Http Notes & Spring Boot Response Entity

HTTP Request and Response Format

Request Format:

- **Request Line:** Contains HTTP method (e.g., GET, POST), URL, and HTTP version.
- **Headers:** Provide metadata (e.g., Content-Type, Authorization, User-Agent).
- **Body:** Optional data sent in POST/PUT requests (e.g., JSON, form data).

Response Format:

- **Status Line:** Includes HTTP version, status code (e.g., 200 OK, 404 Not Found), and reason phrase.
- **Headers:** Metadata about the response (e.g., Content-Type, Cache-Control).
- **Body:** Contains the response data (e.g., JSON, text, or binary content).

Form Data Submission Formats

- **application/x-www-form-urlencoded (default):** Encodes form data as key-value pairs (e.g., name=John&age=30) in the request body.
- **multipart/form-data:** Used for file uploads; divides data into multiple parts.
- **text/plain:** Sends form data as plain text (less common).

HTTP Headers

Common Request Headers:

- Authorization: For authentication.
- Content-Type: Format of the request body (e.g., application/json).
- Accept: Desired response format.
- User-Agent: Information about the client.
- Cookie: Sends session data to the server.

Common Response Headers:

- Content-Type: Format of the response body (e.g., application/json).
- Cache-Control: Controls caching behavior.
- Set-Cookie: Sends cookies to the client.

- Location: Used for redirects.

HTTP Status Codes

1xx: Informational (e.g., 100 Continue). **2xx:** Success (e.g., 200 OK, 201 Created, 204 No Content). **3xx:** Redirection (e.g., 302 Found, 304 Not Modified). **4xx:** Client Error (e.g., 400 Bad Request, 401 Unauthorized, 404 Not Found). **5xx:** Server Error (e.g., 500 Internal Server Error, 503 Service Unavailable).

Example Scenario

Imagine a client (a web browser) is sending a request to a server to log in, and the server responds accordingly.

HTTP Request

Here's what a client might send when the user submits login credentials:

Request Line:

```
POST /api/login HTTP/1.1
```

- **Method:** POST indicates that data is being sent to the server.
- **Path:** /api/login is the resource being accessed.
- **HTTP Version:** HTTP/1.1.

Headers:

```
Host: example.com
Content-Type: application/json
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML
Content-Length: 51
```

```
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36
```

- **Host:** Specifies the domain of the server.

- **Content-Type:** Declares that the body contains JSON data.
- **User-Agent:** Identifies the client software.
- **Content-Length:** Indicates the size of the request body (in bytes).

Body:

```json
{
  "username": "john.doe",
  "password": "12345"
}
```

- The body contains the login credentials in JSON format.

HTTP Response

Here's what the server might send back as a response.

Status Line:

```
HTTP/1.1 200 OK
```

- **HTTP Version:** HTTP/1.1.
- **Status Code:** 200 OK indicates the request was successfully processed.

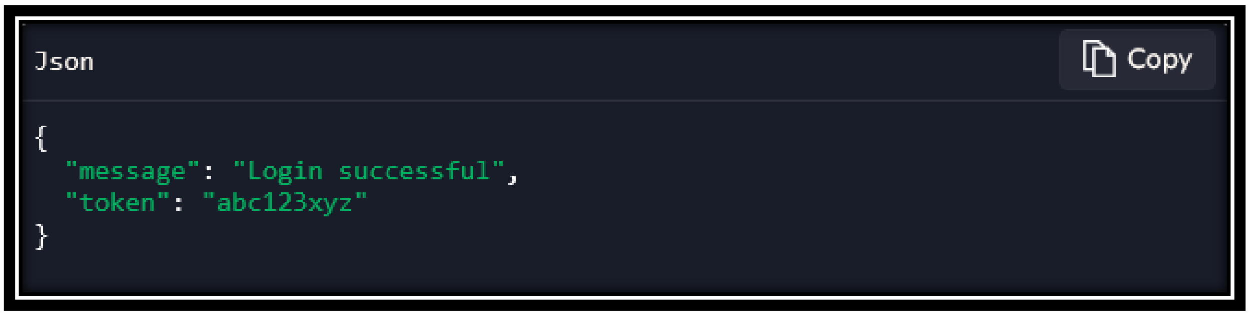
Headers:

```
Content-Type: application/json
Set-Cookie: sessionId=abc123xyz; Path=/; HttpOnly
Content-Length: 60
Cache-Control: no-cache
```

- **Content-Type:** Declares that the response body is JSON.
- **Set-Cookie:** Sends a session cookie to maintain user authentication.
- **Content-Length:** Specifies the size of the response body.

