

LẬP TRÌNH WEB (WEBPR330479)

RESTful API in Spring Boot 3

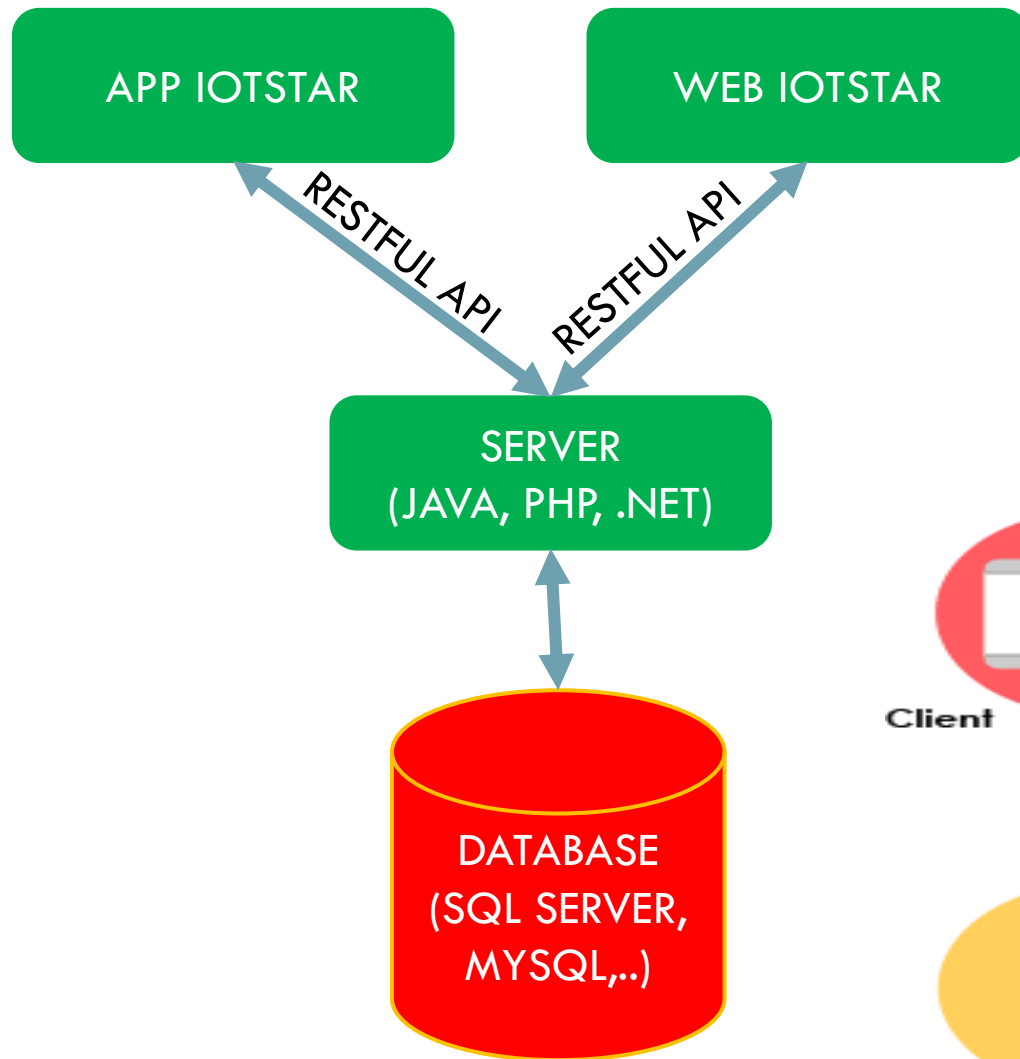


THS. NGUYỄN HỮU TRUNG

- Ths. Nguyễn Hữu Trung
- Khoa Công Nghệ Thông Tin
- Trường Đại học Sư Phạm Kỹ Thuật TP.HCM
- 090.861.7108
- trungnh@hcmute.edu.vn
- <https://www.youtube.com/@baigiai>



