**Spring Boot**

1. Spring and Spring Boot is a framework.
2. Framework is a predefine functionalities provided to the developer using a jar files. In the framework the commonly used functionalities will be provided. Frameworks are use to develop the application faster and with minimum error. Developers can focus on business logic and all the common function will be taken case by framework.
3. Spring and Spring Boot is also known as framework of framework.
4. Can integrate other framework easily like JPA, Hibernate, Struts, JSF etc.
5. In the Full stack development Spring Boot is majorly used to develop a REST APIs.
6. Spring Boot is used to develop micro-service.
7. Using Spring Boot, you can develop the application with minimum configuration.
8. Spring Boot provided auto configuration feature. For example, if you are using DB connection into code then spring provides you H2 embedded database internally and also set all the configuration for H2 database.
9. Spring boot is a highly customized framework.
10. Spring Boot Provides embedded tomcat server. So, you can run your application without any external server.
11. Spring Boot has multiple stater project which is also known as different modules of the spring.
    1. **Spring Bean/IOC**
    2. **Spring Jdbc**
    3. **Spring ORM (JPA/Hibernate)**
    4. **Spring REST APIs**
    5. Spring MVC
    6. Spring Batch
    7. **Spring Cloud**
    8. Spring Security
    9. Spring Cache.
    10. Spring JMS etc.

**How to use spring module**

1. Add the Dependencies (jar files) of the Spring Module.
2. Perform the module configuration.
3. Used the module using the predefine APIs (classes and interfaces) provided by Spring.

**Imp Links**

Spring Properties:

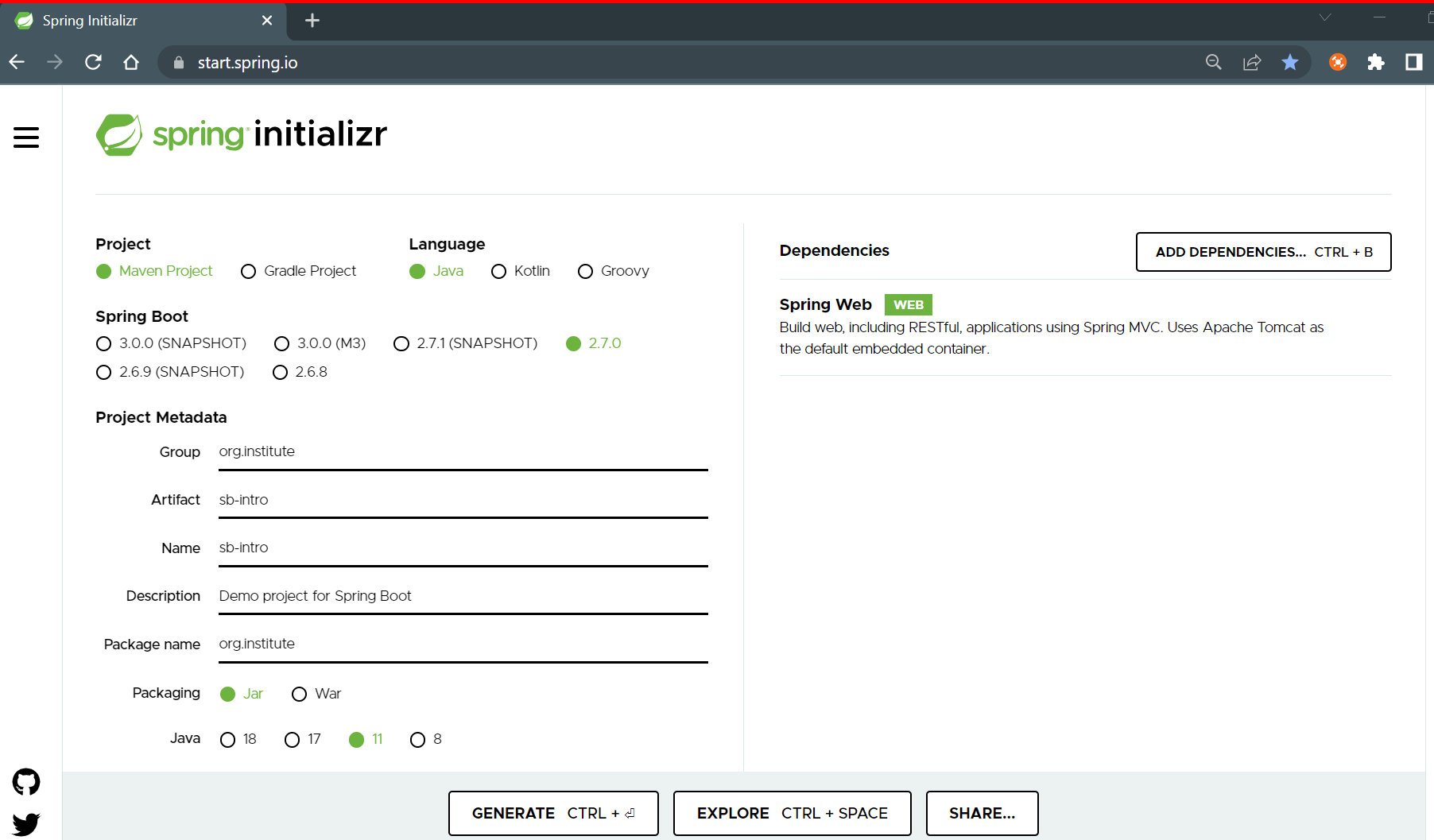
<https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html>

