.

SNO	Date of Version	Document Name and Version:	Document Created By
1	15-01-2022	Sring-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. SrpingBoot with JPA
4. SpringBoot with DataJPA

Spring Annotations

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

Ramkumar , WhatsApp : +91 9701153333

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.

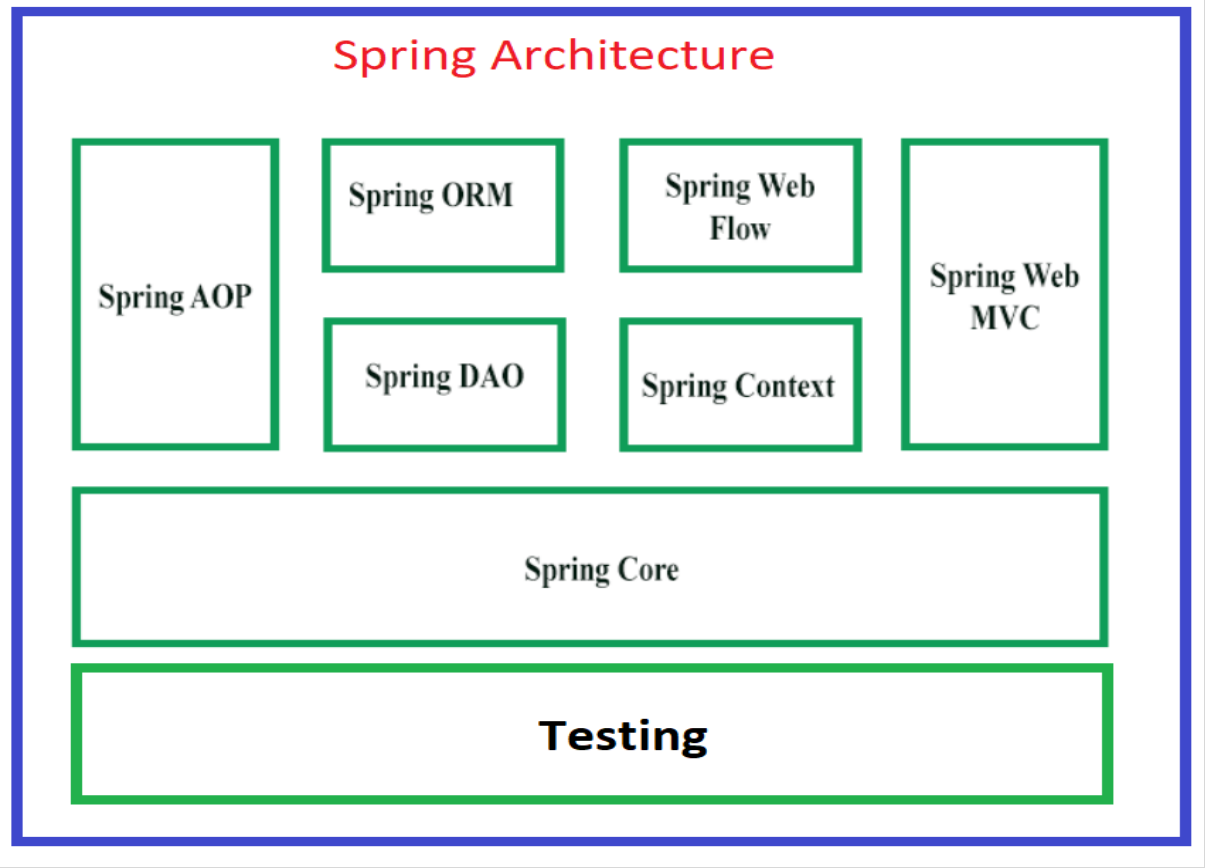
5. Languages: Kotlin, Groovy, dynamic languages.

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

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

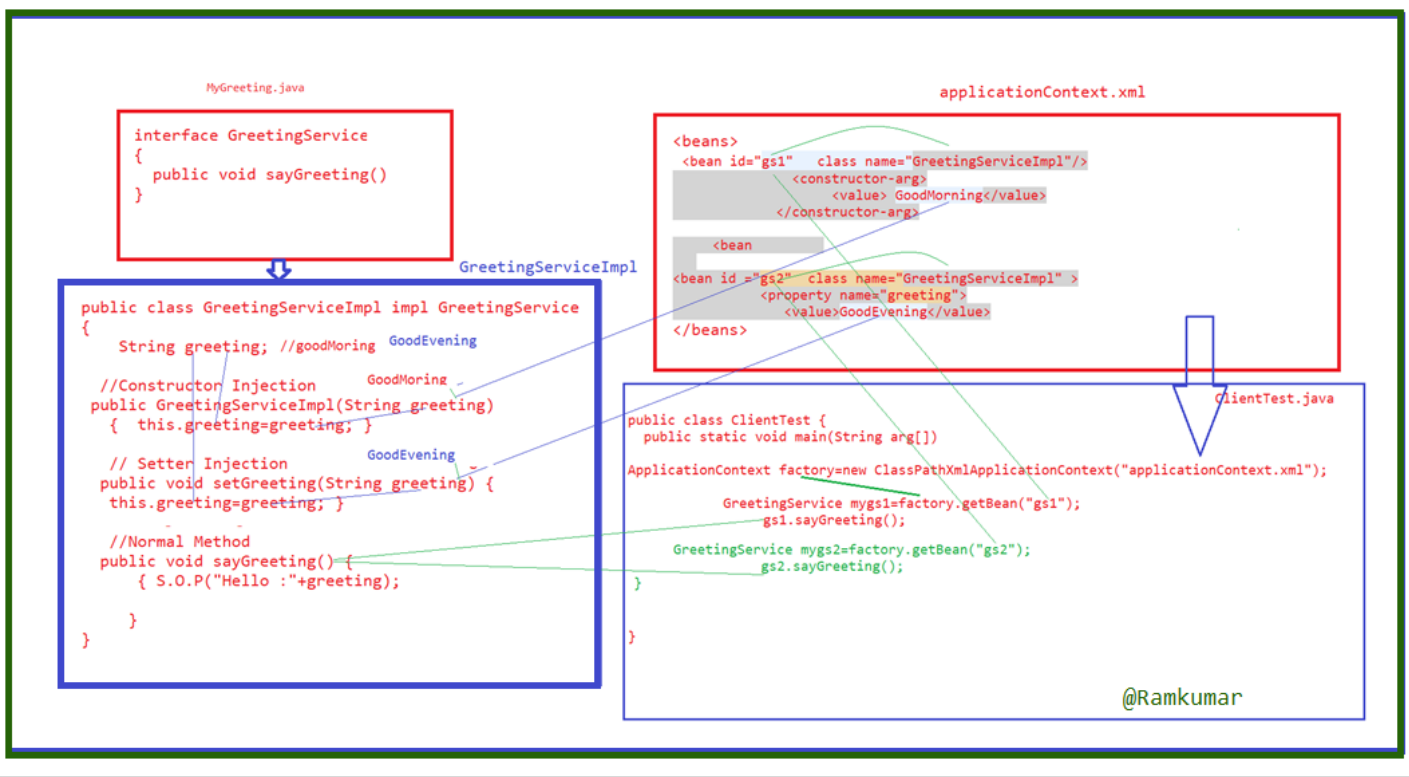
These modules can drastically reduce the development time of an application.

```
IOC (Inversion of Controller)
   Giving control to the contanier to get an instrance of the Objec is
   classed Inversion of Congroller , means insted of you are creating object
   using the new operator , let the container do that for you.

DI(Depenndency Injection);
   Way of injection properties to an object is called Dependency Injection.

   Spring Framework Supports Two types of Depency Injection :
       1. Constructor Injection
       2. Setter Injection
```



Example 1:

S01-Spring-Core-IOC-DI



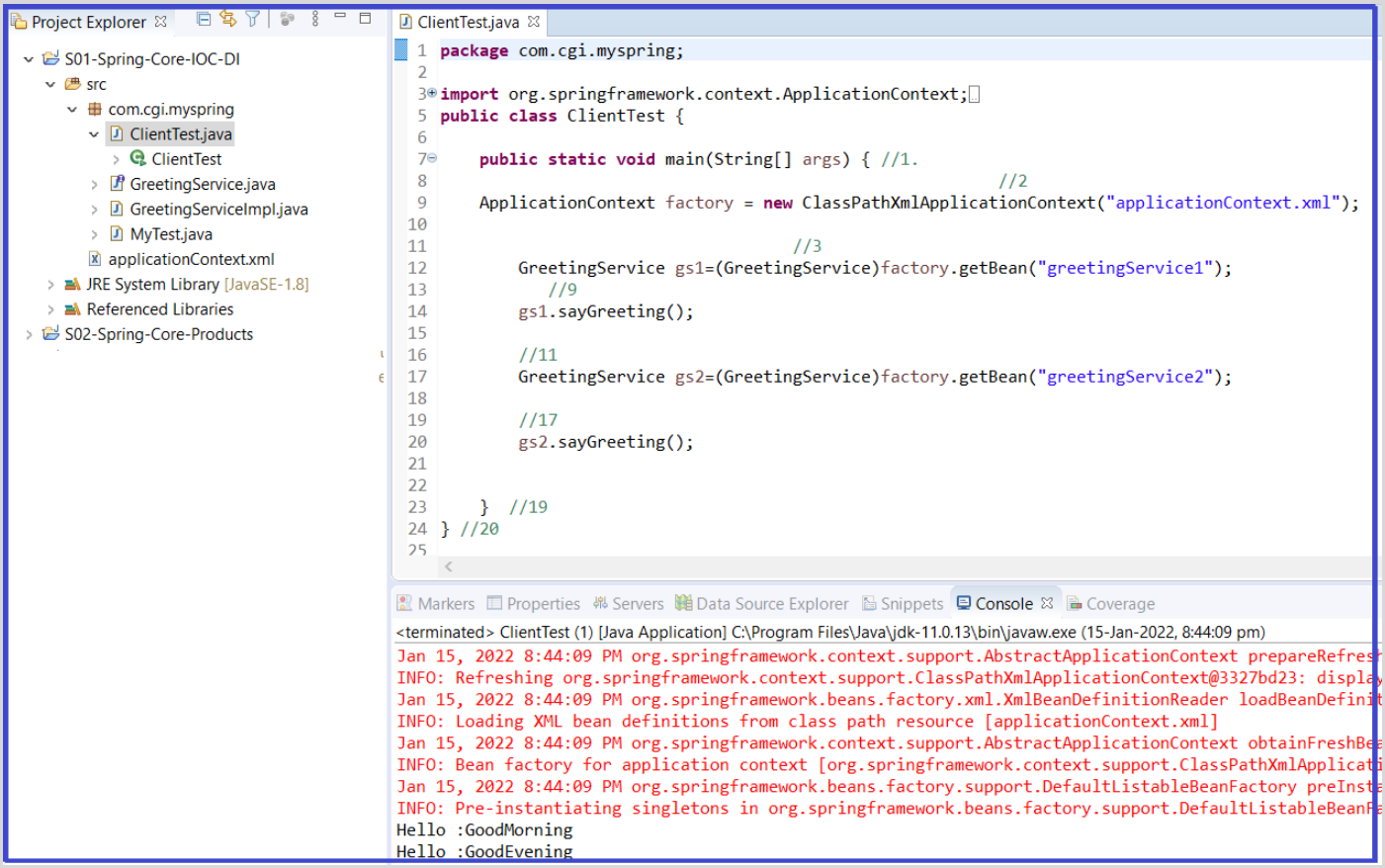
ClientTest.java | applicationContext.xml | GreetingServiceImpl.java

```java
package com.cgi.myspring;

public class GreetingServiceImpl implements GreetingService {
  String greeting; //8. GoodMorning  16//GoodEvening

    public GreetingServiceImpl() {

    }

    //6.
   public GreetingServiceImpl(String greeting)
   {
       this.greeting=greeting;  //7
   }

         //GoodEvening       //14
    public void setGreeting(String greeting)
    {
        this.greeting=greeting; //15
    }

    @Override    //10      //18
    public void sayGreeting() {
        System.out.println("Hello :"+greeting);


        }
    }
```