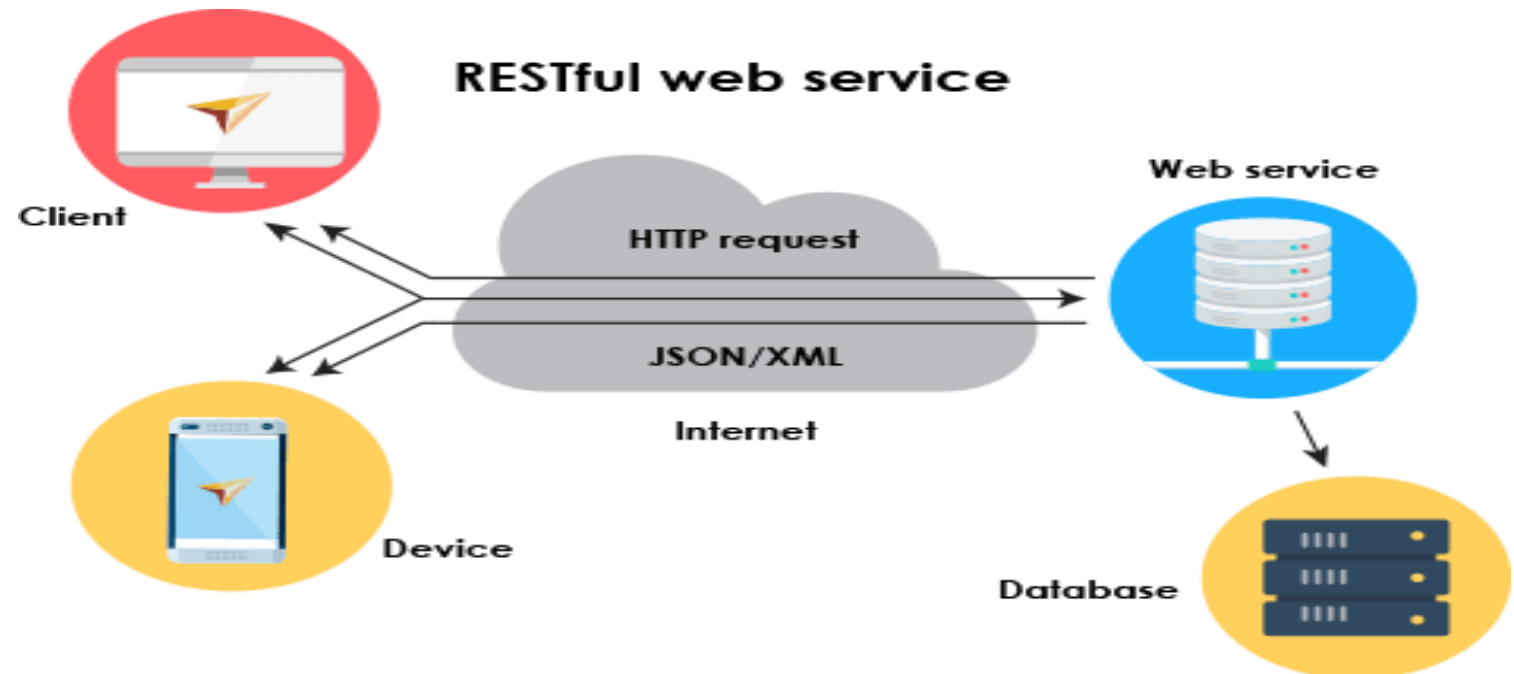
- Restful API
- Restful API trong Spring boot
- Render Rest API to AJAX



