



JaiTechoon Software Solutions Pvt.Ltd.,
JNIT, Ramkumar

SNO	Date of Version	Document Name and Version:	Document Created By
1	15-01-2022	Spring-SpringBoot-Material-v1.0	@Ramkumar
2			

JavaFullStack Training Material

Middle ware Technologies

Server-Side Technologies (Tomcat)

Environment Setup:

- STS, (Spring Suite Tool (Eclipse))

API Testing Tool

- Postman Tool

What is the use of Postman tool?

Postman is an application used for API testing.

It is an HTTP client that tests HTTP

Design Patterns

- DAO (Data Access Object)
- MVC (Model View Controller)

Web Server

- Tomcat Server

Build Tool

- Apache Maven

Spring Technologies 5.0

1. Spring Core
 - a. IOC (Inversion of Controller)
 - b. DI (Dependency Injection)
 - i. Constructor Injection
 - ii. Setter Injection
 - c. Spring Core Programming
2. Spring DAO
 - a. Spring with JDBC Integration Application
3. DataSource Configuration
4. Spring Annotations
5. Spring with MVC
6. `org.springframework.web.servlet.DispatcherServlet`
7. View Resolver
8. JSP Views
9. ModelAndView

Spring Boot

1. SpringBoot
2. SpringBoot Webservice(@RestController)
3. SpringBoot with JPA
4. SpringBoot with DataJPA

Spring Annotations

1. @Autowired
2. @Component
3. @Controller
4. @RestController
5. @RequestMapping(value="employee")

6. @GetMapping(path="/getdetails/{empId}")
7. @PostMapping(path="/create")
8. @DeleteMapping(path="/delete/{empId}")
9. @RequestBody
10. @PathVariable("empId")
11. @Service
12. @Repository
13. @SpringBootApplication
 - a. @EnableAutoConfiguration
 - b. @ComponentScan
 - c. @Configuration

Spring Framework

What Is Spring?

1. The Spring Framework provides a comprehensive programming and configuration model for modern Java-based enterprise applications - on any kind of deployment platform.
2. A key element of Spring is infrastructural support at the application level: Spring focuses on the "plumbing" of enterprise applications so that teams can focus on application-level business logic, without unnecessary ties to specific deployment environments.

Spring Features

1. Core technologies: dependency injection, events, resources, i18n, validation, data binding, type conversion, SpEL, AOP.
2. Testing: mock objects, TestContext framework, Spring MVC Test, WebTestClient.
3. Data Access: transactions, DAO support, JDBC, ORM, Marshalling XML.
4. Spring MVC Web framework.

5. Languages: Kotlin, Groovy, dynamic languages.

Features like Dependency Injection, and out of the box modules like:

- Spring JDBC
- Spring MVC
- Spring Security
- Spring AOP
- Spring ORM
- Spring Test

These modules can drastically reduce the development time of an application.

IOC (Inversion of Controller)

Giving control to the container to get an instance of the Object is called Inversion of Controller, means instead of you are creating object using the new operator, let the container do that for you.

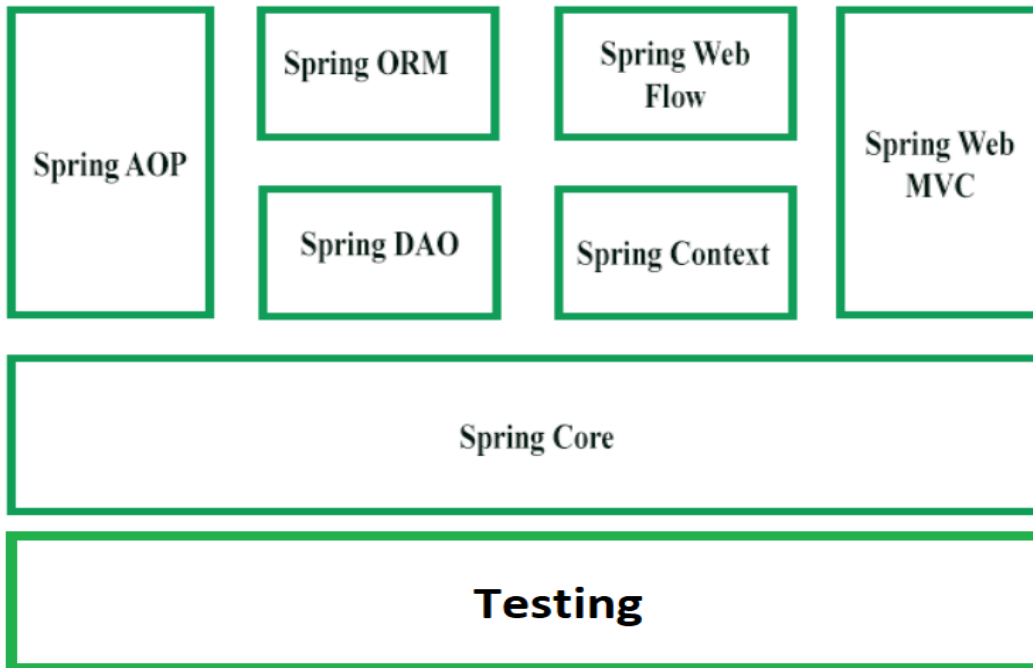
DI(Dependency Injection);

Way of injecting properties to an object is called Dependency Injection.

Spring Framework Supports Two types of Dependency Injection :

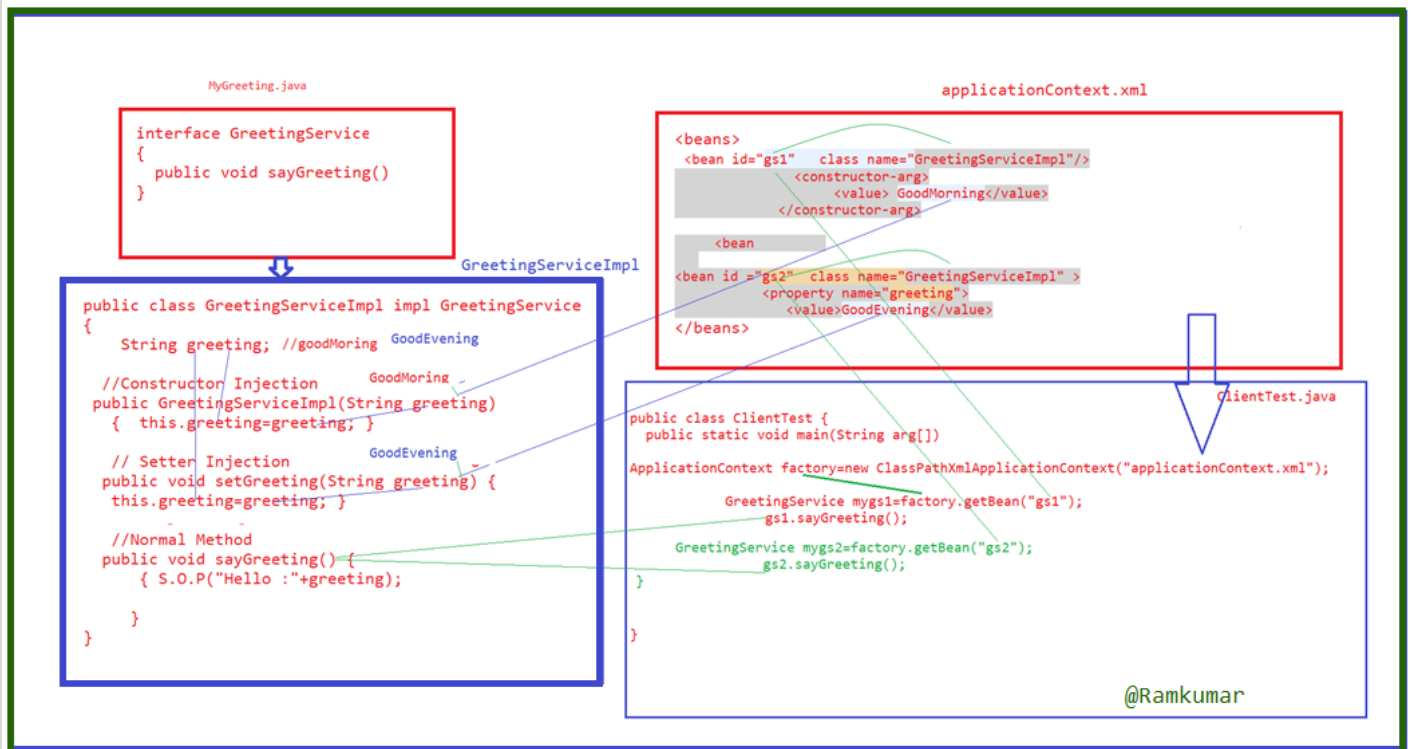
1. Constructor Injection
2. Setter Injection

Spring Architecture



Example 1:

S01-Spring-Core-IOC-DI



```

ClientTest.java  applicationContext.xml  GreetingServiceImpl.java
1 package com.cgi.myspring;
2
3 public class GreetingServiceImpl implements GreetingService {
4     String greeting; //8. GoodMorning 16//GoodEvening
5
6     public GreetingServiceImpl() {
7
8     }
9
10    //6.
11    public GreetingServiceImpl(String greeting)
12    {
13        this.greeting=greeting; //7
14    }
15
16    //GoodEvening //14
17    public void setGreeting(String greeting)
18    {
19        this.greeting=greeting; //15
20    }
21
22    @Override //10 //18
23    public void sayGreeting() {
24        System.out.println("Hello :"+greeting);
25    }
26
27 }
28

```

```

ClientTest.java  applicationContext.xml
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
3     "http://www.springframework.org/dtd/spring-beans-2.0.dtd">
4 <beans>
5     <!-- . Constructor Injection 4. -->
6     <bean id="greetingService1" class="com.cgi.myspring.GreetingServiceImpl">
7         <constructor-arg <!-- 5. -->
8             <value>GoodMorning</value>
9         </constructor-arg>
10    </bean>
11    <!-- . Setter Injection 12 -->
12    <bean id="greetingService2" class="com.cgi.myspring.GreetingServiceImpl">
13        <property name="greeting">
14            <value>GoodEvening</value> <!-- 13 -->
15        </property>
16    </bean>
17
18 </beans>
19
20 </beans>

```

The screenshot shows an IDE with a project explorer on the left and a code editor on the right. The project explorer shows a project named 'S01-Spring-Core-IOC-DI' with a source folder 'src' containing a package 'com.cgi.myspring'. Inside this package, there are files 'ClientTest.java', 'GreetingService.java', 'GreetingServiceImpl.java', 'MyTest.java', and 'applicationContext.xml'. The code editor shows the 'ClientTest.java' file with the following code:

```
1 package com.cgi.myspring;
2
3 import org.springframework.context.ApplicationContext;
4
5 public class ClientTest {
6
7     public static void main(String[] args) { //1.
8         //2
9         ApplicationContext factory = new ClassPathXmlApplicationContext("applicationContext.xml");
10
11         //3
12         GreetingService gs1=(GreetingService)factory.getBean("greetingService1");
13         //9
14         gs1.sayGreeting();
15
16         //11
17         GreetingService gs2=(GreetingService)factory.getBean("greetingService2");
18
19         //17
20         gs2.sayGreeting();
21
22     } //19
23 } //20
24
25
```

