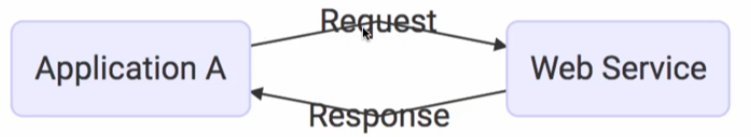
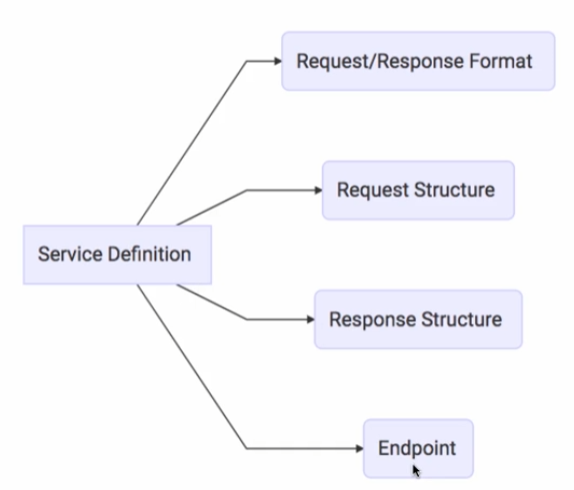
**Udemy Microservices Spring Boot Notes**

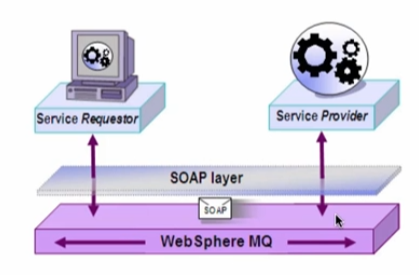
* **Web Service (W3C Definition)** – Software system designed to support interoperable (platform independent) machine-to-machine (or app to app) interaction over a network. Request & Response format should also be platform independent. (XML or JSON)



* **Service Definition** is the contract b/w the Service Provider & Service consumer.
* Every Web Service offers a **Service Definition** to specify the Request & Response format to other applications. It specifies structure of the request & response. It also specifies the endpoint to call the service.



* **Request** is the input to a web service & **Response** is the output from the web service.
* **Message Exchange format** is the format of the Request & the Response like XML, JSON etc.
* **Service Provider or Server** is the one which hosts the web service. **Service Consumer or Client** is the one which consumes the web service.
* Endpoint defines what URL the service is exposed at. Transport defines how a service is called i.e., Is the service exposed over Internet or is the service exposed over Queue. Two popular formats are HTTP & MQ.



* Different types of Web Services

1. SOAP web service
2. REST web service

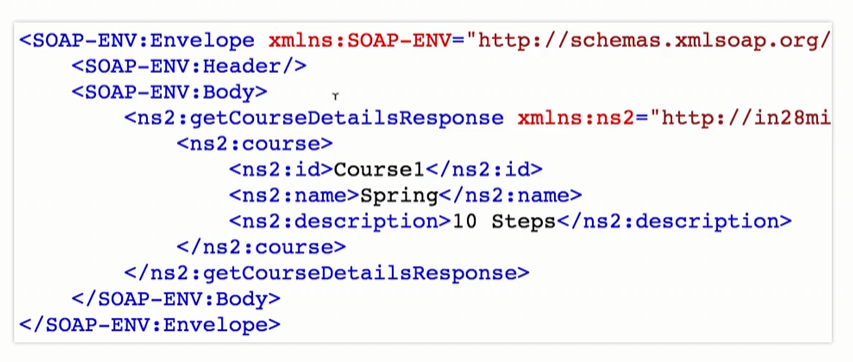
* One thing to remember is the fact that SOAP & REST are not really comparable. REST defines an architectural approach, whereas SOAP poses restrictions on the format of the XML, which is exchanged b/w the service provider & the service consumer.

|  |
| --- |
|  |

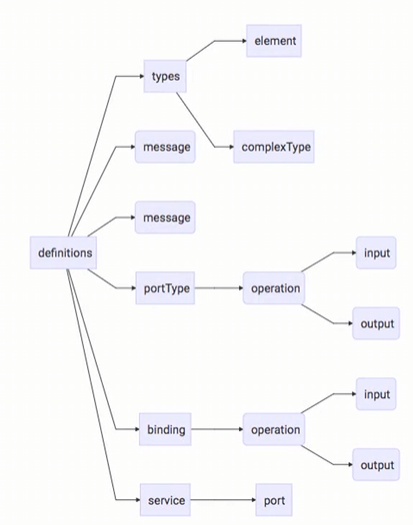
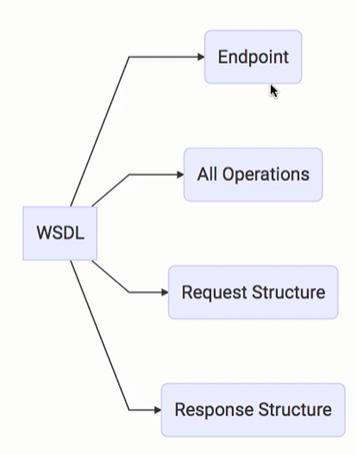
**SOAP Web Service**

* SOAP stands for Simple object access protocol.
* SOAP defines a specific XML request & response structure, First, we have to create a **SOAP envelope** which contains a **SOAP header** & a **SOAP body**. The SOAP header contains meta information like authentication, authorization, signatures etc. & SOAP body is where we put the real content of the request or response.

|  |
| --- |
|  |



* SOAP doesn’t pose any restrictions on the transport. You can either use HTTP or MQ.
* In SOAP, the Service definition is done using **Web Service definition language (WSDL).** WSDL defines the endpoint (where our service is exposed at), all the operations & specifies the request & response structure.



**REST Web Service**

|  |
| --- |
|  |

* REST stands for REpresentational State Transfer.
* The key thing about REST services is the fact that they would want to make best use of HTTP
* HTTP stands for Hyper Text Transfer protocol.
* Using HTTP / Request Methods, we can indicate what action we’re doing. For e.g., Using GET, we can fetch the details of something, Using POST, we can create something, Using PUT, we can update something.

* Using HTTP response status code, we can indicate if the request method calls are successful or not.
* RESTful web services try to define services using the different concepts that are already present in HTTP. The most important abstraction in REST is something called a Resource.

