JAVA WEB

Unit 6: Rest API



Mục lục

Giới thiệu REST API

REST là một trong những kiểu thiết kế API được sử dụng phổ biến ngày nay để cho các ứng dụng (web, mobile...) khác nhau giao tiếp với nhau.

REST không giới hạn bởi ngôn ngữ lập trình ứng dụng, bất kỳ ngôn ngữ hoặc framework nào cũng có thể sử dụng để thiết kế một REST API.

REST hoạt động vào giao thức HTTP với các phương thức sau:

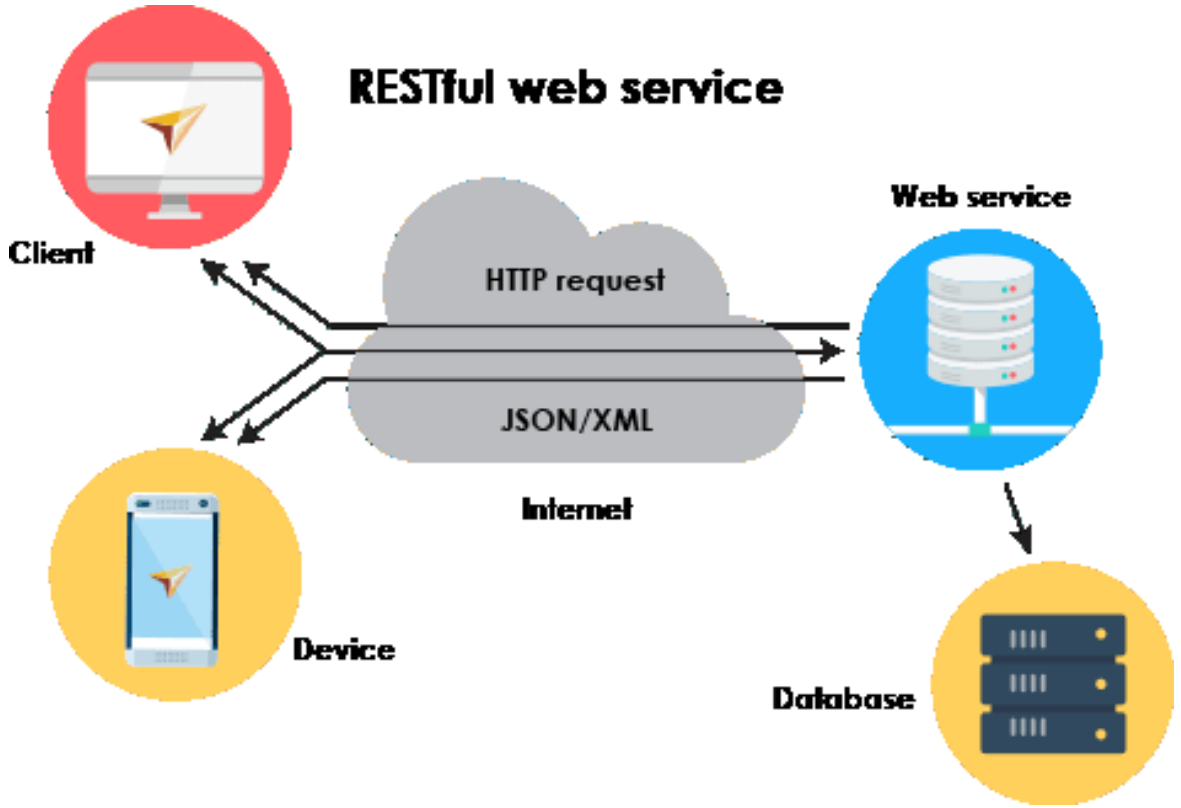
- GET (SELECT): Trả về một Resource hoặc một danh sách Resource.
- POST (CREATE): Tạo mới một Resource.
- PUT (UPDATE): Cập nhật thông tin cho Resource.
- DELETE (DELETE): Xoá một Resource.