The console output at the bottom shows the following messages:

```
<terminated> ClientTest (1) [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (15-Jan-2022, 8:44:09 pm)
Jan 15, 2022 8:44:09 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh
INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@3327bd23: display
Jan 15, 2022 8:44:09 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
INFO: Loading XML bean definitions from class path resource [applicationContext.xml]
Jan 15, 2022 8:44:09 PM org.springframework.context.support.AbstractApplicationContext obtainFreshBeanFactory
INFO: Bean factory for application context [org.springframework.context.support.ClassPathXmlApplicationContext]
Jan 15, 2022 8:44:09 PM org.springframework.beans.factory.support.DefaultListableBeanFactory preInstantiateSingletons
INFO: Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory
Hello :GoodMorning
Hello :GoodEvening
```

Spring – JDBC(DAO) : Integration

S03-Spring-with-DAO

AccountBean

```
public class Account
implements Serializable {
    private int accno;
    private String accName;
    private String accType;
    private double bal;

    public Account() {
    }
}
```

for DatabaseConnection

applicationContext_Dao.xml

```
<beans>
<bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
<property name="driverClassName">
<value>com.mysql.jdbc.Driver</value>
</property>
<property name="url">
<value>jdbc:mysql://localhost:3306/apsitdb</value>
</property>
<property name="username">
<value>root</value>
</property>
<property name="password">
<value>root</value>
</property>
</bean>
<bean id="accountDao" class="AccountDaoImpl" >
<property name="dataSource">
<ref bean="dataSource" />
</property>
</bean>
</beans>
```

```
public class AccountDaoImpl implements AccountDao {
//This Object referecnece take jdbc connects form applicationContext_Dao.xml
private DataSource dataSource;
public DataSource getDataSource() {
return dataSource;
}
public void setDataSource(DataSource dataSource) {
this.dataSource = dataSource; //driverclass ,url ,username,pass
}
public void save(Account account) {
Connection con = null;
PreparedStatement pstmt = null;
try { // Get connection.
con = dataSource.getConnection();
// Create statement.
pstmt = con.prepareStatement("INSERT INTO account VALUES(?,?,?,?)");
// Set parameters.
pstmt.setInt(1, account.getAccno());
pstmt.setString(2, account.getAccName());
pstmt.setString(3, account.getAccType());
pstmt.setDouble(4, account.getBal());
// Execute statement.
pstmt.execute();
// Handle Exceptions
}
}
```

```
public class ClientAccountDao {
public static void main(String[] args) {
ApplicationContext factory = new ClassPathXmlApplicationContext("applicationContext_Dao.xml");
AccountDao accDao = (AccountDao)factory.getBean("accountDao");
Account account = new Account();
account.setAccno(50);
account.setAccName("Kiran");
account.setAccType("Saving1");
account.setBal(7777.00);
accDao.save(account); // save to database
System.out.println("Account No : "+account.getAccno());
System.out.println("Account Name : "+account.getAccName());
System.out.println("Account Type : "+account.getAccType());
System.out.println("Account Balance : "+account.getBal()); }
}
```

S03-Spring-with-DAO

- src
 - (default package)
 - Account.java
 - AccountDao.java
 - AccountDaoImpl.java
 - ClientAccountDao.java
 - applicationContext_Dao.xml
- JRE System Library [jre]
- Referenced Libraries
- SQL-Script

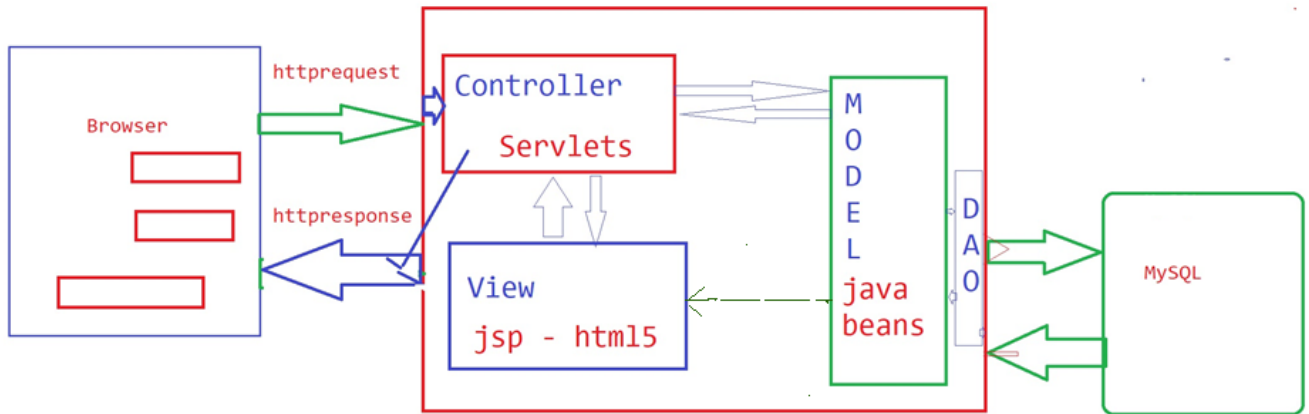
```
1*import org.springframework.context.ApplicationContext;
3
4 //ClientAccountDao.java
5 public class ClientAccountDao {
6     public static void main(String[] args) {
7         ApplicationContext factory = new ClassPathXmlApplicationContext("applicationContext_Dao.xml");
8
9         AccountDao accDao = (AccountDao)factory.getBean("accountDao");
10
11         Account account = new Account();
12         account.setAccno(60);
13         account.setAccName("RamKumar");
14         account.setAccType("Saving1");
15         account.setBal(7777.00);
16         //System.out.println(account);
17
18         accDao.save(account); // save to database
19
20         System.out.println("Account No : "+account.getAccno());
21         System.out.println("Account Name : "+account.getAccName());
22         System.out.println("Account Type : "+account.getAccType());
23         System.out.println("Account Balance : "+account.getBal());
24
25     }
```

Console

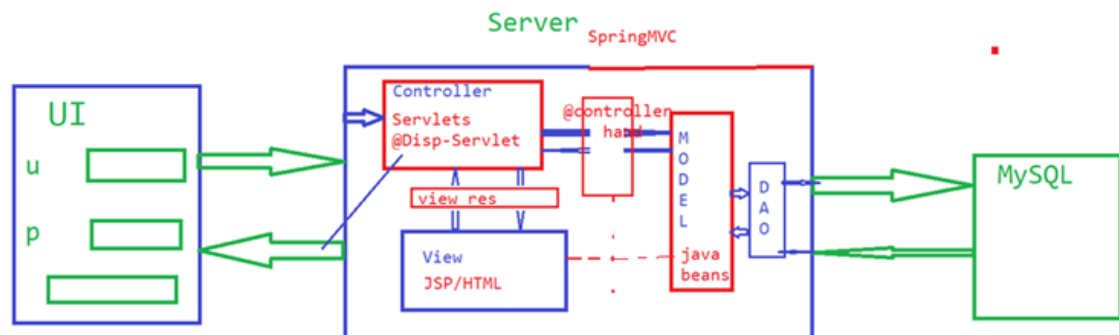
```
<terminated> ClientAccountDao [Java Application] C:\Ramkumar\APSI-Adv\JAVA-Ramkumar\spring-tool-suite-4-4.9.0.RELEASE-\content
INFO: Bean factory for application context [org.springframework.context.support.ClassPathXmlApplicati
Jan 15, 2022 8:00:57 PM org.springframework.beans.factory.support.DefaultListableBeanFactory preInsta
INFO: Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFa
Jan 15, 2022 8:00:57 PM org.springframework.jdbc.datasource.DriverManagerDataSource setDriverClassNam
INFO: Loaded JDBC driver: com.mysql.jdbc.Driver
AccountDaoImpl save()
Account No : 60
Account Name : RamKumar
Account Type : Saving1
Account Balance : 7777.0
```


MVC Design Pattern

MVC Design Pattern



M = Model
V = View
C = Controller



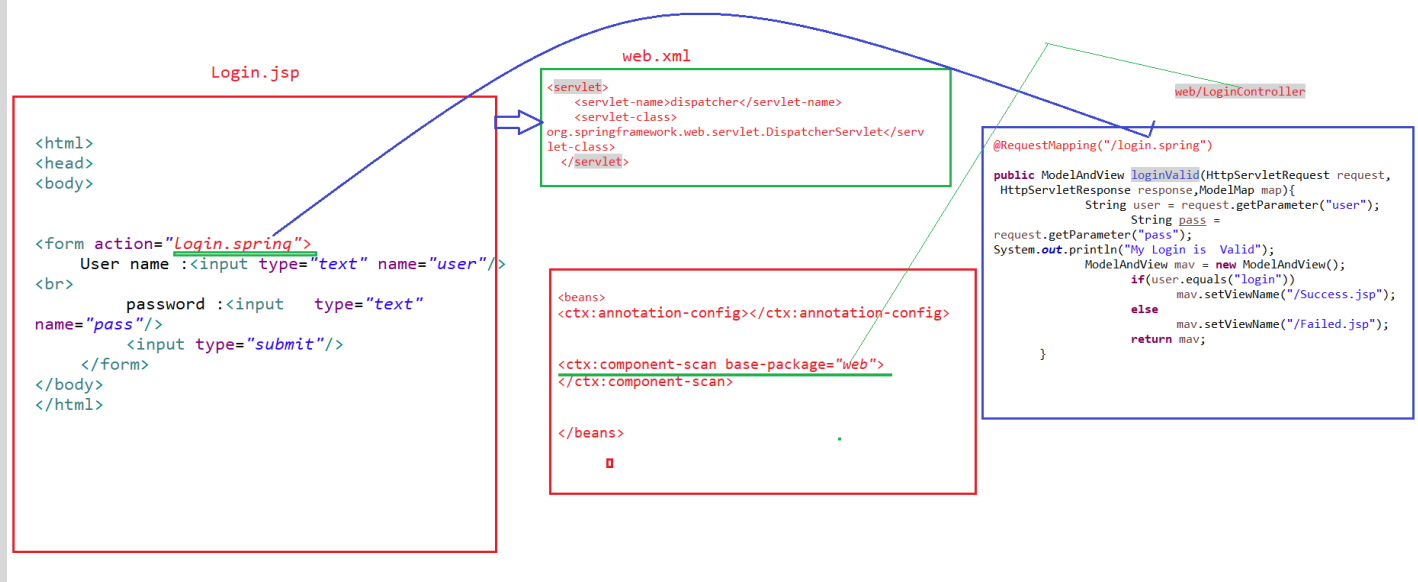
Servers

Webserver
Apache Tomcat
open source

ApplicationServer
Weblogic
JBOSS

http://localhost:8089/09_springmvc/user.spring?user=ram&pass=ram

Spring MVC with Annotations



The screenshot shows the Spring MVC application code in an IDE. The `LoginController.java` file is open, showing the `loginValid` and `userValid` methods. The `loginValid` method is annotated with `@RequestMapping("/login.spring")` and the `userValid` method is annotated with `@RequestMapping("/user.spring")`. The `loginValid` method checks if the user is "ram" and the password is "kumar". If valid, it sets the view name to "Success.jsp"; otherwise, it sets the view name to "Failed.jsp". The `userValid` method checks if the user is "ram" and the password is "kumar". If valid, it sets the view name to "Success.jsp"; otherwise, it sets the view name to "Failed.jsp".