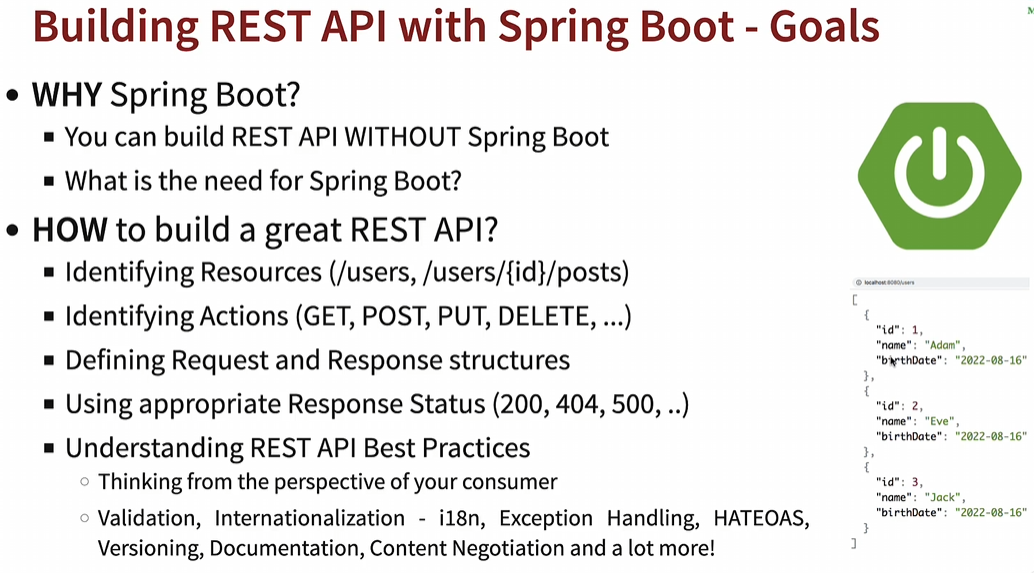
* A Resource is anything that we want to expose to the outside world through our application by assigning a URI to each resource. For e.g., “/user/todos”, “/user/todos/1”
* REST doesn’t worry about how we’re representing our resource i.e., is it XML, HTML, JSON ?
* Imp. Points to remember

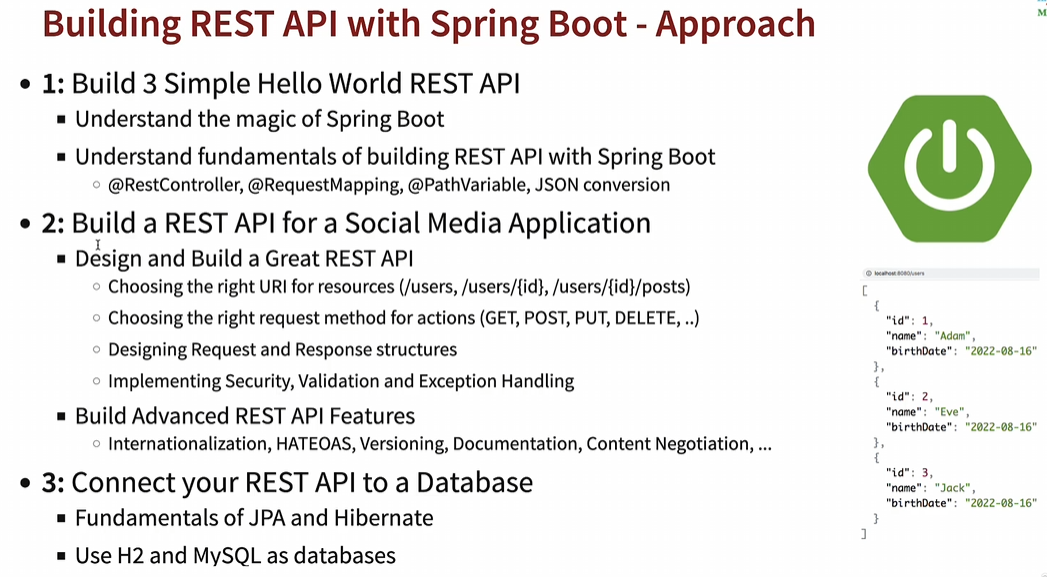
|  |
| --- |
|  |

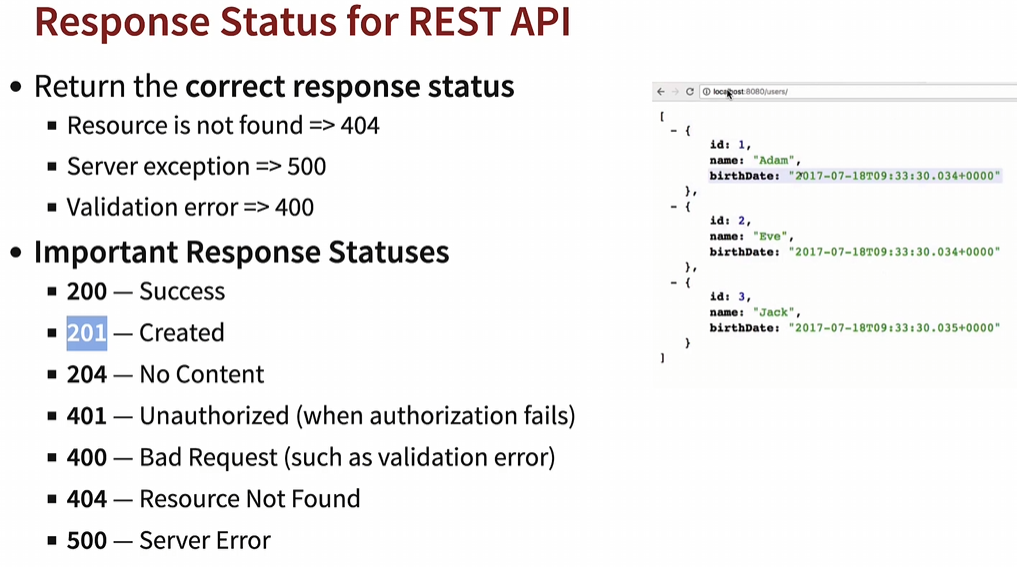
1. In REST, we have to think in terms of resource.
2. REST makes use of HTTP defined methods & status codes.
3. In REST, we don’t have any restriction on the data exchange format though JSON is very popular.
4. In REST, transport is always HTTP.
5. Since REST is completely built on the top of HTTP, there is no standard service definition which is attached with REST. This can be a drawback in some scenarios because when a client wants to consume a service, it needs to understand the request format & response format.
6. Web Application Definition language (WADL), Swagger are some of the formats which we can specify our RESTful web services.

**REST Vs SOAP**

|  |  |  |
| --- | --- | --- |
| **No.** | **SOAP** | **REST** |
| 1. | **Restrictions:** It is a format of an XML. | **Architectural Approach:** REST is an architectural style. |
| 2. | **Data exchange format:** In SOAP, the data exchange format is always XML with SOAP envelope header & body. Both request & response should adhere to the SOAP structure. | In REST, there is no strict data exchange format. We can exchange JSON or XML or any other format we would like to use. |
| 3. | **Service definition:** SOAP uses WSDL (Web Service Definition Language) | REST doesn’t have a standard definition language. WADL, Swagger etc. are some of service definitions for RESTful web services. |
| 4. | **Transport Protocol:** SOAP doesn’t pose any restrictions at all. It can use Web/HTTP or MQ. | REST is very specific about making the best use of HTTP protocol. |
| 5. | **Ease of Implementation:** SOAP services are a bit complex to implement.  Also, with SOAP, we have to define WSDL & there are lots of complexity associated with parsing XMLs. | RESTful services are typically easier to implement than SOAP. They are typically based on JSON which is an easy format to parse & do things.  Also, we don’t really need to mandatorily define a service definition. |