ClientTest.java | applicationContext.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
    "http://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
    <!-- .  Constructor Injection     4. -->
  <bean id="greetingService1" class="com.cgi.myspring.GreetingServiceImpl">
      <constructor-arg>   <!-- 5. -->
                <value>GoodMorning</value>
      </constructor-arg>

  </bean>
   <!-- .  Setter Injection   12 -->
   <bean id="greetingService2"  class="com.cgi.myspring.GreetingServiceImpl">
      <property name="greeting">
         <value>GoodEvening</value>  <!-- 13 -->
       </property>

   </bean>

</beans>
```

Ramkumar , WhatsApp : +91 9701153333



Spring – JDBC(DAO) : Integration

S03-Spring-with-DAO

Ramkumar , WhatsApp : +91 9701153333

AccountBean

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

	public Account() {
	}
```

for DatabaseConnection

applicationContext_Dao.xml

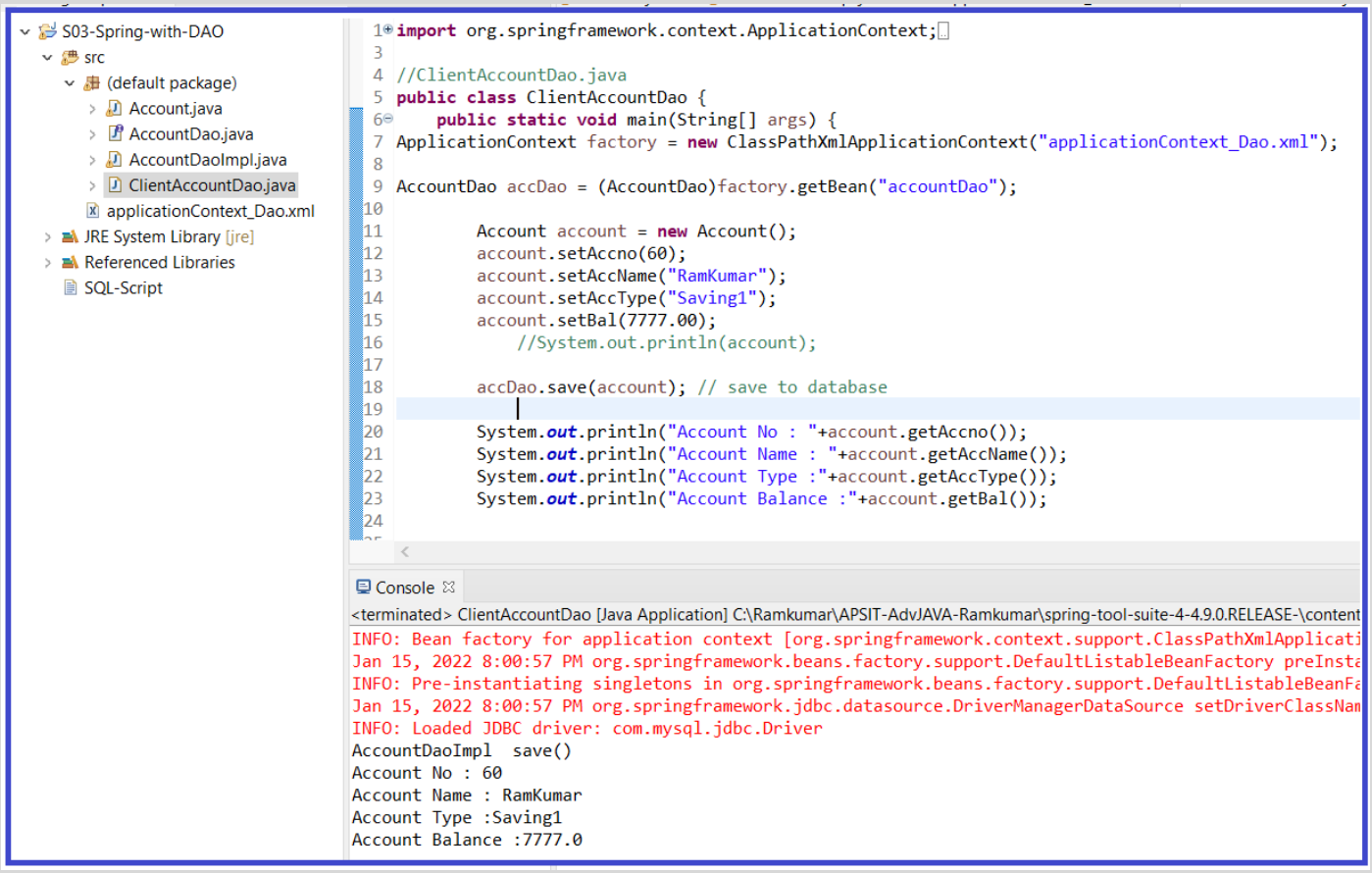
```xml
<beans>
<bean id="dataSource"
	class="org.springframework.jdbc.datasource.DriverManagerDataSource">
	<property name="driverClassName">
		<value>com.mysql.jdbc.Driver</value>
	</property>
	<property name="url">
		<value>jdbc:mysql://localhost:3306/apsitdb</value>
	</property>
	<property name="username">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

AccountDaoImpl.class

