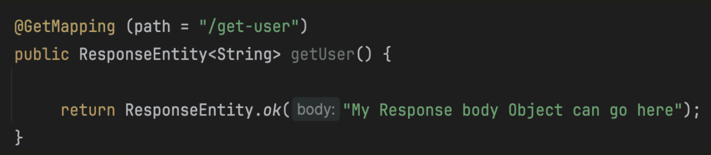
Response Generally contains 3 parts:

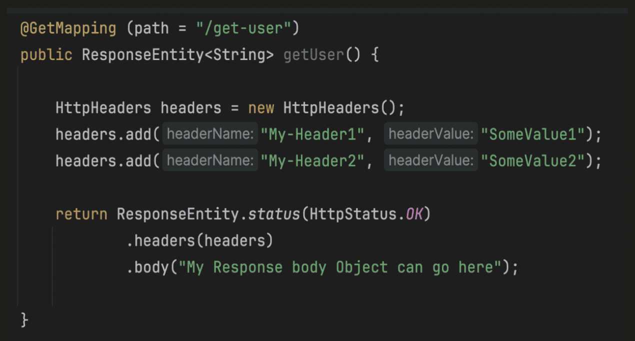
Status Code: HTTP return code like 200 OK, 500 INTERNAL_ERROR etc.
Header: Additional information (Optional)
Body: Data need to be sent in response.

We can use "ResponseEntity<T>" to create Response and in this 'T' represents the type of the 'Body'.



```
@GetMapping (path = "/get-user")
public ResponseEntity<String> getUser() {

    return ResponseEntity.ok( body: "My Response body Object can go here");
}
```

OR



```
@GetMapping (path = "/get-user")
public ResponseEntity<String> getUser() {

    HttpHeaders headers = new HttpHeaders();
    headers.add( headerName: "My-Header1",  headerValue: "SomeValue1");
    headers.add( headerName: "My-Header2",  headerValue: "SomeValue2");

    return ResponseEntity.status(HttpStatus.OK)
            .headers(headers)
            .body("My Response body Object can go here");

}
```

body should be last, actually its kind of using Builder design pattern, so 'status' , 'headers' all are returning Builder object and 'body' method call returns the ResponseEntity object.

```java
public static BodyBuilder status(HttpStatusCode status) {
    Assert.notNull(status, message: "HttpStatusCode must not be null");
    return new DefaultBuilder(status);
}
```

```java
B headers(@Nullable HttpHeaders headers);
```

```java
<T> ResponseEntity<T> body(@Nullable T body);
```

<u>So, what to do, when we don't want to add any body in the response:</u>

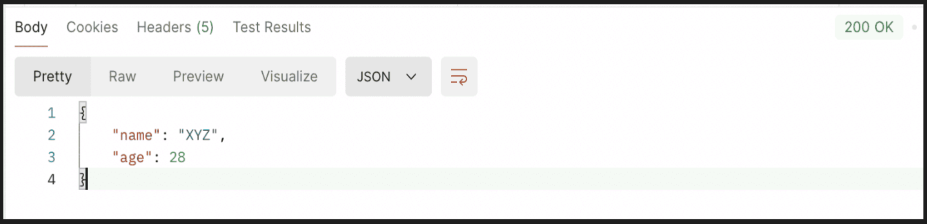
Then, we should use 'build' method

```java
<T> ResponseEntity<T> build();
```

```java
@GetMapping (path = "/get-user")
public ResponseEntity<Void> getUser() {

    HttpHeaders headers = new HttpHeaders();
    headers.add( headerName: "My-Header1",  headerValue: "SomeValue1");
    headers.add( headerName: "My-Header2",  headerValue: "SomeValue2");

    return ResponseEntity.status(HttpStatus.OK)
            .headers(headers).build();

}
```

<u>by-default, 200 Ok is the status code set:</u>

```java
@RestController
@RequestMapping(value = "/api/")
public class UserController {

    @GetMapping (path = "/get-user")
    public User getUser() {
       User responseObj = new User( name: "XYZ", age: 28);
        return responseObj;
    }
}
```