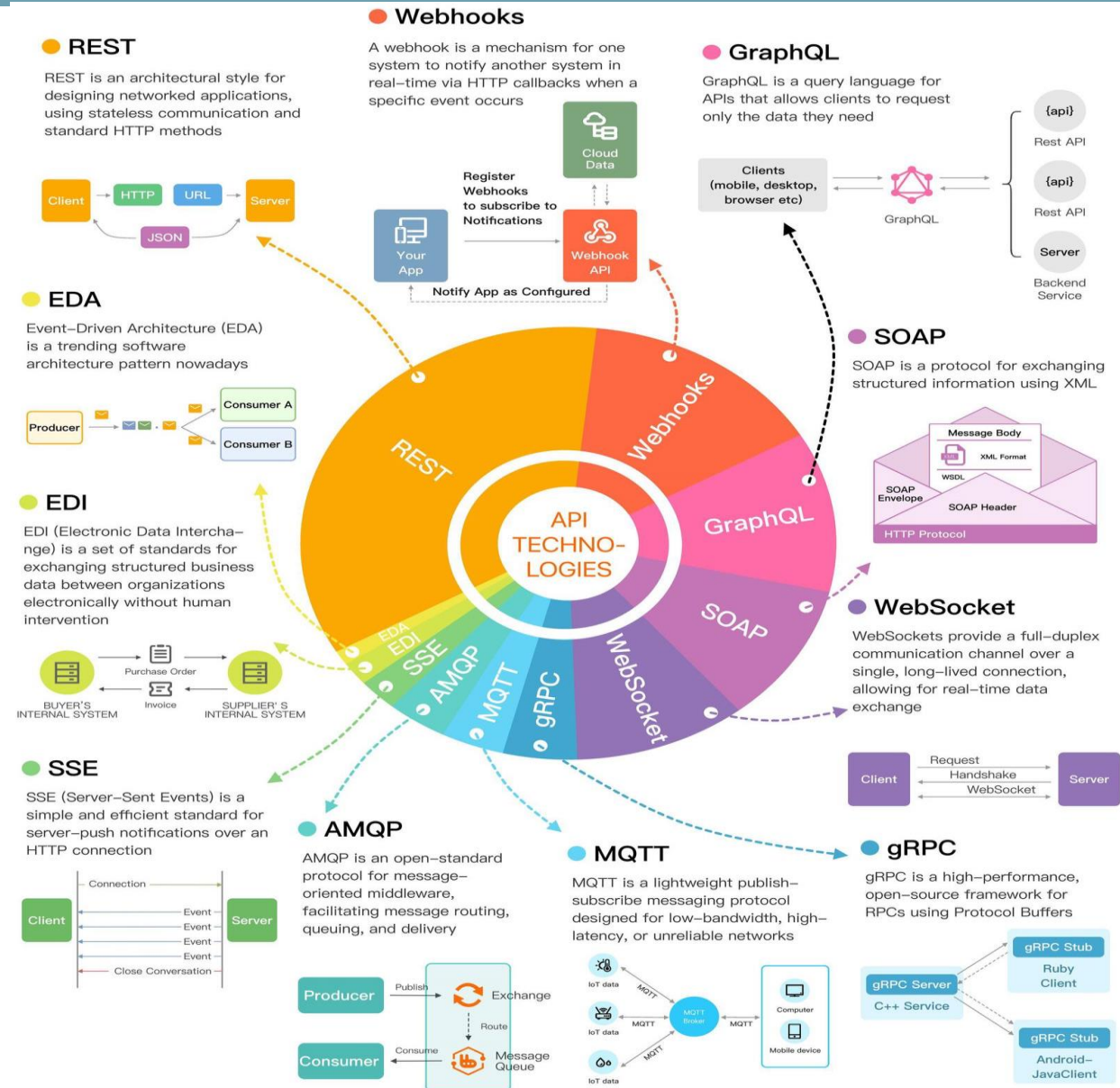
Web Service là tập hợp các giao thức và tiêu chuẩn mở được sử dụng để trao đổi dữ liệu giữa các ứng dụng hoặc giữa các hệ thống.

Web service thì có 2 dạng:

- SOAP: Sử dụng XML
- REST: Sử dụng JSon



- API (Application Programming Interface): là một giao diện lập trình phần mềm trung gian hỗ trợ các ứng dụng giao tiếp với nhau.
- Chúng cho phép các ứng dụng khác giao tiếp hiệu quả với nhau và dùng được cho web-based system, operating system, database system, computer hardware, or software library.
- Web API là mô hình được tạo ra nhằm mục đích hỗ trợ MVC như: routing, controller, action result, filter, IoC container, model binder, unit test, injection. Đồng thời còn cung cấp tính năng cho phép Restful đầy đủ các phương thức: **Get/ Post/ Put/ Delete** dữ liệu.



- **RESTful API** là một tiêu chuẩn dùng trong việc thiết kế API cho các ứng dụng web (thiết kế Web services) để tiện cho việc quản lý các resource. Nó chú trọng vào tài nguyên hệ thống (tệp văn bản, ảnh, âm thanh, video, hoặc dữ liệu động...), bao gồm các trạng thái tài nguyên được định dạng và được truyền tải qua HTTP.
- Dưới đây là 4 lệnh phổ biến nhất cho phép người dùng truy cập vào RESTful API:
 - ▣ **GET**: cho phép truy vấn object
 - ▣ **POST**: giúp tạo object mới
 - ▣ **PUT**: hỗ trợ sửa đổi hoặc thay thế một object
 - ▣ **DELETE**: loại bỏ một object
- Hiện tại có 3 cơ chế Authorize chính:
 - ▣ HTTP Basic
 - ▣ JSON Web Token (JWT)
 - ▣ OAuth2

- Khi chúng ta request một API nào đó thường thì sẽ có vài status code để nhận biết sau:
 - ▣ 200 OK – Trả về thành công cho những phương thức GET, PUT, PATCH 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.
 - ▣ 404 Not Found – Không tìm thấy resource từ URI
 - ▣ 405 Method Not Allowed – Phương thức không cho phép với user hiện tại.
 - ▣ 410 Gone – 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

- **JSON**(JavaScript **O**bject **N**otation) là một kiểu định dạng dữ liệu tuân theo một quy luật nhất định mà hầu hết các ngôn ngữ lập trình hiện nay đều có thể đọc được. **JSON** là một tiêu chuẩn mở để trao đổi dữ liệu trên web.

