ResponseEntity

■ What is ResponseEntity?

ResponseEntity is a class in Spring Framework used to represent the entire HTTP response.
It gives you full control over:

- HTTP status code (200, 201, 400, 404, etc.)
- Response body (data you want to return)
- HTTP headers (extra metadata)

■ Why use ResponseEntity?

- By default, Spring returns only the response body (using @RestController).
- Sometimes, you need more control like custom status codes, headers, or error handling.

■ Why Do We Need ResponseEntity?

In real-world applications, just returning the body is not enough. ResponseEntity is needed for:
1. To Control HTTP Status Codes
```
return ResponseEntity.status(HttpStatus.NOT_FOUND).build();
```
Without ResponseEntity, every response would default to 200 OK, even for errors.

2. To Add Custom Headers
```
HttpHeaders headers = new HttpHeaders(); headers.add("X-Total-Count", "100");
return ResponseEntity.ok().headers(headers).body(userList);
```
Useful for adding metadata like pagination, file info, or security tokens.

3. To Handle Empty or No Content Responses
```
return ResponseEntity.noContent().build();
```
For DELETE or other operations where no body is required.

4. To Provide Consistent Error Handling
```
return ResponseEntity.badRequest().body("Invalid input data");
```
Lets you send structured error messages with the correct status code.

5. To Improve API Clarity and REST Compliance
REST APIs need proper communication with clients via status + headers, not just body.

■ Syntax

```
ResponseEntity response = new ResponseEntity<>(body, headers, status);
```

■ Common Examples

1. Returning 200 OK with body
```
@GetMapping("/hello") public ResponseEntity sayHello() { return
ResponseEntity.ok("Hello, World!"); }
```

2. Returning 201 Created with body
```
@PostMapping("/users") public ResponseEntity createUser(@RequestBody User user) {
User savedUser = userService.save(user); return
ResponseEntity.status(HttpStatus.CREATED).body(savedUser); }
```

3. Returning with custom headers

```
@GetMapping("/download") public ResponseEntity downloadFile() { HttpHeaders
headers = new HttpHeaders(); headers.add("File-Type", "text/plain");
headers.add("File-Name", "sample.txt"); return
ResponseEntity.ok().headers(headers).body("This is file content"); }
```

4. Returning 404 Not Found

```
@GetMapping("/users/{id}") public ResponseEntity getUser(@PathVariable Long id) {
Optional user = userService.findById(id); if (user.isPresent()) { return
ResponseEntity.ok(user.get()); } else { return
ResponseEntity.status(HttpStatus.NOT_FOUND).build(); } }
```

5. Returning empty response (204 No Content)

```
@DeleteMapping("/users/{id}") public ResponseEntity deleteUser(@PathVariable Long
id) { userService.delete(id); return ResponseEntity.noContent().build(); }
```

■ Key Methods of ResponseEntity

- ok(body) → 200 OK with body
- status(HttpStatus) → set custom status
- body(body) → set body
- headers(HttpHeaders) → add headers
- noContent() → 204 No Content
- notFound() → 404 Not Found
- badRequest() → 400 Bad Request

■ In short: ResponseEntity gives you fine-grained control over HTTP responses in Spring Boot.