Spring Boot Official Documentation:

<https://docs.spring.io/spring-boot/docs/current/reference/html/index.html>

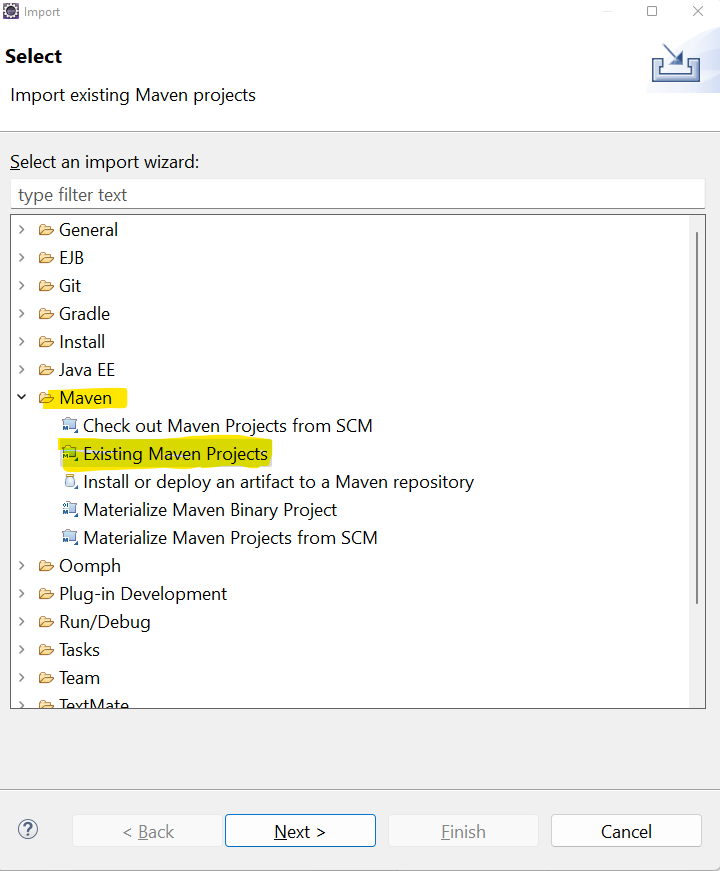
**Create first Spring Boot Application**

1. Options to Create Spring Application
   1. STS tool (Spring Tool Suit) IDE
      1. <https://spring.io/tools>
   2. Spring Boot CLI (Command Line Interface)
      1. <https://docs.spring.io/spring-boot/docs/current/reference/html/cli.html>
   3. Spring Initilizer
      1. This is a web application provided by spring community to create spring application without any local setup.
      2. <https://start.spring.io/>