XML

```
<empinfo>
  <employees>
    <employee>
      <name>James Kirk</name>
      <age>40</age>
    </employee>
    <employee>
      <name>Jean-Luc Picard</name>
      <age>45</age>
    </employee>
    <employee>
      <name>Wesley Crusher</name>
      <age>27</age>
    </employee>
  </employees>
</empinfo>
```

JSON

```
{ "empinfo" :
  {
    "employees" : [
      {
        "name" : "James Kirk",
        "age" : 40,
      },
      {
        "name" : "Jean-Luc Picard",
        "age" : 45,
      },
      {
        "name" : "Wesley Crusher",
        "age" : 27,
      }
    ]
  }
}
```


- JSON sử dụng các cặp *key – value* để định dạng. Nó hỗ trợ các cấu trúc dữ liệu như đối tượng và mảng.

Ví dụ một tập tin có tên `iotstar.json` với nội dung như ở dưới đây sử dụng format kiểu JSON để lưu trữ thông tin:

```
{ "name" : "iotstar",  
  "title" : "Nguyễn Hữu Trung"  
}
```

- Hai thư viện Json phổ biến: Gson và Jackson (Spring boot đã được tích hợp)

- `@RestController` annotation là sự kết hợp giữa `@Controller` và `@ResponseBody` annotations.
- `@RequestMapping("/api")` để khai báo các url của các api trong controller này sẽ bắt đầu với '/api'
- `@ResponseStatus` để chỉ định controller sẽ trả về status nào đó, ngay cả khi thực hiện thành công
- **`ResponseEntity<T>`**: Đây là cách sử dụng response tốt nhất và mình khuyến khích sử dụng. Nó có ưu điểm hơn hai cách trước như sau:
 - ▣ Không cần code dài dòng, class này có sử dụng builder nên dùng khá tiện
 - ▣ Với `@ResponseStatus` thì chỉ set cứng status code, tuy nhiên nếu mình muốn status code khác nhau tùy vào điều kiện thì sao. `ResponseEntity<T>` sẽ giải quyết được hết.
 - ▣ Có thể vừa đặt data trong body, vừa tùy chỉnh header, vừa,... cùng lúc được luôn.
 - ▣ Trong code thì các bạn chỉ cần đổi return type của method từ return gì đó (tạm gọi là X) thành kiểu `ResponseEntity<X>` là được.

Sử dụng @Controller và @ResponseBody Annotation

11

- Với REST API, dữ liệu khi trả về sẽ nằm trong response body, dạng JSON. Nếu dùng @RestController thì không cần @ResponseBody vì nó mặc định có rồi, còn ngược lại @Controller thì phải chỉ định rõ dữ liệu nằm ở đâu trong response.

@Controller

@RequestMapping("v1/api/category")

```
public class CategoryApiController {
```

```
    @Autowired
```

```
    private CategoryRepository repository;
```

```
    @RequestMapping("")
```

```
    @ResponseBody
```

```
    public ResponseEntity<?> list() {
```

```
        return ResponseEntity.ok().body(repository.findAll());
```

```
    }
```

```
}
```

The screenshot displays a REST client interface for the endpoint `http://localhost:8088/v1/api/category`. The request method is `GET`. The response status is `200 OK` with a response time of `125 ms` and a body size of `740 B`. The response body is shown in JSON format, containing an array of two objects representing categories.

Key	Value	Description
categoryId	1	
name	"Giày 01a"	
products	[]	

```
1 [
2   {
3     "categoryId": 1,
4     "name": "Giày 01a",
5     "products": []
6   },
7   {
8     "categoryId": 2,
9     "name": "Giày 01",
10    "products": []
11  },
12 ]
```

Sử dụng @RestController Annotation

12

- Với REST API, dữ liệu khi trả về sẽ nằm trong response body, dạng JSON. Nếu dùng @RestController thì không cần @ResponseBody vì nó mặc định có rồi.

```
@RestController
```

```
@RequestMapping("v1/api/category")
```

```
public class CategoryApiController {
```

```
    @Autowired
```

```
    private CategoryRepository repository;
```

```
    @RequestMapping("")
```

```
    public ResponseEntity<?> list() {
```

```
        return ResponseEntity.ok().body(repository.findAll());
```

```
    }
```

```
}
```

http://localhost:8088/v1/api/category

GET http://localhost:8088/v1/api/category

Params Authorization Headers (7) Body Scripts Settings Cookies

Headers 7 hidden

Key	Value	Description	Bulk Edit	Presets
-----	-------	-------------	-----------	---------