- **Cache-Control:** Prevents the response from being cached.

Body:

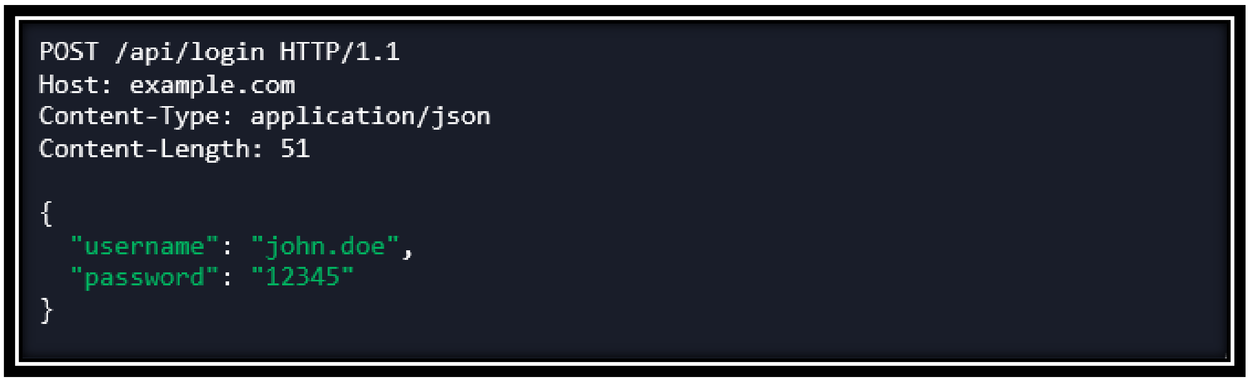


```json
{
  "message": "Login successful",
  "token": "abc123xyz"
}
```

- The body contains a success message and an authentication token.

Summary of Request and Response

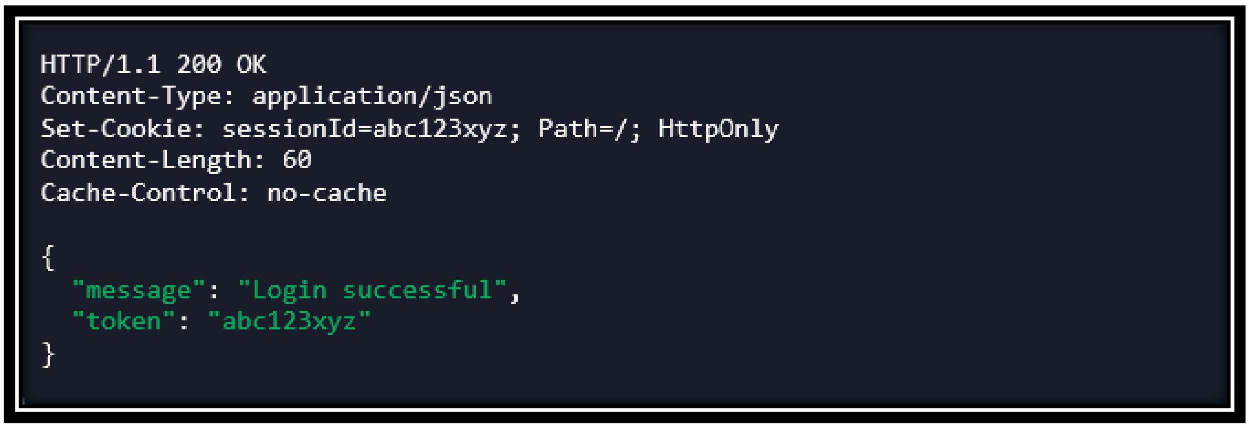
Request:



```
POST /api/login HTTP/1.1
Host: example.com
Content-Type: application/json
Content-Length: 51

{
  "username": "john.doe",
  "password": "12345"
}
```

