Spring Cloud & Microservices

* Spring Boot
* Spring Microservices
* Deploying the microservices on cloud using Docker & Kubernates

Pre-requisites

* Java
* Spring Framework

OOPS concepts

Encapsulation: Where data is private and the only way you can access it is through public methods (setters & getters)

Spring Framework:

It is an application framework used to develop any kind of applications like

* Desktop/Console based
* Web applications
* Enterprise applications
* Mobile applications
* Cloud based applications

Framework: It allows you to develop complex applications in a simple way, framework takes care of all the repeating tasks in the application, it provides all the common features which every application needs, so that development time will be less and production will be more

Benefits we get from Spring Framework

* Dependency Injection Feature (Object creation)
* MVC based web applications (layered architecture)
* REST API’s
* Spring Boot
* Microservices
* Cloud deployment

Spring Framework examples has to be done with Maven project, because maven avoids dependency problems

i.e., Developers may forget to add some jars into their classpath when working on their project, this can be solved with Maven, as maven keeps all the dependencies of jars in its repository

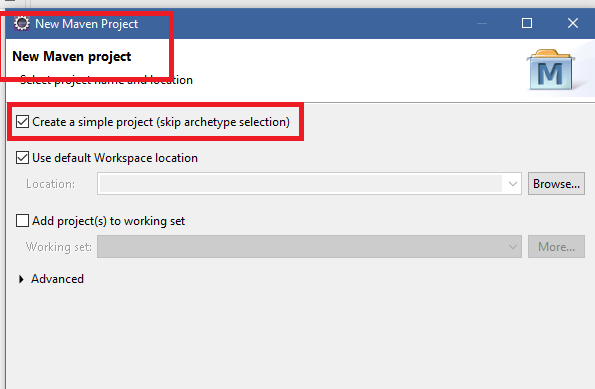
Things to understand in maven project

1. group-id: organization id
2. artifact-id: project-id or project-name
3. version: version number of the release
4. packaging: jar or war

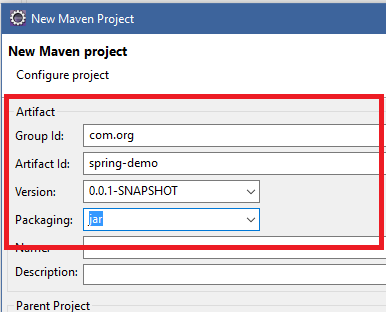
First Spring Framework program

1. Create maven project
2. Select create a simple project (skip archetype)
3. Enter project informations like group-id, artifact-id, packaging etc
4. Once project is created add dependencies i.e., jars for the project like spring-context, spring-mvc, mysql-connector and etc in pom.xml
5. You can also add java version in pom.xml if your maven project doesn’t have the right version of java matching to your system

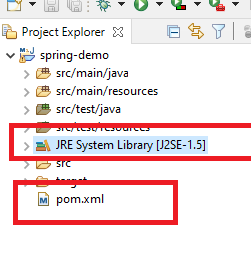
Step1:



Step2:

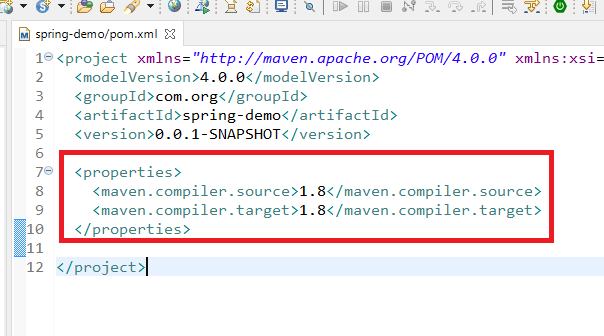


Step3 checking pom.xml & java version



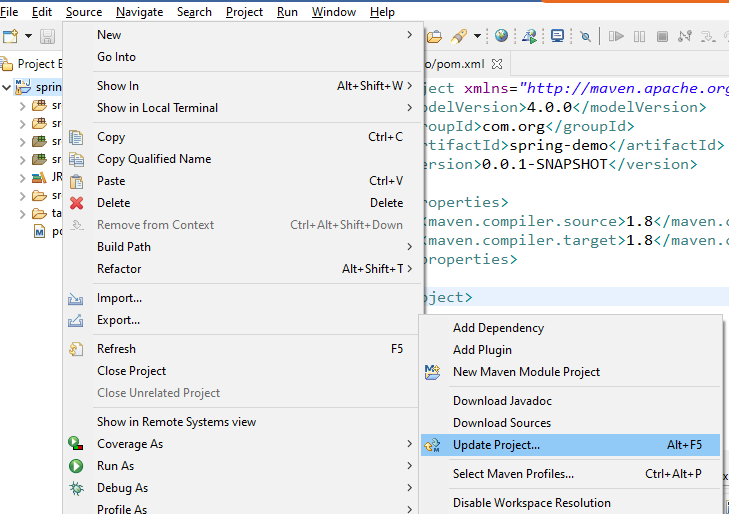
Since Maven project is having Java 1.5 we need to configure pom.xml to use 1.8

pom.xml

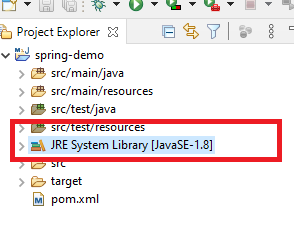


Update maven project

Project -> Maven -> Update Project



Now you can notice 1.8 in maven project



Adding Spring dependencies into pom.xml

List of dependencies we need are:

Spring Modules : dependency-names

1. Spring Core: spring-context
2. Spring MVC: spring-webmvc
3. Spring ORM: spring-orm
4. Spring AOP: spring-aop

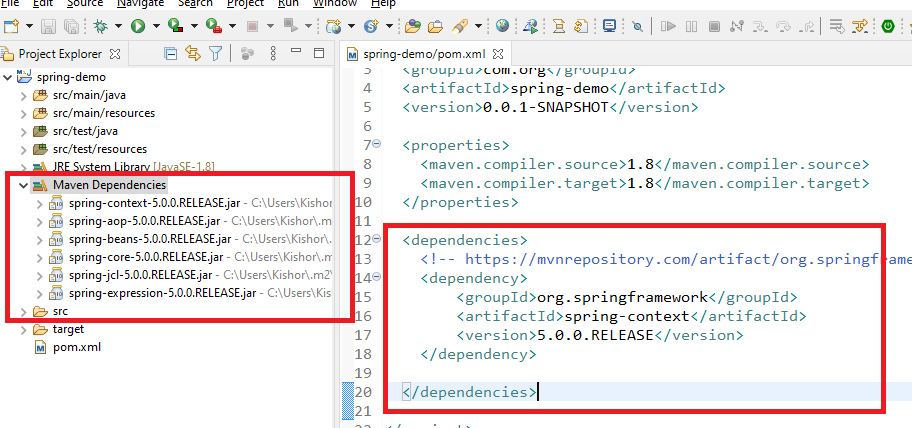
Spring Core: Here you can achieve Dependency Injection