Body Cookies Headers (5) Test Results

200 OK • 125 ms • 740 B • Save Response

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "categoryId": 1,
4     "name": "Giày 01a",
5     "products": []
6   },
7   {
8     "categoryId": 2,
9     "name": "Giày 01",
10    "products": []
11  },
12 ]
```

- **ResponseEntity<T>**: Đây là cách sử dụng response tốt nhất và mình khuyến khích sử dụng. Nó có ưu điểm hơn hai cách trước như sau:
 - ▣ Không cần code dài dòng, class này có sử dụng builder nên dùng khá tiện
 - ▣ Với @ResponseStatus thì chỉ set cứng status code, tuy nhiên nếu mình muốn status code khác nhau tùy vào điều kiện thì sao. ResponseEntity<T> sẽ giải quyết được hết.
 - ▣ Có thể vừa đặt data trong body, vừa tùy chỉnh header, vừa,... cùng lúc được luôn.
 - ▣ Trong code thì các bạn chỉ cần đổi return type của method từ return gì đó (tạm gọi là X) thành kiểu ResponseEntity<X> là được.

Sử dụng ResponseEntity<T>

14

```
@GetMapping("/get")
```

```
public ResponseEntity<CategoryEntity>
```

```
getCategory(@RequestParam("name") String categoryname) {
```

```
// Tìm Category trong database bằng name
```

```
Optional<CategoryEntity> category =
```

```
categoryService.findByName(categoryname);
```

```
// Nếu không tìm thấy, trả về message lỗi 404 Not found
```

```
if (category == null)
```

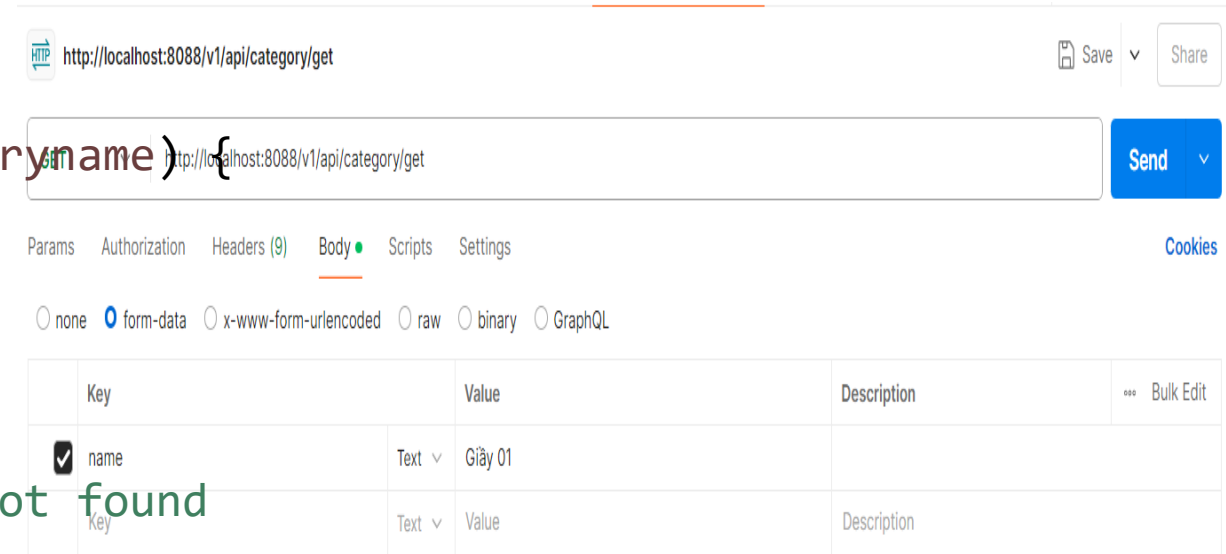
```
// Tạm thời là vậy, thực tế người ta dùng AOP để bắt exception
```

```
return ResponseEntity.notFound().build();
```

```
// Nếu tìm thấy return 200 OK
```

```
return ResponseEntity.ok(category.get());
```

```
}
```

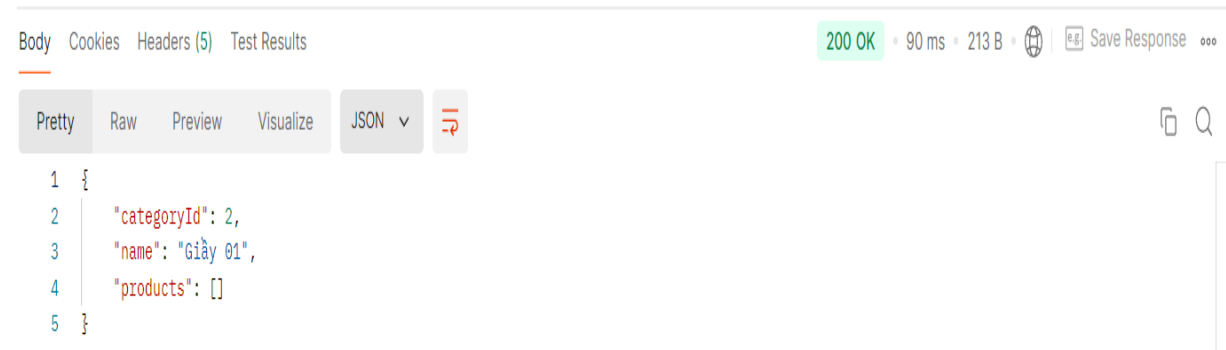


HTTP GET http://localhost:8088/v1/api/category/get

Params Authorization Headers (9) Body Scripts Settings Cookies

☐ none ☒ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/> name	Text <input type="text" value="Giày 01"/>			
Key	Text <input type="text" value="Value"/>	Description		



Body Cookies Headers (5) Test Results 200 OK • 90 ms • 213 B • Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "categoryId": 2,
3   "name": "Giày 01",
4   "products": []
5 }
```