Response:



```
HTTP/1.1 200 OK
Content-Type: application/json
Set-Cookie: sessionId=abc123xyz; Path=/; HttpOnly
Content-Length: 60
Cache-Control: no-cache

{
  "message": "Login successful",
  "token": "abc123xyz"
}
```

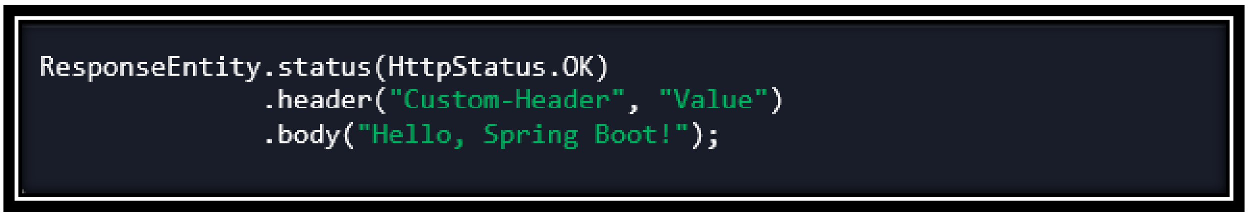
Http Notes & Spring Boot Response Entity

ResponseEntity in Spring Boot

- Represents the full HTTP response (status, headers, body).
- Customizable with methods like .status(), .header(), .body().

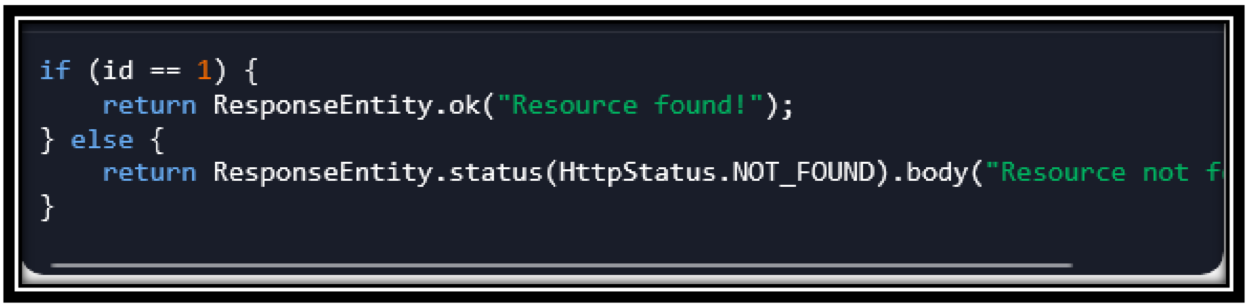
Examples:

1. **Custom Status and Headers:**



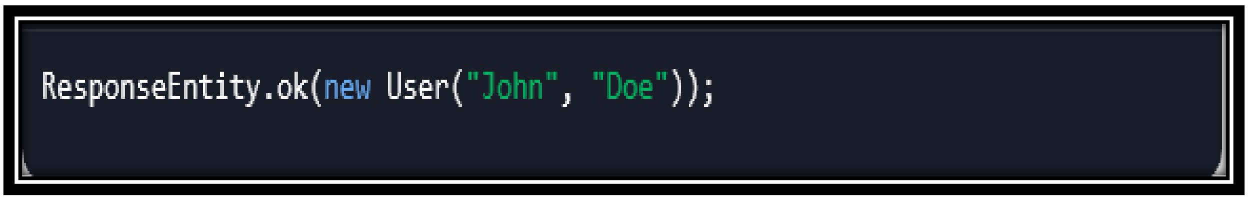
```
ResponseEntity.status(HttpStatus.OK)
              .header("Custom-Header", "Value")
              .body("Hello, Spring Boot!");
```

2. **Handling Different Status Codes:**



```
if (id == 1) {
    return ResponseEntity.ok("Resource found!");
} else {
    return ResponseEntity.status(HttpStatus.NOT_FOUND).body("Resource not f
}
```