A browser window is open showing the URL `http://localhost:8089/09_springmvc/user.spring?user=ram&pass=kumar`. The browser displays the message "Success" and the URL `user.spring` is highlighted in red.

Layered architecture

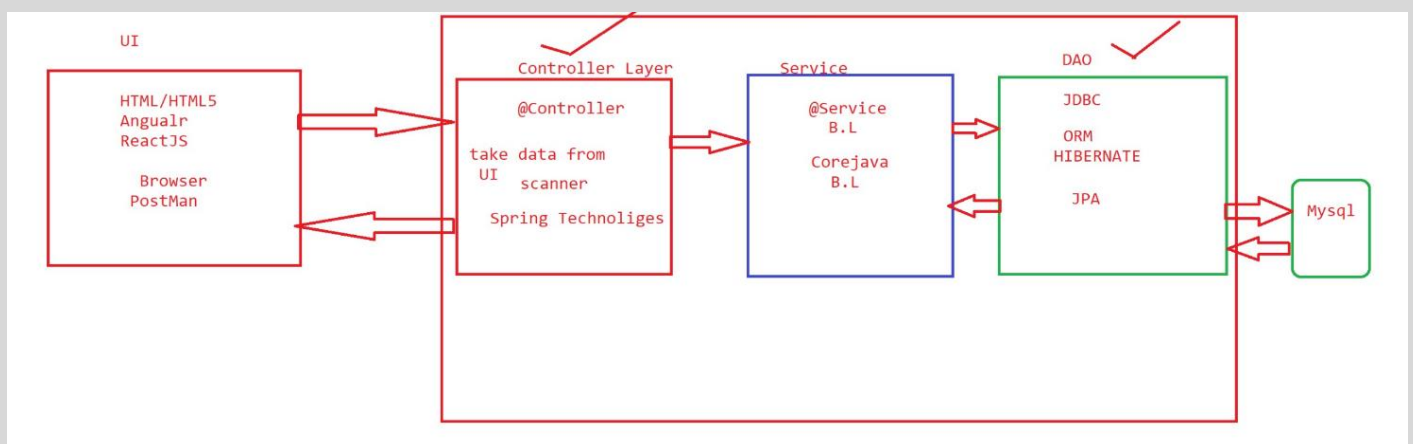
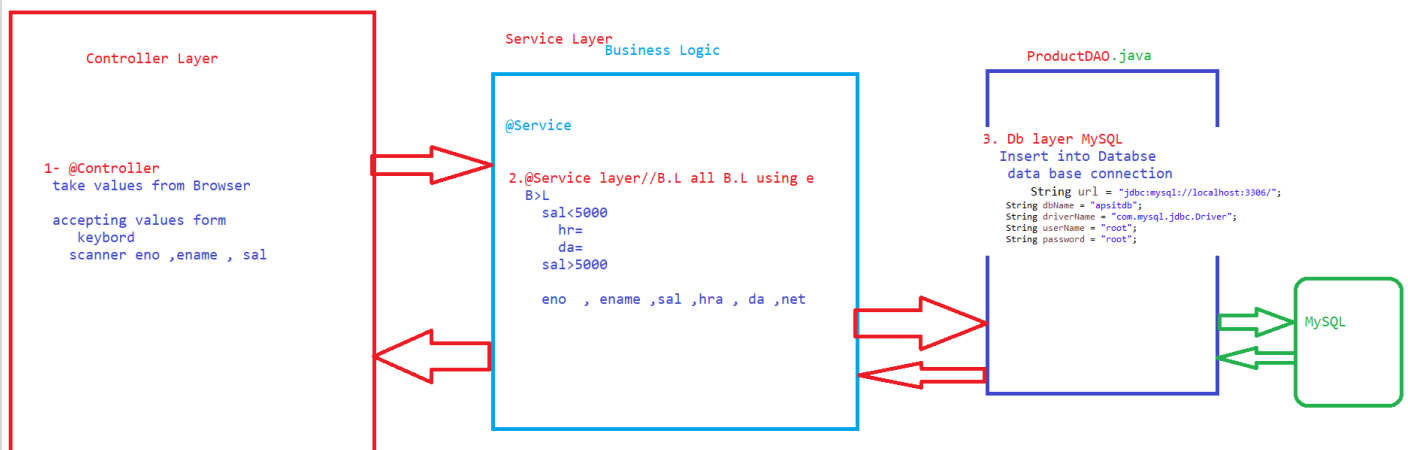
```
1- @Controller
take values from Browser

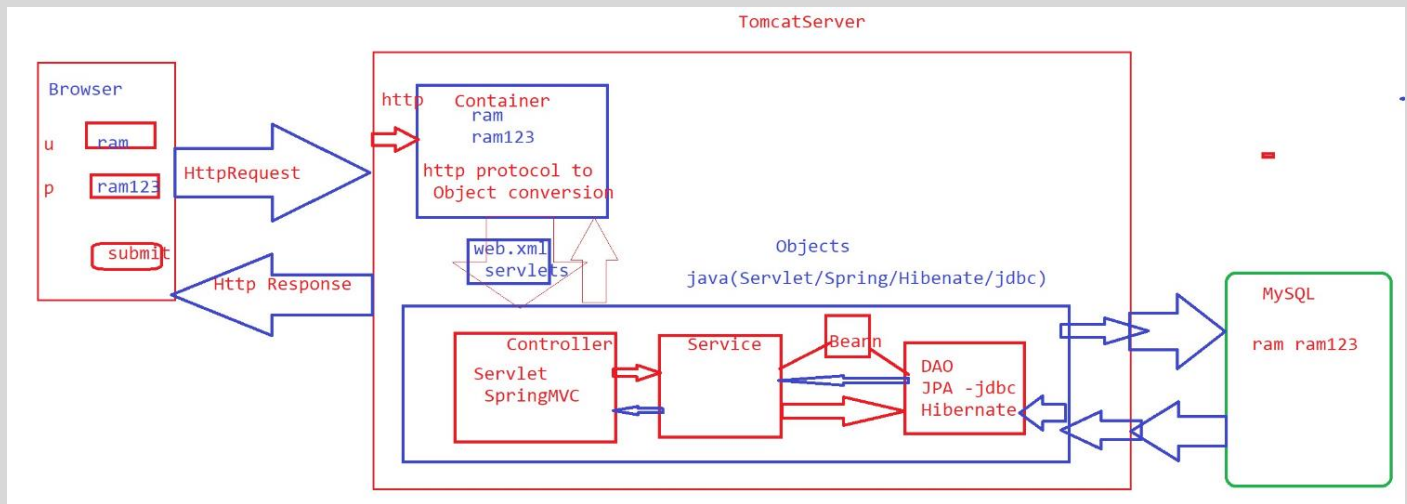
accepting values form
keyboard
scanner eno ,ename , sal

2.@Service layer//B.L all B.L using e
B>L
sal<5000
hr=
da=
sal>5000

eno , ename ,sal ,hra , da ,net

3. Db layer MySQL
Insert into Database
data base connection
String url = "jdbc:mysql://localhost:3306/";
String dbName = "apsitdb";
String driverName = "com.mysql.jdbc.Driver";
String userName = "root";
String password = "root";
```





Spring Annotations

@Autowired

1. The Spring framework enables automatic dependency injection. In other words, **by declaring all the bean dependencies in a Spring configuration file, Spring container can autowire relationships between collaborating beans.** This is called Spring bean autowiring.
2. After enabling annotation injection, **we can use autowiring on properties, setters, and constructors**
3. Fields are injected right after construction of a bean, before any config methods are invoked.

Example :

```
public class Customer {
    @Autowired
    private Person person;
    private int type;
}
```

```
1 package com.cg.demo.controller;
2 import java.util.List;
20
21 @RestController
22 @RequestMapping(value="employee")
23
24 //Connecting ... to Angular
25 @CrossOrigin("http://localhost:4200")
26
27 public class EmployeeController {
28
29     @Autowired
30     private EmployeeServiceI service;
31
32
33     @PostMapping(path="/create")
34     public ResponseEntity<Boolean> CreateEmployee(@RequestBody Employee emp) {
35         service.CreateEmployee(emp);
36         ResponseEntity<Boolean> responseEntity = new ResponseEntity<Boolean>(true, HttpStatus.OK);
37         System.out.println("response entity=" + responseEntity);
38         return responseEntity;
39     }
40
41     @GetMapping(path="/getdetails/{empId}")
42     public ResponseEntity<Employee> findEmployeeById(@PathVariable("empId") long empId) {
43         Employee emp=service.findEmployeeById(empId);
44         if (emp == null) {
45             throw new EmployeeNotFoundException("employee not found for id=" + empId);
46         }
47         return new ResponseEntity<Employee>(emp,new HttpHeaders(),HttpStatus.OK);
48     }
49
50     @GetMapping(path="/getAll")
51     public ResponseEntity<List<Employee>> findAll(){
52         List<Employee> list=service.findAllEmployees();
53
54         return new ResponseEntity<List<Employee>>(list,new HttpHeaders(),HttpStatus.OK);
55     }
56 }
```

@Configuration

1. This annotation is used on classes which define beans.
2. It is an analogy for XML configuration file – it is configuration using Java class. Java class annotated with @Configuration
3. It is a configuration by itself and will have methods to instantiate and configure the dependencies.

@Controller

- The @Controller annotation is used to indicate the class is a Spring controller. This annotation can be used to identify controllers for Spring MVC

@Service

- @Service annotation is used with classes that provide some business functionalities.
- Spring Service annotation can be applied only to classes. It is used to mark the class as a service provider.

```
DemoApplica... *EmployeeCo... application... iReadMe *EmployeeSe...
1 package com.cg.demo.service;
2 import java.util.List;
9
10 @Service
11 public class EmployeeServiceImpl implements EmployeeServiceI{
12
13     @Autowired
14     private EmployeeDaoI dao;
15
16     public Employee CreateEmployee(Employee emp) {
17         return dao.CreateEmployee(emp);
18     }
19
20     public Employee findEmployeeById(long empId) {
21         return dao.findEmployeeById(empId);
22     }
23     public Employee updateEmployee(Employee emp) {
24         return dao.updateEmployee(emp);
25     }
26     public List<Employee> findAllEmployees(){
27         return dao.findAllEmployees();
28     }
29     public void deleteEmployee(long empId) {
30         dao.deleteEmployee(empId);
31     }
32 }
33
```

@Repository

- Spring @Repository annotation is used to indicate that the class provides the mechanism for storage, retrieval, search, update and delete operation on objects.