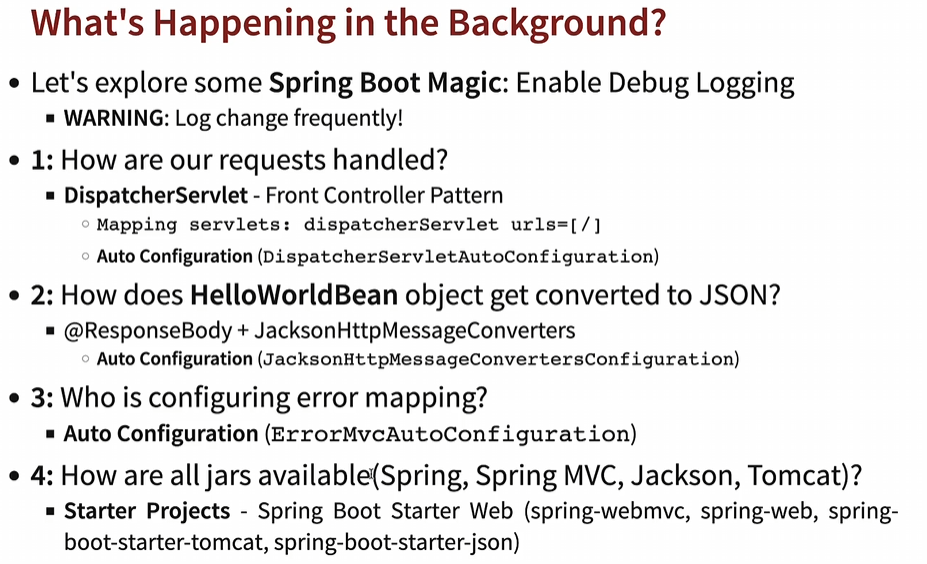
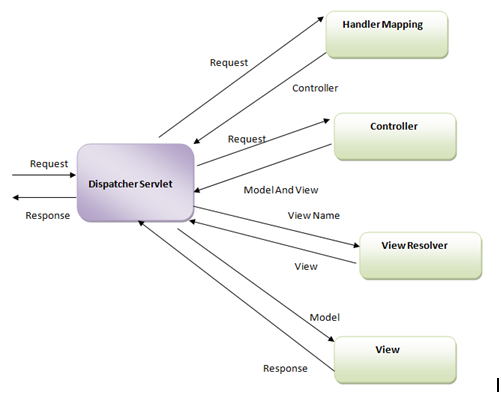






* We can enable debug logging for springframework by adding below property to **application.properties** to view the AutoConfiguration beans & Dispatcher servlets

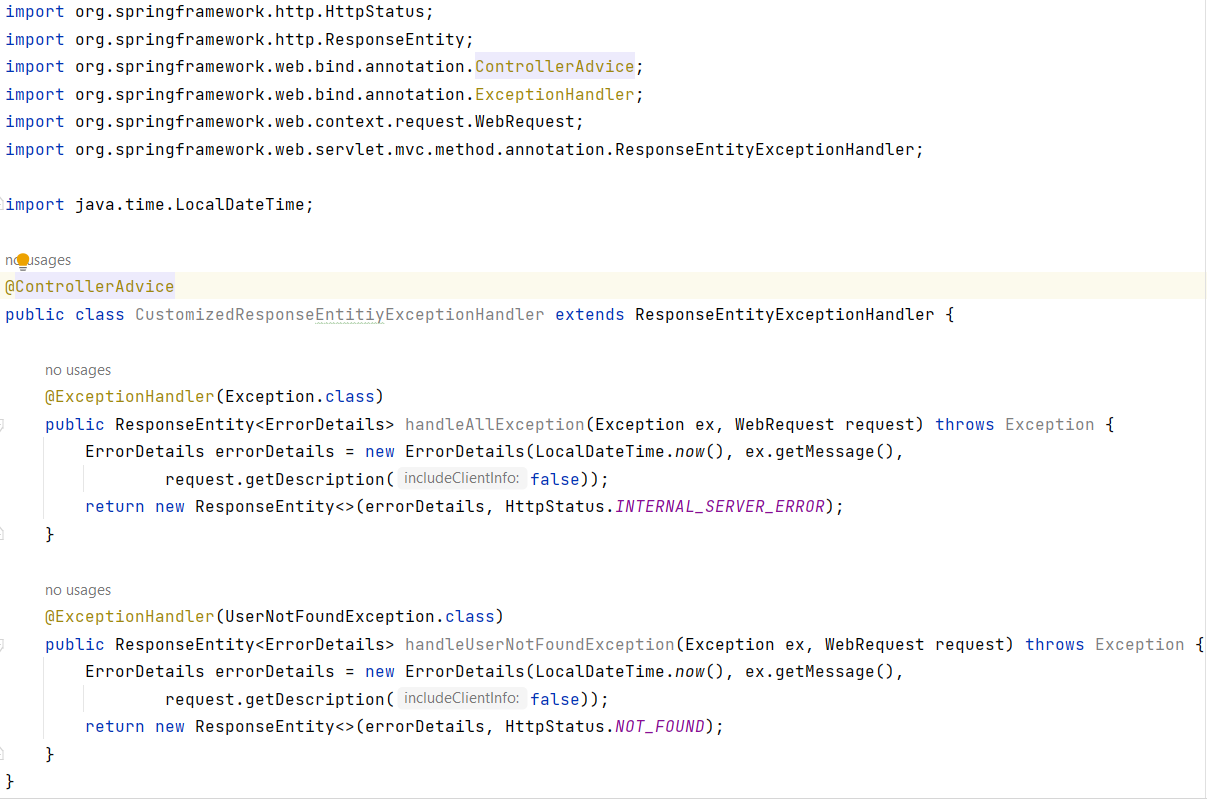
*logging.level.org.springframework=DEBUG*