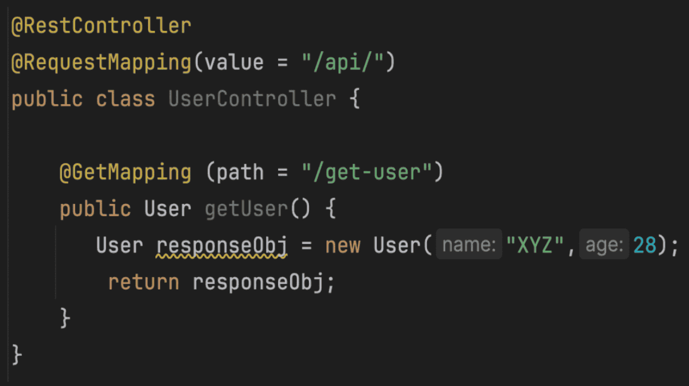
```json
{
    "name": "XYZ",
    "age": 28
}
```

@ResponseBody

When we return Plain string or POJO directly from the class, then @ResponseBody annotation is required.

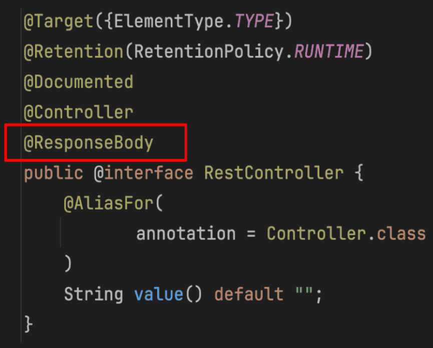
Why?
It tells to considered value as Response Body and not the View.

But in above example (mentioning below also), we did not use @ResponseBody



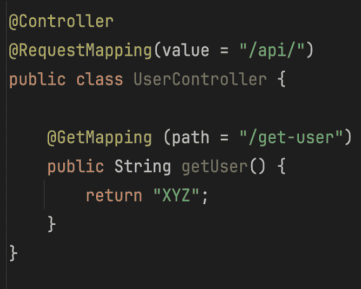
```java
@RestController
@RequestMapping(value = "/api/")
public class UserController {

    @GetMapping (path = "/get-user")
    public User getUser() {
       User responseObj = new User( name: "XYZ", age: 28);
        return responseObj;
    }
}
```

Its because, @RestController, automatically puts @ResponseBody to all the methods

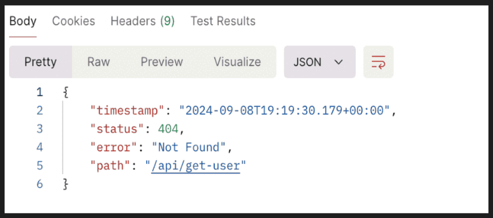


```java
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Controller
@ResponseBody
public @interface RestController {
    @AliasFor(
            annotation = Controller.class
    )
    String value() default "";
}
```

But lets say, if you use @Controller instead, then below code will throw exception

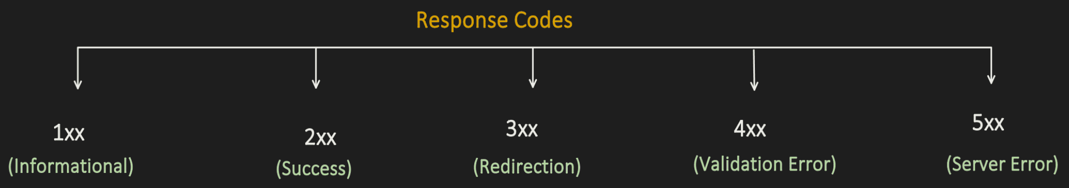


```java
@Controller
@RequestMapping(value = "/api/")
public class UserController {

    @GetMapping (path = "/get-user")
    public String getUser() {
        return "XYZ";
    }
}
```



```
{
    "timestamp": "2024-09-08T19:19:30.179+00:00",
    "status": 404,
    "error": "Not Found",
    "path": "/api/get-user"
}
```

Because, Return value is treated as "view" and spring boot will try to look for file with the given name "XYZ" which do not exist.



2xx (Success)
Request received from Client is received and processed successfully.

