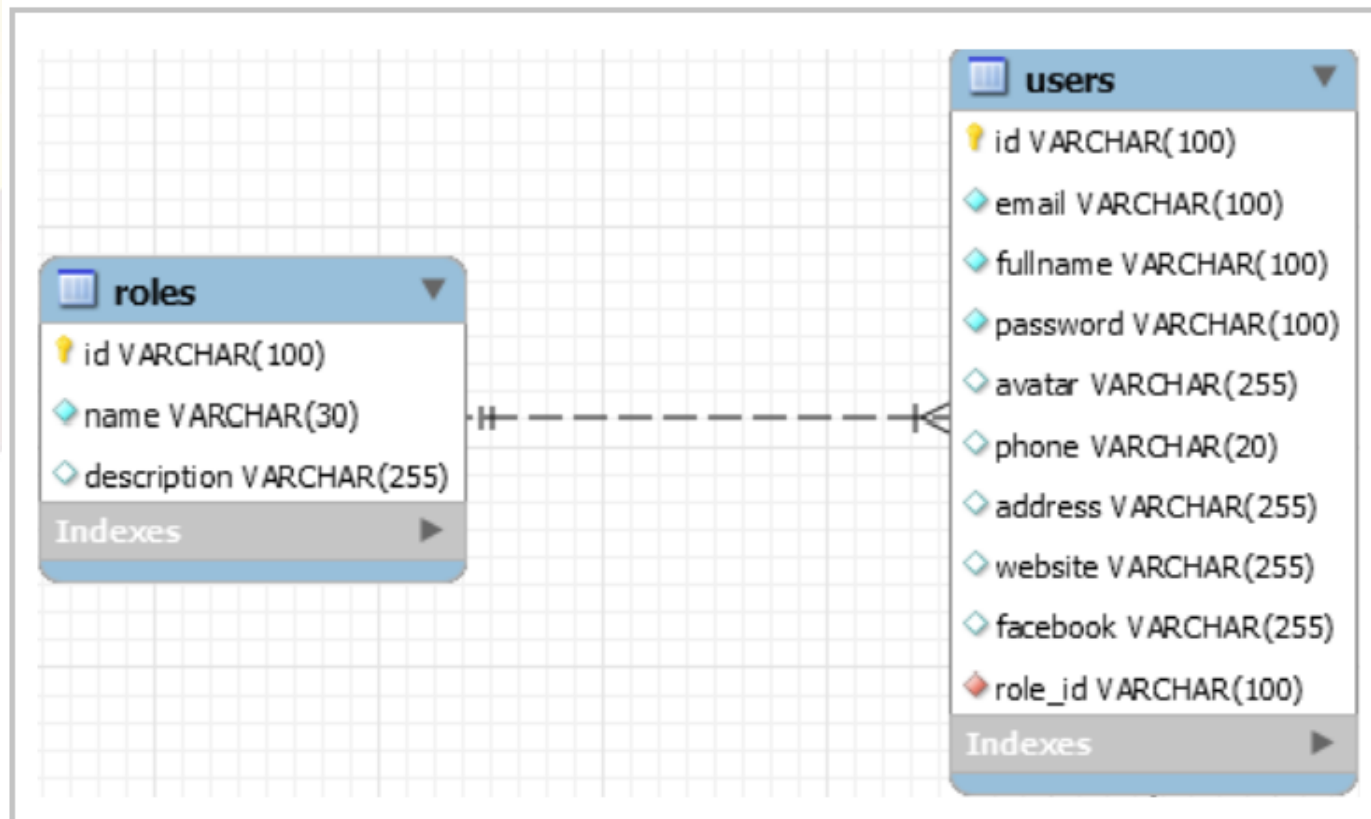

Api xóa

```
@DeleteMapping(value = "/products/{id}")  
public ResponseEntity<Product> delete(@PathVariable("id") Integer id) {  
    productService.removeById(id);  
    return new ResponseEntity<>(HttpStatus.OK);  
}
```

Bài tập

❑ Xây dựng API quản lý User và quản lý Role (CRUD).