REST API không sử dụng session và cookie, nó sử dụng một access_token với mỗi request. Dữ liệu trả về thường là JSON.

@RestController

@RestController được giới thiệu từ phiên bản Spring 4.0 để đơn giản hóa việc tạo ra các Rest API=> Nó là sự kết hợp của annotaiton **@Controller và @ResponseBody.**

@RestController trả về dữ liệu dưới dạng JSON.



Tạo các Rest API

Phương thức	**Đường dẫn**	**Miêu tả**
GET	/api/v1/products	Lấy về tất cả sản phẩm
GET	/api/v1/products/1	Lấy về sản phẩm có id bằng 1
POST	/api/v1/products	Thêm một sản phẩm
PUT	/api/v1/products/1	Cập nhật sản phẩm có id là 1
DELETE	/api/v1/products/1	Xoá sản phẩm có id là 1

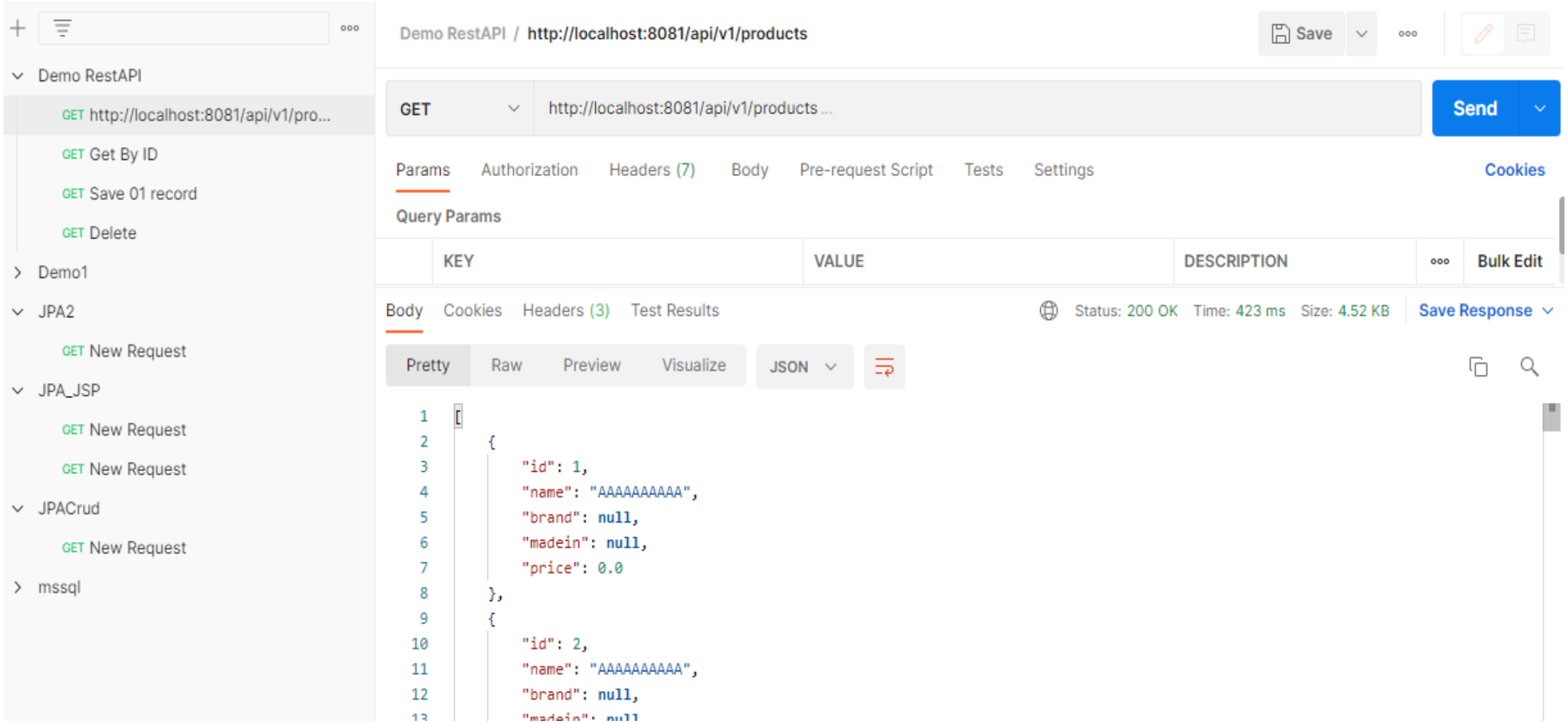
Tạo các Rest API

Thực hiện tạo ứng dụng Spring boot và thêm các dependency cần thiết:

```
@RestController
@RequestMapping("/api/v1/")
public class AppController {
    @Autowired  private ProductService service;
    @GetMapping("/products")
    public List<Product> getALL() {
        List<Product> listProducts = service.listAll();  return listProducts;
    }
    @GetMapping("/getProdByID/{id}")
    public Product getProdById(@PathVariable Long id) {
      Product prod = service.get(id); return prod;
    }
    @PostMapping("/products")
    public Product createProduct(@RequestBody Product prod) throws Exception {
       try { service.save(prod);  return service.get(prod.getId());
        } catch( Exception e) {  throw new Exception("Đã có");  }
    }
    @PutMapping("/products")
    public Product updateProduct(@RequestBody Product prod) throws Exception {
        try {  service.save(prod); return service.get(prod.getId());
        } catch( Exception e) {  throw new Exception("Đã   có"); }
    }
    @DeleteMapping("/products/{id}")
       public String  deleteProduct(@PathVariable Long id) {
        service.delete(id);
        Map<String, Boolean> response = new HashMap<>();
        response.put("deleted", Boolean.TRUE); return response;
    }
}
```