```java
public class AccountDaoImpl implements AccountDao {
 //This Object refecrnece 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
		}}
```

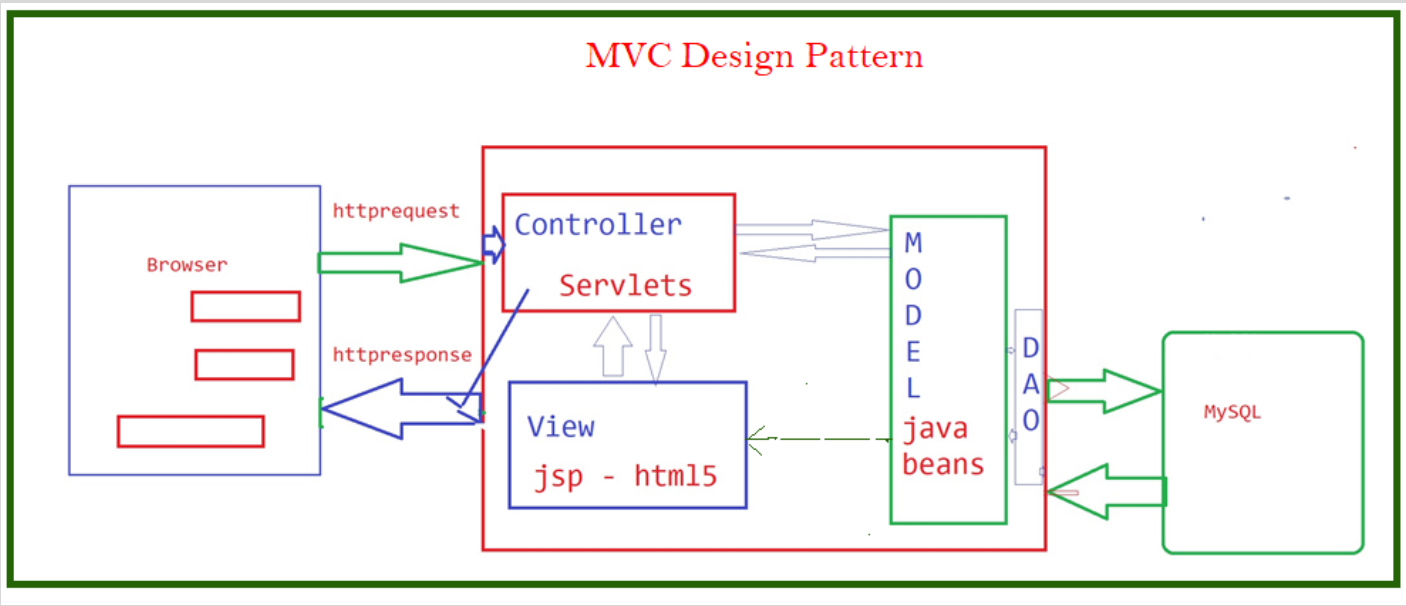
ClientAccountDao.class

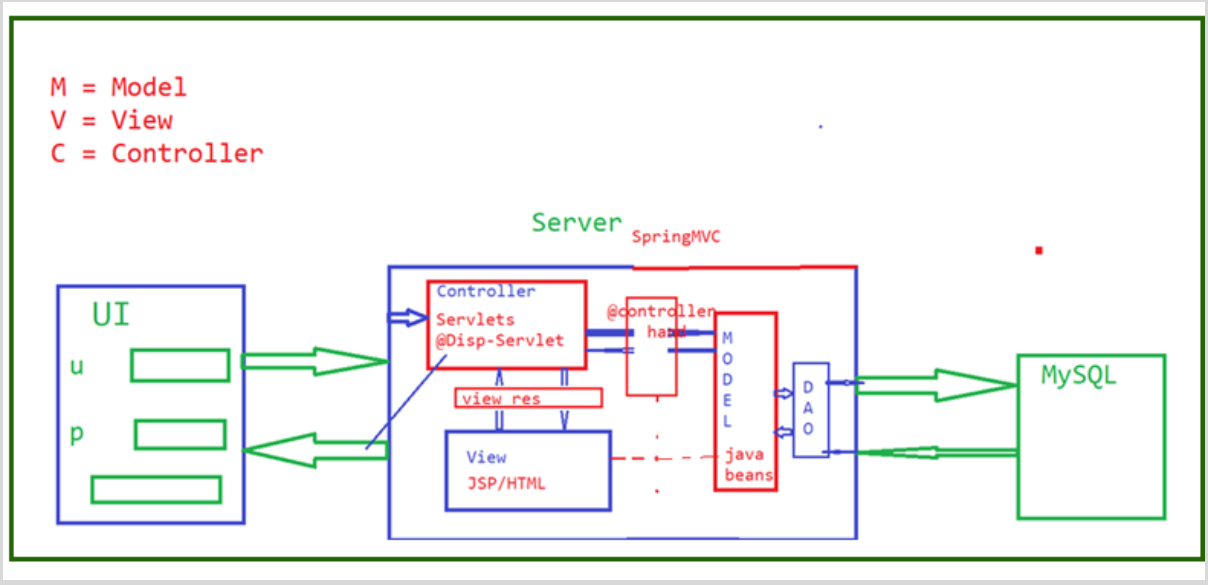
```java
public class ClientAccountDao {
	public static void main(String[] args) {
ApplicationContext factory = new ClassPathXmlApplicationContext("applicationContext_Dao.xml");
	AccountDao accDao = (AccountDao)factory.getBean("accountDao");
		Account account = new Account();
		account.setAccno(50);
		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());  }}
```

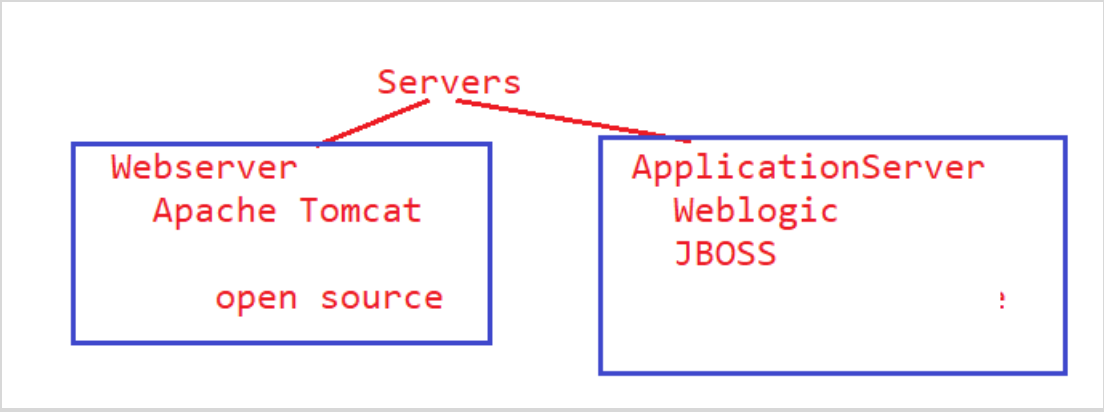


Ramkumar , WhatsApp : +91 9701153333

MVC Design Pattern

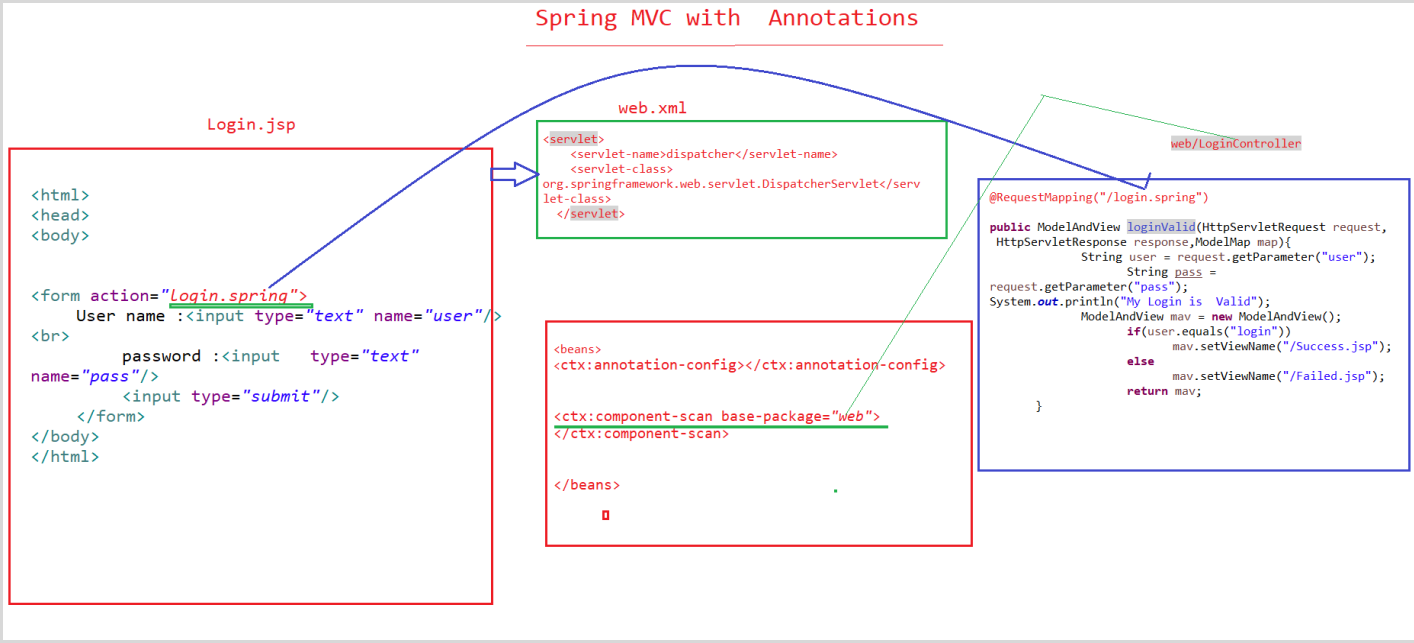


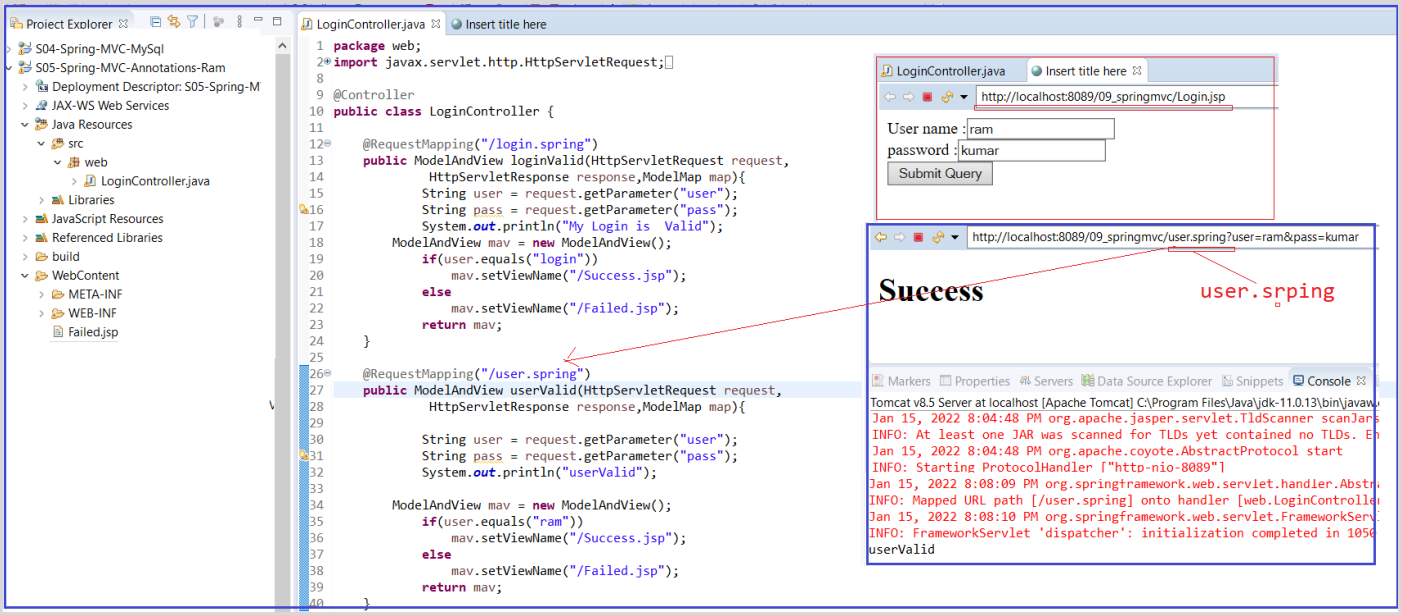




Ramkumar , WhatsApp : +91 9701153333

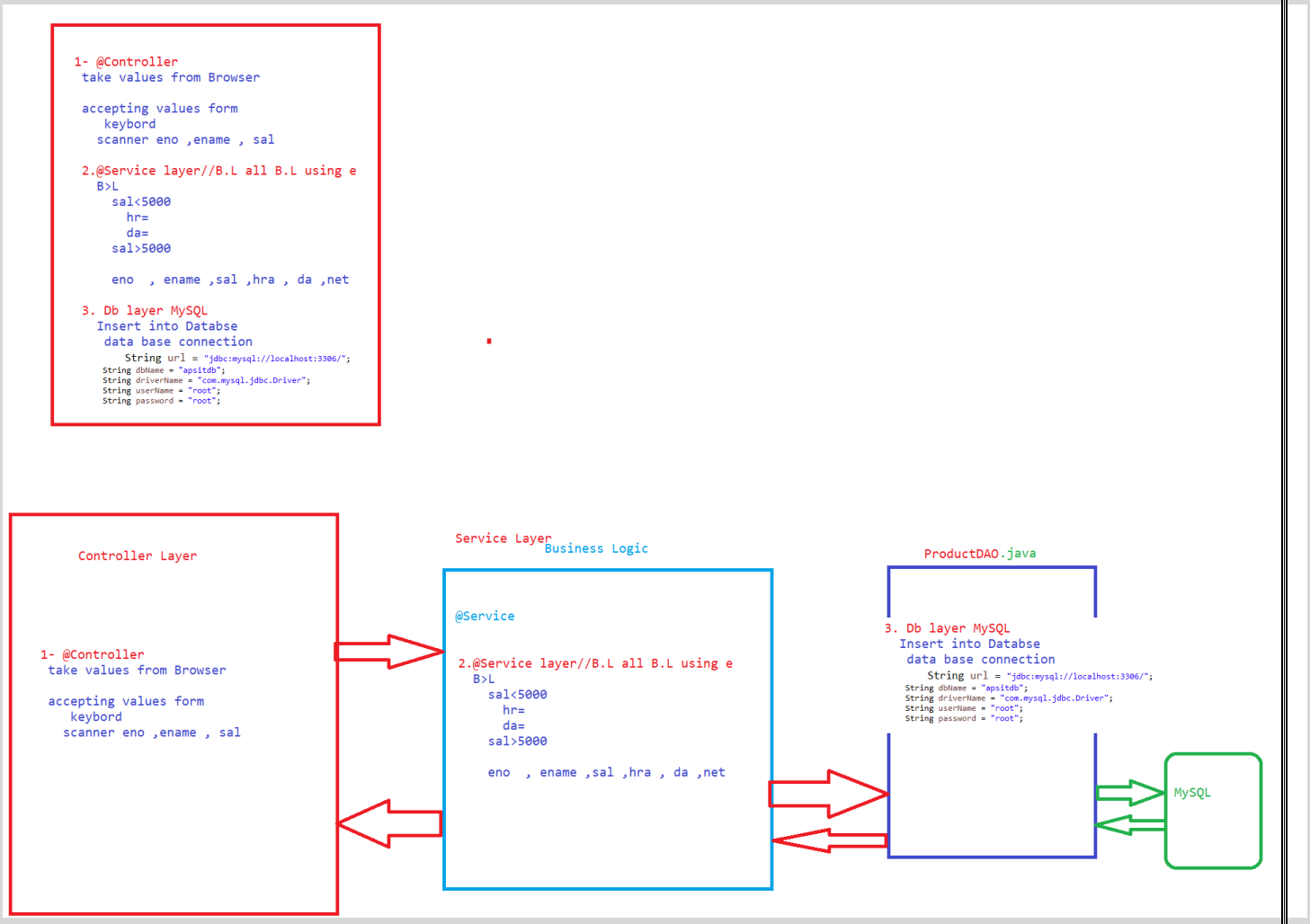
http://localhost:8089/09_springmvc/user.spring?user=ram&pass=ram

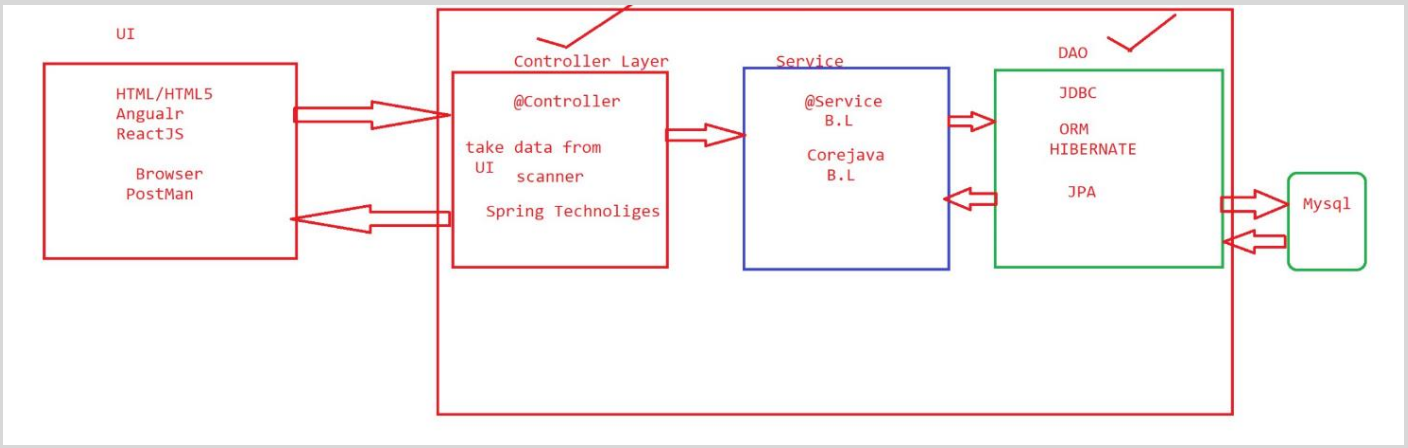


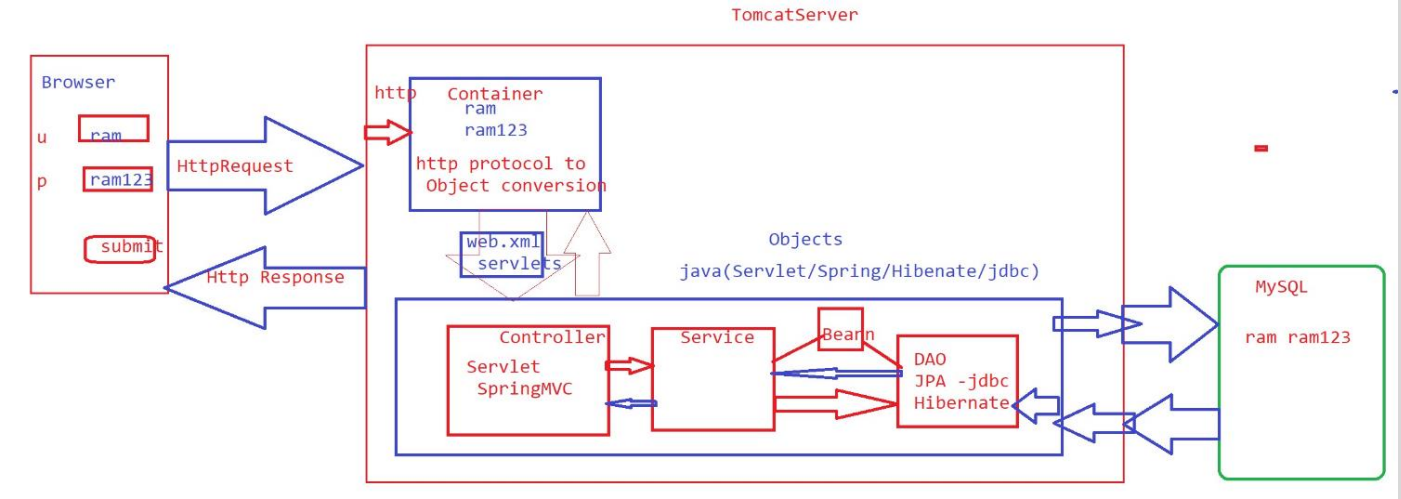


Ramkumar , WhatsApp : +91 9701153333

Layered architecture







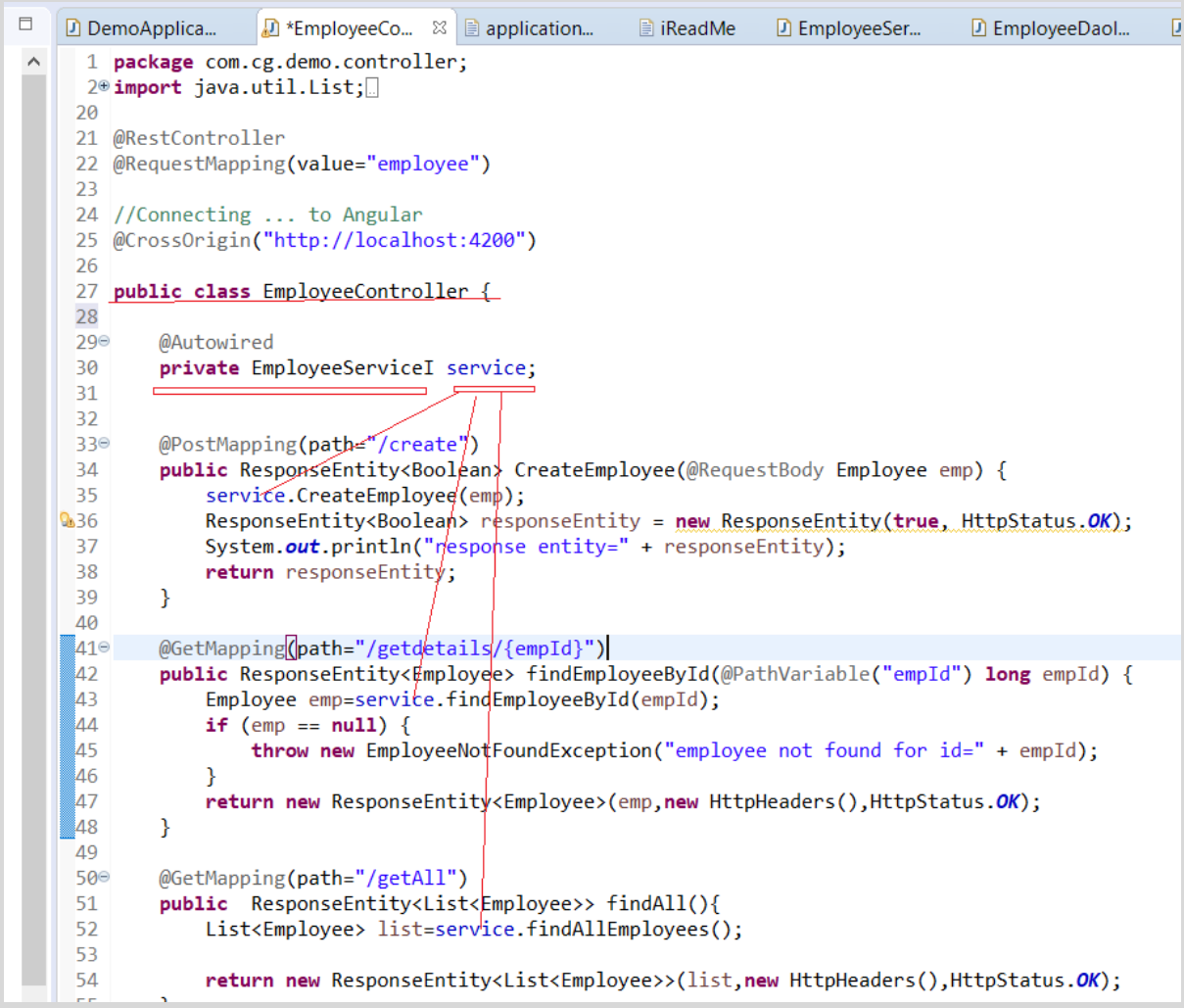
Spring Annotations

@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;
}
```