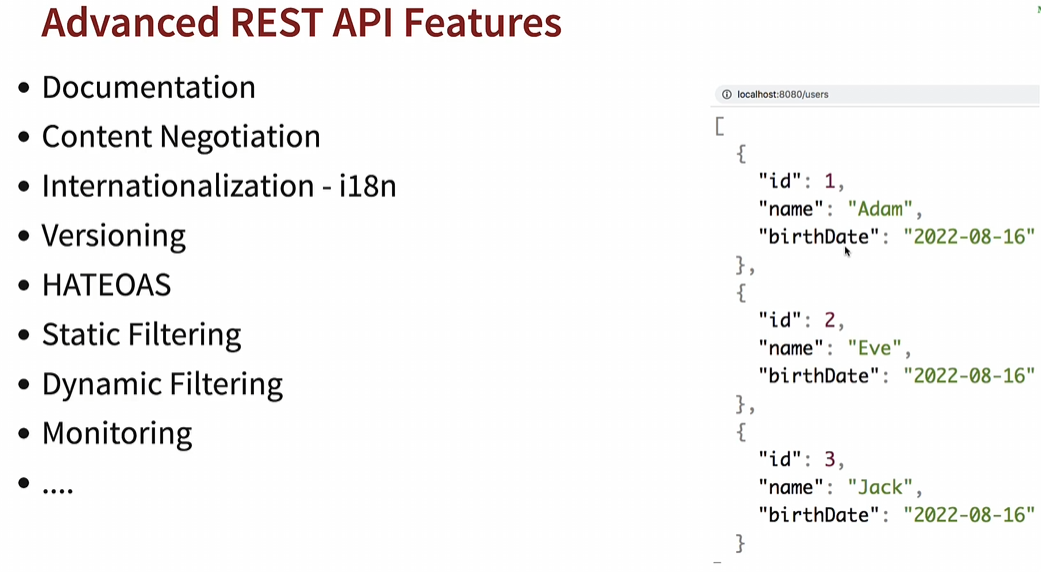
* Dispatcher servlet acts as the front controller for Spring-based web application i.e., all the incoming requests first go to the dispatcher servlet which maps the request to the right controller & determines the view & returns the response object to the client.
* Dispatcher servlet handles an incoming HttpRequest, delegates the request & processes that request according to the configured **HandlerAdapter** interfaces that have been implemented within the spring application with accompanying annotations specifying handlers, controller endpoints & response object.

**Implementing Generic Exception Handling for all Resources/URI**

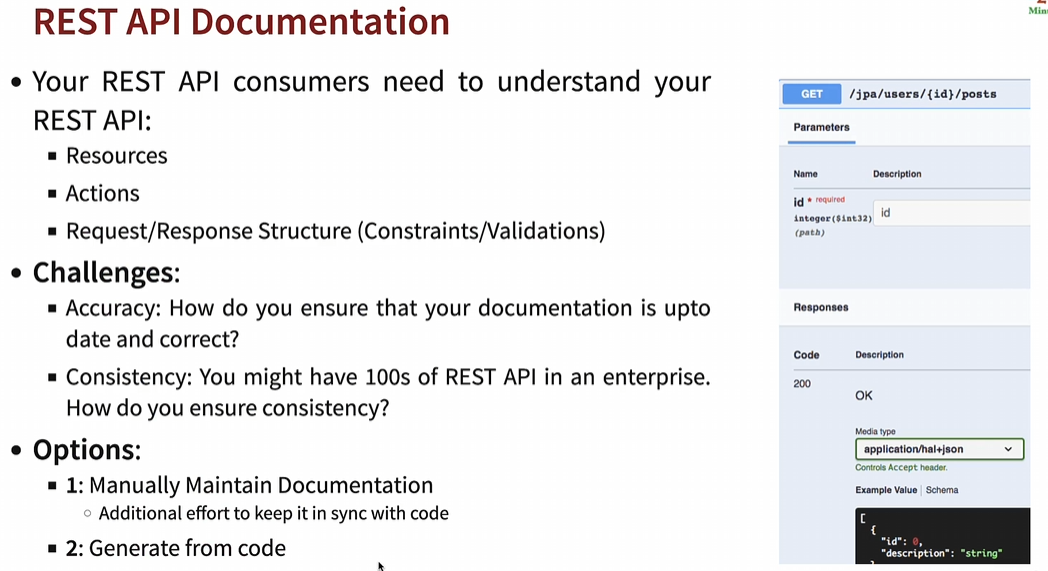
* If our controller methods returning ResponseEntity class object, then we can extend **ResponseEntityExceptionHandler** class & provide exception Handler methods for all exceptions or specific exceptions.



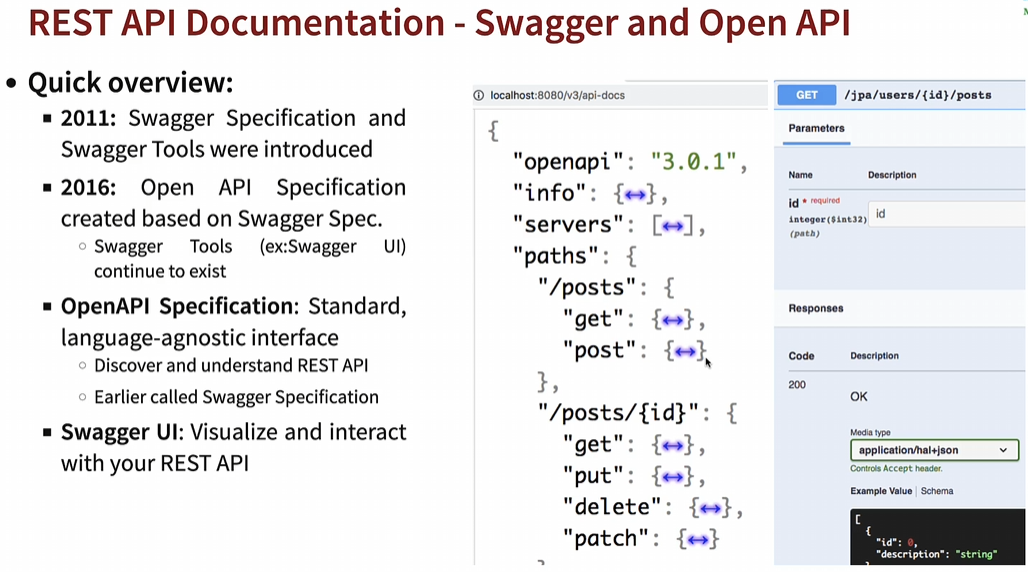
* We need to mark the class with **@ControllerAdvice** (it is specialization of @Component for the classes that declare **@ExceptionHandler** methods to be shared across multiple @Controller classes).