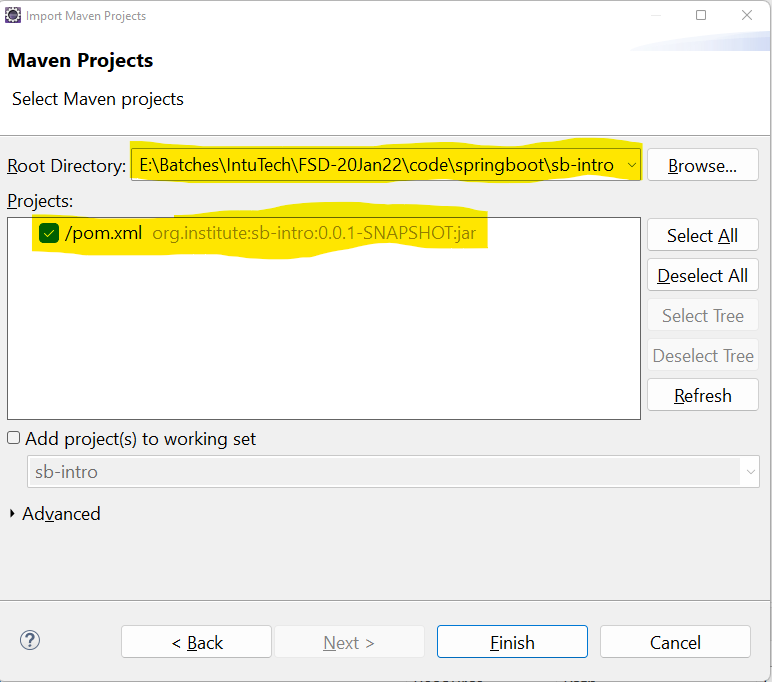


**Open Project in the Eclipse IDE**

1. After generating the project from Spring Initializer web application. Go to Downloads.
2. Extract the ZIP file.
3. Copy the extracted folder and paste it into Eclipse workspace location.
4. Go to “File” Menu -> click on “Import…” option
5. Search and expand “Maven” option
6. Select “Existing Maven Projects” option

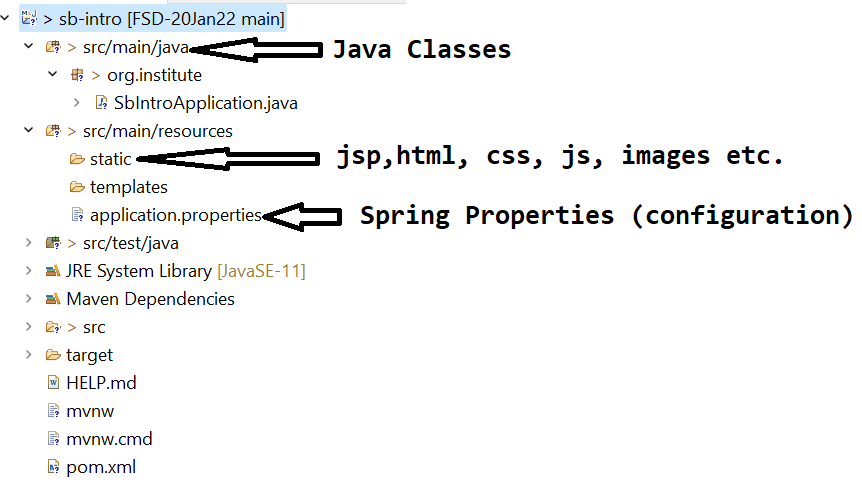
****

1. Click on “Next” button
2. Browser and select for a “Root directory”. (It is a path of the extracted folder and select a folder path in which pom.xml file is present.)



1. Click on “Finish” button
2. Wait till the project import completed. (for the first time it will take some time to download all Spring boot jar files. Make sure that your internet connection is proper and stable.)
3. If download not happened properly due to network issue then you might get an error on project. Follow the septs mention below.
   1. Right click on the project.
   2. Go to “Maven” option -> click on “Update Project…” option
   3. Select the check bob on the new window “Force Update of Snapshots/Releases”
   4. Click in “Ok” button

**Spring Boot Project Structure**



**@SpringBootApplication**

1. This annotation is a combination of 3 annotations
   1. **@Configuration**: This is use to mark any class as a configuration class. These classes will be loaded at the start of spring boot application.
   2. **@EnableAutoConfiguration:** This annotation is use to activate the auto configuration feature provided by spring boot. For Example, if you added a JDBC or JPA dependency then Spring Boot will provide the H2 database and connection will be establish with the H2 Database automatically. Another Example is, if you added spring security dependency, then spring boot provides you the login, logout page with default user name and password.
   3. **@ComponentScan:** Using this annotation spring will search and instantiate for the spring bean/component classes in the current package. Bean classes are the java classes which object will be created by spring and manage by spring. This type of classes can be created by using following annotation.

**@Component, @Controller, @RestController, @Service, @Repository, @ControllerAdvice** etc.

**Dependency Injection (DI)**

1. It is a process of injecting the object of one class into another class.
2. In core java term it is use to satisfy the HAS-A relation.
3. This can be achieved using **@Autowired** annotation

**Spring Container**

1. The Objects which are created by Spring will be store and manage inside spring Container.
2. Using the container, you can get the object created by spring. To get this object spring container has provided one method **getBeans()**
3. There are two main types of containers in spring
   1. ApplicationContext
      1. This container is majorly used for a large scale application like web application.
      2. This container support advance features like AOP (Aspect Oriented Programming), transaction management etc.
   2. BeanFactory
      1. This container is used for a small scale application like core application.
      2. This container does not support the functionalities like AOP and transaction Management.

