**Introduction in Java Spring Boot**

Done by Opariuc Rares

**@SpringBootApplication** - convenience annotation that adds all of the following

**@Configuration** - Tags the class as a source of bean definitions for the application context

**@EnableAutoConfiguration** - Tells Spring Boot to start adding beans based on classpath settings, other beans, and various property settings

**@ComponentScan** - Tells Spring to look for other components, configurations, and services in the com/example package, letting it find the controllers

1. Spring Initializer -> latest version of installation; Dependencies -> (Spring Web + Spring Data JPA + MS SQL Server Driver). You can add the Spring Security and Java Mail sender and comment it in the pom.xml.
2. Intelijj IDE Ultimate edition (normal version works
3. Comment the spring-boot-starter-data-jpa/ spring-boot-starter-data-jdbc from the pom.xml (when we don’t have a database prepared) IMPORTANT: we will decomment it when we are ready to connect to our database

* If you have this error (*Web server failed to start. Port 8080 was already in use*) => resources/application.properties and write server.port=number\_port.

1. Simple API with Spring Boot
2. Vom implementa un UserService ca sa informatiile din UserController. (de exemplu: informatiile din baza de date). Deocamdata, avem eroare

**@RestController** - it is ready for use by Spring MVC to handle web requests; it will creat Rest endpoints

* combines **@Controller** and **@ResponseBody**, two annotations that results in web requests returning data rather than a view

**@GetMapping -** maps / to the index() method and it will return pure text

To see the result, go to browser and type Localhost:number\_port (that you definied in a previous step).

1. Create a class
2. Controller for the class (it will have all the resources for the API)
3. We will create UserService class to get the information from the UserController class. We do like that when we get the information from the database. We have an error because we don’t have an instance of StudentsService. (SOL: Dependency Injection)
4. Dependency Injection

The solution to the previous problem **@Autowired**

* The StudentsService shoul be autowired for the constructor.

Now we have to tell the class UserService it will be instantied. **(@Component/@Service**) -> same meaning -> we should **@Service** for readability

1. Property file. You need these data to connect to the database

spring.datasource.url=jdbc:sqlserver://sqlsrv\\sqlexpress;databaseName=customer

spring.datasource.username=username

spring.datasource.password=password

spring.datasource.driver-class-name=com.microsoft.sqlserver.jdbc.SQLServerDriver

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.format\_sql=true

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.org.hibernate.dialect.SQLServer2008Dialect

Open the applications.properties and paste it here.Also, decomment in the pm.xml the dependency with jpa. For now, we don’t have a working database.

1. JPA and Entity

We go to the class where is stored in the Database and map it. (**@Entity** -> hibernate; **@Table** -> Database; **@Id** -> id; **@GeneratedValue** -> for MSSQL)

1. JPA in action -> create a database in Intellij (How? -> link in Bibliography)
2. JPA Repository (acces the Database)

Create an interface called UserRepository and extends to JpaRepository <T, id>(T – type of object to work with; id – type that we want) => <User, Integer> (id is an int). Also add **@Repository.** Next, we create a new object of type UserRepository, create the constructor with this parameter and **@Autowired**. Next, we return in getUser with the new created object.

1. Saving Users

Create a new class (UserConfig) and add the **@Configuration.** Create a commandLineRunner and add the **@Bean**. In the args from the commandLineRunner, you introduce new Users in the database.

1. Transient (import javax.persistence -> ALWAYS)

**@Transient** – does not need to be a column to the database. (a variable in the database). For example, you have date in the database, you don’t need to store the age, you can calculate it.

1. Post Mapping

Create a new method called registerNewUser in the UserService and add the **@PostMapping.** In the method where is **@PostMapping** use **@RequestBody** (we take the request body and map it to user). Then, right click on **@PostMapping** and open in client. Paste this code to see if the post works.

###  
GET http://localhost:8060/  
  
###  
POST http://localhost:8060/api/v1/user  
*Content-Type*: application/json  
  
{  
 "name": "Ben",  
 "email" : "tare@gmail.com"  
}  
  
###  
PUT http://localhost:8060/api/v1/user/registration  
*Content-Type*: application/json  
  
{  
 "name": "Ben",  
 "email" : "tare@gmail.com",  
 "password" : "pass"  
}

1. Writing Business Logic

In the UserRepository, create the **@Query** and implement the addNewUser method to save the new user.

1. Testing Post Request

Add to application.properties this so that can show you the errors.

server.error.include-message=*always*

1. Deleting Users

In the UsserController, create the method deleteUser and add the **@DeleteMapping.**  We use the path and the **@PathVariable** gets the userId. In the UserService, we verify if the user exists and if not, we throw an exception. If it is true, we delete the user. Also, in the client http, we create the DELETE => DELETE <http://localhost:8060/api/v1/user/1>

The last digit represent the id of the user.

**Bibliography**

Source1 - <https://spring.io/guides/gs/spring-boot/>

Source2 - <https://www.youtube.com/watch?v=9SGDpanrc8U>

How to connect to Microsoft SQL Server -><https://www.codejava.net/frameworks/spring-boot/connect-to-microsoft-sql-server-examples>

Connect your Microsoft SQL Server in Intellij - <https://www.youtube.com/watch?v=_yD-LYLoWF0>