Example :

```
MyTesting-Eclipse-Workspace - 07-SpringBoot-Data/JPA-UI-JSP/src/main/java/com/apsit/springboot/data/EmployeeRepository.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
07-SpringBoot-Data/JPA-UI-JSP
  src/main/java
    com.apsit.springboot
    com.apsit.springboot.controller
    com.apsit.springboot.data
      EmployeeRepository.java
    com.apsit.springboot.model
  src/main/resources
  src/test/java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
  pom.xml
  ReadMe

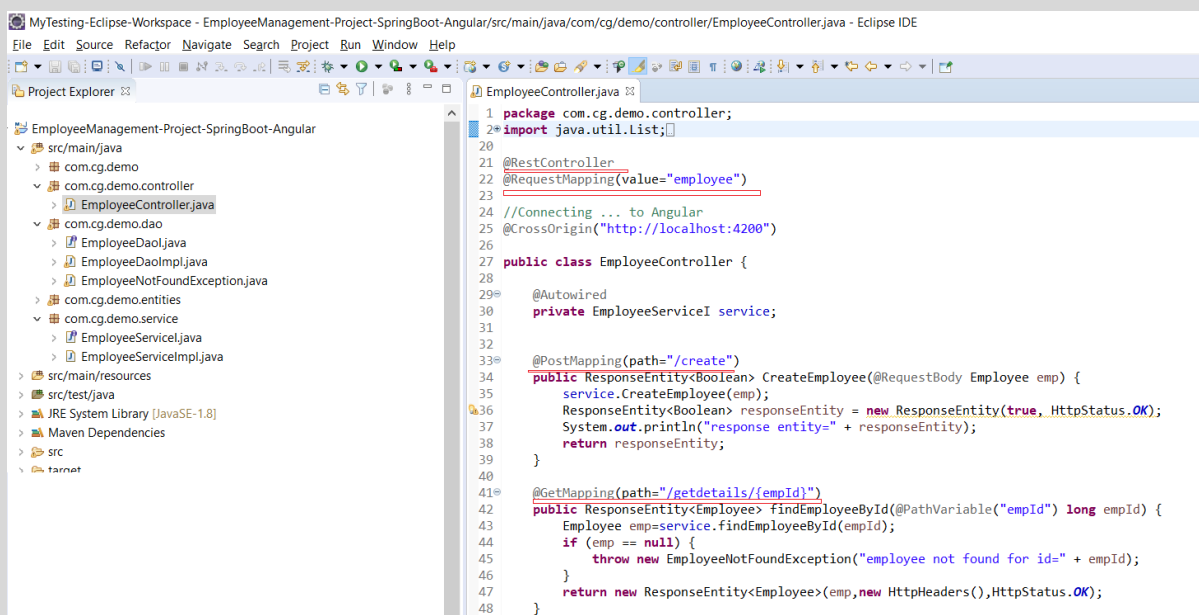
EmployeeRepository.java
1 package com.apsit.springboot.data;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5 @Repository
6 public interface EmployeeRepository extends JpaRepository<Employee, Long> {
7
8 }
9
10
11
12
13
```

Request Mapping Variants

@RestController

1. By annotating a class with this annotation you no longer need to add @ResponseBody to all the RequestMapping method.
2. @RestController is a convenience annotation which combines @Controller and @ResponseBody

Example :



```
1 package com.cg.demo.controller;
2 import java.util.List;
3
4 @RestController
5 @RequestMapping(value="employee")
6 //Connecting ... to Angular
7 @CrossOrigin("http://localhost:4200")
8
9 public class EmployeeController {
10
11     @Autowired
12     private EmployeeServiceI service;
13
14     @PostMapping(path="/create")
15     public ResponseEntity<Boolean> CreateEmployee(@RequestBody Employee emp) {
16         service.CreateEmployee(emp);
17         ResponseEntity<Boolean> responseEntity = new ResponseEntity(true, HttpStatus.OK);
18         System.out.println("response entity=" + responseEntity);
19         return responseEntity;
20     }
21
22     @GetMapping(path="/getdetails/{empId}")
23     public ResponseEntity<Employee> findEmployeeById(@PathVariable("empId") long empId) {
24         Employee emp=service.findEmployeeById(empId);
25         if (emp == null) {
26             throw new EmployeeNotFoundException("employee not found for id=" + empId);
27         }
28         return new ResponseEntity<Employee>(emp,new HttpHeaders(),HttpStatus.OK);
29     }
30 }
```

@RequestMapping

1. This annotation maps HTTP requests to handler methods of MVC and REST controllers.
2. To configure the mapping of web requests, you use the @RequestMapping annotation.
3. The @RequestMapping annotation can be applied to class-level and/or method-level in a controller.
4. The class-level annotation maps a specific request path or pattern onto a controller.
5. The Spring MVC @RequestMapping annotation is capable of handling HTTP request methods, such as GET, PUT, POST, DELETE, and PATCH.
6. Example: @RequestMapping(method = RequestMethod.GET)
@RequestMapping(method = RequestMethod.POST)

@Controller

```
@RequestMapping("/welcome")
public class WelcomeController{
@RequestMapping(method = RequestMethod.GET)
public String welcomeAll(){
return "welcome all";
}
```

@PathVariable

1. The `@PathVariable` spring annotation on a method argument to bind it to the value of a URI template variable.

@PostMapping

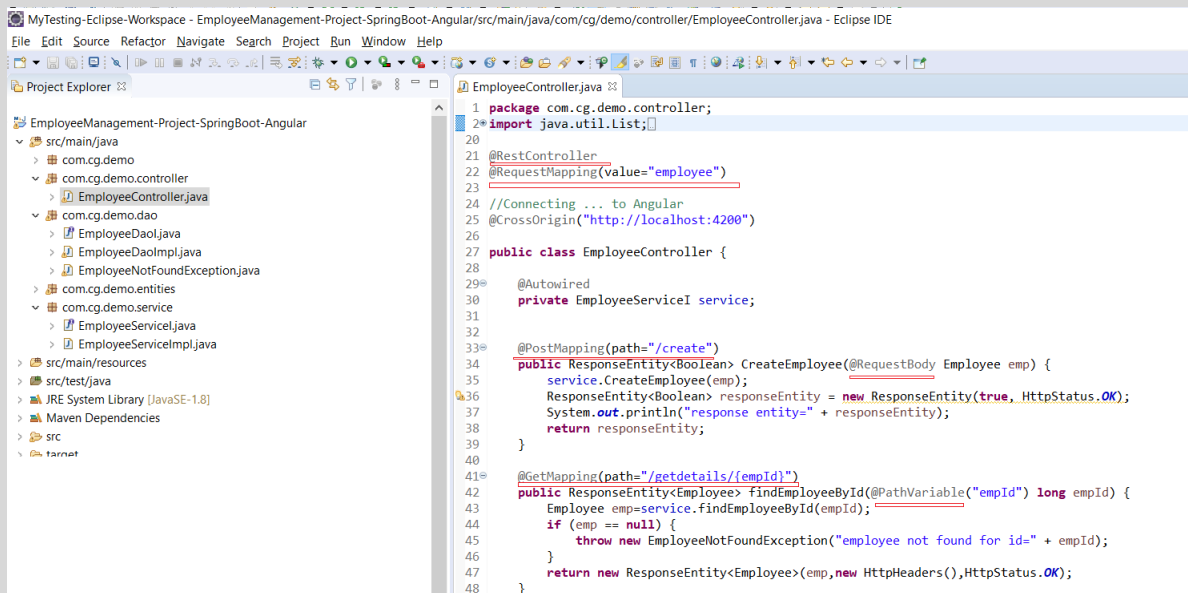
2. `@PostMapping` is a *composed annotation* that acts as a shortcut for `@RequestMapping(method = RequestMethod.POST)`.
3. Annotation for mapping HTTP POST requests onto specific handler methods.
4. Should only be used to create new resource.
5. Example: `@PostMapping("/home")`

@GetMapping

1. **@GetMapping** is specialized version of [@RequestMapping](#) annotation that acts as a shortcut for `@RequestMapping(method = RequestMethod.GET)`.
2. [@GetMapping](#) annotated methods handle the HTTP GET requests matched with given URI expression.
3. Should only be used to read the resource.
4. Example: `@GetMapping("/home")` `@GetMapping("/members/{id}")`

@PutMapping

1. **@PutMapping** is specialized version of [@RequestMapping](#) annotation that acts as a shortcut for `@RequestMapping(method = RequestMethod.PUT)`.
2. `@PutMapping` annotation is used to accept HTTP Put request containing a Request Body with JSON or XML payload, so that you can perform let's say an update user details operation.
3. Example: `@PutMapping("/{userId}")`

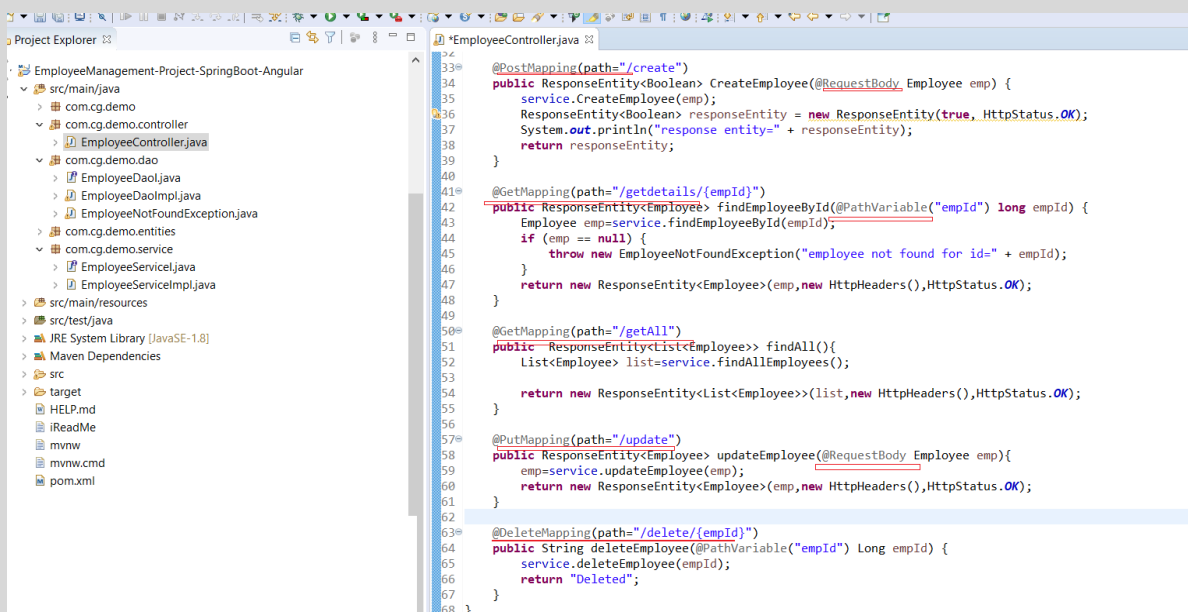


```
1 package com.cg.demo.controller;
2 import java.util.List;
3
4 @RestController
5 @RequestMapping(value="employee")
6 //Connecting ... to Angular
7 @CrossOrigin("http://localhost:4200")
8
9 public class EmployeeController {
10
11     @Autowired
12     private EmployeeServiceI service;
13
14     @PostMapping(path="/create")
15     public ResponseEntity<Boolean> CreateEmployee(@RequestBody Employee emp) {
16         service.CreateEmployee(emp);
17         ResponseEntity<Boolean> responseEntity = new ResponseEntity<Boolean>(true, HttpStatus.OK);
18         System.out.println("response entity=" + responseEntity);
19         return responseEntity;
20     }
21
22     @GetMapping(path="/getdetails/{empId}")
23     public ResponseEntity<Employee> findEmployeeById(@PathVariable("empId") long empId) {
24         Employee emp=service.findEmployeeById(empId);
25         if (emp == null) {
26             throw new EmployeeNotFoundException("employee not found for id=" + empId);
27         }
28         return new ResponseEntity<Employee>(emp,new HttpHeaders(),HttpStatus.OK);
29     }
30 }
```