Spring MVC: You can develop MVC based application & REST APIs

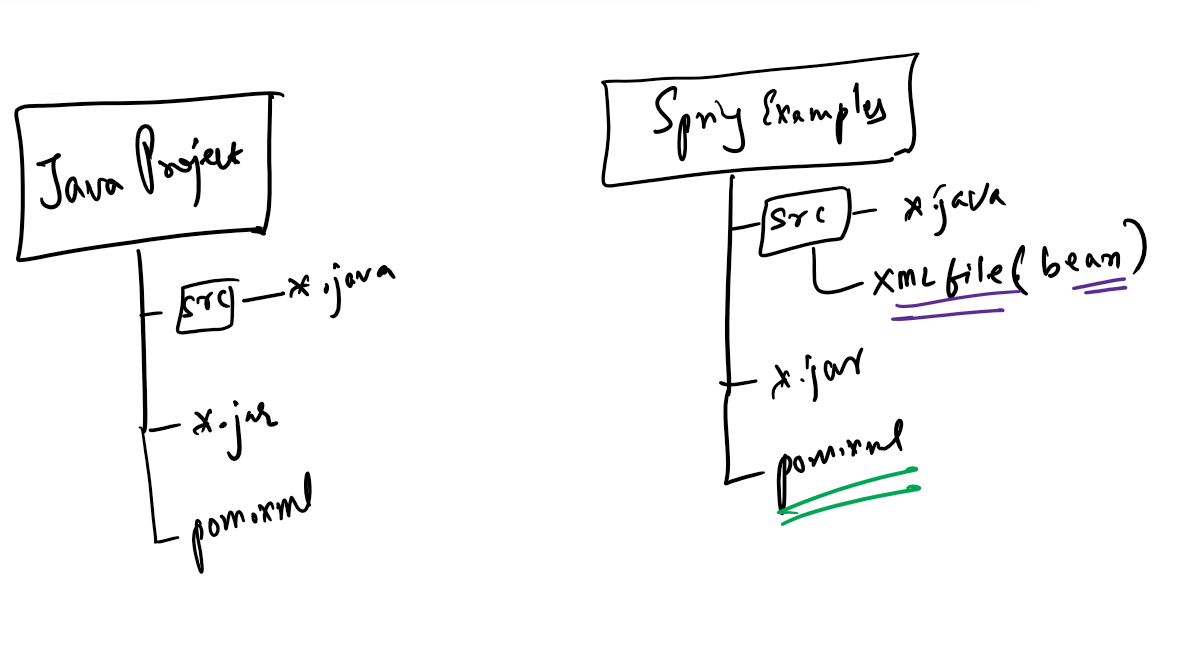
Spring ORM: You can interact with any database using Object Relational Mapping

Spring AOP: It is for Aspect Oriented Programming (Out of scope)

Update pom.xml to have dependency of spring-context



Project Structure for Spring Examples



Normal Java project doesn’t need xml file for bean configurations, however spring example use one xml file which will have bean configurations.

Bean Configuration:

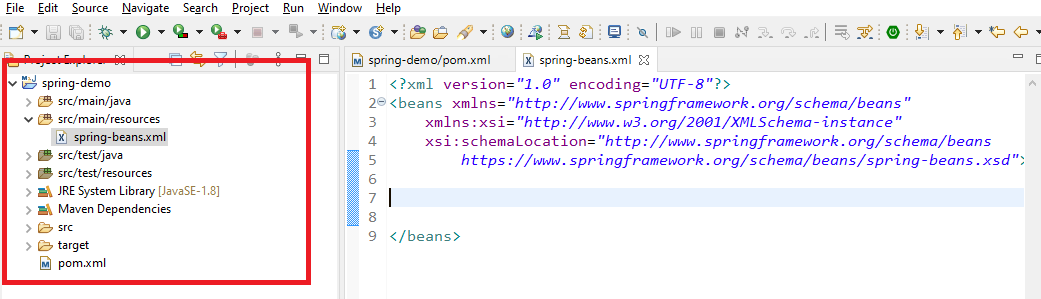
This configures the classes that needs to instantiated by spring framework to manage the objects that depends on other objects, this provides dependency injection feature to us

Sample code of bean configuration file

<beans>  
 <bean id = “x” class = “className”></bean>  
 <bean id = “y” class = “className”></bean>  
</beans>

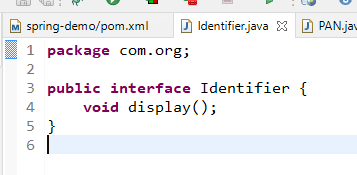
Note: The name of the xml file can be anything like spring-beans.xml, beans.xml, spring-config.xml, abc.xml and so on.

A simple bean configuration file with the name spring-beans.xml looks like

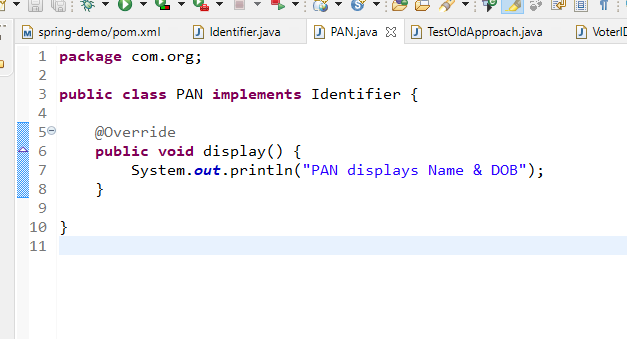


Note: This xml file has to be downloaded either from internet or from spring website