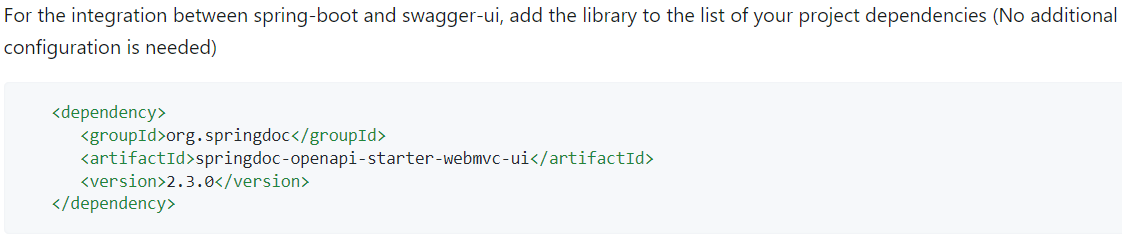
**1. Rest API Documentation**

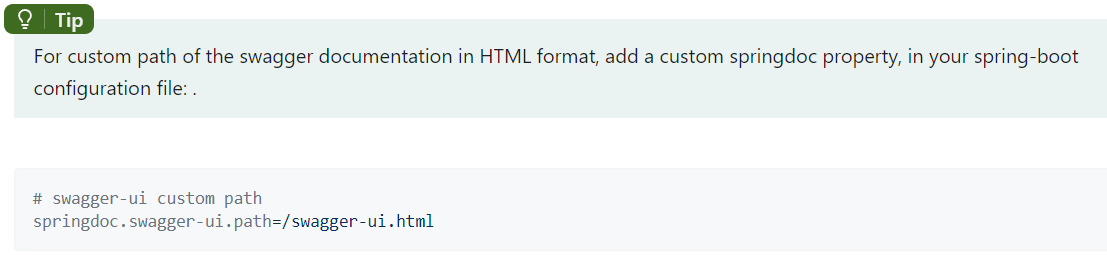


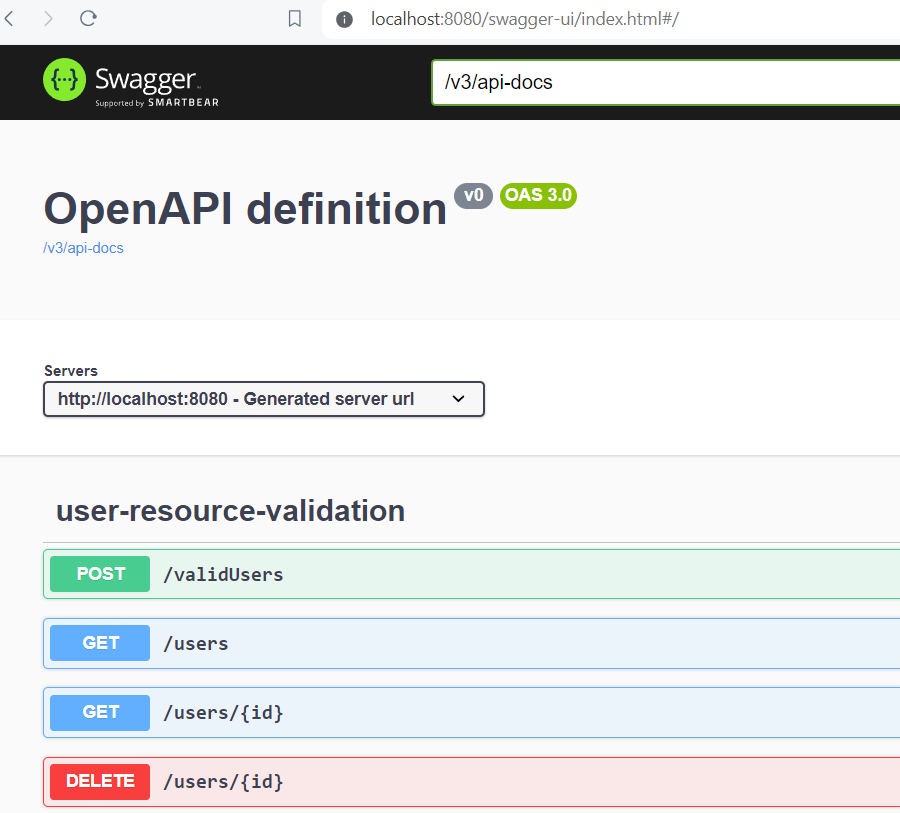
* Here let’s explore how to generate API documentation from Code – Swagger & Open API



Simple step to configure OpenAPI in our application. Just add openAPI dependency in pom.xml & you will get all the functionality along with Swagger UI. Springdoc OpenAPI Link - <https://springdoc.org/>

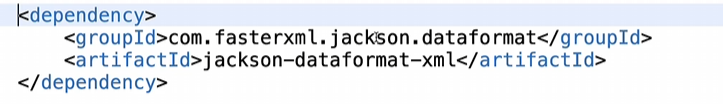


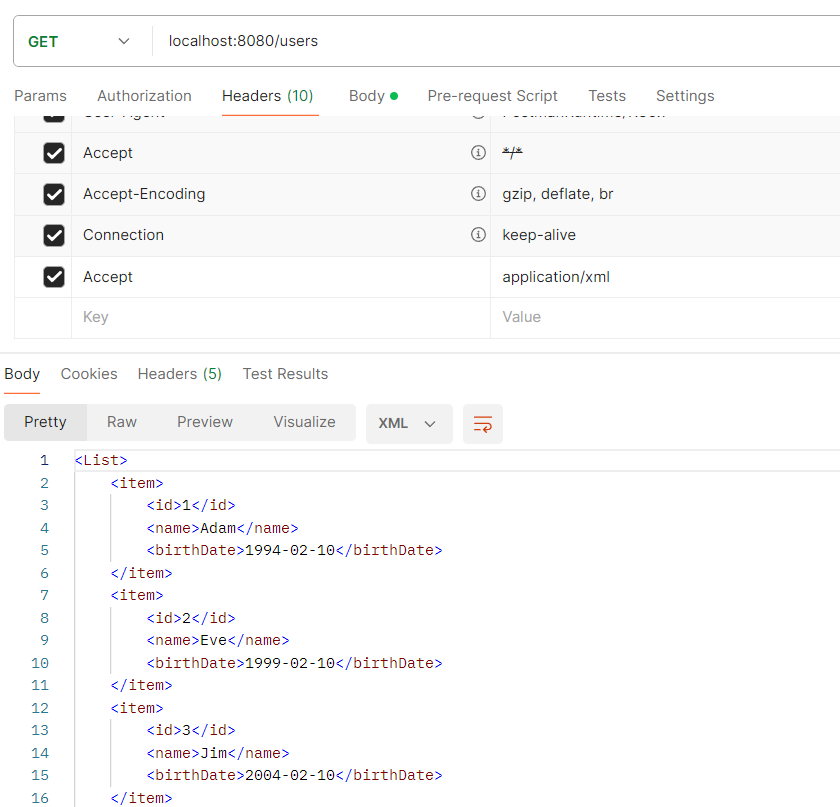


**2. Content Negotiation**

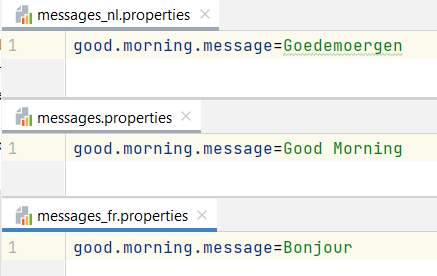
* For the same resource/URI, we might have different representation in terms of content type or different language like English, Dutch etc.
* In Content Negotiation, the consumer & the service provider negotiate with each other on what kind of representation consumer would want like XML, JSON response or language like Dutch etc.
* E.g., Accept header (MIME types – application/xml, application/json, …)
* E.g., Accept-Language header (en, nl, fr, …)
* Add a simple dependency to pom.xml to support different kind of representations. For XML, add