@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.

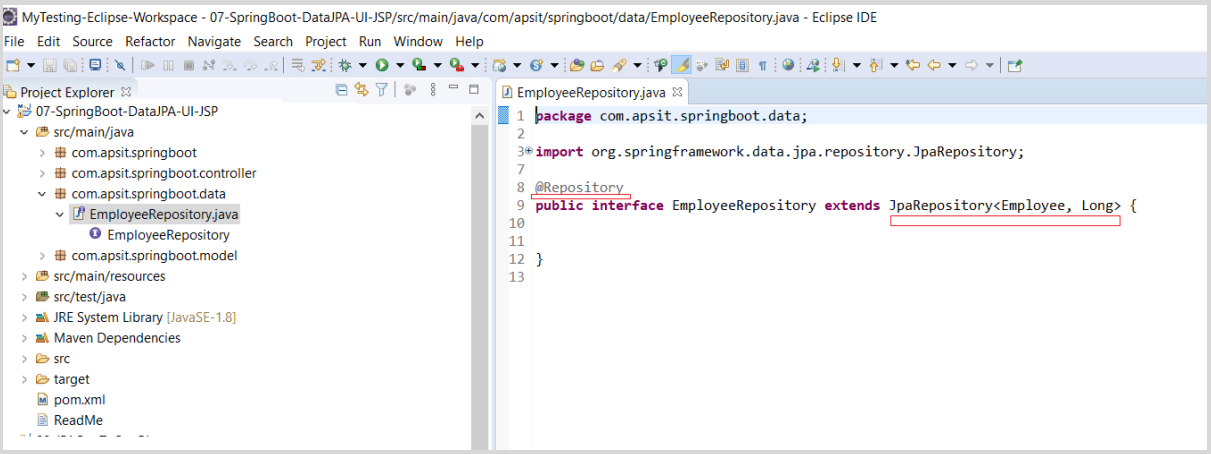
DemoApplica... *EmployeeCo... application... iReadMe *EmployeeSe...

```java
package com.cg.demo.service;
import java.util.List;

@Service
public class EmployeeServiceImpl implements EmployeeServiceI{

    @Autowired
    private EmployeeDaoI dao;

    public Employee CreateEmployee(Employee emp) {
        return dao.CreateEmployee(emp);
    }

    public Employee findEmployeeById(long empId) {
        return dao.findEmployeeById(empId);
    }
    public Employee updateEmployee(Employee emp) {
        return dao.updateEmployee(emp);
    }
    public List<Employee> findAllEmployees(){
         return dao.findAllEmployees();
    }
    public void deleteEmployee(long empId) {
         dao.deleteEmployee(empId);
    }
}

```

@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 :

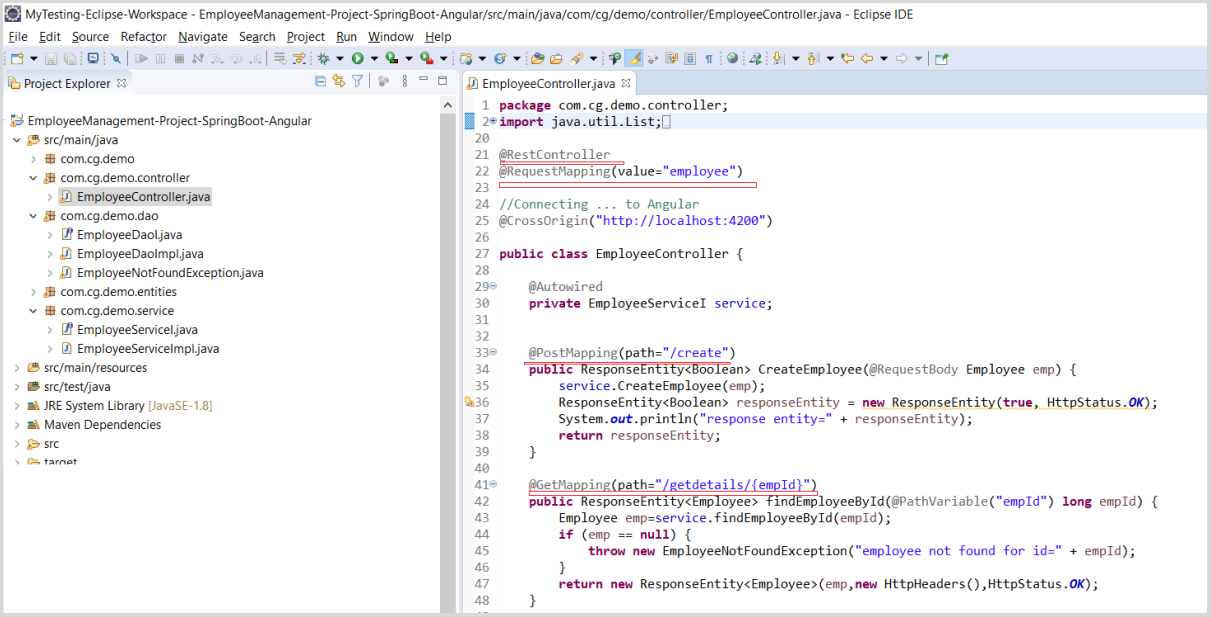


Request Mapping Variants

@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 :