□ ResponseEntity<T> có thể dùng 3 cách:

- ▣ Tạo mới object dạng `new ResponseEntity<>(category, HttpStatus.OK)`
- ▣ Dùng dạng static method `ResponseEntity.ok(category)`
- ▣ Dùng dạng builder `ResponseEntity.status(HttpStatus.BAD_REQUEST).body("User not found").build()`

```
@GetMapping("/get")
public ResponseEntity<CategoryEntity> getCategory(@RequestParam("name") String
categoryname) {
    // Tìm Category trong database bằng name
    Optional<CategoryEntity> category = categoryService.findByName(categoryname);
    // Nếu không tìm thấy, trả về message lỗi 404 Not found
    if (category == null)
        // Tạm thời là vậy, thực tế người ta dùng AOP để bắt exception
        // return ResponseEntity.notFound().build();
        return ((HeadersBuilder<BodyBuilder>)
ResponseEntity.status(HttpStatus.BAD REQUEST).body("User not found")).build();
    // Nếu tìm thấy return 200 OK
    // return ResponseEntity.ok(category.get());
    return new ResponseEntity<CategoryEntity>(category.get(), HttpStatus.OK);
}
```


Sử dụng @RequestBody annotation

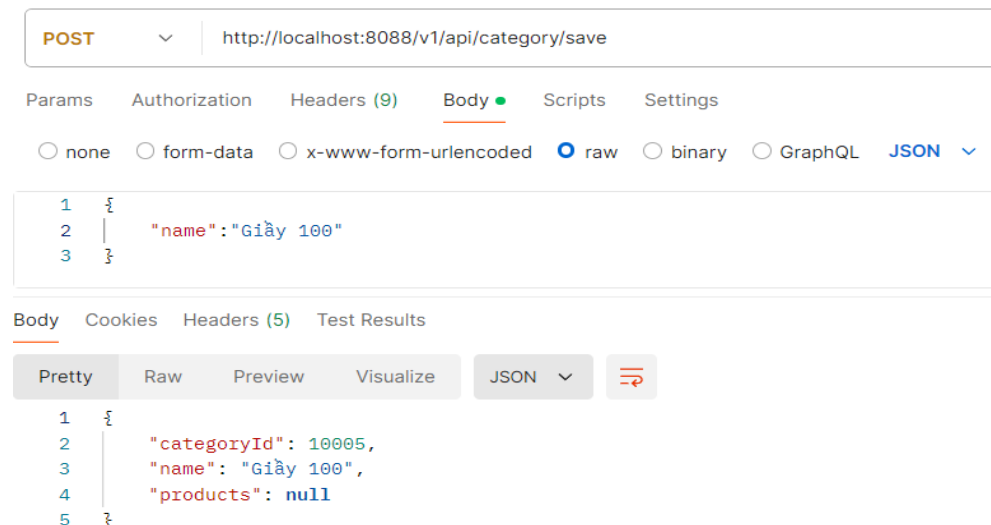
16

- Request method PUT, POST mới có request body, đây là nơi chứa data chính để gửi lên. Thường thì request body sẽ ở dạng JSON hoặc form-data, khi vào controller sẽ được tự động parse ra thành Object (ví dụ Entity, DTO, Models chẳng hạn).

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "Categories")
public class CategoryEntity implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long categoryId;
    @Column(name = "category_name", length = 200, columnDefinition = "nvarchar(200) not null")
    private String name;

    @OneToMany(mappedBy = "category", cascade = CascadeType.ALL )
    private Set<ProductEntity> products;
}
```

```
@PostMapping(value = "/save")
public CategoryEntity saveCate(@Valid
@RequestBody CategoryEntity cate) {
    return categoryService.save(cate);
}
```



Sử dụng @RequestBody annotation

17

□ Update category