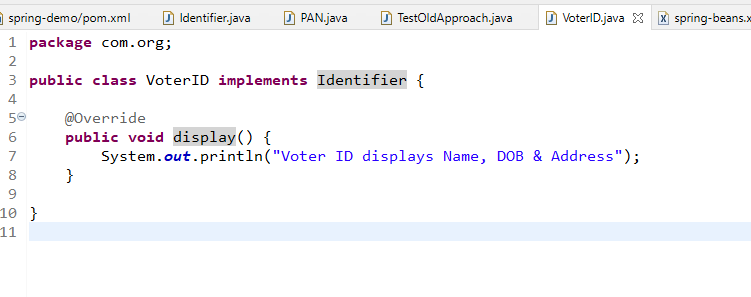
Identifier.java



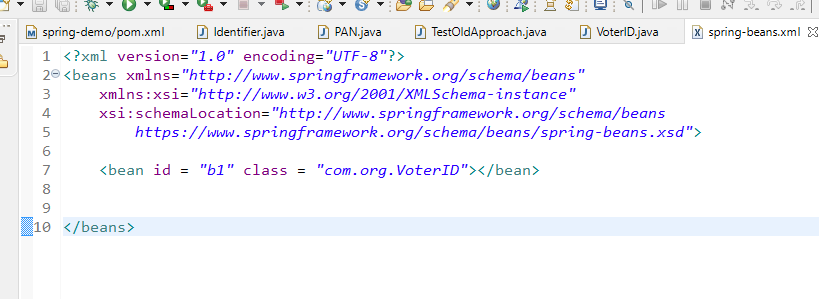
PAN.java



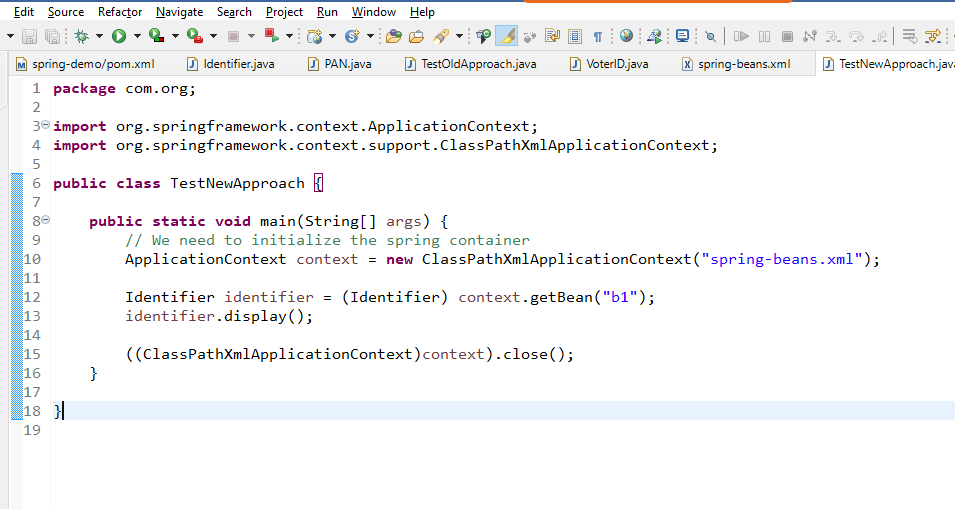
VoterID.java



spring-beans.xml

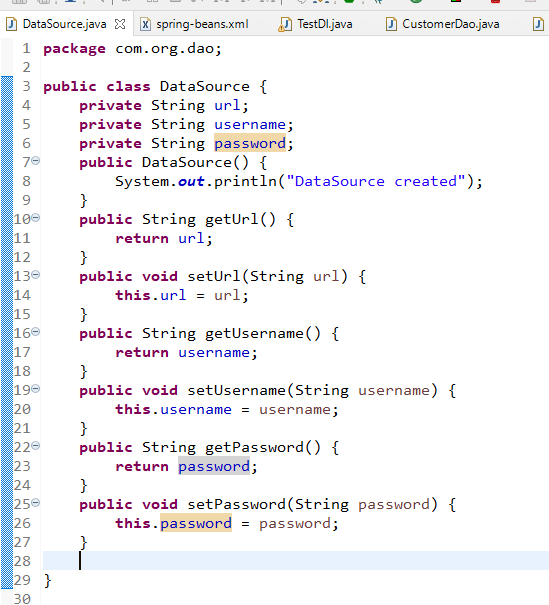


TestNewApproach.java

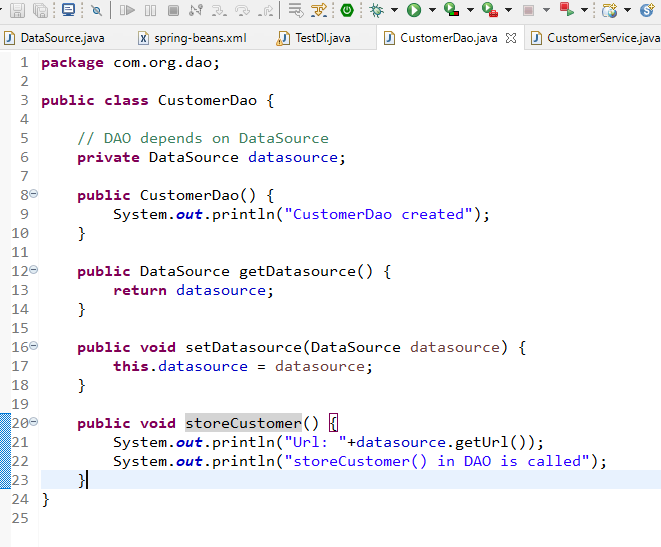


Dependency injection of Datasource->DAO->Service with XML

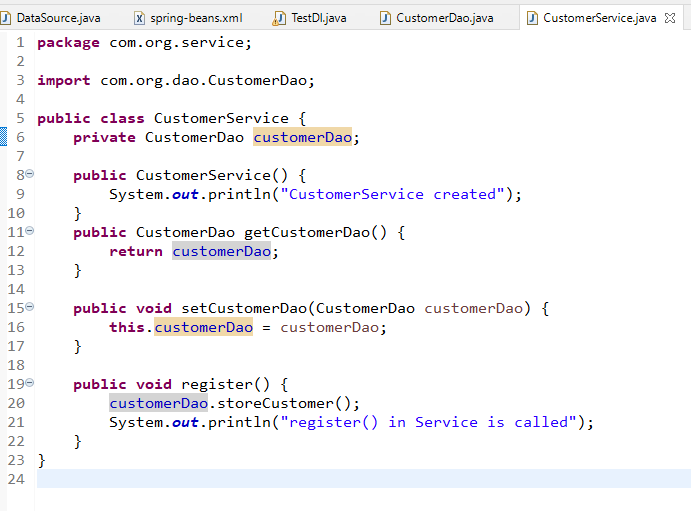
Datasource.java



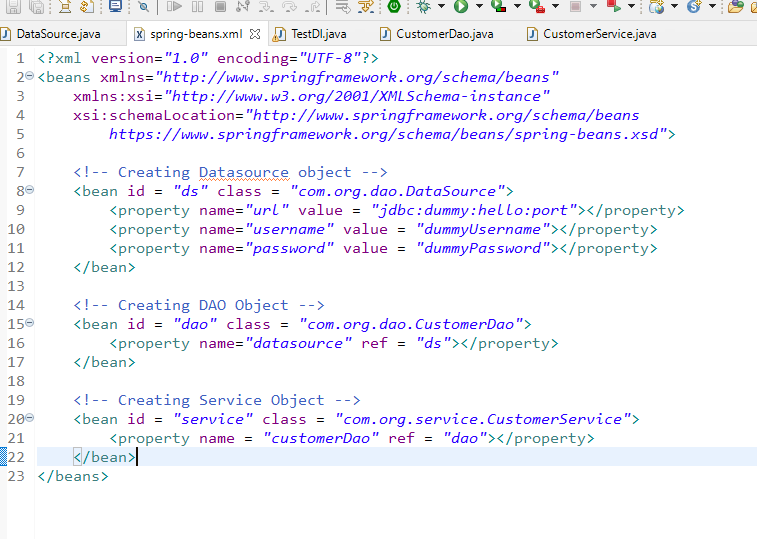
CustomerDao.java



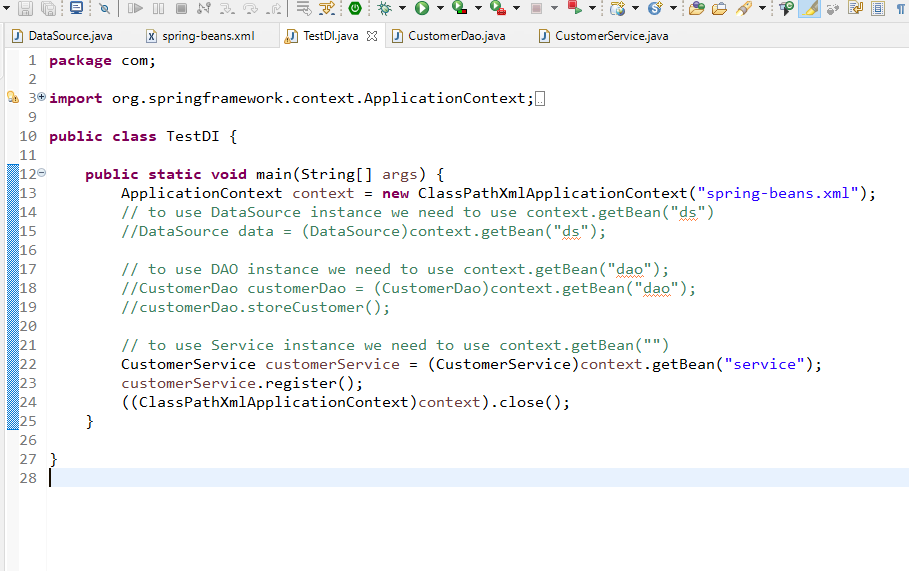
CustomerService.java



spring-beans.xml



TestDI.java



Spring Framework uses annotation instead of <bean> configuration, so that spring container can maintain the object

Some of the important annotations

1. @Component: similar to <bean>
2. @Controller: to mark the controller class
3. @RestController: to mark the RESTApi class
4. @Service: to mark the service layer classes
5. @Repository: to mark the DAO layer classes