@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

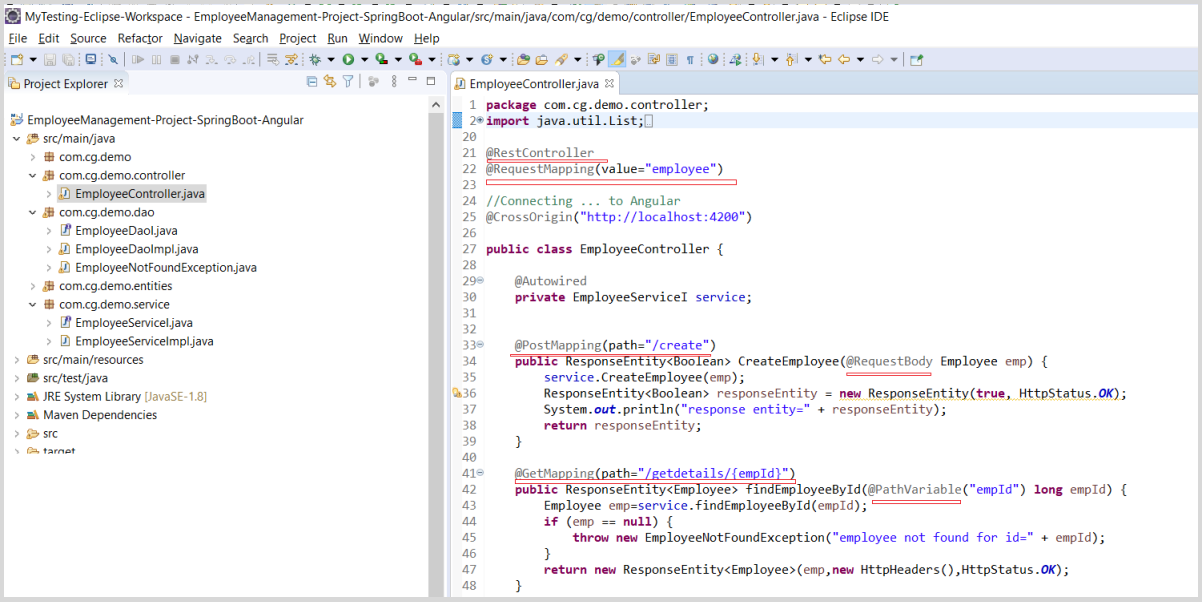
2. @PostMapping is a *composed annotation* that acts as a shortcut for @RequestMapping(method = RequestMethod.POST).
3. Annotation for mapping HTTP POST requests onto specific handler methods.
4. Should only be used to create new resource.
5. Example: @PostMapping("/home")

@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

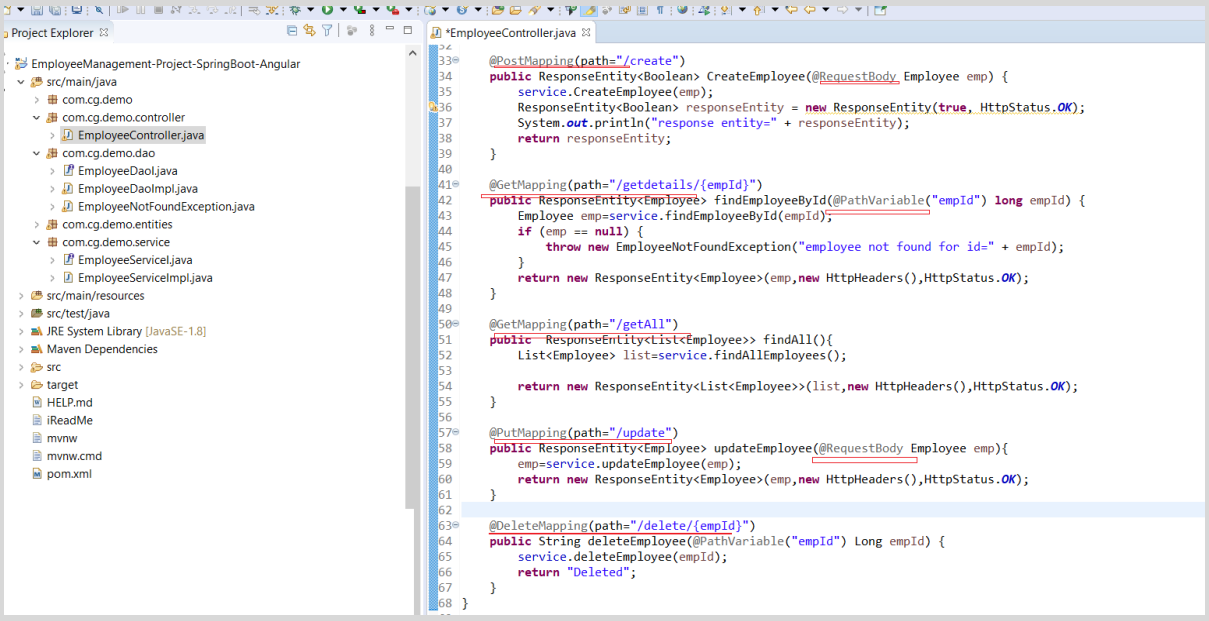
1. **@PutMapping** is specialized version of @RequestMapping annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.PUT).
2. @PutMapping annotation is used to accept HTTP Put request containing a Request Body with JSON or XML payload, so that you can perform let's say an update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



@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 :



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?

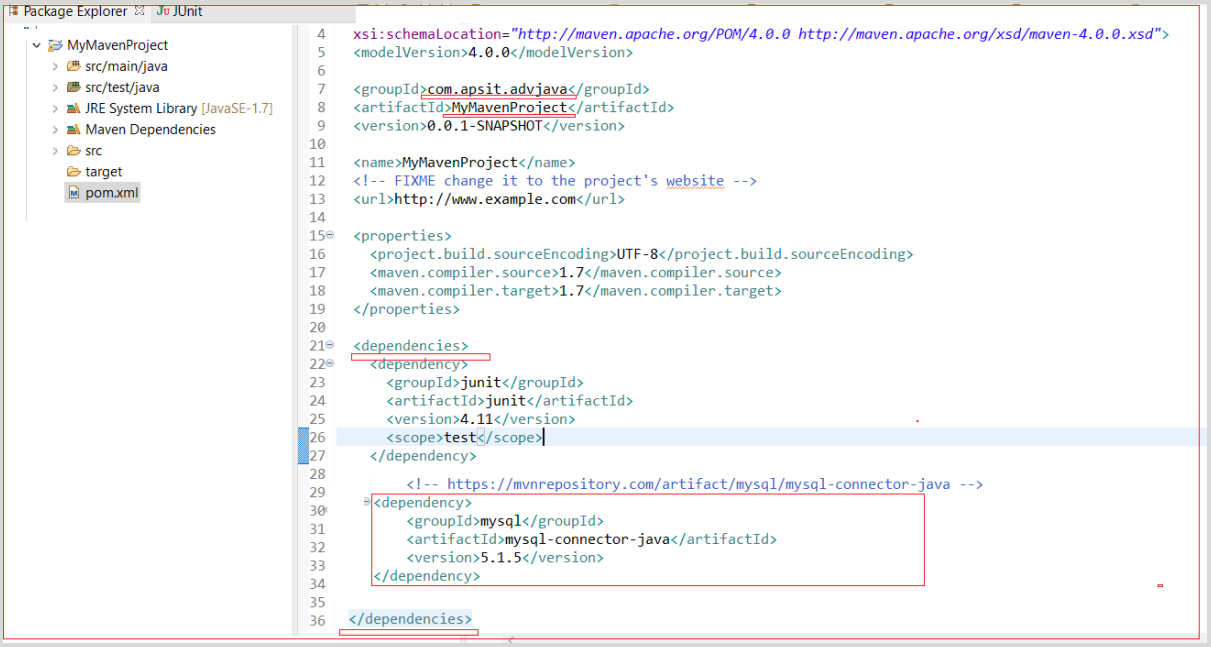
1. A Project Object Model or POM is the fundamental unit of work in Maven.
2. It is an XML file that contains information about the project and configuration details used by Maven to build the project.
3. It contains default values for most projects.
4. Examples for this is the build directory, which 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:

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'.

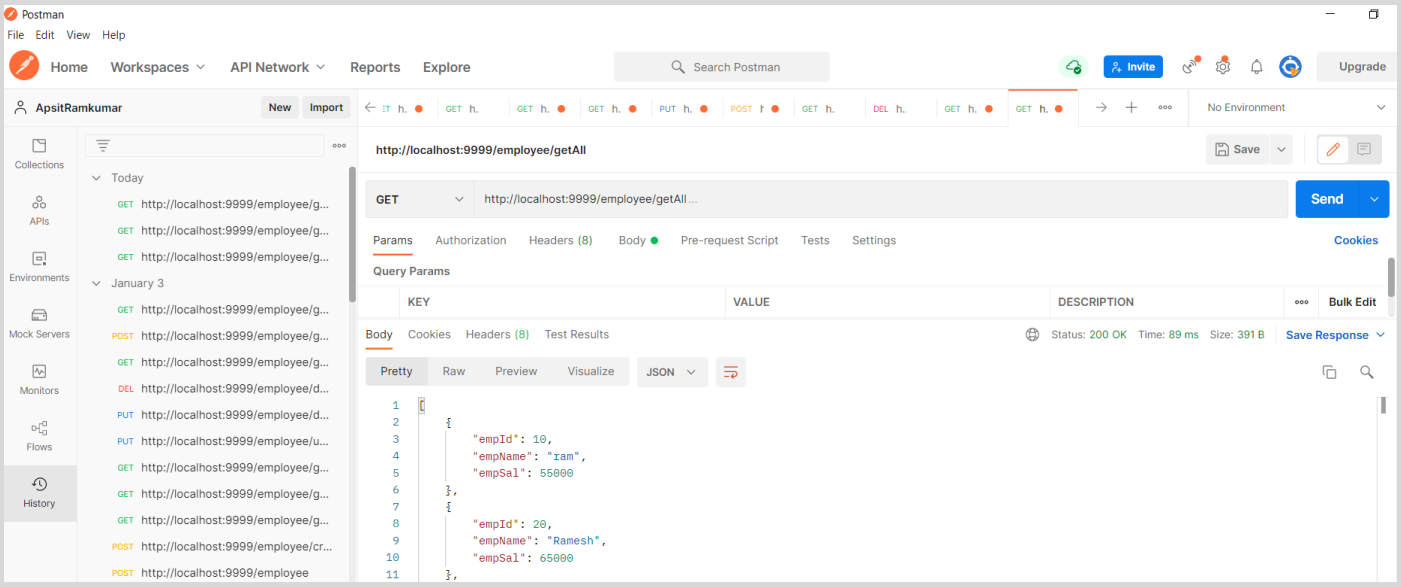
500 Series > Server error responses, for example, '500 Internal Server Error.'

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.