@DeleteMapping

1. @DeleteMapping is a *composed annotation* that acts as a shortcut for @RequestMapping(method = RequestMethod.DELETE).
2. Annotation for mapping HTTP DELETE requests onto specific handler methods.
3. It is basically used to delete a resource from the project.
4. Example: @DeleteMapping(value = "/posts/{id}")

Example :



```
33 @PostMapping(path="/create")
34 public ResponseEntity<Boolean> CreateEmployee(@RequestBody Employee emp) {
35     service.CreateEmployee(emp);
36     ResponseEntity<Boolean> responseEntity = new ResponseEntity<Boolean>(true, HttpStatus.OK);
37     System.out.println("response entity=" + responseEntity);
38     return responseEntity;
39 }
40
41 @GetMapping(path="/getdetails/{empId}")
42 public ResponseEntity<Employee> findEmployeeById(@PathVariable("empId") long empId) {
43     Employee emp=service.findEmployeeById(empId);
44     if (emp == null) {
45         throw new EmployeeNotFoundException("employee not found for id=" + empId);
46     }
47     return new ResponseEntity<Employee>(emp,new HttpHeaders(),HttpStatus.OK);
48 }
49
50 @GetMapping(path="/getAll")
51 public ResponseEntity<List<Employee>> findAll(){
52     List<Employee> list=service.findAllEmployees();
53
54     return new ResponseEntity<List<Employee>>(list,new HttpHeaders(),HttpStatus.OK);
55 }
56
57 @PutMapping(path="/update")
58 public ResponseEntity<Employee> updateEmployee(@RequestBody Employee emp){
59     emp=service.updateEmployee(emp);
60     return new ResponseEntity<Employee>(emp,new HttpHeaders(),HttpStatus.OK);
61 }
62
63 @DeleteMapping(path="/delete/{empId}")
64 public String deleteEmployee(@PathVariable("empId") Long empId) {
65     service.deleteEmployee(empId);
66     return "Deleted";
67 }
68 }
```

1. Maven (Build Tool)
2. STS (Spring Tool Suite)
3. Postman (Testing Tool)

Maven is Build tool

Development Environment

Testing Environment

Add dependency for jar files

Apache Maven

1. Maven is an automation and management tool developed by Apache Software Foundation.
2. Maven used for building and managing any Java-based project.
3. day-to-day work of Java developers easier and generally help with the comprehension of any Java-based project.

Maven's Objectives :

1. Maven's primary goal is to allow a developer to comprehend the complete state of a development effort in the shortest period of time.
2. In order to attain this goal, Maven deals with several areas of concern:

- Making the build process easy
- Providing a uniform build system
- Providing quality project information
- Encouraging better development practices

Making the build process easy

- While using Maven doesn't eliminate the need to know about the underlying mechanisms, Maven does shield developers from many details.

What is a POM?

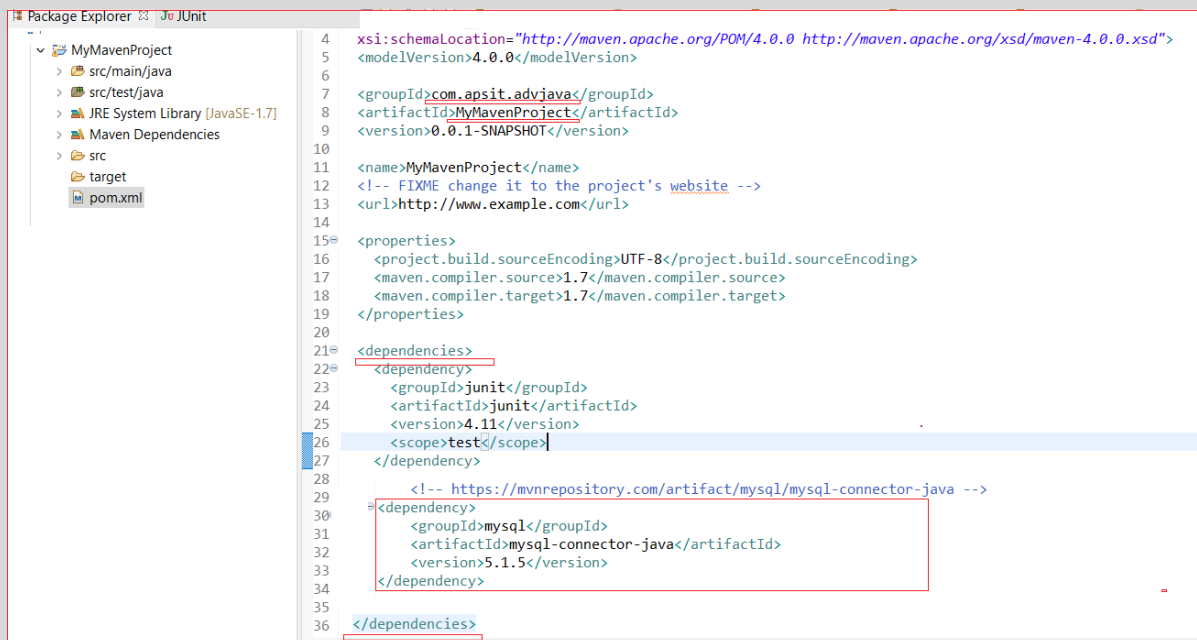
1. A **Project Object Model** or POM is the fundamental unit of work in Maven.
2. It is an XML file that contains information about the project and configuration details used by Maven to build the project.
3. It contains default values for most projects.
4. Examples for this is the build directory, which is target; the source directory, which is src/main/java; the test source directory, which is src/test/java; and so on.
5. When executing a task or goal, Maven looks for the POM in the current directory.
6. It reads the POM, gets the needed configuration information, then executes the goal.

The minimum requirement for a POM are the following:

- `project` root
- `modelVersion` - should be set to 4.0.0
- `groupId` - the id of the project's group.
- `artifactId` - the id of the artifact (project)
- `version` - the version of the artifact under the specified group

Example of POM

```
1. <project>
2.   <modelVersion>4.0.0</modelVersion>
3.
4.   <groupId>com.mycompany.app</groupId>
5.   <artifactId>my-app</artifactId>
6.   <version>1</version>
7. </project>
```



Postman:

Postman is an interactive and automatic tool for verifying the APIs of your project.

What is Postman API Test?

1. Postman is an application used for API testing.
2. It is an HTTP client that tests HTTP requests, utilizing a graphical user interface, through which we obtain different types of responses that need to be subsequently validated.

Methods

Postman offers many endpoint interaction methods. The following are some of the most used, including their functions:

- GET: Obtain information
- POST: Add information
- PUT: Replace information
- PATCH: Update certain information
- DELETE: Delete information

Response Codes :

When testing APIs with Postman, we usually obtain different response codes. Some of the most common include:

Ramkumar , WhatsApp : +91 9701153333

100 Series > Temporal responses, for example, '102 Processing'.

200 Series > Responses where the client accepts the request and the server processes it successfully, for instance, '200 Ok'.

300 Series > Responses related to URL redirection, for example, '301 Moved Permanently.'

400 Series > Client error responses, for instance, '400 Bad Request'.

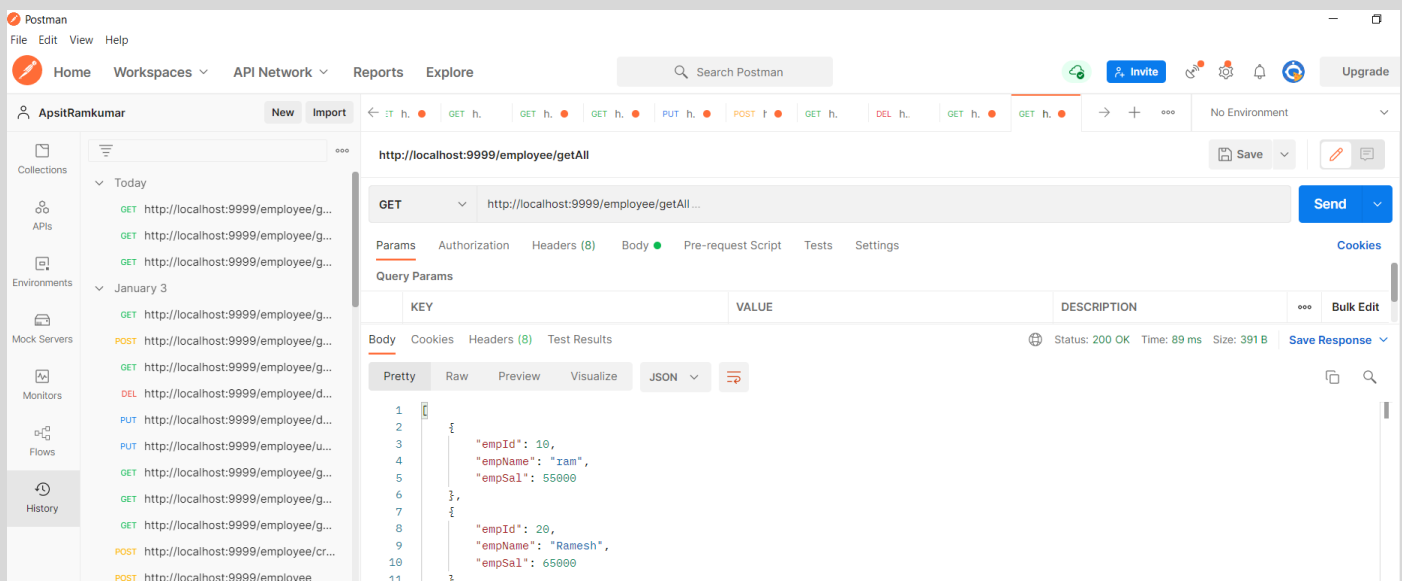
500 Series > Server error responses, for example, '500 Internal Server Error.'

What is the HTTP status codes?