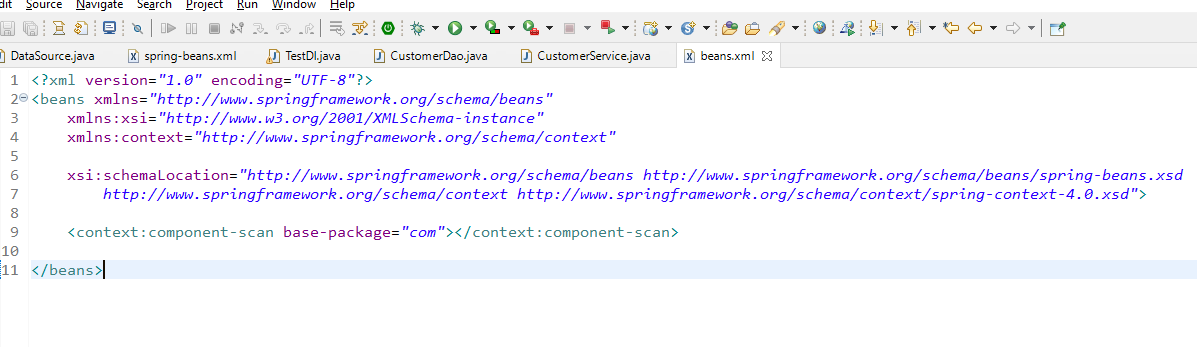
All these annotations should be written on top of the classes, the spring container will create object for these classes

To configure the beans through annotations you need to use a tag called

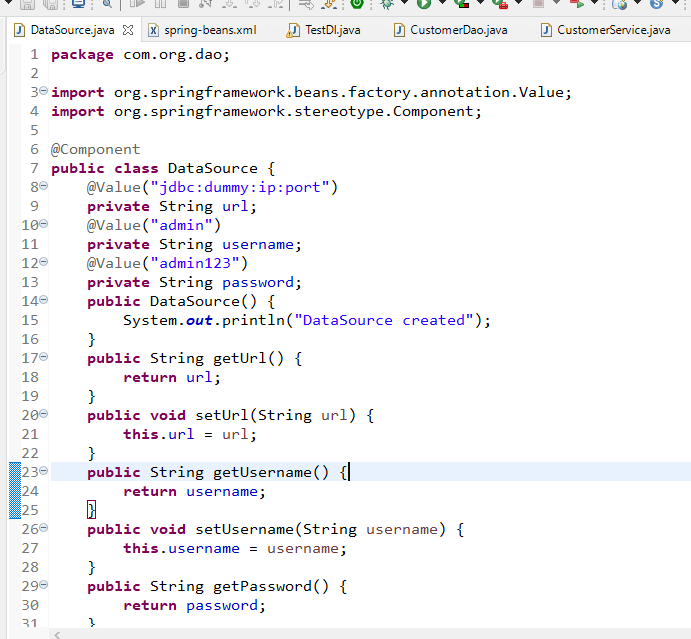
<context:component-scan package = “com”>

@Component  
public class Test { }   
@Controller  
public class Demo { }   
@Service  
public class EmpService { }  
@Repository  
public class EmpDao {}  
@RestController  
public class EmpController {}

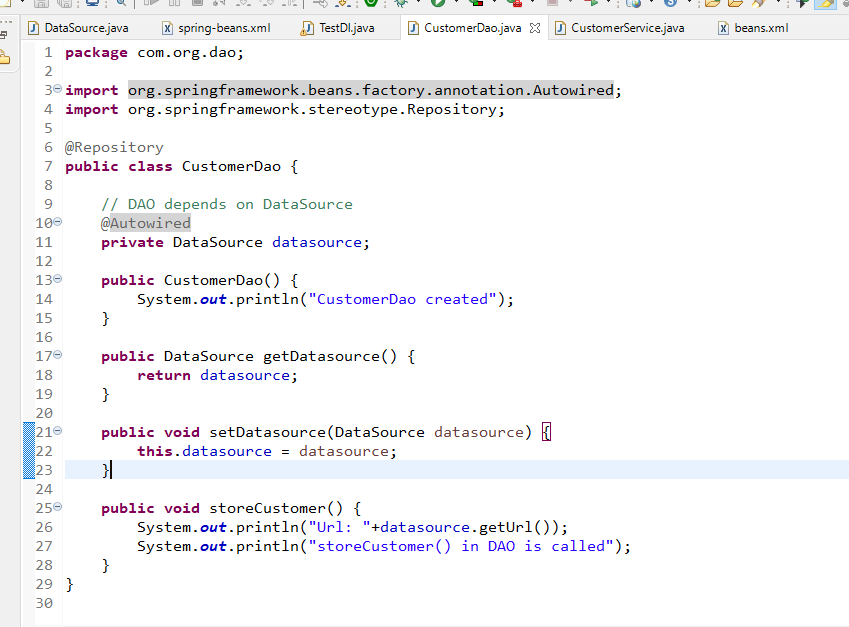
beans.xml



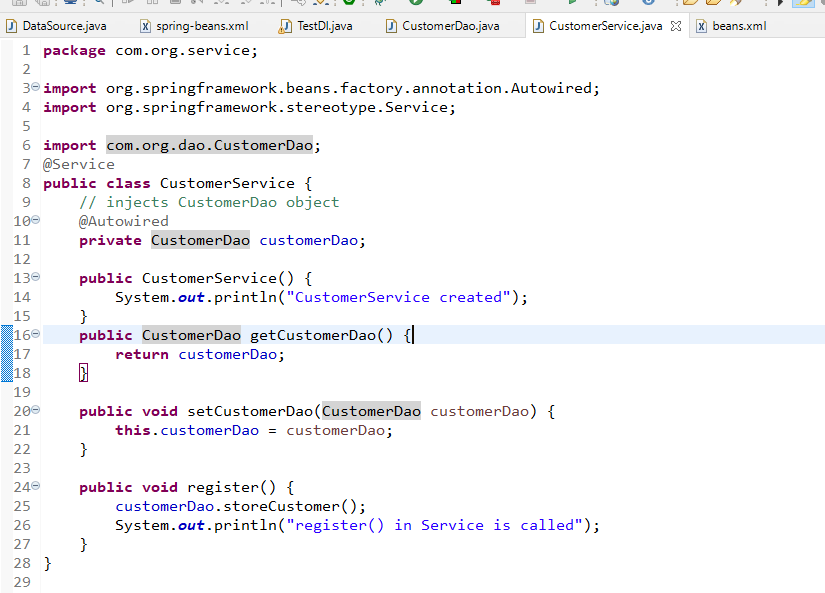
DataSource.java



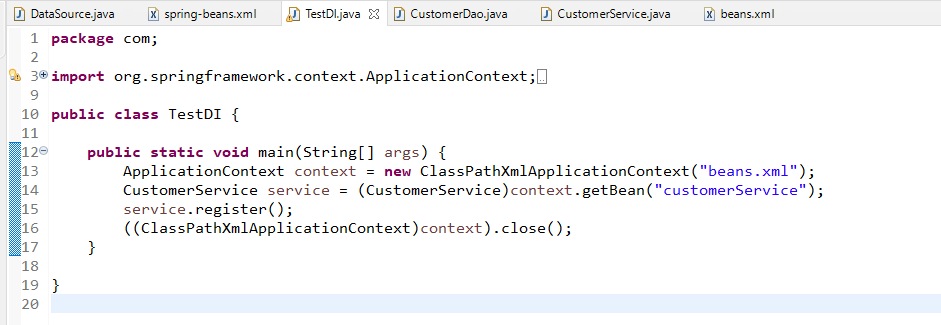
CustomerDao.java



CustomerService.java



TestDI.java



Spring MVC

* Web applications
* RESTful webservices based applications
* Configure servers

ReST stands for Representational State Transfer, it is a webservice.