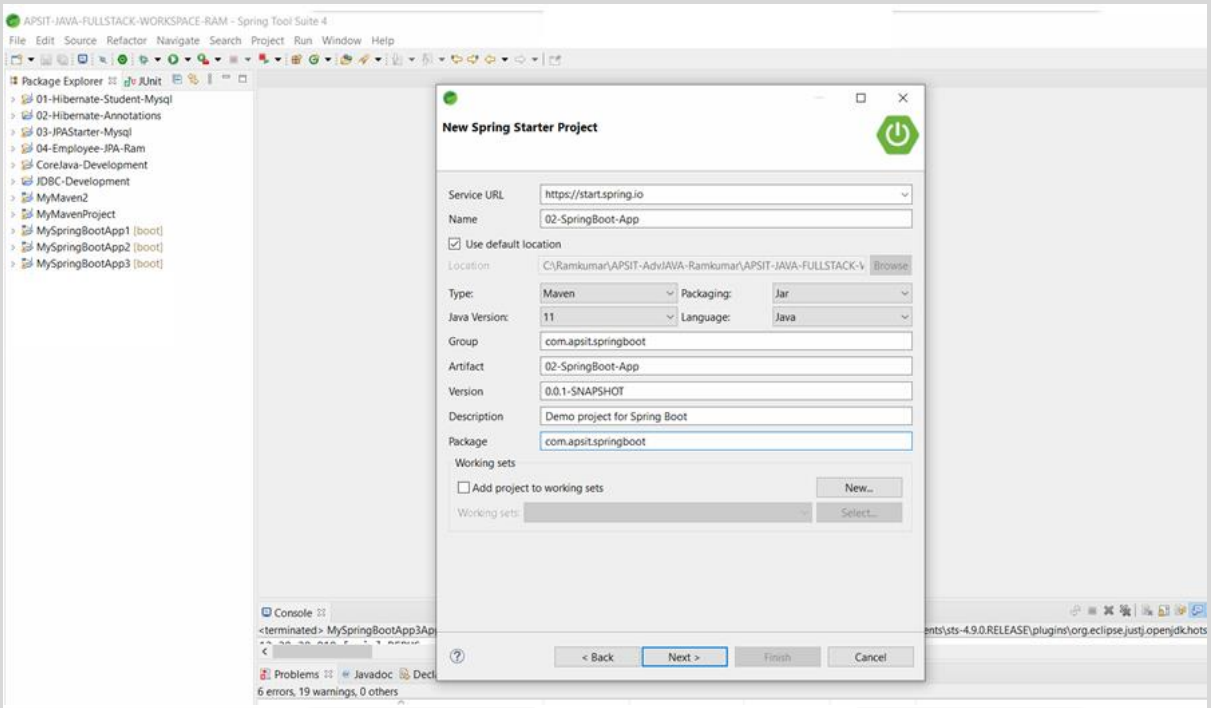
3. Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".

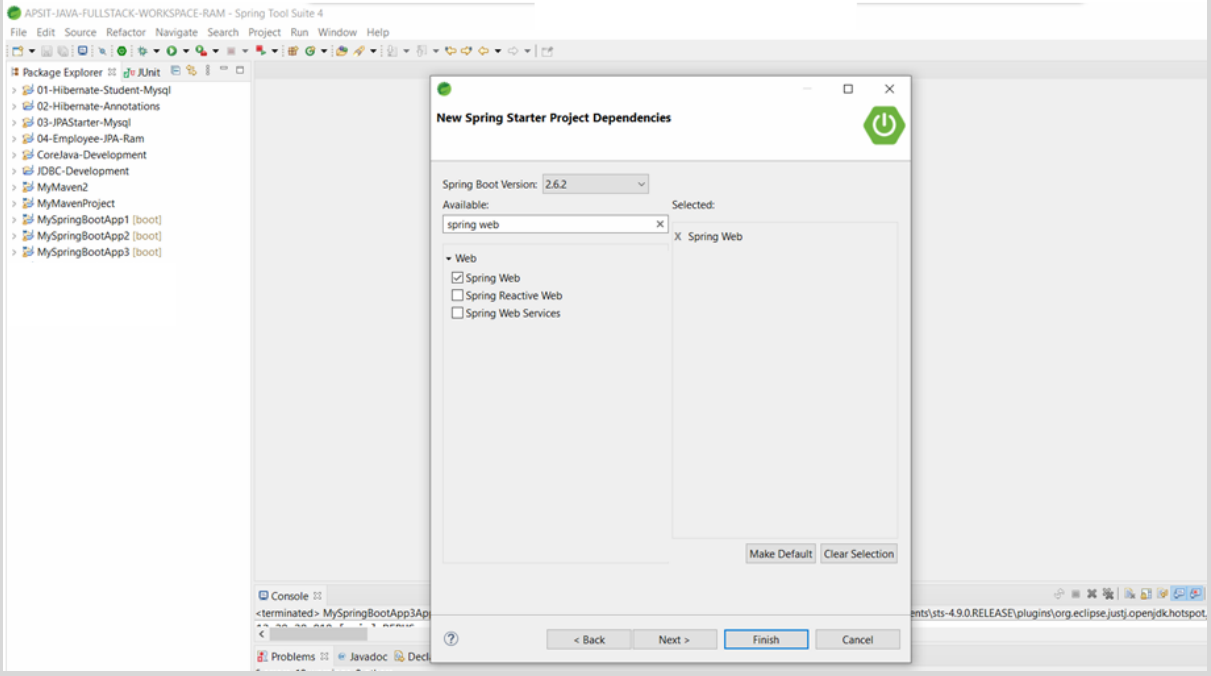
4. We take an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need minimal Spring configuration.

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

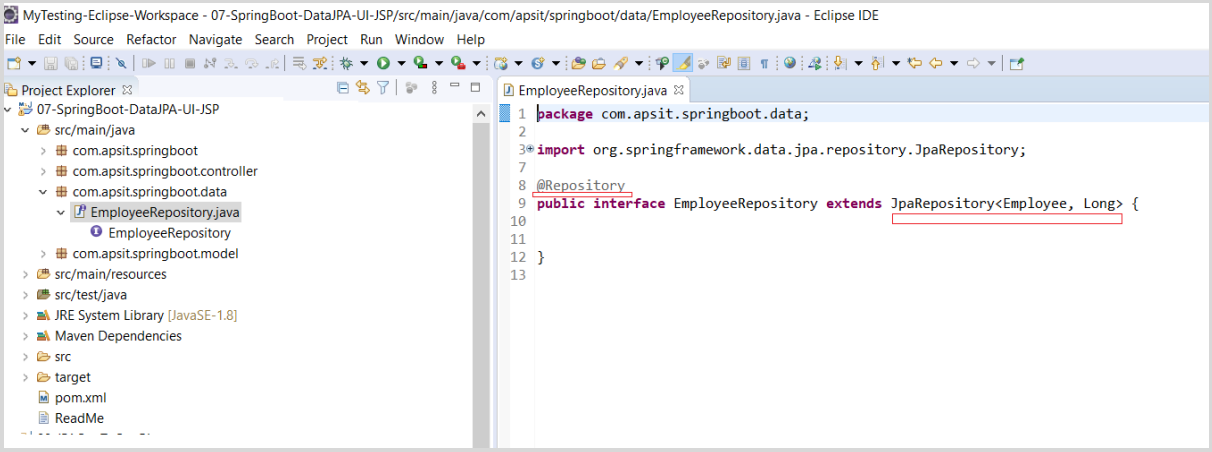
How to create SpringBoot Application:





Ramkumar , WhatsApp : +91 9701153333

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 :

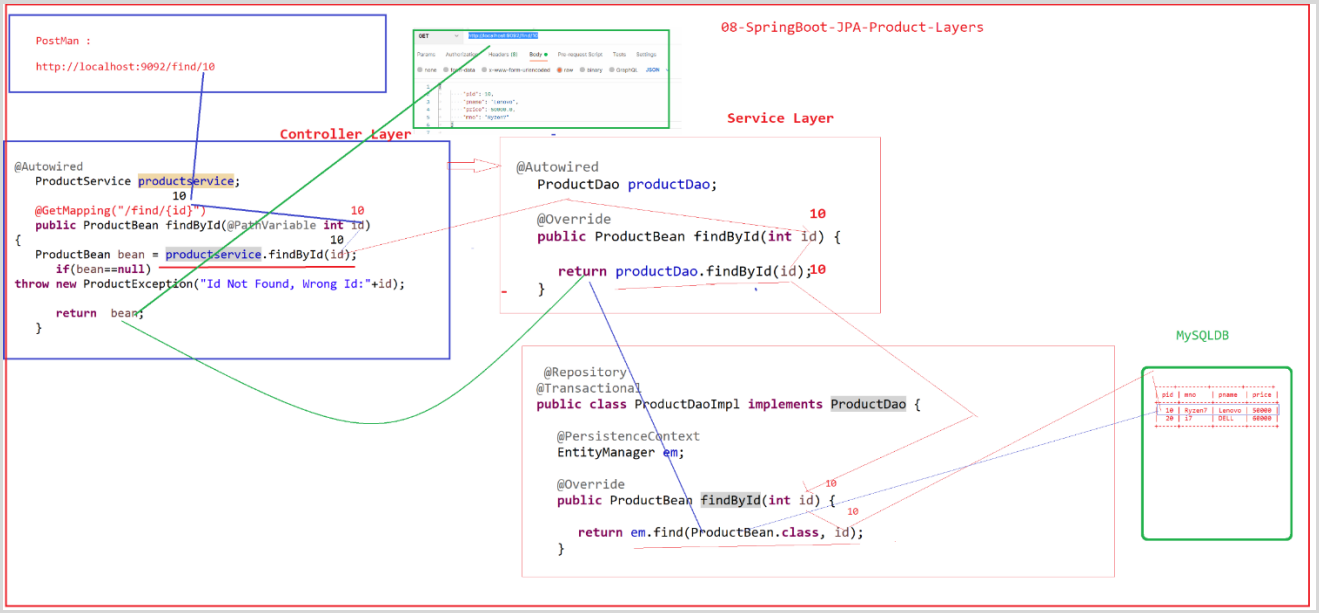
@Table(name="product_masters")

] 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'

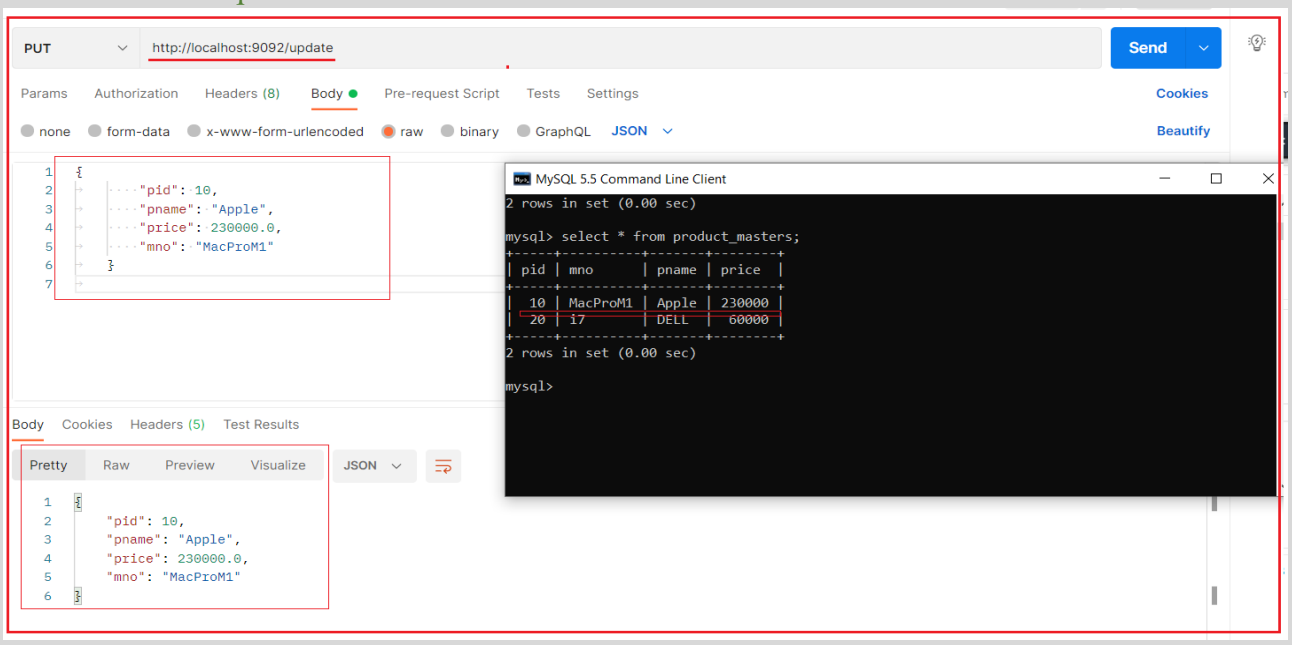
2021-12-30 11:40:23.371 INFO 30240 --- [nio-9092-exec-2] o.s.web.servlet.DispatcherServlet : Completed initialization in 10 ms