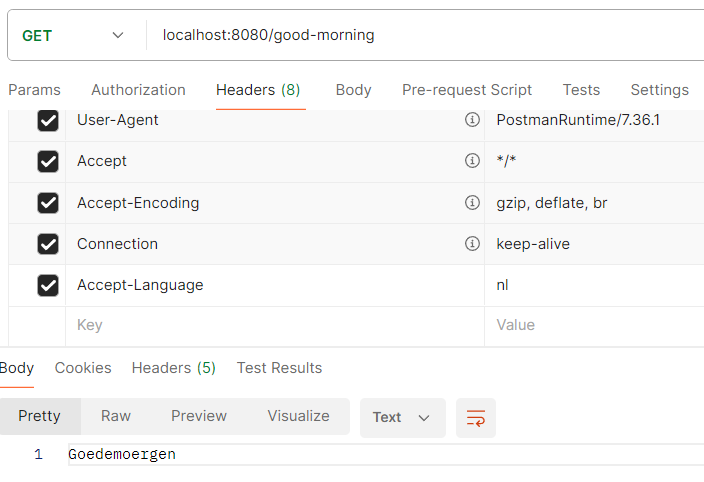
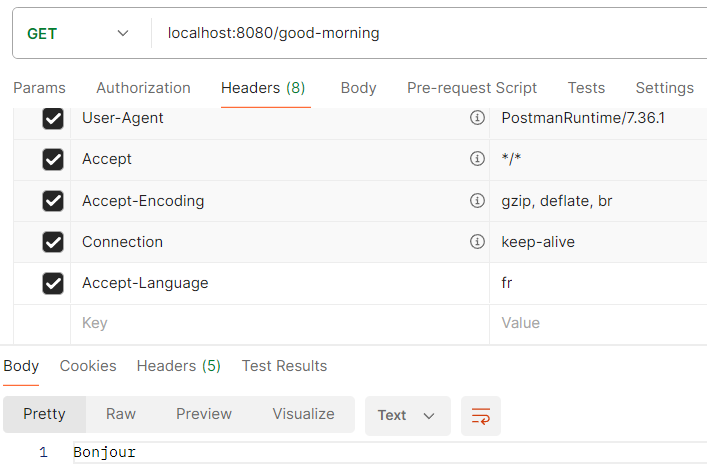




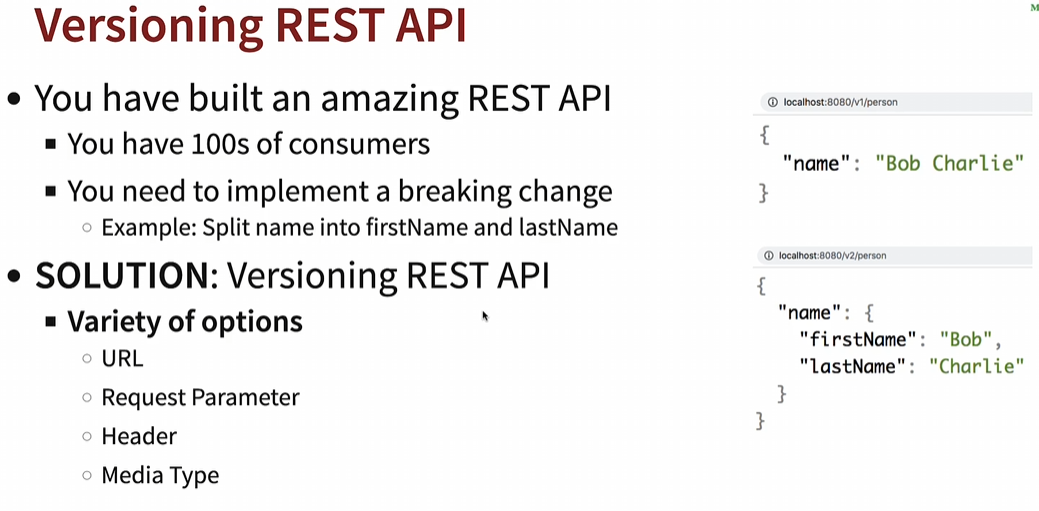
**3. Internationalization (i18n)**

* Typically, HTTP Request Header (Accept-Language) is used. Accept-Language indicates natural language & locale that the consumer prefers.
* For e.g., en – English, nl – Dutch, fr – French etc.
* To add this functionality, simple create message.properties file for en, message\_nl.properties file for Dutch & so on & add all the required translations.



**4. Versioning**



1. **URI Versioning (used by Twitter)**

* <http://localhost:8080/v1/person>
* <http://localhost:8080/v2/person>

1. **Request Parameter versioning (used by Amazon)**

* <http://localhost:8080/person?version=1>
* <http://localhost:8080/person?version=2>

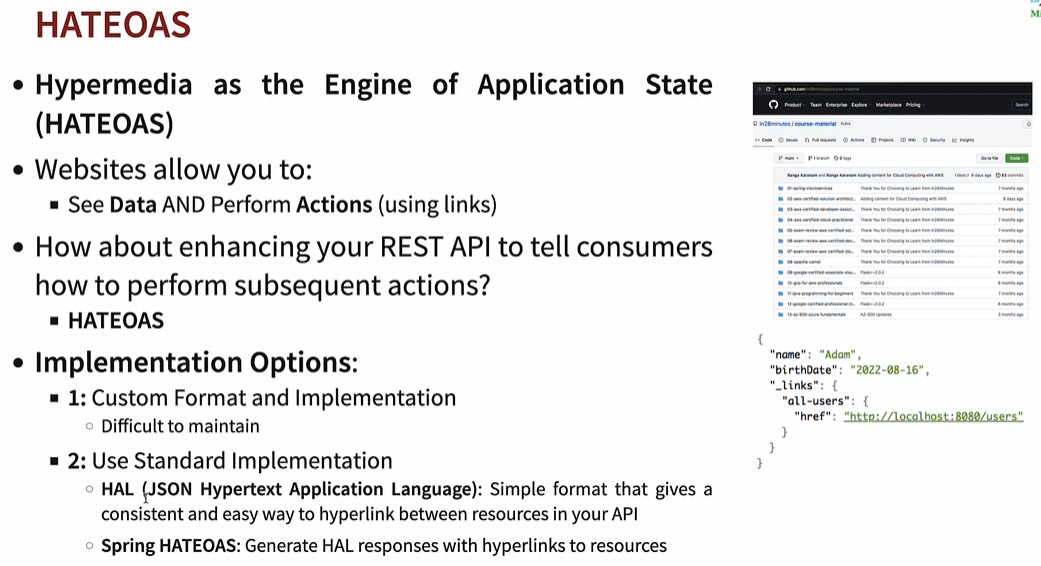
1. **(Custom) headers versioning (used by Microsoft)**

* SAME-URL headers=[X-API-VERSION=1]
* SAME-URL headers=[X-API-VERSION=2]