How to create REST based applications

1. You need to have methods with URI i.e., end points
2. You need to use HTTP methods like GET, POST, PUT, DELETE as the data is exchanged using HTTP protocol, both consumer & producer needs to use HTTP methods
3. You need to provide the structure of the data like JSON, XML, CSV, HTML, TEXT and so on, but more preferred structure by consumer & producer is JSON
4. You can use XML based configuration as well as annotation based configuration to create the webservices
5. With Spring Boot you can quickly develop spring based applications

Spring Boot

It is used to develop spring based application without any extra configurations, it will give you ready to run spring applications

Benefits of spring boot

* Most of the things are automated
* No need of XML files
* Spring Module Version compatibility issues are avoided
* Provides auto-configuration based on the library you add
* Spring boot provides some starter libraries to achieve auto-configuration, they will take care of performing auto-configuration based on the starter library

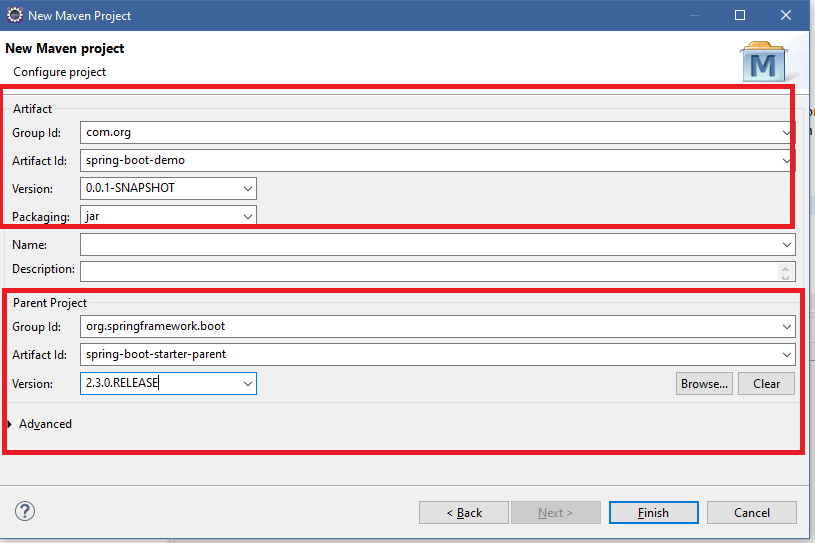
Starter libraries

1. spring-boot-starter-web
   1. It automatically provides component scanning, embedded server, dependency management, RESTful based application API’s, downloads all the necessary compatible libraries for web like spring-context, spring-webmvc, spring-aop, log4j and etc.
2. spring-boot-starter-jpa
   1. It automatically provides dependency management to interact with database
   2. It reads application.properties to establish database connection

How to create spring boot application

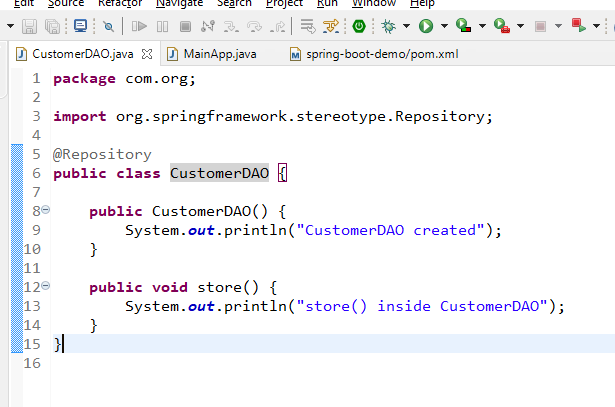
1. You need to create maven project
2. You need to use Spring Initializr which gives you maven project

Creating spring boot with maven project

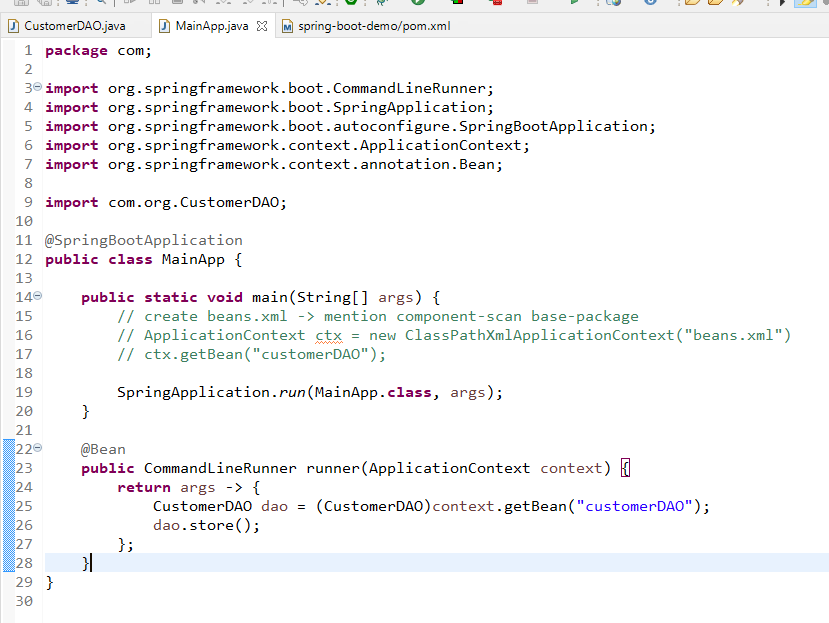


In Spring Boot Application Context will automatically created, you don’t have to create it, you are developing web application you can use components directly, however if it’s a console based application then you need an interface called CommandLine Runner instance to use ApplicationContext

CustomerDAO.java



MainApp.java



How to create Restful API’s

You need a class with @RestController

You need to provide the url using @RequestMapping

You can mention the HTTP methods using some of the annotation like

@GetMapping, @PutMapping, @PostMapping, @DeleteMapping

How to call webservices

You can call webservices using URI and HTTP methods with the following tools, softwares and applications

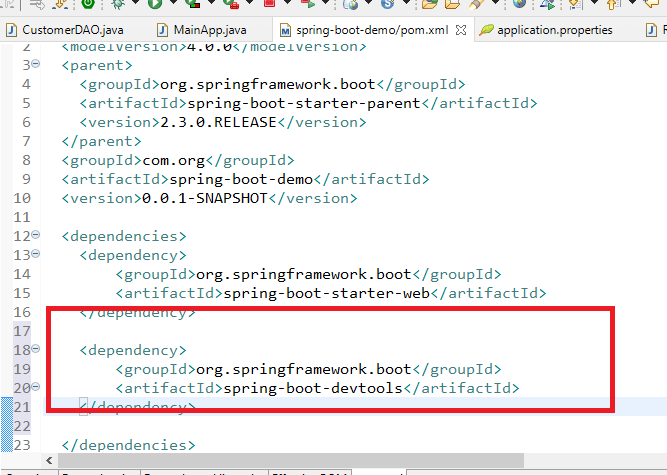
1. Browser: You can only call http GET
2. Softwares & Tools: you can call all the HTTP methods like GET, POST, PUT, DELETE

Since you can’t write program in all the languages to test your webservice you can use a testing tool like Postman, which provides a UI to enter the URL and select HTTP methods.