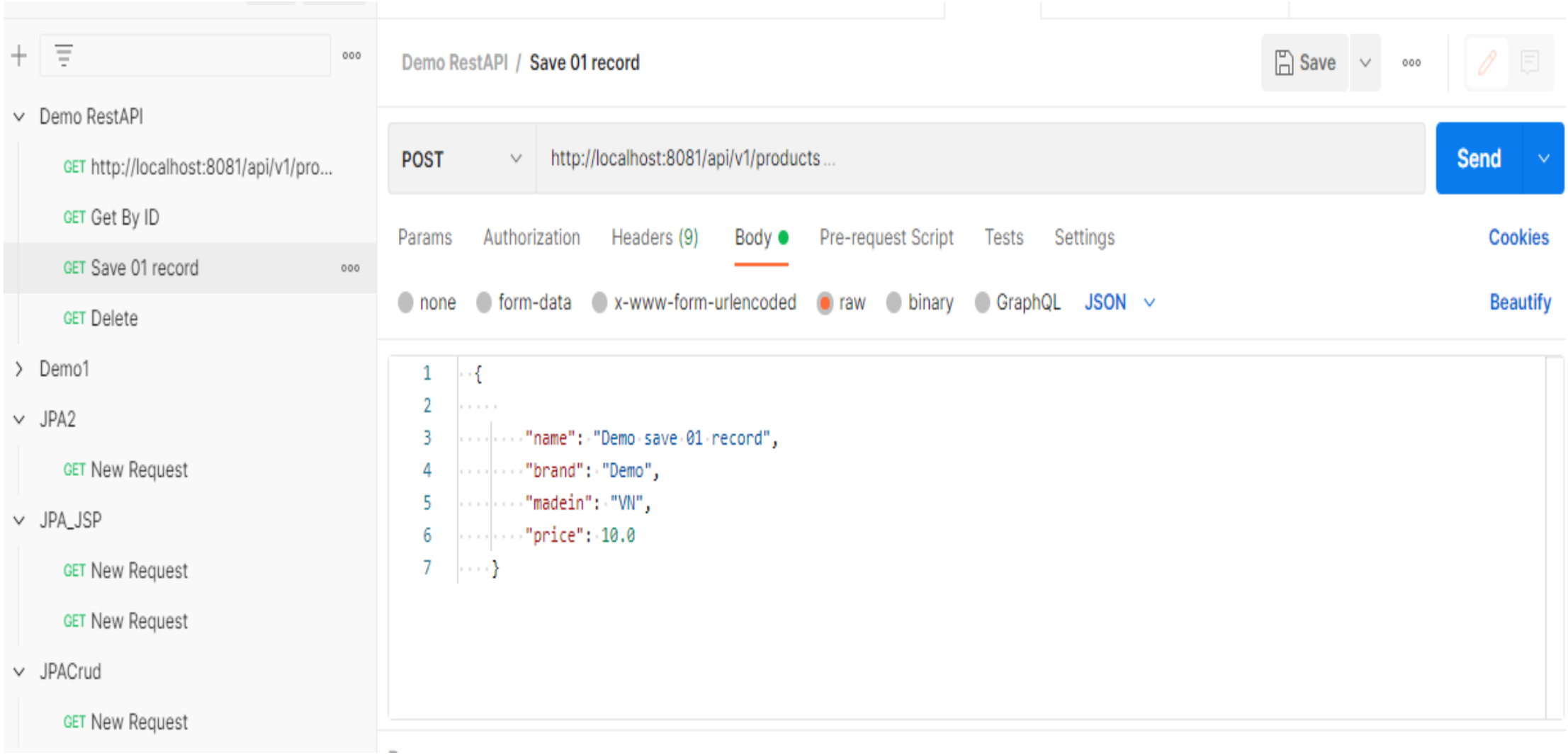
Thử nghiệm các Rest API trên PostMan

Lấy về tất cả sản phẩm



Thử nghiệm các Rest API trên PostMan

Tạo mới một sản phẩm



CORS trong Spring Boot

Chia sẻ tài nguyên giữa các nguồn gốc khác nhau (Cross-Origin Resource Sharing -CORS) là kĩ thuật khi JS client truy cập tài nguyên thông qua Rest APIs.

Thông thường, host chạy JS (Ví dụ: example.com) khác biệt với host chạy phần dịch vụ API (Ví dụ: apiDemo.com). Trong trường hợp này, CORS cho phép giao tiếp giữa các domain khác nhau (cross-domain) hoạt động.

Spring cung cấp cách đơn giản để thiết lập CORS trong các ứng dụng Spring hoặc Spring Boot

Thiết lập CORS trong controller

Thiết lập CORS cho phương thức của controller: để kích hoạt CORS cho một phương thức chúng ta cần thêm annotation @CrossOrigin:

- Trường hợp không tham số, ví dụ:@CrossOrigin() sẽ cho phep JS client truy cập từ mọi nơi.
- Trường hợp có tham số origins, ví dụ: @CrossOrigin (origins = http://example.com) sẽ chỉ có phép JS truy cập từ host: http://example.com

Có thể thiết lập CORS cho cả lớp controller.

Cấu hình Global CORS

```
@SpringBootApplication
public class RestServiceCorsApplication {
  public static void main(String[] args) {
                    SpringApplication.run(RestServiceCorsApplication.class, args);
  }
  @Bean
  public WebMvcConfigurer corsConfigurer() {
     return new WebMvcConfigurer() {
                  @Override
                   public void addCorsMappings(CorsRegistry registry) {
                            // Cho phép truy cập từ mọi nơi đến mọi API
                            registry.addMapping("/**");

                            // Cho phép truy cấn đến "…/greeting" từ http://localhost:8080
                            //registry.addMapping("/greeting").allowedOrigins("http://localhost:8080");
                   }
            };
     }

}
```

Rest API exception handling

Spring Boot cung cấp các annotations: @RestControllerAdvice và @ExceptionHandler để xử lý ngoại lệ toàn cục.