✓ Role entity và User entity là các POJO class có các thuộc tính như hình bên dưới.



FT
RINH

Swagger

- ❑ **Swagger** là một phần mềm mã nguồn mở được sử dụng để **phát triển, thiết kế, xây dựng** và **làm tài liệu cho các RESTful Web Service**.
- ❑ Có một số phần mềm **Swagger** như Swagger Editor, Swagger Codegen, Swagger Inspector, Swagger UI.
- ❑ **Swagger UI** được sử dụng nhiều nhất, nó hỗ trợ tự động làm tài liệu, sinh code và sinh test case.
- ❑ Link demo với Swagger UI: <http://petstore.swagger.io>

Swagger

pet Everything about your Pets

Find out more: <http://swagger.io>



POST

/pet Add a new pet to the store



PUT

/pet Update an existing pet



GET

/pet/findByStatus Finds Pets by status



GET

/pet/findByTags Finds Pets by tags



GET

/pet/{petId} Find pet by ID



POST

/pet/{petId} Updates a pet in the store with form data



DELETE

/pet/{petId} Deletes a pet



POST

/pet/{petId}/uploadImage uploads an image



store Access to Petstore orders



GET

/store/inventory Returns pet inventories by status



Cài đặt Swagger

❑ Thư viện sử dụng



```
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger2</artifactId>
  <version>2.8.0</version>
</dependency>
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger-ui</artifactId>
  <version>2.8.0</version>
</dependency>
```

- ❖ Thư viện **springfox-swagger-ui** giúp bạn nhúng sẵn phần mềm **swagger-ui** vào project, phần mềm **swagger ui** sẽ được **start cùng project**.

Cấu hình Swagger

❑ Tạo file **SwaggerConfig** để cấu hình Swagger (package: **com.myclass.config**)

```
@Configuration
@EnableSwagger2
public class SwaggerConfig {
    @Bean
    public Docket api() {
        return new Docket(DocumentationType.SWAGGER_2)
            .select()
            .apis(RequestHandlerSelectors.basePackage("com.myclass.controller"))
            .build()
            .apiInfo(apiInfo());
    }

    private ApiInfo apiInfo() {
        return new ApiInfo(
            "My REST API",
            "Some custom description of API.",
            "API TOS",
            "Terms of service",
            new Contact("NGUYỄN TIẾN HOÀNG", "www.tienhoang.com", "tienhoang@gmail.com"),
            "License of API", "API license URL", Collections.emptyList());
    }
}
```

Cấu hình Swagger

- ❑ **Bean Docket** sẽ xác định các **class trong package nào được tạo document**, thông tin là gì... (ví dụ ở đây mình tạo document cho các class trong package **com.myclass.controller**)
- ❑ Method **ApiInfo()** sẽ trả về thông tin hiển thị trên swagger.
 - ✓ **title**: tên API.
 - ✓ **description** : thông tin mô tả về API, có thể viết thành nhiều dòng & hỗ trợ cú pháp Markdown.
 - ✓ **info** : thông tin liên hệ, chứng chỉ, điều khoản sử dụng và những thông tin khác.
 - ✓ **version**: phiên bản API.
- ❑ Khởi động **Swagger** <http://localhost:8080/swagger-ui.html>

Cấu hình Static Resource

```
public void addResourceHandlers(ResourceHandlerRegistry registry) {  
    registry  
        .addResourceHandler("/**")  
        .addResourceLocations("/WEB-INF/static/");
```

Thêm cấu hình
cho Swagger

```
    registry  
        .addResourceHandler("swagger-ui.html")  
        .addResourceLocations("classpath:/META-INF/resources/");  
  
    registry  
        .addResourceHandler("/webjars/**")  
        .addResourceLocations("classpath:/META-INF/resources/webjars/");
```

```
}
```

Bài tập về nhà

❑ Xây dựng các API quản lý các đối tượng như hình bên dưới.

❖ Lưu ý: **user_courses** chưa cần làm nhé.

❖ Yêu cầu làm lại project mới hoàn toàn, đặt tên là **ElearningApp**.