Tools will send or receive the intermediate data between the producer & consumer

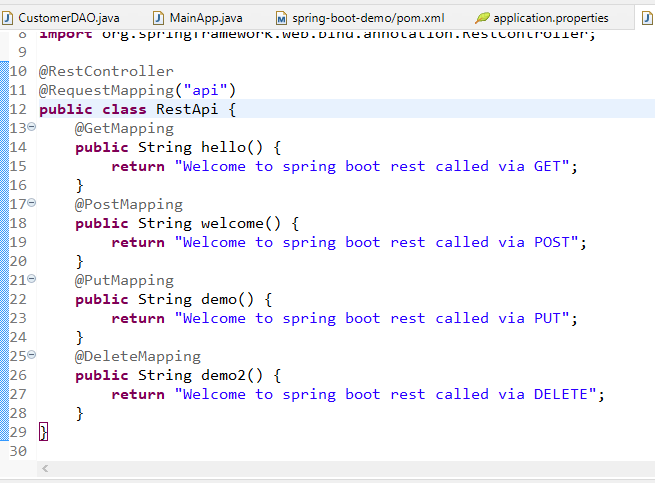
Spring DevTools  
It makes server to detect the changes while you modify the code

pom.xml



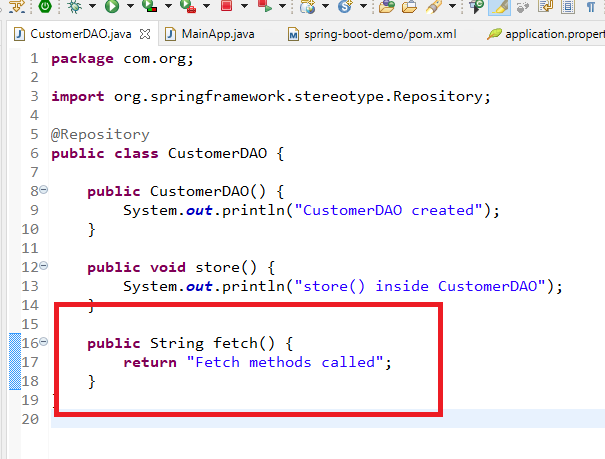
Create webservices with POST, PUT, DELETE, GET methods

RestApi.java



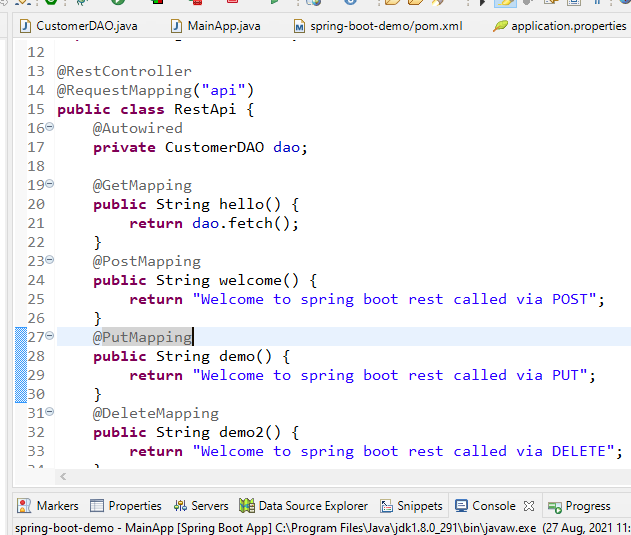
Note: Now onwards you don’t need CommandLineRunner

Modify CustomerDAO.java

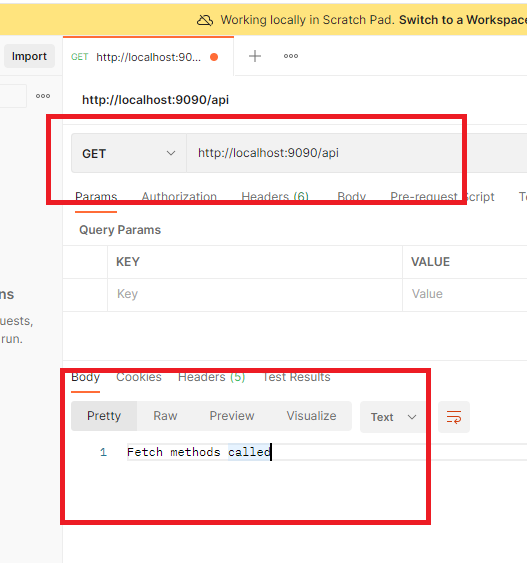


We can autowire CustomerDAO in Service layer but as of now we will autowire in Controller layer

RestApi.java



Output:



The above example is to just understand REST API’s, but we need to create DAO layer, Service Layer, Controller layer(RESTAPI layer) and interact with the database to see the real time examples.

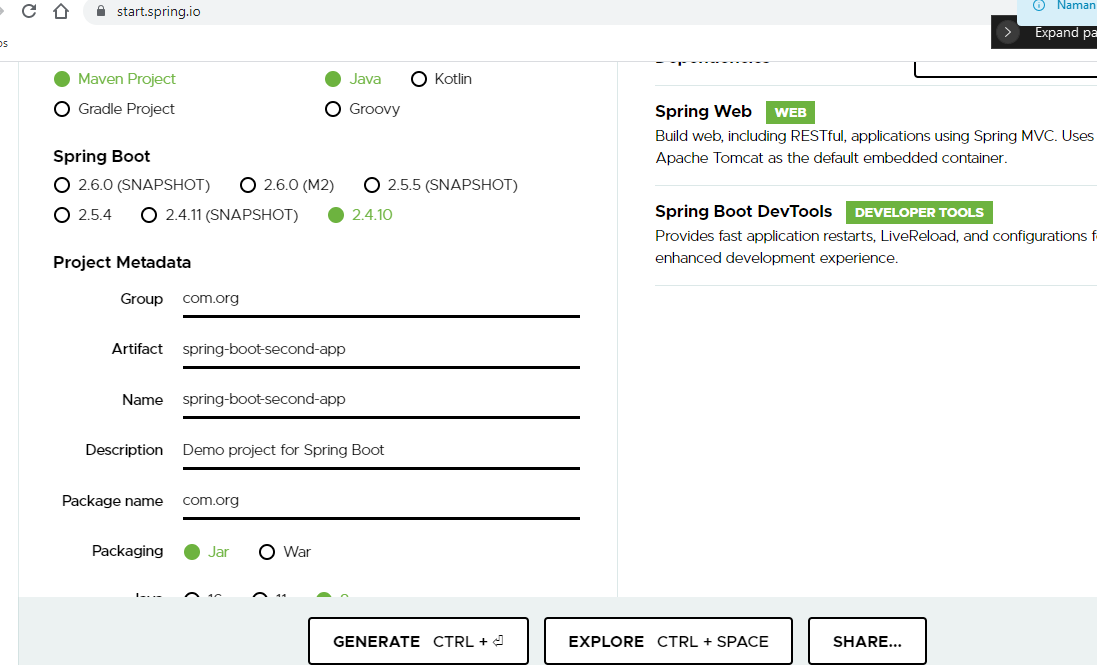
Firstly we create Service layer and DAO layer without Database then we can use Database

How to produce & consume JSON from REST

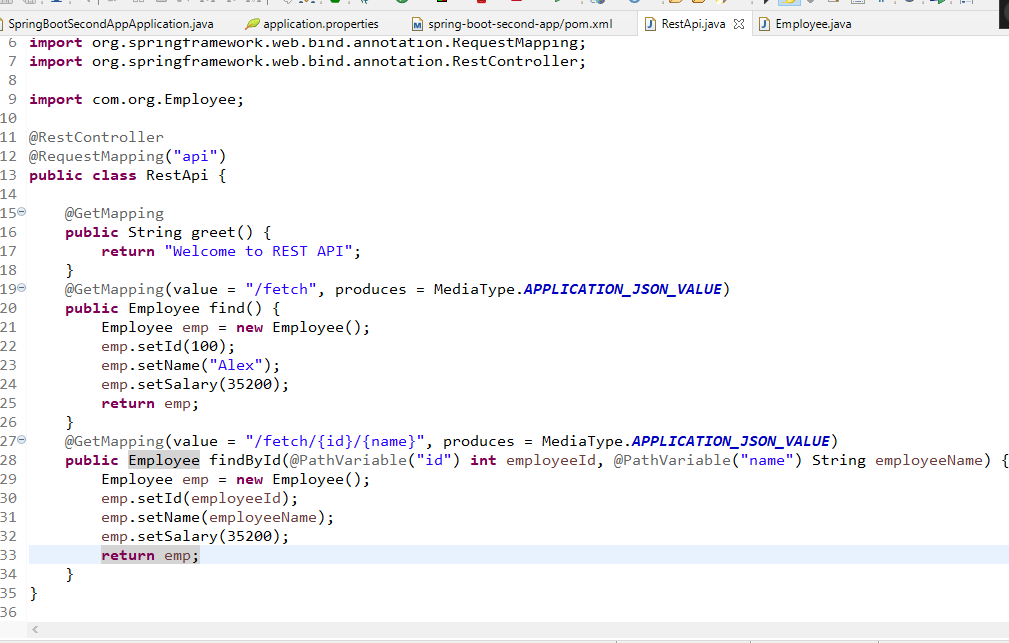
If you have employee data like id=100, name=Alex, salary=25000 then in java you create employee object but in json it will be like