Status Code	Reason	Mostly used in	More details
200	Ok	GET, POST (Idempotent calls)	Request is successful and we are returning the response body.
201	Created	POST	Request is successful and new resource is created.
202	Accepted	POST	Request is successfully accepted but processing is not yet completed. Batch processing like Export, Import etc.
204	No Content	DELETE	Request is successful and we are NOT returning any data in response body.
206	Partial Content	POST	Request is partial successful, say during Bulk Addition of 100 Users , 95 passed and 5 requests failed, so this Response code can be used.

3xx (Redirection)

Client must take additional action to complete the request

Status Code	Reason	Mostly used in	More details
301	Moved Permanently	When we migrate from Legacy API to new API (Old Status code, new one is 308)	All request should directed to the new URI.
308	Permanent Redirect	When we migrate from Legacy API to new API	Same as 301, but it do not allow HTTP Method to change while redirect (for ex: if Old API call is POST, then NEW API should also be POST, which is relaxed in 301)
304	Not Modified	GET PATCH ✗ (try to avoid using it with PATCH for example: you trying to update a name of the USER, but lets say name is already same in the DB, so no update required. In that case, we should not throw 304 (NOT_MODIFIED) error code, instead 204(NO_CONTENT) or 200(OK) is more appropriate.)	1. Client makes a GET call, Server returned it with Last-Modified time in header. 2. Client cache the response. 3. Client make a GET call, pass this Last modified time in "If-Modified-Since" header. 4. Server check the particular resource last update time with what client provided, if resource is not updated, server simply returns 304 (NOT_MODIFIED). 5. If Modified, server process the request as usual and returns the new values.

4xx (Validation Errors)

Client need to pass correct request to server

Status Code	Reason	Mostly used in	More details
400	Bad Request	GET, POST, PATCH, DELETE	Client is not passing the required details to process the request.
401	Unauthorized	GET, POST, PATCH, DELETE	Any API, which require Authentication(like Bearer token, Basic authentication etc..) and client try to access it without providing authentication details.
403	Forbidden	GET, POST, PATCH, DELETE	Lets say, only ADMIN can perform certain operation. But if API get invoked apart from ADMIN. We should throw 401 status code as clients (apart from ADMIN) do not have permission to access the resource.
404	Not Found	GET, PATCH, DELETE	The requested resource which client passes, is not found in DB by the server. For ex: GET the user details with ID: 123, but in DB there is not such ID present.
405	Method Not Allowed	GET, POST, PATCH, DELETE	Ex: Hitting GET API, but with POST HTTP Method. In Springboot, dispatcher servlet might throw this error, as control not even reach to controller.
422	Un-processable Entity	GET, POST, PATCH, DELETE	Your application Business validation: Like France Users should not be allowed to open an account. (as country is not supported yet)
429	Too Many Requests	GET, POST, PATCH, DELETE	Lets say: our rule is: 1 user can max make 10 calls in a minute. if User:12345 makes the 11th call in a minute then this 11th call should get failed and we can throw 429 error code.

5xx (Server Errors)
Request got failed at Server, even though client passed the valid request. Means Something wrong at Server.

Status Code	Reason	Mostly used in	More details
500	Internal Server Error	GET, POST, PATCH, DELETE	Generic error code when no more specific error code is suitable.
501	Not Implemented	GET, POST, PATCH, DELETE	API lacks the ability to fulfill the request. Or say, API is in development and in future it will be available.
502	Bad Gateway	GET, POST, PATCH, DELETE	Server acting as a proxy and while calling upstream got invalid response. Example: My application is deployed behind Reverse Proxy (Nginx). If NginX is not able to communicate with my application (because of misconfiguration of port number or something), then it is eligible to throw 502 Bad Gateway.

1xx (Informational)
Interim response to communicate request progress or its status before processing the final request.

Status Code	Reason	Mostly used in	More details
100	Continue	POST	Before sending the request, client check with server, if it can handle the request and ready: 1. Client add few things in the header first, like: - content length : 1048576 - content type : multipart/form-data - Expect: 100-continue 2. Server, checks that in header, 'Expect:100-continue' is present, means, client is just checking. So server validate everything (authentication, authorization, content type, length etc.) 3. If Server is okay, it return 100 CONTINUE status code 4. Client receives it and then invokes the API again without Expect and server process the request.