```
@RequestMapping(value = "/update/{id}",
method = RequestMethod.PUT)
public ResponseEntity<CategoryEntity>
updateCategory(@PathVariable(value = "id")
Long categoryid,
@Valid @RequestBody CategoryEntity
category) {
Optional<CategoryEntity> cate =
categoryService.findById(categoryid);
if(cate == null) {
return ResponseEntity.notFound().build();
}
cate.get().setName(category.getName());
CategoryEntity updatedContact =
categoryService.save(cate.get());
return ResponseEntity.ok(updatedContact);
}
```

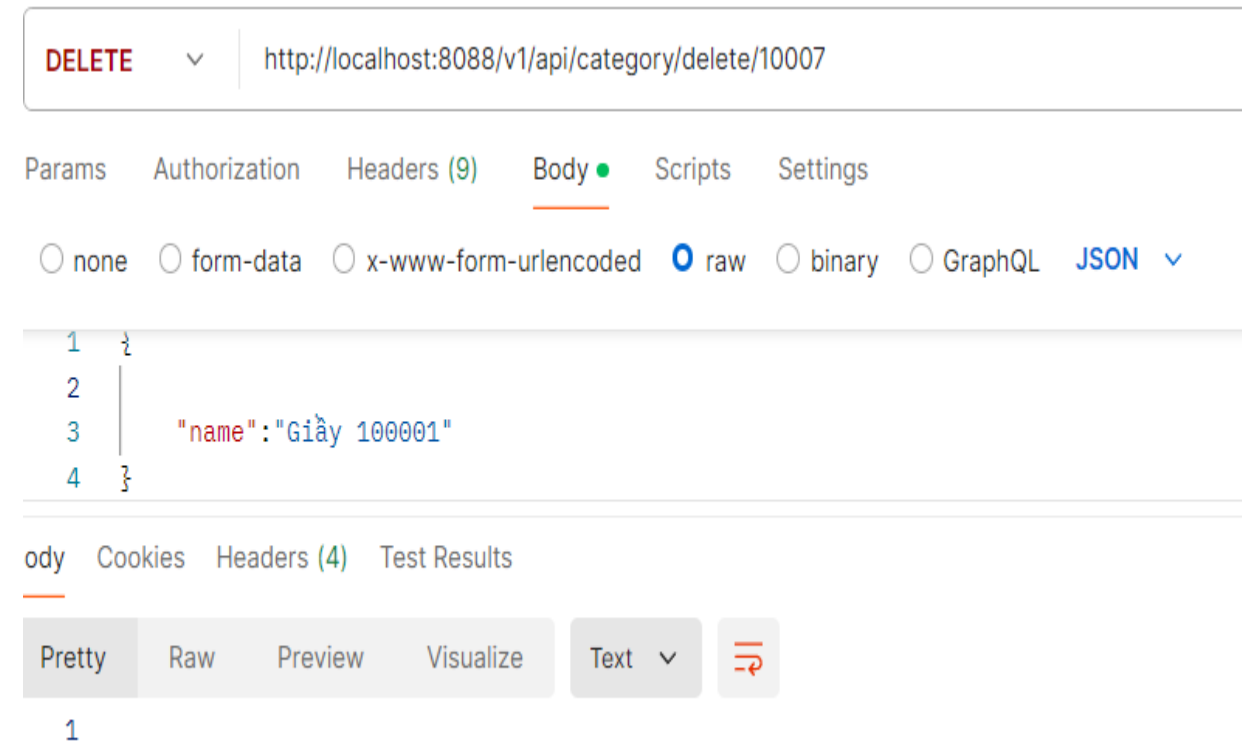
The screenshot shows a REST client interface with a PUT request to `http://localhost:8088/v1/api/category/update/10005`. The 'Body' tab is selected, showing a JSON payload: `{ "categoryId": 10005, "name": "Giày 100001" }`. Below this, another tab shows the response body in JSON: `{ "categoryId": 10005, "name": "Giày 100001", "products": [] }`.

Sử dụng @RequestBody annotation

18

❑ Delete category

```
@DeleteMapping(value = "/contact/{id}")
public ResponseEntity<CategoryEntity>
deleteCategory(@PathVariable(value = "id")
Long categoryid) {
Optional<CategoryEntity> cate =
categoryService.findById(categoryid);
if(cate == null) {
return ResponseEntity.notFound().build();
}
categoryService.delete(cate.get());
return ResponseEntity.ok().build();
}
```



□ Tạo class Response.java để chứa các trường thông tin muốn tùy chỉnh khi response về client.

```
7 @Data
8 @NoArgsConstructor
9 @AllArgsConstructor
10
11 public class Response {
12
13     private Boolean status;
14
15     private String message;
16
17     private Object body;
18
19 }
20
```

The screenshot shows a REST client interface with a GET request to `http://localhost:8088/v1/api/category/list`. The response is displayed in JSON format, showing a successful status and a list of categories.