{“id”:100, “name”:”Alex”, “salary”:25000}

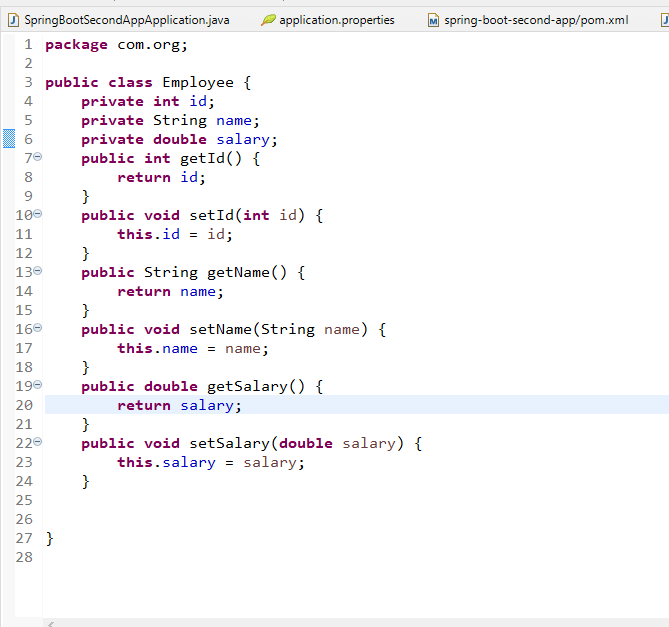
Creating the project through spring initializr



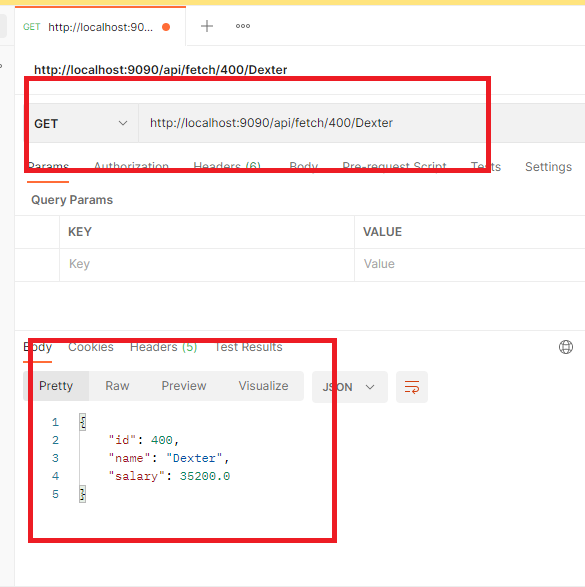
RestApi.java



Employee.java



Output:

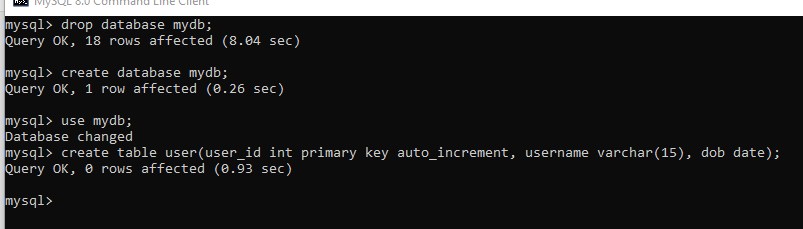


Interacting with the database using Spring Boot & Spring Data Jpa

Spring Data JPA directly maps the object to the table using ORM Framework, you need to just use the interface provided by SpringDataJpa, you don’t have to implement that, the only thing you need to do is mention the Entity class in the interface.

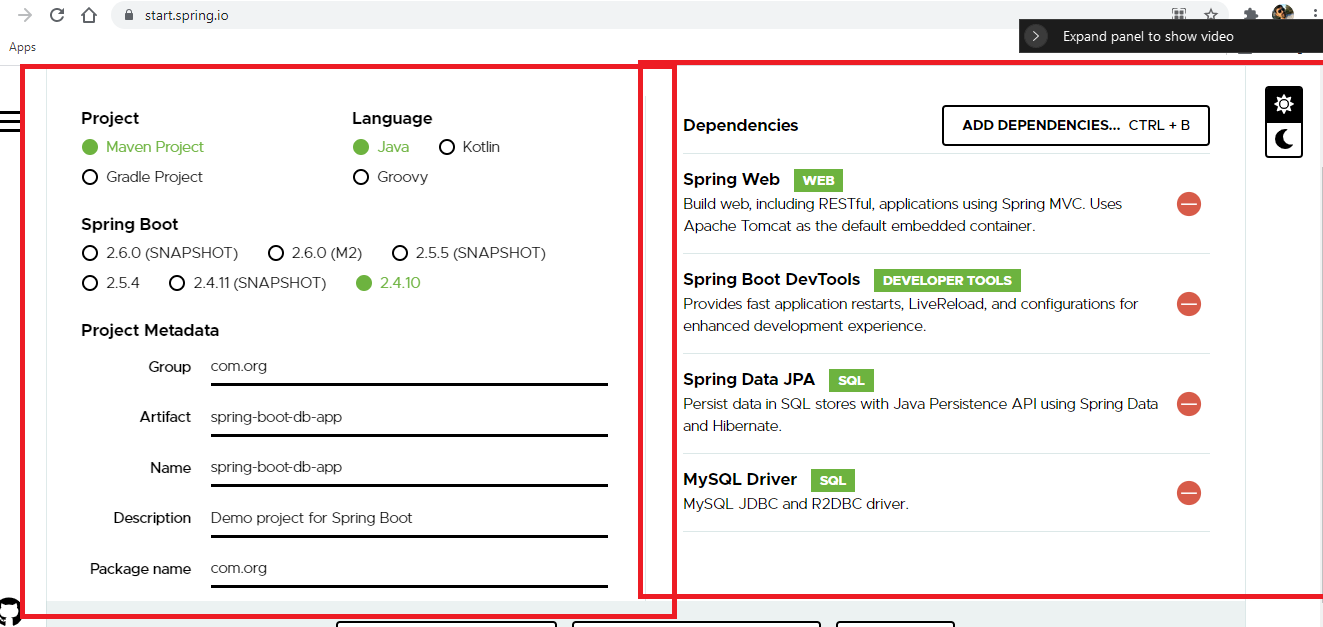
Entity class: These are normal classes that will have table & column information, i.e, mapping of classes & their variables on tables & columns.