HTTP response status codes:

- Informational responses (100 – 199)
- Successful responses (200 – 299)
- Redirection messages (300 – 399)
- Client error responses (400 – 499)
- Server error responses (500 – 599)

Postman Example:



Spring Boot

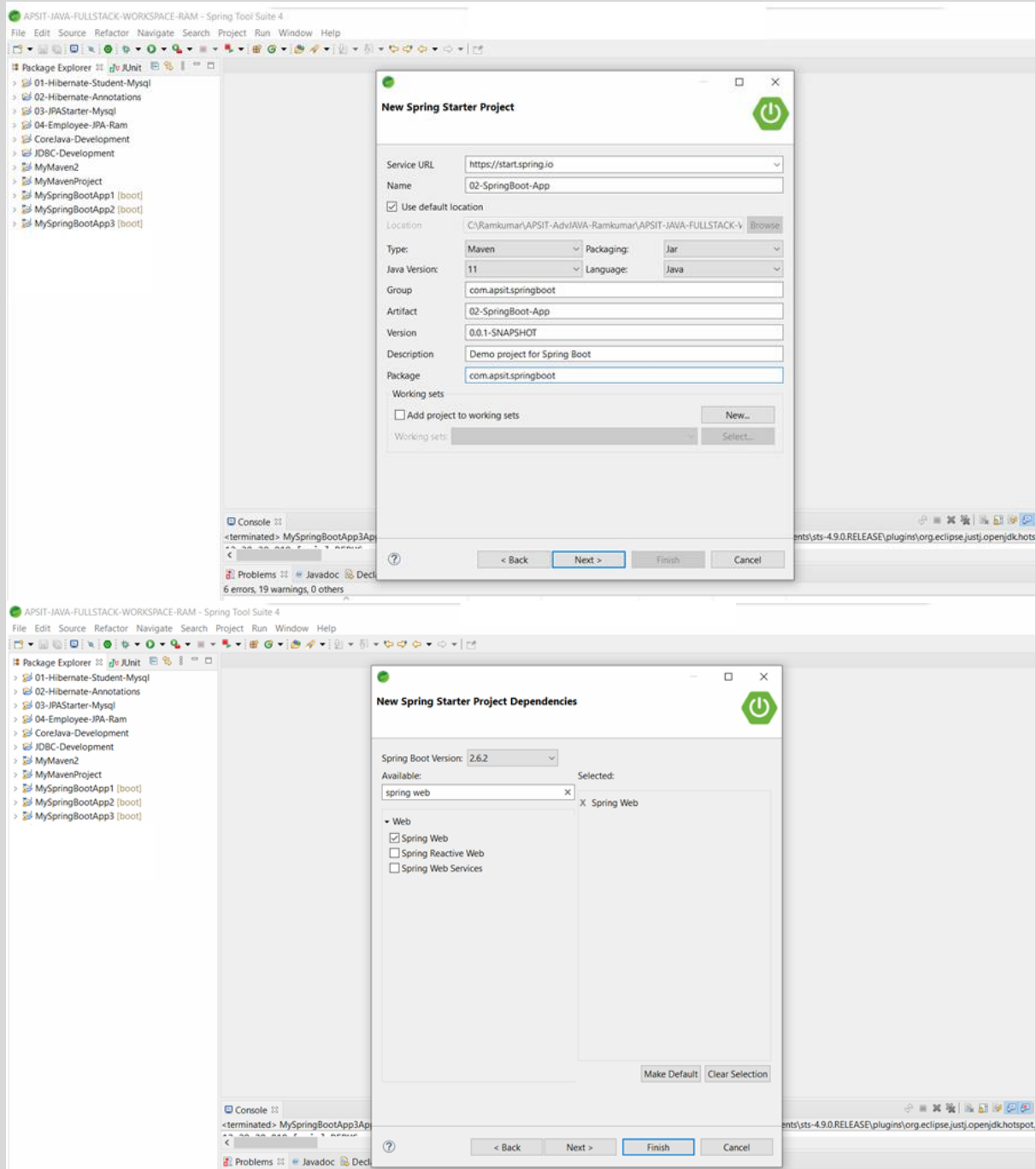
What Is Spring Boot?

1. Spring Boot is basically an extension of the Spring Framework.
2. It eliminates the XML configuration required to set up Spring applications, paving the way for a faster and more efficient development ecosystem.
3. Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".
4. We take an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need minimal Spring configuration.

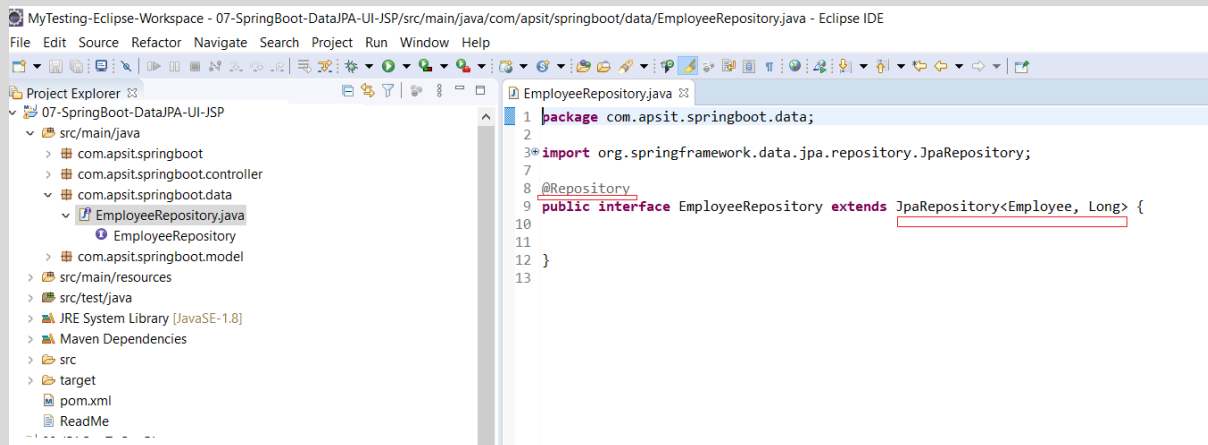
Spring Boot Features :

- Create stand-alone Spring applications
- Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files)
- Provide opinionated 'starter' dependencies to simplify your build configuration
- Automatically configure Spring and 3rd party libraries whenever possible
- Provide production-ready features such as metrics, health checks, and externalized configuration
- Absolutely no code generation and no requirement for XML configuration

How to create SpringBoot Application:



SpringBoot with DataJPA



08-SpringBoot-JPA-Product-Layers

Server Running info...

Bootstrapping Spring Data JPA repositories in DEFAULT mode.

: Finished Spring Data repository scanning in 20ms. Found 0 JPA repository interfaces.

Tomcat initialized with port(s): 9092 (http)

2021-12-30 11:32:09.823 INFO 30240 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]

2021-12-30 11:32:09.823 INFO 30240 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.33]

2021-12-30 11:32:09.971 INFO 30240 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext

HHH000400: Using dialect: org.hibernate.dialect.MySQL55Dialect

Using JtaPlatform implementation:

Initialized JPA EntityManagerFactory for persistence unit 'default'

Tomcat started on port(s): 9092 (http) with context path "

Table Name :

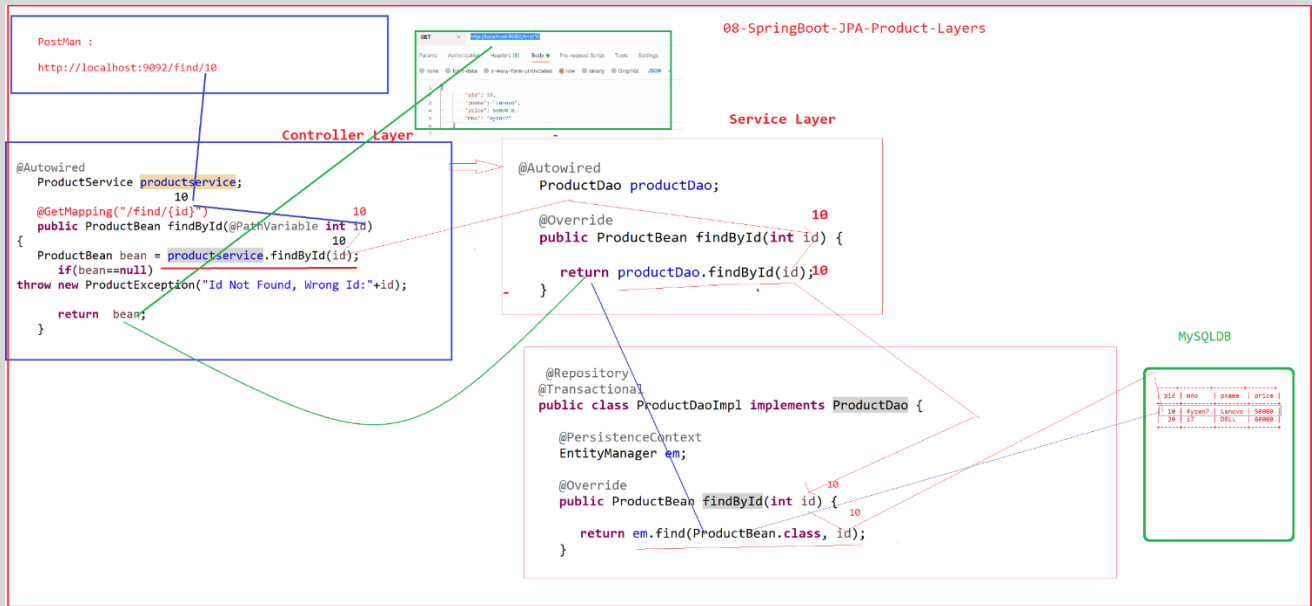
@Table(name="product_masters")

] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'

2021-12-30 11:40:23.361 INFO 30240 --- [nio-9092-exec-2] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'

2021-12-30 11:40:23.371 INFO 30240 --- [nio-9092-exec-2] o.s.web.servlet.DispatcherServlet : Completed initialization in 10 ms

Application Flow :



Postman Example :

PUT `http://localhost:9092/update` Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```

1 {
2   ...."pid": 10,
3   ...."pname": "Apple",
4   ...."price": 230000.0,
5   ...."mno": "MacProM1"
6 }
7

```

MySQL 5.5 Command Line Client

```

2 rows in set (0.00 sec)

mysql> select * from product_masters;

+----+-----+-----+-----+
| pid | mno  | pname | price |
+----+-----+-----+-----+
| 10  | MacProM1 | Apple | 230000 |
| 20  | DELL  |      | 60000  |
+----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql>

```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```

1 {
2   "pid": 10,
3   "pname": "Apple",
4   "price": 230000.0,
5   "mno": "MacProM1"
6 }

```

CURD:

C: Create (create tables)

U : Update (insert or update data)

R : Retrieve (retrieve data form database (select * from employee))

D : Delete (Delete record)

Spring Boot Annotations

@SpringBootApplication

```
<ctx:annotation-config></ctx:annotation-config>
```

```
<ctx:component-scan base-package="web"></ctx:component-scan>
```

[org.springframework.boot.autoconfigure.SpringBootApplication](#)

[@SpringBootConfiguration](#)

[@EnableAutoConfiguration](#)

[@ComponentScan\(excludeFilters={@Filter\(type=CUSTOM, classes={TypeExcludeFilter.class}\), @Filter\(type=CUSTOM, classes={AutoConfigurationExcludeFilter.class}\)}\)](#)

@SpringBootApplication

- a. This annotation is used on the application class while setting up a Spring Boot project.
- b. The class that is annotated with the @SpringBootApplication must be kept in the base package.
- c. The one thing that the @SpringBootApplication does is a component scan.
- d. But it will scan only its sub-packages.
- e. As an example, if you put the class annotated with @SpringBootApplication in `co`. Example then @SpringBootApplication will scan all its sub-packages, such as `com.example.a`, `com.example.b`, and `com.example.a.x`.