```
GET http://localhost:8088/v1/api/category/list

Params Authorization Headers (9) Body Scripts Settings
none form-data x-www-form-urlencoded raw binary GraphQL JSON

Body Cookies Headers (5) Test Results
Pretty Raw Preview Visualize JSON

1 {
2   "status": true,
3   "message": "Thành công",
4   "body": [
5     {
6       "categoryId": 1,
7       "name": "Giày 01a",
8       "products": []
9     },
10    {
11      "categoryId": 2,
12      "name": "Giày 01",
13      "products": []
14    },
15    {
16      "categoryId": 6,
17      "name": "Nguyễn Hữu Trung 1",
18      "products": []
19    }
20  ]
21 }
```

❑ Tạo class Response.java để chứa các trường thông tin muốn tùy chỉnh khi response về client.

```
@GetMapping(path = "/get3")
public ResponseEntity<> getCategory(@Validated @RequestParam("id") Long categoryid) {
    Optional<CategoryEntity> category = categoryService.findById(categoryid);

    if (category.isPresent()) {
        //return ResponseEntity.ok().body(category.get());
        return new ResponseEntity<Response>(new Response(true, "Thành công", category.get()), HttpStatus.OK);
    } else {
        //return ResponseEntity.notFound().build();
        return new ResponseEntity<Response>(new Response(false, "Thất bại", null), HttpStatus.NOT_FOUND);
    }
}
```

GET http://localhost:8088/v1/api/category/get3

Params Authorization Headers (9) **Body** Scripts Settings

☐ none ☒ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

	Key	Value
<input checked="" type="checkbox"/>	id	Text 1

body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "status": true,
3   "message": "Thành công",
4   "body": {
5     "categoryId": 1,
6     "name": "Giày 01a",
7     "products": []
8   }
9 }
```

LẬP TRÌNH WEB (WEBPR330479)

Render Rest API to AJAX



THS. NGUYỄN HỮU TRUNG

Gọi thư viện JQuery và viết hàm thực hiện render

22

- ❑ `<script src="https://cdn.jsdelivr.net/npm/jquery@3.7.1/dist/jquery.min.js"></script>`
- ❑ `<script src="/js/main.js"></script>`
- ❑ Hàm xử lý render sẽ viết trong file main.js

```
1=$(document).ready(function() {
2    //Hiển thị thông tin người dùng đăng nhập thành công
3    $.ajax({
4        type: 'GET',
5        url: '/users/me',
6        dataType: 'json',
7        contentType: "application/json; charset=utf-8",
8        beforeSend: function(xhr) {
9            if (localStorage.token) {
10                xhr.setRequestHeader('Authorization', 'Bearer ' + localStorage.token);
11            }
12        },
13        success: function(data) {
14            var json = JSON.stringify(data, null, 4);
15            // $('#profile').html(json);
16            $('#profile').html( data.fullName);
17            $('#images').html(document.getElementById("images").src=data.images);
18            //console.log("SUCCESS : ", data);
19            //alert('Hello ' + data.email + '! You have successfully accessed to /api/profile.');
```


Gọi thư viện JQuery và viết hàm thực hiện render

23

- ❑ `<script src="https://cdn.jsdelivr.net/npm/jquery@3.7.1/dist/jquery.min.js"></script>`
- ❑ `<script src="/js/main.js"></script>`
- ❑ Hàm xử lý render sẽ viết trong file main.js

```
28 //Hàm đăng xuất
29 $('#logout').click(function() {
30     localStorage.clear();
31     window.location.href = "/login";
32 });
33 //hàm Login
34 $('#Login').click(function() {
35     var email = document.getElementById('email').value;
36     var password = document.getElementById('password').value;
37     var basicInfo = JSON.stringify({
38         email:email,
39         password:password |
40     });
41     $.ajax({
42         type: "POST",
43         url: "/auth/login",
44         dataType: 'json',
45         contentType: "application/json; charset=utf-8",
46         data: basicInfo,
47         success: function(data) {
48             localStorage.token = data.token;
49             // alert('Got a token from the server! Token: ' + data.token);
50             window.location.href = "/user/profile";
51         },
52         error: function() {
53             alert("Login Failed");
54         }
55     });
56 });
57 });
```

Gọi thư viện JQuery và viết hàm thực hiện render

24

- `<script src="https://cdn.jsdelivr.net/npm/jquery@3.7.1/dist/jquery.min.js"></script>`
- `<script src="/js/main.js"></script>`
- Gọi ra view

```
<div class="container" style="min-height: 500px">
<form action="" method="post">
<label>Email</label>
<input type="email" name="email" id="email" required="required">
<label>Password</label>
<input type="password" name="password" id="password"
required="required" autocomplete="on">
<button id="Login" type="button">Login</button>
</form>
</div>
```

```
<div class="container" style="min-height: 500px">
<div class="starter-template">
<h1>Spring Boot REST API with AJAX Example</h1>
<img id="images" src="" alt="" width="100">
<div id="profile"></div>
<button id="Logout">Logout</button>
</div>
</div>
```