Create a User table

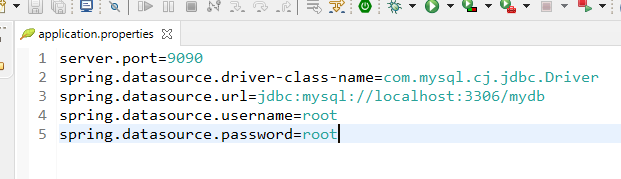


List of dependencies required inorder to interact with the database through Spring Data JPA

1. Spring Boot Starter Web
2. Spring Boot Data JPA
3. Spring Boot DevTools
4. MySQL Driver

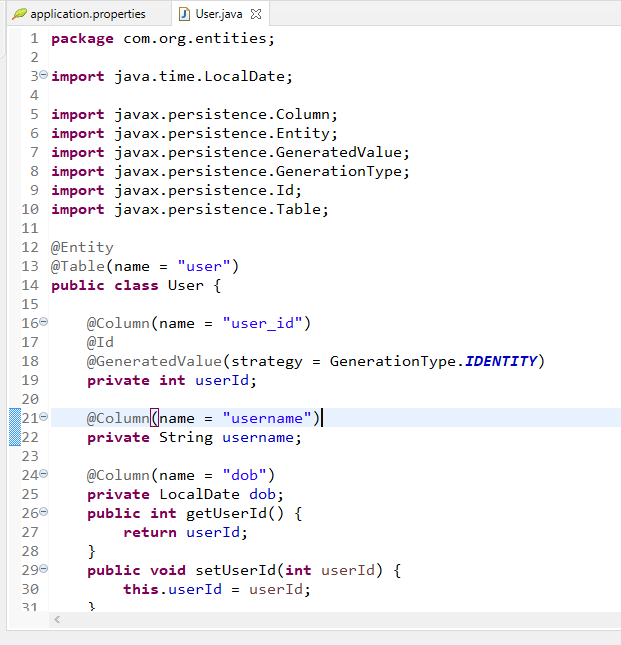


Spring Boot automatically create database connection by looking at the datasource information in the application.properties



Creating the entity class to map to User table

User.java



In Spring Data Jpa you need to use some interfaces like CrudRepository<T, ID>, JpaRepository<T, ID>, which takes Entity class & Primary key class names, these needs to be extends so that you can call the methods present in it.

Here T is the entity class name & ID is the wrapper class name of the primary key, ex: T is User and ID is Integer

CrudRepository<T, ID> provides methods like

save(T)  
upate(T)  
delete(T)  
findById(ID)

findAll()

JpaRepository<T, ID> extends CrudRepository, PagingSortingRepository, provides methods like

sort()

and all the methods of CrudRepository.

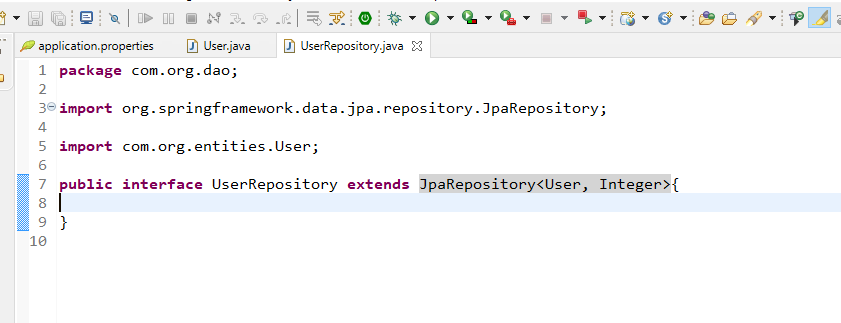
Developers need not implement it, they just need to extend the Interface so that spring boot understands on which entity it has to perform operation.

interface UserRepo extends JpaRepository<User, Integer>{}

Spring Boot creates the instance for UserRepo and you can autowire it in service/controller layer

Note: You don’t have to create @Repository class at all

UserRepository.java



Note: Spring Boot provides implementation to UserRepository at runtime and also creates the instance of that implementation, it means developers need not to write any logics for database interaction, they just need to call the methods of JpaRepository through the UserRepository reference.

List of methods you can use from JpaRepository

@Autowired

UserRepository userDao;

userDao.save(userObject); // stores user Object

userDao.findAll(); // returns all the user object in List<User>

userDao.findById(id); // returns a user object matching to the id else null or null pointerexception

Note: In real-time we need to @Autowire DAO to the Service & Service to the Controller, but here we are directly using @Autowire of DAO in controller

HTTP Methods operations

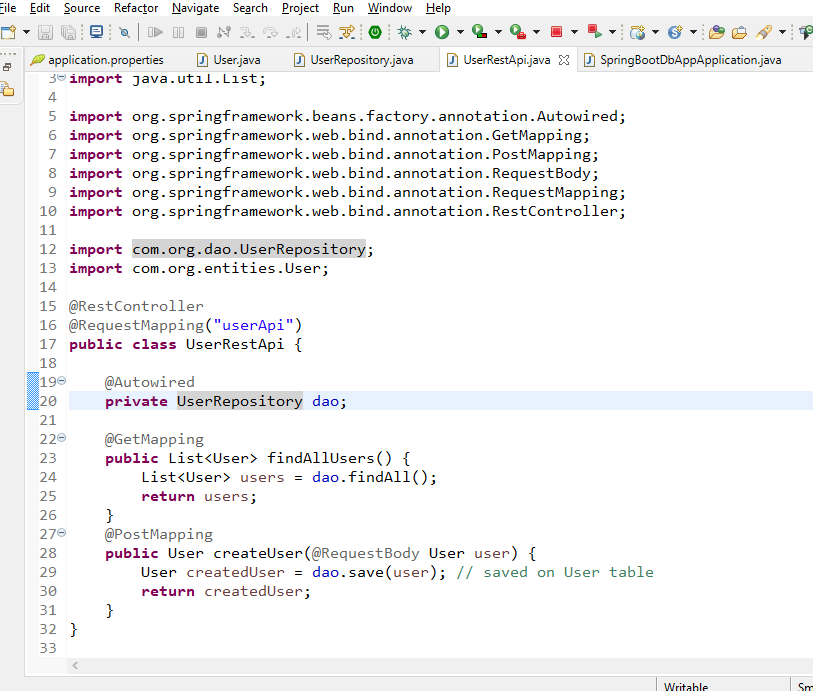
GET: Fetch

POST: Create

PUT: Update

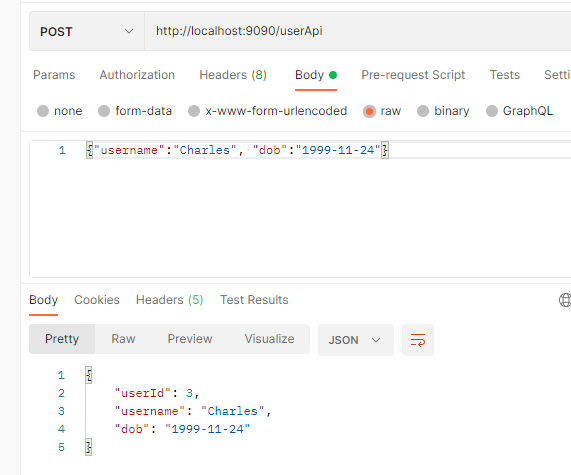
DELETE: Delete

UserRestApi.java

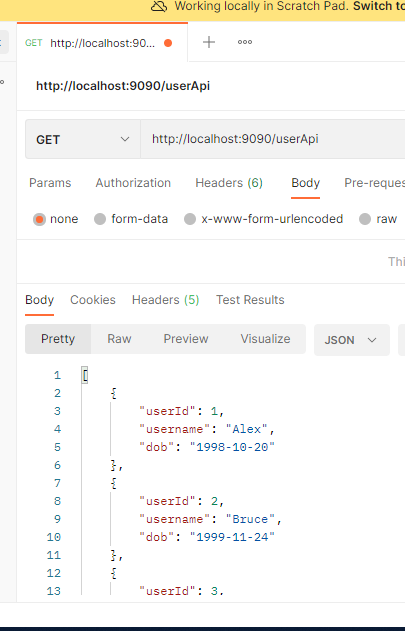


Output:

POST operation



GET operations



Finding a particular entity