The @SpringBootApplication is a convenient annotation that adds all the following:

@Configuration

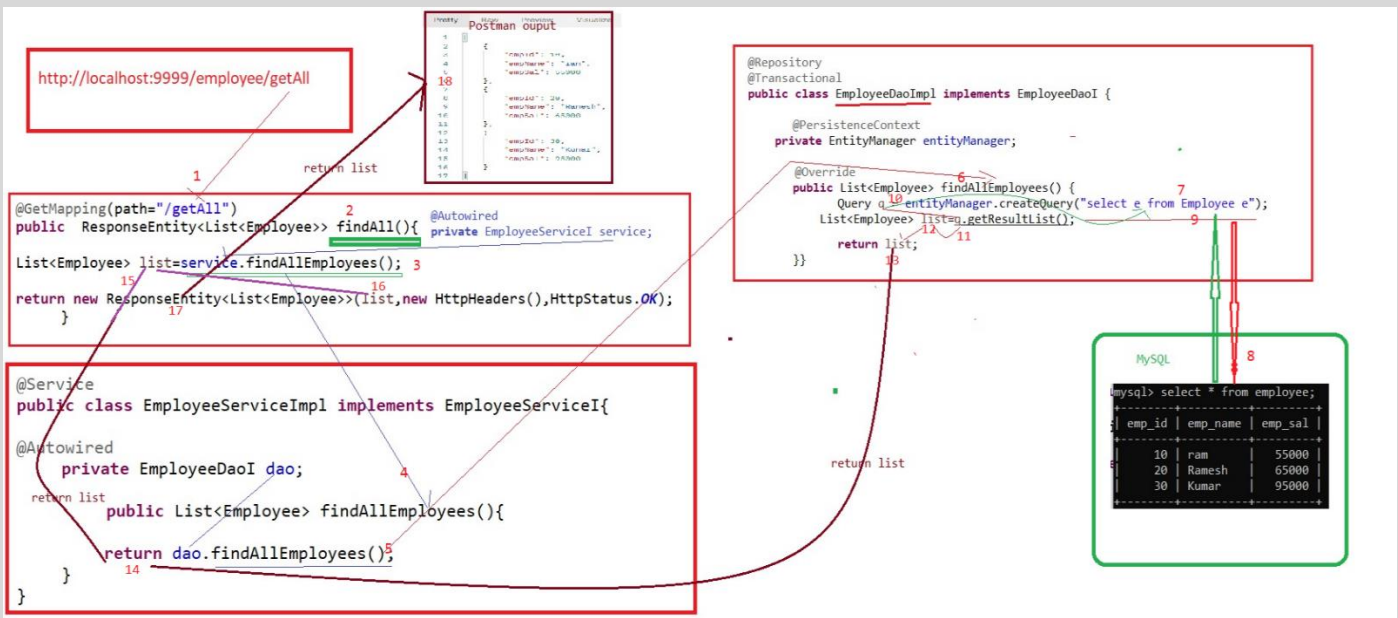
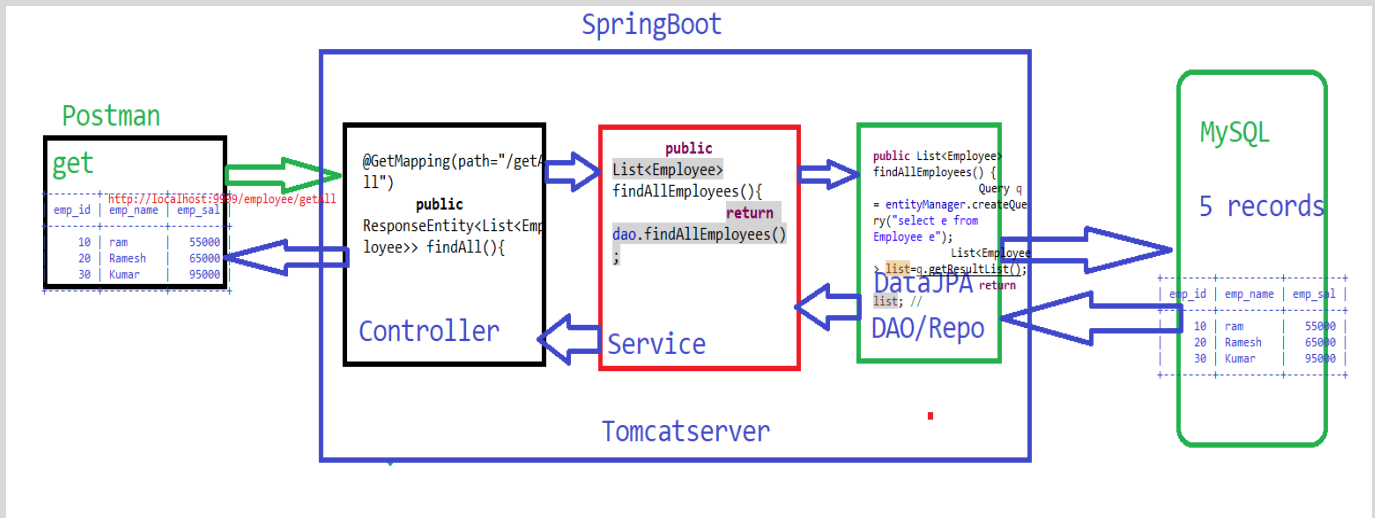
@EnableAutoConfiguration

@ComponentScan

@EnableAutoConfiguration :

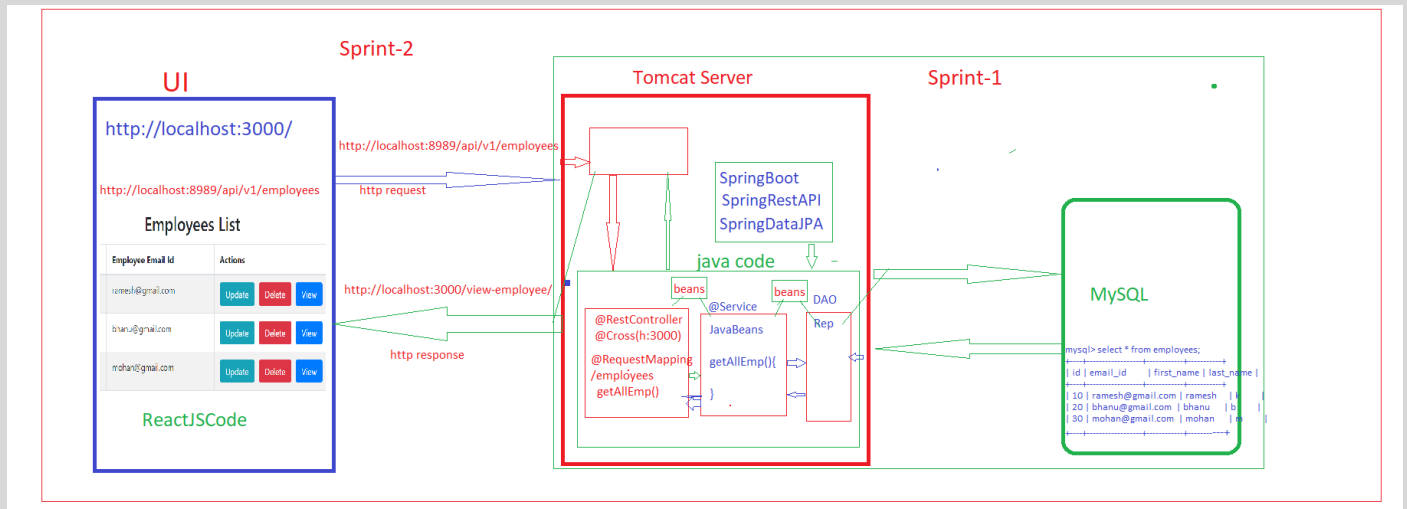
1. This annotation is usually placed on the main application class.
2. The @EnableAutoConfiguration annotation implicitly defines a base “search package”.
3. This annotation tells Spring Boot to start adding beans based on classpath settings, other beans, and various property settings.

Employee Management Project..