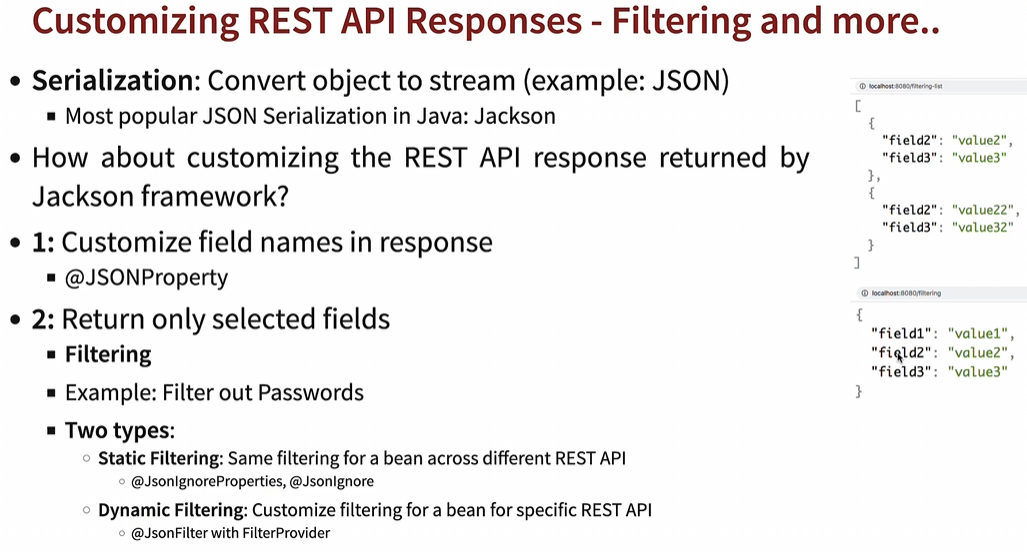
1. **Media type versioning (a.k.a “content negotiation” or “accept header”) - Github**

* SAME-URL produces=application/vnd.company.app-v1+json
* SAME-URL produces=application/vnd.company.app-v2+json

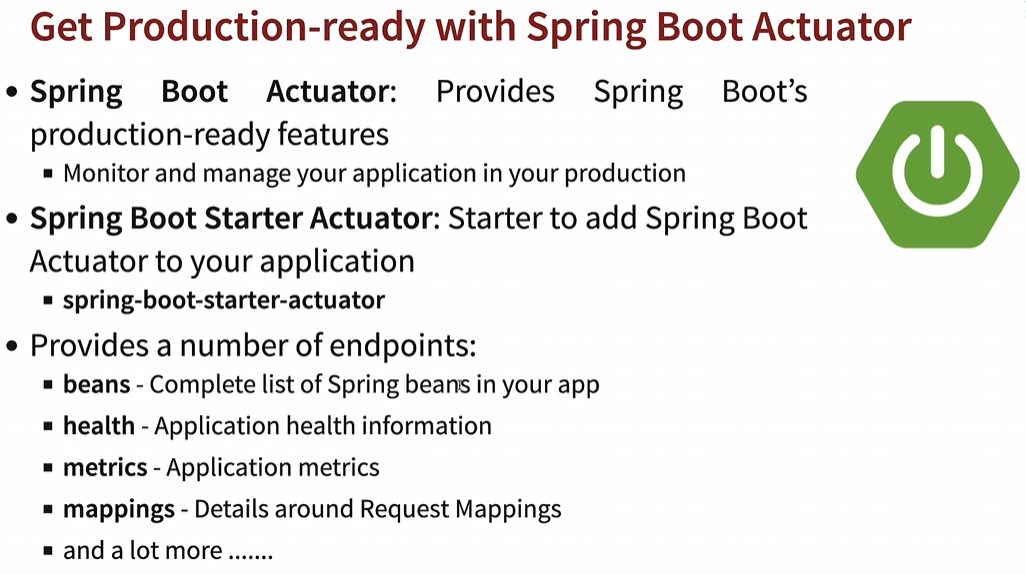
**5. Hateoas for REST API**



**6. Static & Dynamic Filtering for REST API**



**7. Monitoring APIs with Spring Boot Actuator**



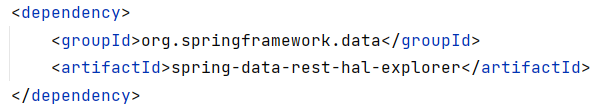
* By default, the only thing which actuator exposes is the health of the application. If you want to expose more information as part of actuator, you add “management.endpoints.web.exposure.include=\*” to application.properties file.

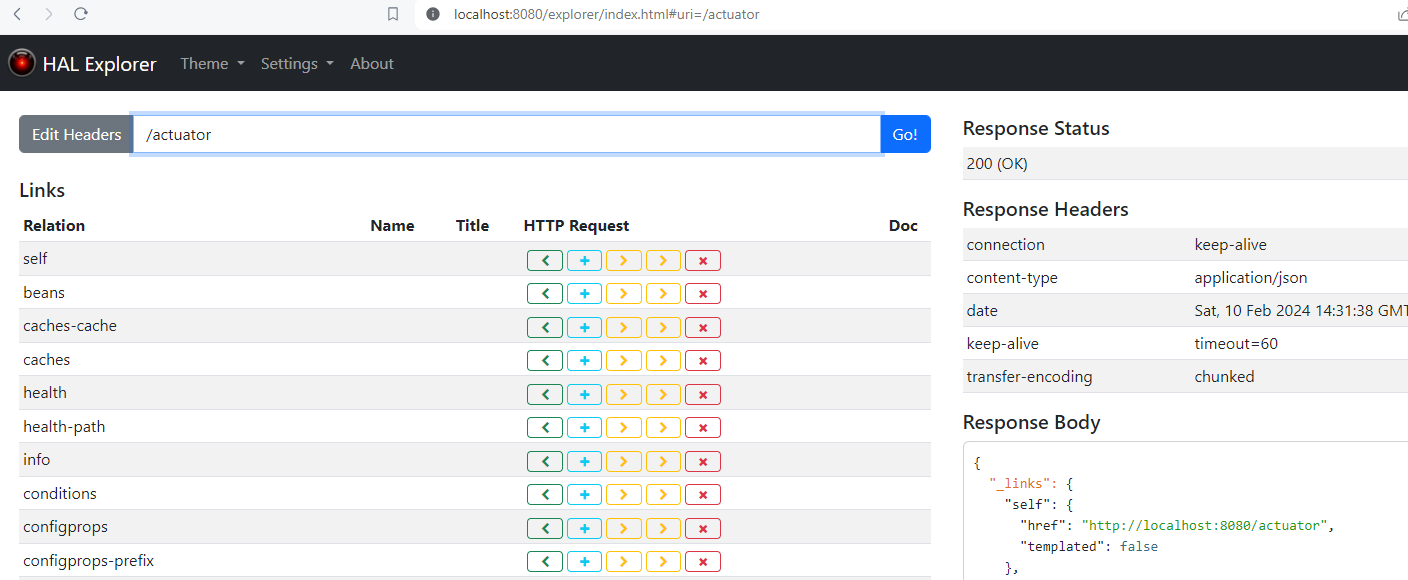


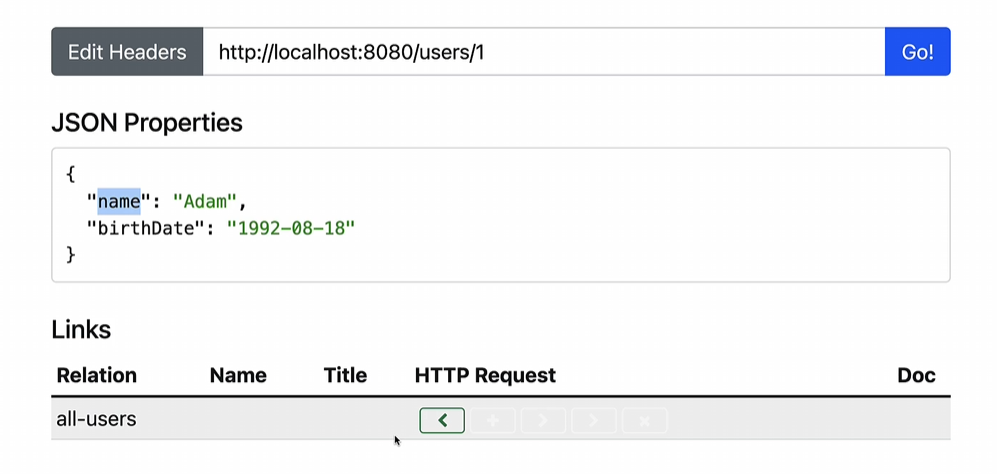


* **HAL Explorer** is an API explorer for restful Hypermedia APIs, which are using **HAL (Hypertext Application language)**