{"statusCode":10100,"message":"Đối tượng không tồn tại"}

{"statusCode":10100,"message":"!!!1SQL Exception"}

Rest API exception handling

Tạo các lớp cơ bản:

```java
@Data
@AllArgsConstructor
public class Todo {
    private String title;
    private String detail;
}
```

```java
@Data
@AllArgsConstructor
public class ErrorMessage {
    private int statusCode;
    private String message;
}
```

```java
@RestController
@RequestMapping("/api/v1")
public class RestAPIController {
    private List<Todo> todoList;
    @PostConstruct
    public void init() {
        // Tạo các đối tượng Todo từ 0 đến 10
        todoList = IntStream.range(0, 10)
                .mapToObj(i -> new Todo( title: "title-" + i, detail: "detail-" + i))
                .collect(Collectors.toList());
    }
    /*   http://localhost:8081/api/v1/todo/11
   ==> Phát sinh ngoại lệ IndexOutOfBoundsException vì đối tượng 11 không tồn tại     */
    @GetMapping("/todo/{todoId}")
    public Todo getTodo(@PathVariable(name = "todoId") Integer todoId) throws SQLException {

        return todoList.get(todoId);
    }
    /*    http://localhost:8081/api/v1/data/0      */
    @GetMapping("/data/{todoId}")
    public String getData(@PathVariable(name = "todoId") Integer todoId) throws SQLException {
        if (todoId==0)  throw new SQLException( " " );
        return "done";
    }
}
```

Rest API exception handling

Sử dụng @RestControllerAdvice và @ExceptionHandler để xử lý ngoại lệ

```java
@RestController
@RequestMapping("/api/v1")
public class RestAPIController {
    private List<Todo> todoList;
    @PostConstruct
    public void init() {
        // Tạo các đối tượng Todo từ 0 đến 10
        todoList = IntStream.range(0, 10)
                .mapToObj(i -> new Todo( title: "title-" + i, detail: "detail-" + i))
                .collect(Collectors.toList());
    }
    /*   http://localhost:8081/api/v1/todo/11
    ==> Phát sinh ngoại lệ IndexOutOfBoundsException vì đối tượng 11 không tồn tại     */
    @GetMapping("/todo/{todoId}")
    public Todo getTodo(@PathVariable(name = "todoId") Integer todoId) throws SQLException {

        return todoList.get(todoId);
    }
    /*    http://localhost:8081/api/v1/data/0      */
    @GetMapping("/data/{todoId}")
    public String getData(@PathVariable(name = "todoId") Integer todoId) throws SQLException {
        if (todoId==0)  throw new SQLException( " " );
        return "done";
    }
```

```java
@RestControllerAdvice
public class ApiExceptionHandler {    /**
     * Tất cả các Exception không được khai báo sẽ được xử lý tại đây */
    @ExceptionHandler(Exception.class)
    @ResponseStatus(value = HttpStatus.INTERNAL_SERVER_ERROR)
    public ErrorMessage handleAllException(Exception ex, WebRequest request) {
        // quá trình kiểm soat lỗi diễn ra ở đây
        return new ErrorMessage( statusCode: 10000, ex.getLocalizedMessage());
    }
    /**
     * Ngoại lệ: IndexOutOfBoundsException sẽ được xử lý riêng tại đây
     */
    @ExceptionHandler(IndexOutOfBoundsException.class)
    @ResponseStatus(value = HttpStatus.BAD_REQUEST)
    public ErrorMessage TodoException(Exception ex, WebRequest request) {
        return new ErrorMessage( statusCode: 10100,  message: "Đối tượng không tồn tại");
    }
    @ExceptionHandler(SQLException.class)
    @ResponseStatus(value = HttpStatus.BAD_REQUEST)
    public ErrorMessage TodoSQLException(Exception ex, WebRequest request) {
        return new ErrorMessage( statusCode: 10100,  message: "!!!1SQL Exception");
    }
}
```