Ramkumar , WhatsApp : +91 9701153333

Application Flow :



Postman Example :



C U R D:

C: Create (create tables)

U : Update (insert or update data)

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

D : Delete (Delete record)

Ramkumar , WhatsApp : +91 9701153333

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

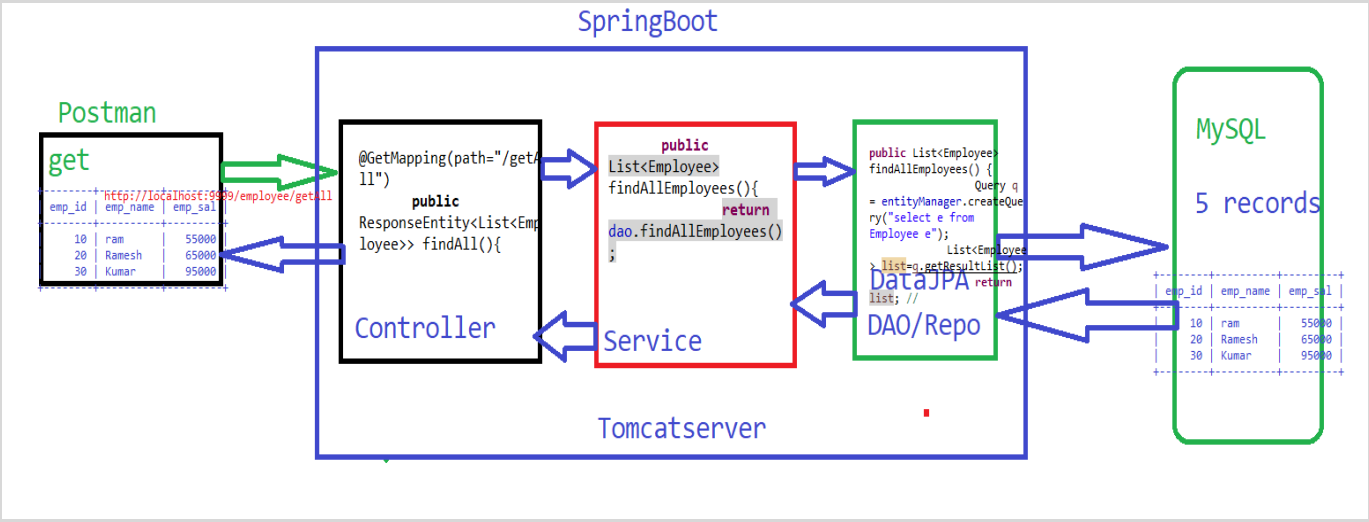
@EnableAutoConfiguration

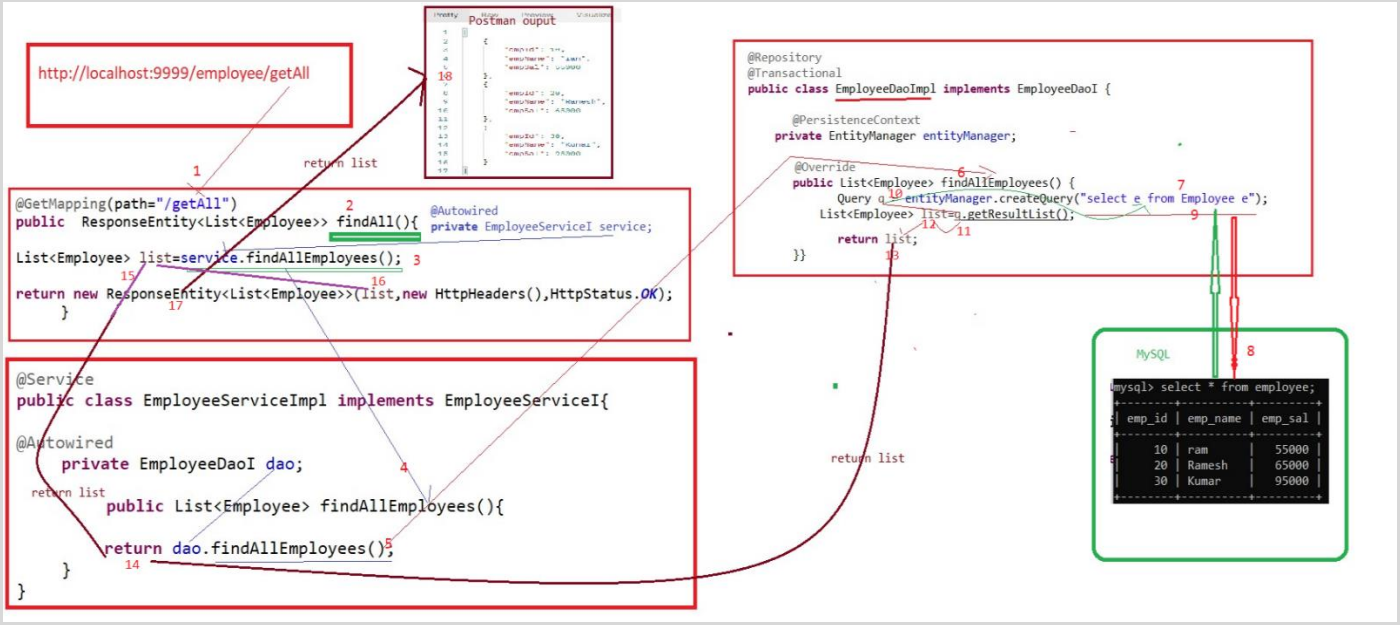
@ComponentScan

@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..