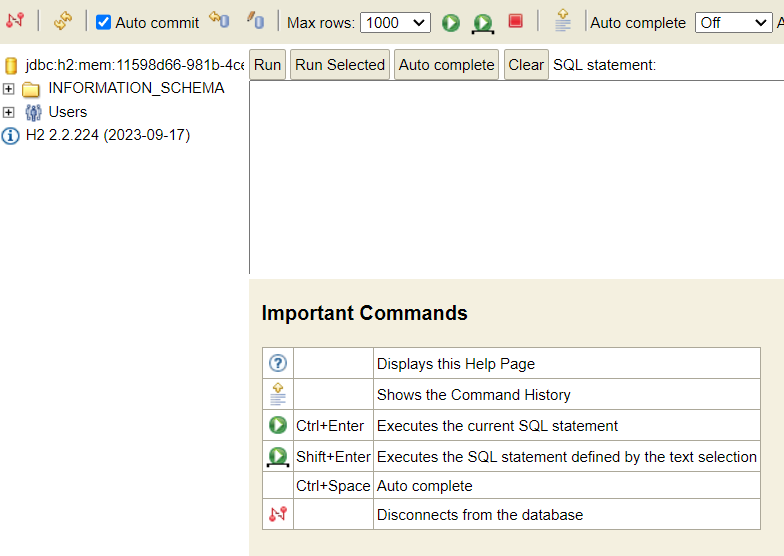
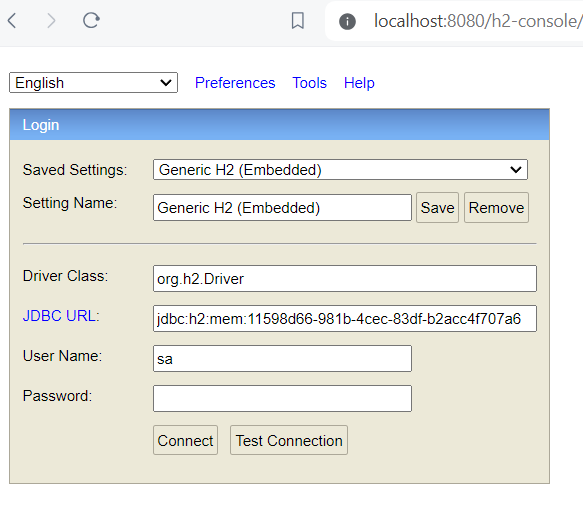




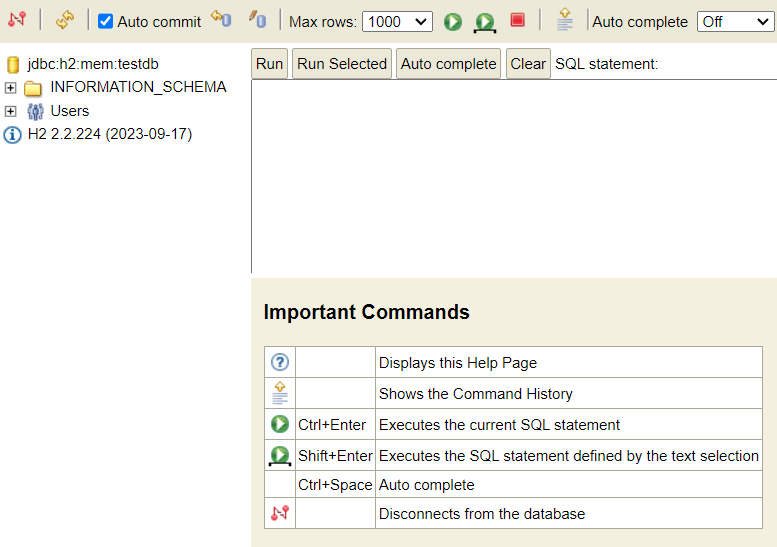


**Spring JPA & Hibernate**

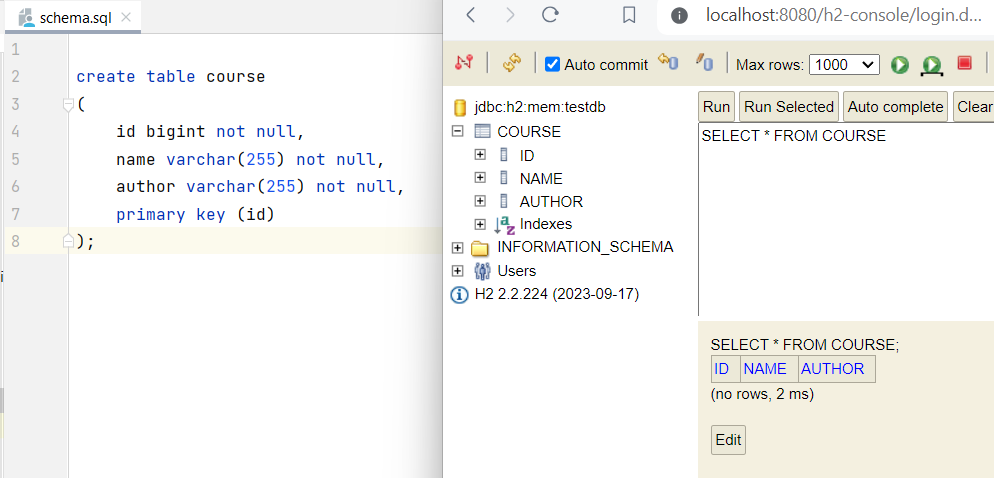
* If we’re using H2 database, we can access h2 console by enable property “**spring.h2.console.enabled=true**” in application.properties & we can access the console by: <http://localhost:8080/h2-console>
* In JDBC Url, add the **url you will find in IDE console i.e., dynamic JDBC URL** & login



* We can configure static JDBC url by providing jdbc url as “**spring.datasource.url=jdbc:h2:mem:testdb**” in **application.properties** file.

* Whenever we’re making use of Spring data JPA starter, it would automatically pickup the file called schema.sql to create new table.



**Spring JDBC, JPA, Spring Data JPA**