SpringBoot and ReactJS Integration

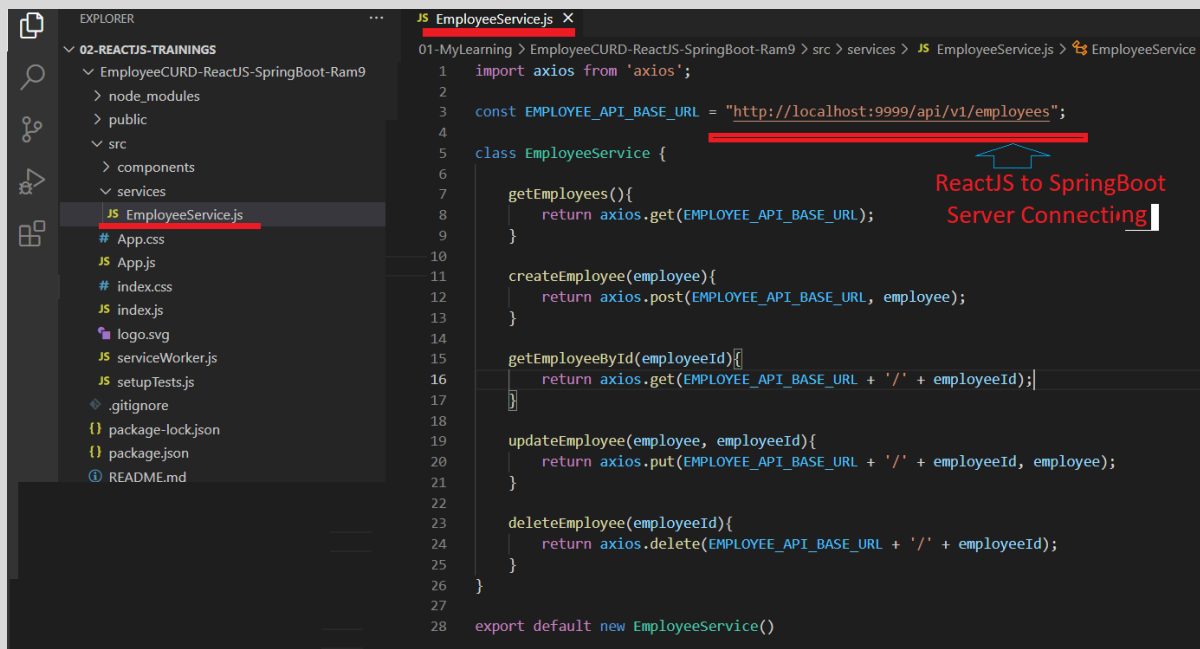
@Ramkumar



Axios is a Javascript library used to make HTTP requests from node. js or XMLHttpRequests from the browser and it supports the Promise API

```
MyTesting-Eclipse-Workspace - EmployeeCURD-SpringBoot-ReactJS-Ram9/src/main/java/com/springboot/empcurd/ram/controller/EmployeeController.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
> 01-SpringBoot-App
> 02-SpringBoot-Controller
> 03-SpringBoot-Controller-JSON
> 04-SpringBoot-Service-JSON
> 05-SpringBoot-Service-JSON-SingleParam
> 06-SpringBoot-Service-JSON-Add-POST-Postman
> 07-SpringBoot-DataJPA-UI-JSP
> 08-JPAOneToOneBI
> 08-SpringBoot-JPA-Product-Layers
> 09-Employee-SpringBoot-CURD-All-Ram
> CoreJavaDevelopment
> EmployeeCURD-SpringBoot-ReactJS-Ram9
  > src/main/java
    > com.springboot.empcurd.ram
      > SpringbootBackendApplication.java
      > com.springboot.empcurd.ram.controller
        > EmployeeController.java
      > com.springboot.empcurd.ram.exception
      > com.springboot.empcurd.ram.model
      > com.springboot.empcurd.ram.repository
  > src/main/resources
    > application.properties
  > src/test/java
  > JRE System Library [JavaSE-1.8]
  > Maven Dependencies
  > src
  > target
  > mvnw
  > pom.xml

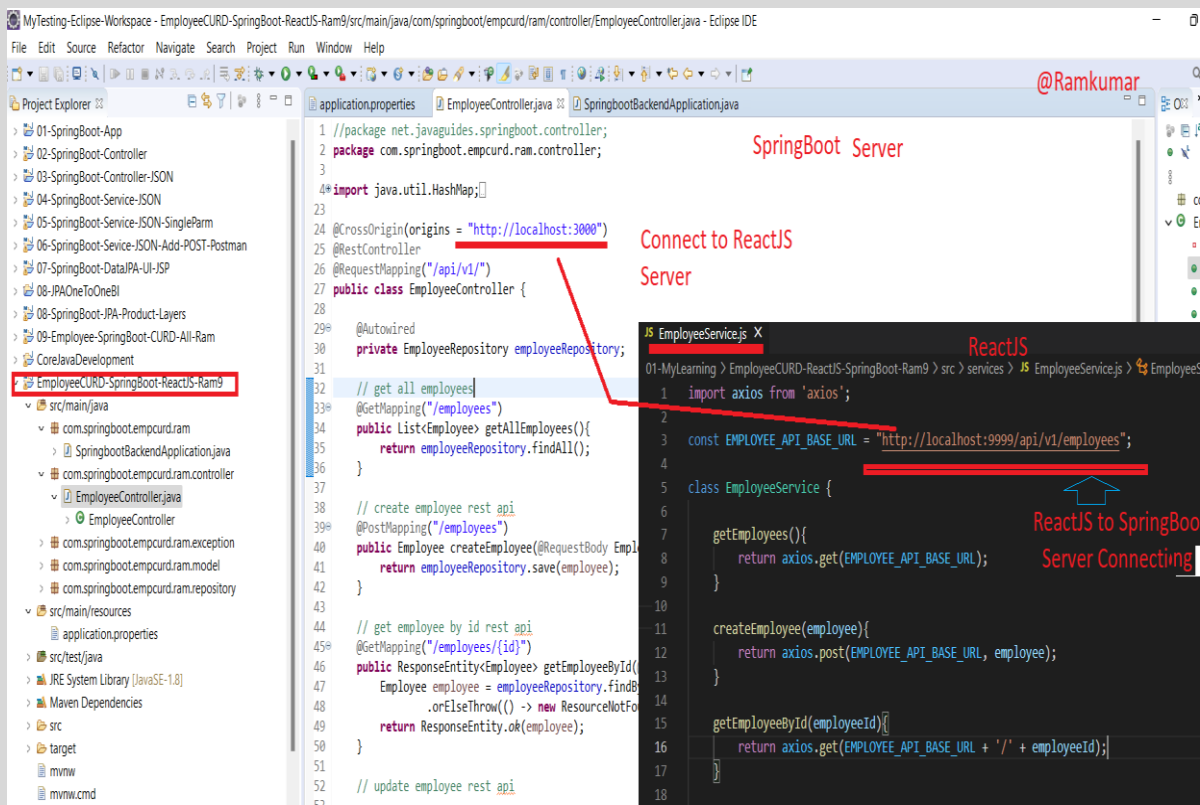
application.properties EmployeeController.java SpringbootBackendApplication.java
1 //package net.javaguides.springboot.controller;
2 package com.springboot.empcurd.ram.controller;
3
4 import java.util.HashMap;
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24 @CrossOrigin(origins = "http://localhost:3000")
25 @RestController
26 @RequestMapping("/api/v1/")
27 public class EmployeeController {
28
29     @Autowired
30     private EmployeeRepository employeeRepository;
31
32     // get all employees
33     @GetMapping("/employees")
34     public List<Employee> getAllEmployees(){
35         return employeeRepository.findAll();
36     }
37
38     // create employee rest api
39     @PostMapping("/employees")
40     public Employee createEmployee(@RequestBody Employee employee) {
41         return employeeRepository.save(employee);
42     }
43
44     // get employee by id rest api
45     @GetMapping("/employees/{id}")
46     public ResponseEntity<Employee> getEmployeeById(@PathVariable Long id) {
47         Employee employee = employeeRepository.findById(id)
48             .orElseThrow(() -> new ResourceNotFoundException("Employee not exist with id : " + id));
49         return ResponseEntity.ok(employee);
50     }
51
52     // update employee rest api
```



This screenshot shows the VS Code editor with the file Explorer on the left and the code editor in the center. The file Explorer shows a project structure for '02-REACTJS-TRAININGS' with a sub-project 'EmployeeCURD-ReactJS-SpringBoot-Ram9'. The code editor displays the 'EmployeeService.js' file. The code defines an axios instance, a base URL, and an EmployeeService class with methods for getting, creating, updating, and deleting employees. A red arrow points from the text 'ReactJS to SpringBoot Server Connecting' to the axios.get call in the getEmployees method.

```
1 import axios from 'axios';
2
3 const EMPLOYEE_API_BASE_URL = "http://localhost:9999/api/v1/employees";
4
5 class EmployeeService {
6
7   getEmployees(){
8     return axios.get(EMPLOYEE_API_BASE_URL);
9   }
10
11   createEmployee(employee){
12     return axios.post(EMPLOYEE_API_BASE_URL, employee);
13   }
14
15   getEmployeeById(employeeId){
16     return axios.get(EMPLOYEE_API_BASE_URL + '/' + employeeId);
17   }
18
19   updateEmployee(employee, employeeId){
20     return axios.put(EMPLOYEE_API_BASE_URL + '/' + employeeId, employee);
21   }
22
23   deleteEmployee(employeeId){
24     return axios.delete(EMPLOYEE_API_BASE_URL + '/' + employeeId);
25   }
26 }
27
28 export default new EmployeeService();
```

axios



This screenshot shows the Eclipse IDE with the Project Explorer on the left and the code editor in the center. The Project Explorer shows a project structure for '01-SpringBoot-App' with a sub-project 'EmployeeCURD-SpringBoot-ReactJS-Ram9'. The code editor displays the 'SpringBootBackendApplication.java' file. The code defines a SpringBootBackendApplication class with a main method and a controller. A red arrow points from the text 'Connect to ReactJS Server' to the axios.get call in the getEmployees method. Another red arrow points from the text 'ReactJS to SpringBoot Server Connecting' to the axios.get call in the getEmployees method.

```
1 //package net.javaguides.springboot.controller;
2 package com.springboot.empcurd.ram.controller;
3
4 import java.util.HashMap;
5
6 @CrossOrigin(origins = "http://localhost:3000")
7 @RestController
8 @RequestMapping("/api/v1/")
9 public class EmployeeController {
10
11   @Autowired
12   private EmployeeRepository employeeRepository;
13
14   // get all employees
15   @GetMapping("/employees")
16   public List<Employee> getAllEmployees(){
17     return employeeRepository.findAll();
18   }
19
20   // create employee rest api
21   @PostMapping("/employees")
22   public Employee createEmployee(@RequestBody Employee employee){
23     return employeeRepository.save(employee);
24   }
25
26   // get employee by id rest api
27   @GetMapping("/employees/{id}")
28   public ResponseEntity<Employee> getEmployeeById(@PathVariable Long id){
29     Employee employee = employeeRepository.findById(id).orElseThrow(() -> new ResourceNotFoundException(id));
30     return ResponseEntity.ok(employee);
31   }
32
33   // update employee rest api
34 }
```

The screenshot shows a web browser at localhost:3000/employees displaying an 'Employee Management ReactApp'. The page has a header 'Employee Management ReactApp' and a main title 'Employees List'. Below the title is a blue 'Add Employee' button. A table lists employee data with columns: Employee First Name, Employee Last Name, Employee Email Id, and Actions. The table contains one row for 'Ramkumar' with last name 'K' and email 'ramkumar.k999@gmail.com'. The Actions column has 'Update', 'Delete', and 'View' buttons. In the foreground, a MySQL 5.5 Command Line Client window shows the command 'select * from employees;' and its output, which matches the data in the web application table.

Employee First Name	Employee Last Name	Employee Email Id	Actions
Ramkumar	K	ramkumar.k999@gmail.com	<button>Update</button> <button>Delete</button> <button>View</button>

```
MySQL 5.5 Command Line Client
1 row in set (0.00 sec)

mysql> select * from employees;
+----+-----+-----+-----+
| id | email_id | first_name | last_name |
+----+-----+-----+-----+
| 1 | ramkumar.k999@gmail.com | Ramkumar | K |
+----+-----+-----+-----+
1 row in set (0.00 sec)

mysql>
```

What is Axios?

1. Axios is used to communicate with the backend and it also supports the Promise API that is native to JS ES6.
2. It is a library which is used to make requests to an API, return data from the API, and then do things with that data in our React application.
3. Axios is a Javascript library used to make HTTP requests from node. js or XMLHttpRequests from the browser and it supports the Promise API
4. Axios allows you to communicate with the APIs in your React project.
5. Axios has function names that match any HTTP methods. To perform a GET request, you use the .get() method.

C:\EmployeeCURD-ReactJS-SpringBoot-Ram9> npm install axios/>npm install axios

Example :

```

JS EmployeeService.js X
01-MyLearning > EmployeeCURD-ReactJS-SpringBoot-Ram9 > src > services > JS EmployeeService.js > ...
1  import axios from 'axios';
2
3  const EMPLOYEE_API_BASE_URL = "http://localhost:9999/api/v1/employees";
4
5  class EmployeeService {
6
7      getEmployees(){
8          return axios.get(EMPLOYEE_API_BASE_URL);
9      }
10
11      createEmployee(employee){
12          return axios.post(EMPLOYEE_API_BASE_URL, employee);
13      }
14
15      getEmployeeById(employeeId){
16          return axios.get(EMPLOYEE_API_BASE_URL + '/' + employeeId);
17      }
18
19      updateEmployee(employee, employeeId){
20          return axios.put(EMPLOYEE_API_BASE_URL + '/' + employeeId, employee);
21      }
22
23      deleteEmployee(employeeId){
24          return axios.delete(EMPLOYEE_API_BASE_URL + '/' + employeeId);
25      }
26  }
27
28  export default new EmployeeService()

```

```

JS EmployeeService.js X
01-MyLearning > EmployeeCURD-ReactJS-SpringBoot-Ram9 > src > services > JS EmployeeService.js > ...
1  import axios from 'axios';
2
3  const EMPLOYEE_API_BASE_URL = "http://localhost:9999/api/v1/employees";
4
5  class EmployeeService {
6
7      getEmployees(){
8          return axios.get(EMPLOYEE_API_BASE_URL);
9      }
10
11      createEmployee(employee){
12          return axios.post(EMPLOYEE_API_BASE_URL, employee);
13      }
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

node + v

PS C:\Ramkumar\UI-Development\02-ReactJS-Trainings\01-MyLearning\EmployeeCURD-ReactJS-SpringBoot-Ram9> npm install axios

npm WARN deprecated chokidar@2.1.8: Chokidar 2 does not receive security updates since 2019. Upgrade to chokidar 3 with 15x fewer dependencies

[] / reify:readable-stream: timing reifyNode:node_modules/watchpack-chokidar2/node_modules/chokidar Completed in 103ms