3. **Returning JSON (with Jackson):**



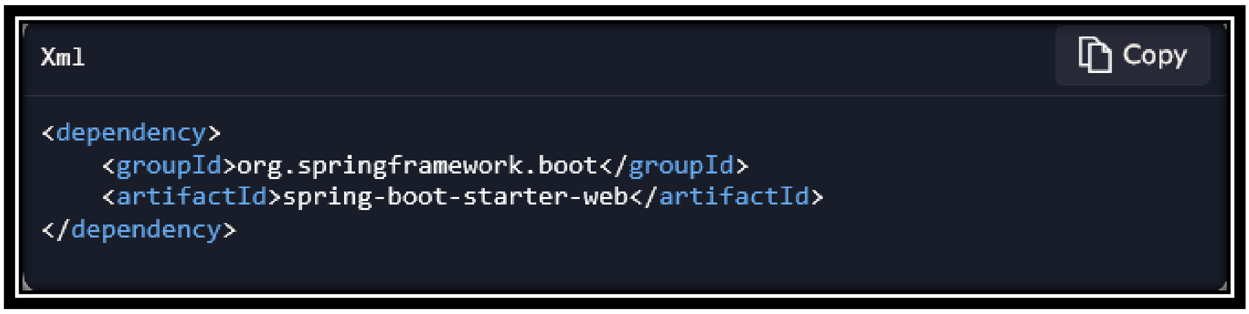
```
ResponseEntity.ok(new User("John", "Doe"));
```

POJO Conversion to JSON

- **Jackson (default library in Spring Boot):**
    - Automatically converts POJOs to JSON.
- **Works with:**
    - ResponseEntity
    - @ResponseBody
    - Directly returning an object in a @RestController.

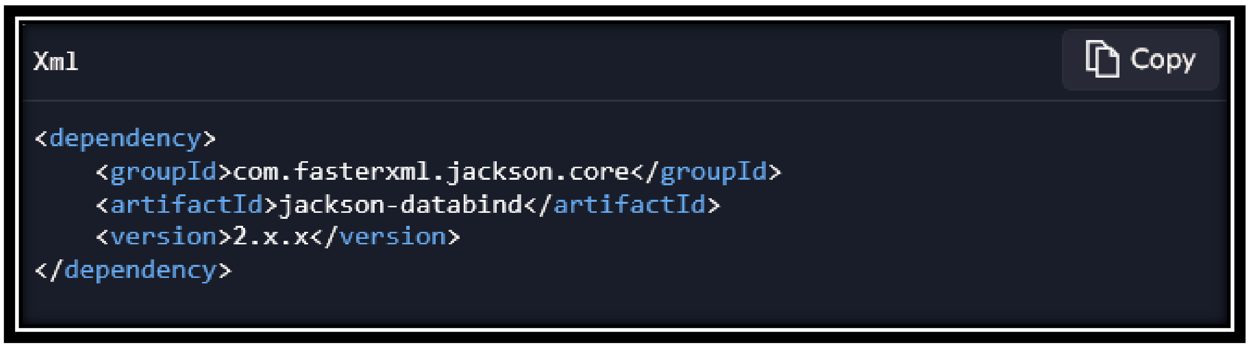
Dependencies

Jackson Library: Included in spring-boot-starter-web dependency.



```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Can also be added directly with:



```xml
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.x.x</version>
</dependency>
```

Question) Automatic Conversion of POJO to JSON and vice Versa only Happens with @RestController Annotation?

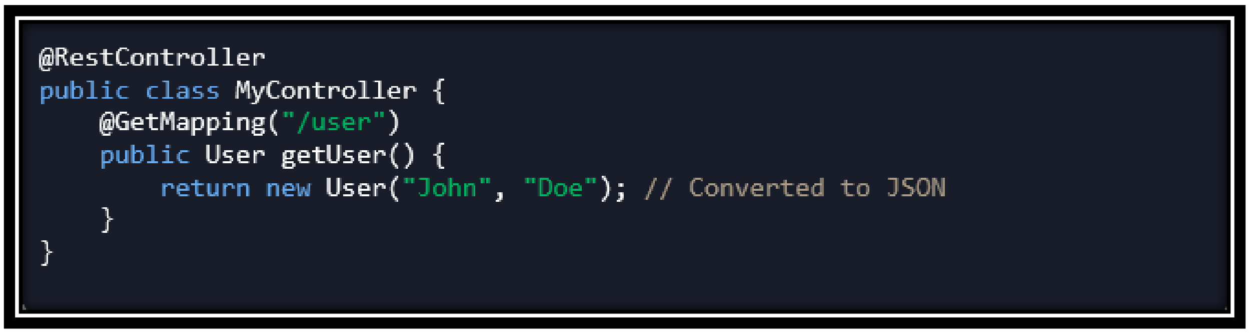
Ans: Not exactly! While the **automatic conversion of POJOs to JSON** in Spring Boot does work seamlessly with the **@RestController annotation**, it's not exclusive to it.

Here's how it works:

1. Using @RestController

When you annotate a class with @RestController, Spring automatically assumes that all methods within the class will produce RESTful responses. The **POJO-to-JSON conversion** happens automatically for any method returning an object.

Example:



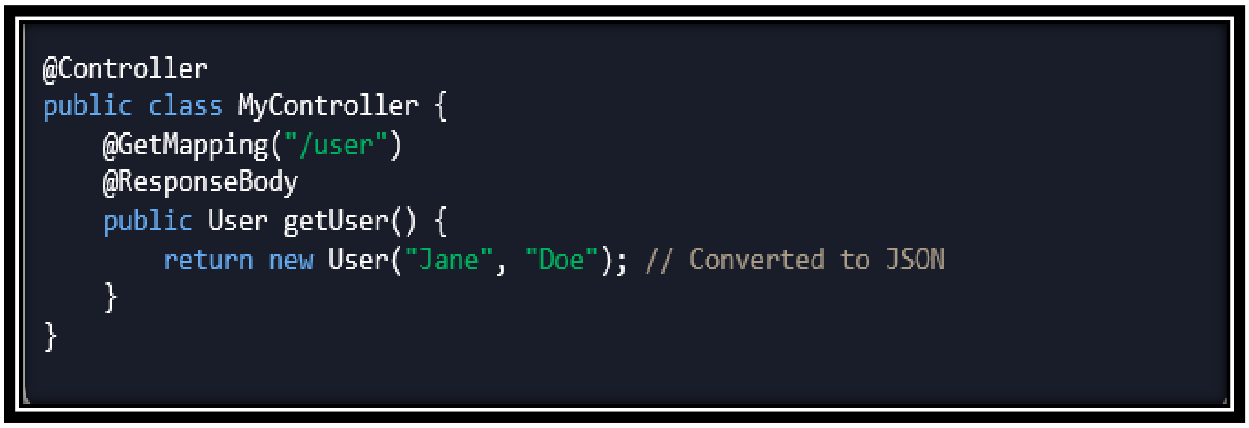
```java
@RestController
public class MyController {
    @GetMapping("/user")
    public User getUser() {
        return new User("John", "Doe"); // Converted to JSON
    }
}
```

Jackson handles the conversion, and the response is sent as JSON.

2. Using @ResponseBody

The @ResponseBody annotation is another way to enable automatic conversion. It binds the return value of a method directly to the HTTP response body, bypassing any view resolution mechanism.

Example:



```java
@Controller
public class MyController {
    @GetMapping("/user")
    @ResponseBody
    public User getUser() {
        return new User("Jane", "Doe"); // Converted to JSON
    }
}
```

In this case, the controller doesn't need to be annotated with **@RestController**. The use of **@ResponseBody** achieves the same result.

---

3. Implicit Behavior

If you use **@Controller** without **@ResponseBody**, the automatic conversion will **not happen** unless you explicitly configure it. In this case, Spring expects the returned value to map to a view (e.g., a JSP).

Key Role of Jackson

The automatic conversion to JSON is possible because:

- **Spring Boot includes Jackson** as the default JSON library (via spring-boot-starter-web).
- Jackson is configured out-of-the-box to serialize POJOs.