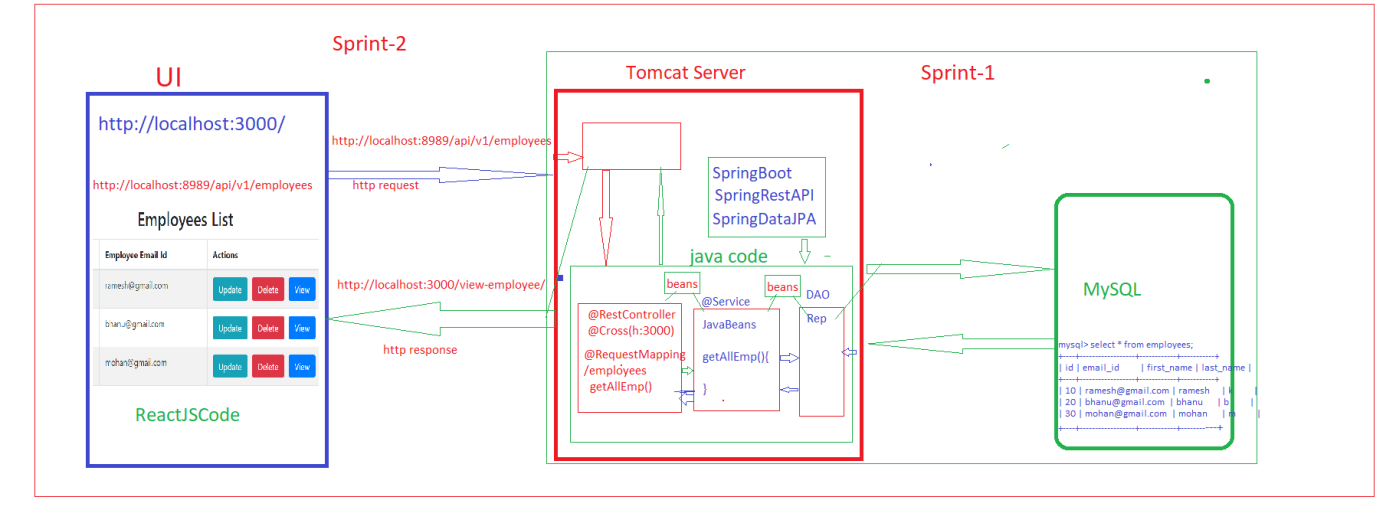




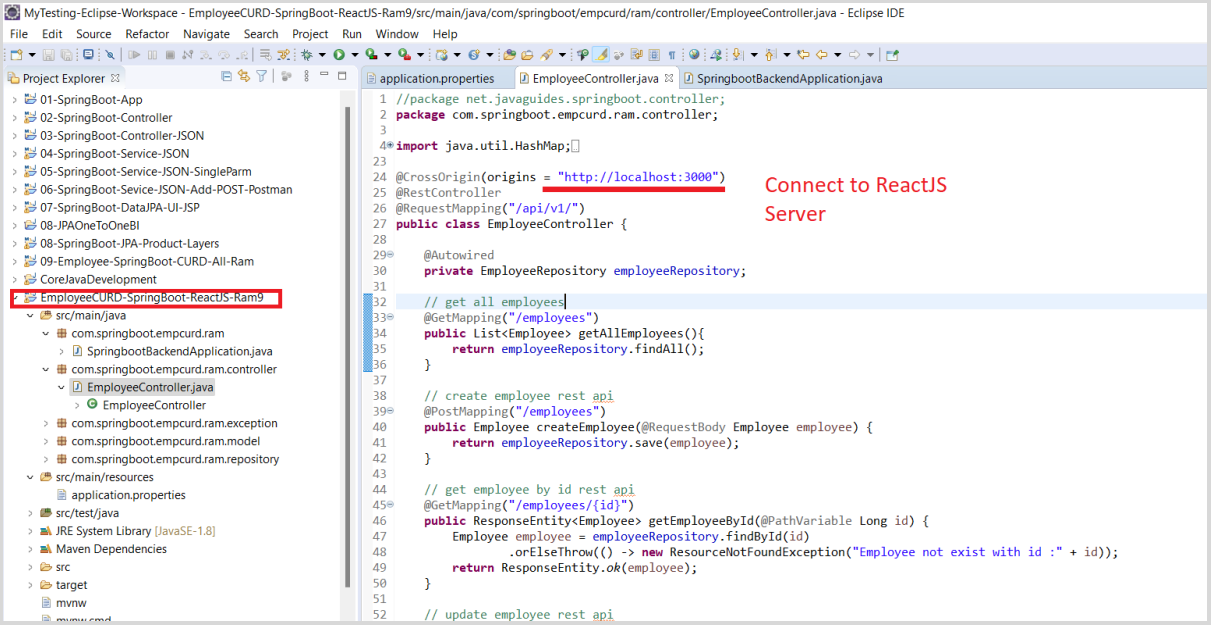
Ramkumar , WhatsApp : +91 9701153333

SpringBoot and ReactJS Integration

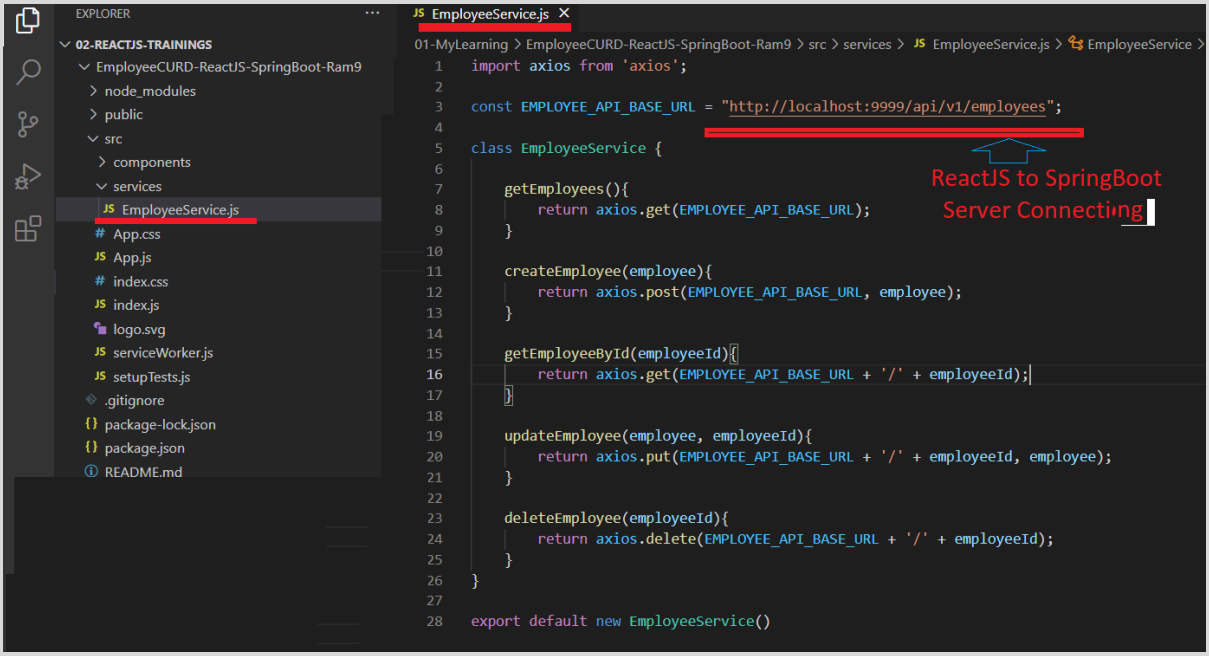
@Ramkumar



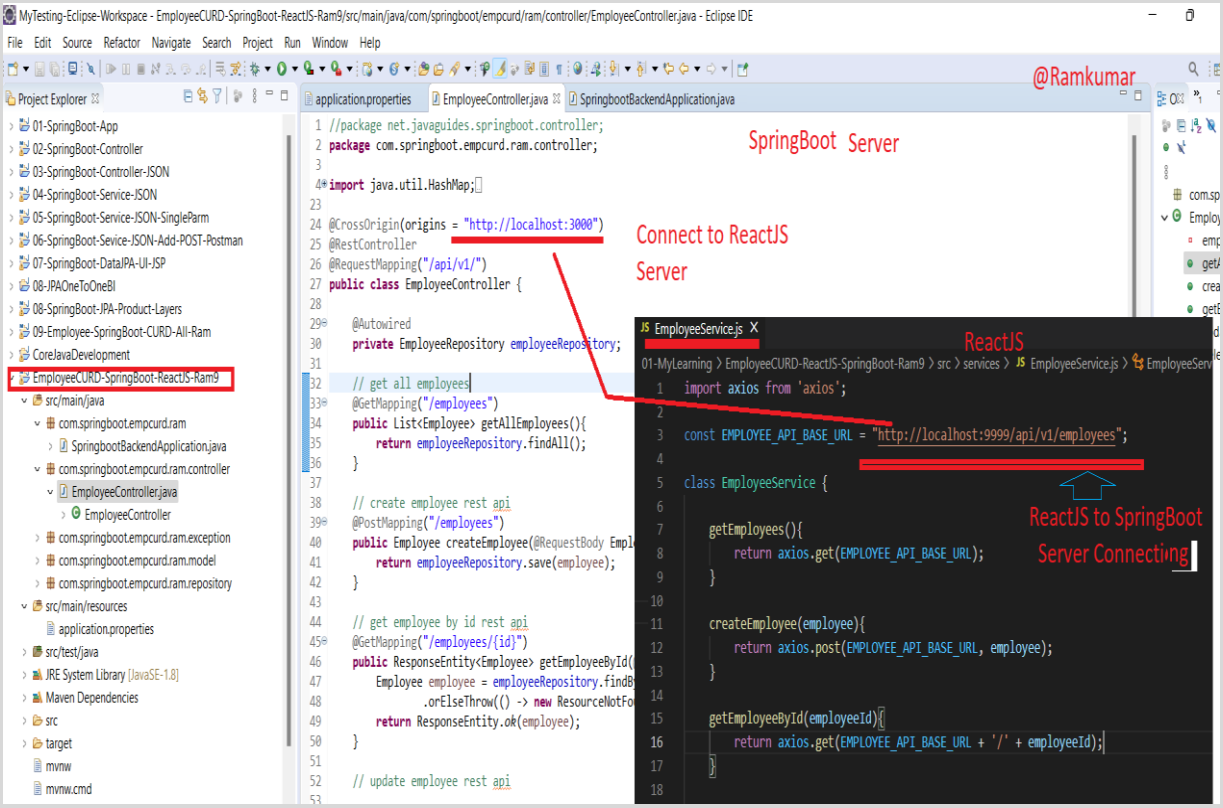
Axios is a Javascript library **used to make HTTP requests from node. js or XMLHttpRequests from the browser** and it supports the Promise API



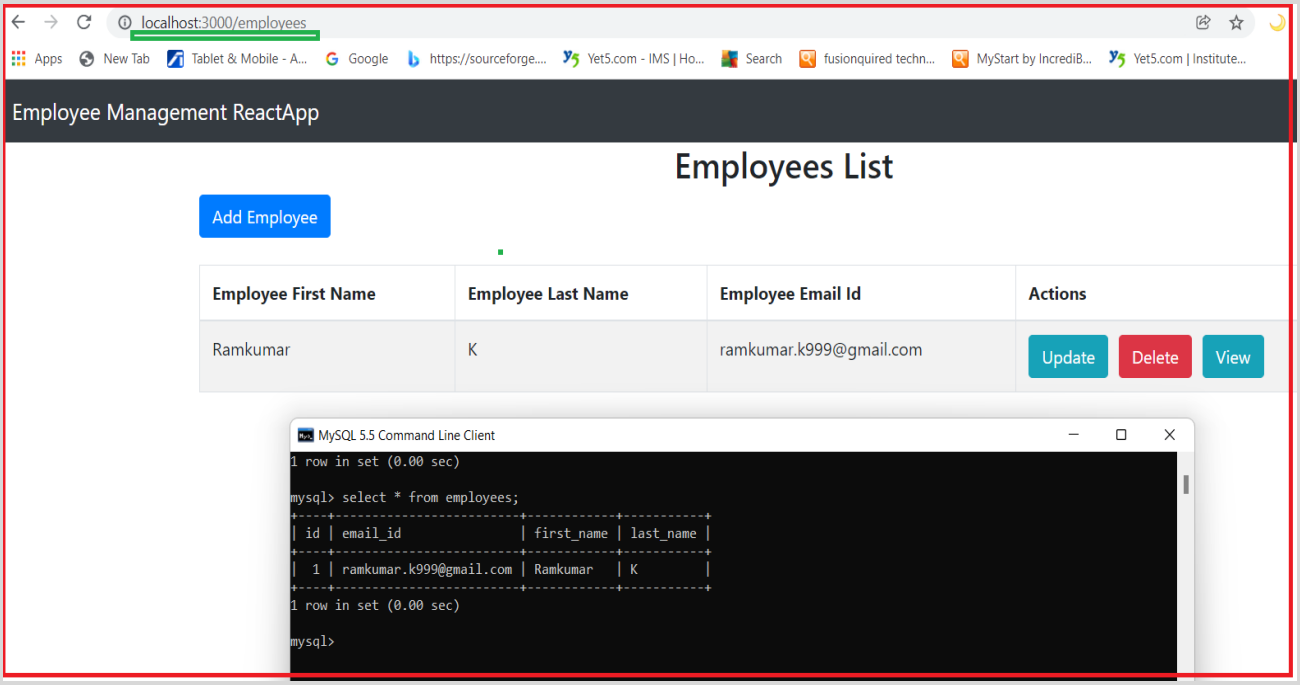
Ramkumar , WhatsApp : +91 9701153333



axios



Ramkumar , WhatsApp : +91 9701153333



What is Axios?

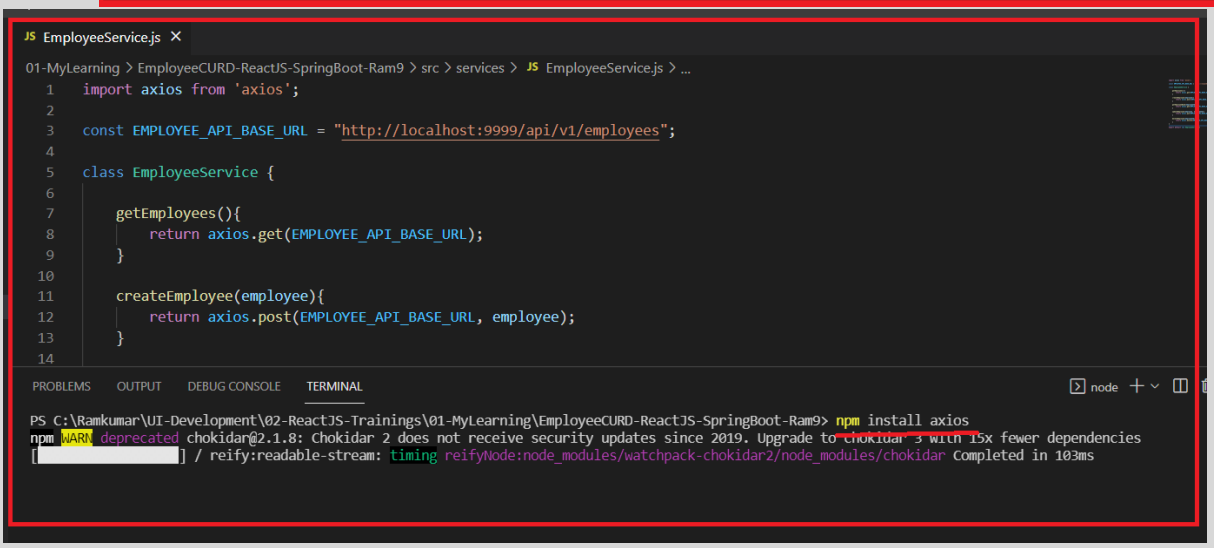
1. Axios is used to communicate with the backend and it also supports the Promise API that is native to JS ES6.
2. It is a library which is used to make requests to an API, return data from the API, and then do things with that data in our React application.
3. Axios is a Javascript library used to make 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 :

EmployeeService.js

01-MyLearning > EmployeeCURD-ReactJS-SpringBoot-Ram9 > src > services > 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()
```



Ramkumar , WhatsApp : +91 9701153333