Spring Boot

Dispatcher Servlets

1. **What is it….?** when a URL is hit the application receives it first request at the root of application and this is accepted by a Controller initially(FrontController) then forwards the request to respective Controller. FrontController is called Dispatcher. Spring Framework will have a in-built and we can reuse for our specific project.
2. **How to config it….?** Done using SpringBoot AutoConfiguration
3. **What it do..?** It is accessed from root of web-application [/]
4. **How bean convert to JSON…?** Done using SpringBoot AutoConfiguration by messageConverter and JacksonBeans are initialized.
5. **Error map config….?** Done using SpringBoot AutoConfiguration.

DAO-data access object / Repository

The level which accepts the flow of data to & from DB.

RestServices(Step 9 & 10)

POST

We will must create a page or a response to indicate that the service had changed the application and then we update the status code also.
So we make a created response and further this can be created as page or to update component in the view.

Exception (Step 11 to 13)

Here we used 404 Exception to mention that the User is not found instead of default 401(unauthorized).
Exception structure can be customized.

Validation

Validation annotation is present SpringBoot Starter Web(Validation API) <- @Valid.

HATEOAS(Hypermedia As Engine of Application State)

Link to the user and further details are provided
Retrieve all the users.
We have used for **Resource** we used (EntityModel) defined using <- Class.of(instance) and further use **Linkto** to connect the resource to the corresponding Class.
Now the retrieve method return object changes to **EntityModel<Customer>** from Customer object.

Internationalization(I18N)

Implementing the application to give the responses as per the client location.
For this we add **SessionLocaleResolver** and **ResourceBundleMessageSource** beans in Main class and add local properties in resources and further connect as the Accepted-Language Header changes.

```
##### Configuration
- LocaleResolver
   - Default Locale - Locale.US
- ResourceBundleMessageSource

##### Usage
- Autowire MessageSource
- @RequestHeader(value = "Accept-Language", required = false) Locale locale
- messageSource.getMessage("helloWorld.message", null, locale)
```

If we do not want the

paramenters for the methods we update getMessage(“”,null, LocalecontextHolder.getlocale()); and also the **SessionLocaleResolver** Bean to **AcceptLocalResolver**.

Content Negotiation

Representation of data is handled. With different formats such as XML, JSON, Text

Implementation

Add the corresponding dependencies to the gradle/maven/jar to the project.

Documentation using Swagger

We use swagger to document and provide this to the client to check the API request and responses.

Implementation:

1. Spring 3 does not support springfox-swagger2 (v2) so we add the dependencies for Swagger-3.
2. Created own API with custom description
3. Unable to define request and response (types MIME Types) using addProduce & addConsume(See later)

Maintaining the Documentation(Using HAL, and Spring Actuator)

1. Enable endpoints exposure in application.properties
    1. Check the results in: localhost:8080/actuator

Filter API

Used when the field has to be ignored from the client or viewed in front-end.

Implementation ways:

A. Static(Always same fields are ignored)
   A. Implementing In Filtering controller.
   B. Customer controller for pass(password filed).

B. Dynamic
   A. Defining MappingJackson, FilterProvider and FilterBean customized to corresponding method.

Versioning

Using **Header** Options:

* Multiple ways of versioning by distinguishing different methods
* **Request parameters (paramPerson{V1})**
  * Sending the version parameters in the url will change the version of the whole Response
  * URL = localhost:8080/person/param?**version=1 changes the version**
  * **Annotation: @GetMapping(value = "/person/param", params =“version=2")**
* **Headers (public PersonV2 headerPerson{V1})**
  * User header of the request
  * **Annotation: @GetMapping(value = "/person/header", headers =“X-API-VERSION=1“)**
* **Producers (public PersonV1 producerPerson{V1})**
  * **Annotation: @GetMapping(value =“/person/produces”, produces = “"application/vnd.company.app-v2+json”)**

Real Cases of Versioning:

```
- Media type versioning (a.k.a “content negotiation” or “accept header”)
  - GitHub
- (Custom) headers versioning
  - Microsoft
- URI Versioning
  - Twitter
- Request Parameter versioning
  - Amazon
```

Security

Different way to secure API is

1. Basic Authentication: USER ID and password
    1. Use dependency: Spring-boot-starter-security
2. Digest Authentication: password
3. OAuth :

JPA -> Java Persistent API

Create entity and create a JPA -> Repository / DB
JPA is part of the Enterprise JavaBeans (EJB) 3.0 specification and provides a POJO persistence model for object-relational mapping. It's made up of two parts: a mapping subsystem that maps classes onto relational tables, and an EntityManager API that provides access to objects, defines and executes queries, and more.

Implementation:
To create a table using Post API we need Repository to get connected.
Repository is also called as DAO and vice versa.

Defining Relationships

1. @ManytoOne etc….
2. @relation(value fetch = Fetch_Type.LAZY) will define how the query should perform and this is used on joining the tables to avoid StackOverflow by calling each other table
3. mappedBy = ‘customer’ will define the relation and it is defined on the Entity class that the mapped column should be on.

Further -> add validation and check garbage values are not added to DB(throw Error and Exception)

Errors and Debugging

1. When we do not use getter method in Beans we get “WhiteLabel Error” -> no converter found.
2. 405 Method Not Allowed-> either method arguments are mismatched or default constructor is not created.
3. Caused by: org.springframework.jdbc.datasource.init.ScriptStatementFailedException: Failed to execute SQL script statement #4 of file [/Users/youniqdexterous/Documents/GitHub/Spring-Boot/Spring-Boot/rest_v2/build/resources/main/data.sql]: insert into cusorder values(1001,1101,'Grooming items’) —> **Happens when data.sql file executes before h2 initialization** <— solve by adding properties : **“spring.jpa.defer-datasource-initialization=true” & “spring.sql.init.mode=always “** also by Defining table name at the class annotation **@Entity(name=“table_name”)**

Also check for different models to design the project connection. In this project we have checked with Richardson Maturity Model.

Best Practises

1. Consumer First: Mobile App, Web Application and Web Application with future expand to Mobile App
2. Documentation maintain
3. Correct use of HTTP methods, Response Status.

- 200 - SUCCESS
- 404 - RESOURCE NOT FOUND
- 400 - BAD REQUEST
- 201 - CREATED
- 401 - UNAUTHORIZED
- 500 - SERVER ERROR

4. No sensitive info in URI(User SSN, API Keys)
5. Use Plurals -> Customers
6. Use noun for Resources -> Customer
7. For Exceptions -> Define a consistent approach (“/search”, “PUT /gists/{id}/star”)

Application Fired/Hit Flow

Client -> CustomerResource (imple Customer)-> uses CustDAO (implement Customer) -> DB

Terms to be clarified

Name	Additional Info	
@Autowired		
Interface	Why no body	
Exception		