**Spring Bean Scope**

1. The scope of bean defines how long that object will be accessible to the user.
2. There are 5 bean scopes
   1. **singleton**: this is the default scope of spring bean. In this case only one object of the bean class available through the container. This scope is applicable for the core application.
   2. **prototype**: in this case multiple objects will be return to every user request. This scope is applicable for the core application.
   3. **request**: This scope is applicable for the web application only. In this case for every user request the new bean class object will be provided.
   4. **session**: This scope is applicable for the web application only. In this case there is a single provided for every session.
   5. **globalSession**: This scope is applicable for the web application only. In this case there is only one object present through the application.
3. To Define a scope of the Bean class you can use the **@Scope** Annotation at class level.

**Customize the Bean class Object creation**

1. In the idea case spring creates an object of all the bean classes and manage the object of classes inside container.
2. In this case you will not have any control on the object creation.
3. To get a control on the Object creation of the class you can make use of **@Bean** annotation.
4. This object can be created by programmatically and this will be managed by spring container.

**Spring REST APIs**

1. REST APIs are use to consume from any platform and from any languages.
2. REST APIs communication can be happed using XML or JSON. That is REST APIs accept the request data through XML/JSON and return response in form if XML/JSON.
3. REST APIs used HTTP protocol for the communication.
4. REST APIs can create in Spring using following Annotation
   1. **@RestController**: Is use to create a controller which has REST APIs.
   2. **@RequestMapping**: This annotation is use to create a REST APIs and assign the URL for the API. There are HTTP method Specific annotations also available like @GetMapping, @PostMapping, @DeleteMapping etc.

**JSON**

1. JSON stand for JavaScript Object Notation.
2. This is a common language which can be understand by all the application based on any other languages.
3. JSON can be interpreted by any programming languages.
4. In case of java GSON and Jakson libraries are used to generate JSON from object and convert JSON into Java Object
5. Spring Boot Internally used Jakson lib. to interpret JSON.
6. JSON has Key and values pair which called as JSON element. JSON key are always in string.
7. JSON has a Array or Object in which these elements will be created.

**JSON Object ({})**

{ -----> JSON Object

"firstName":"John", -----> JSON element

"lastName":"Doe",

"salary":34534.34

}

**JSON Array ([])**

["Abc", "Xyz", "LMN"]

**JSON Array Of Objects**

[ ---------> JSON Array

{ -----> JSON Object

"firstName":"John", -----> JSON element

"lastName":"Doe",

"salary":34534.34

},

{ -----> JSON Object

"firstName":"John", -----> JSON element

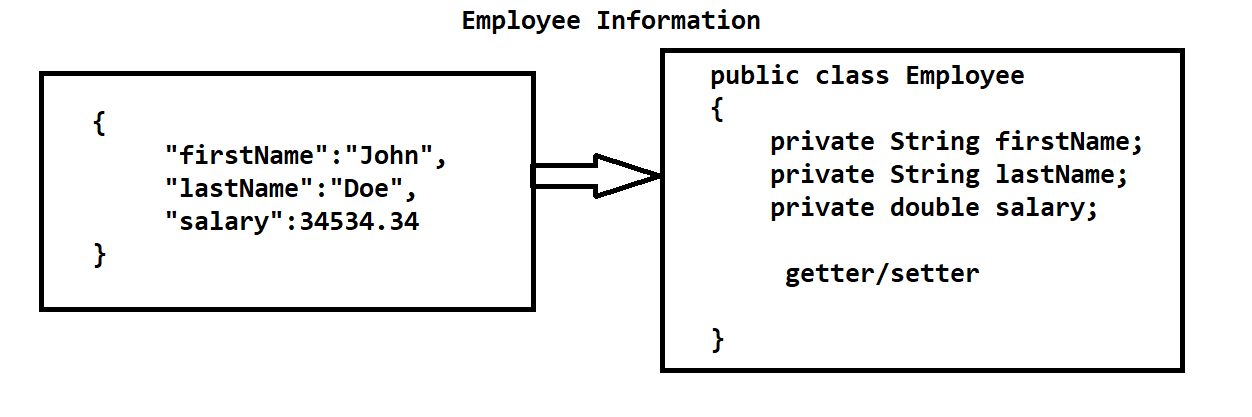
"lastName":"Doe",

"salary":34534.34

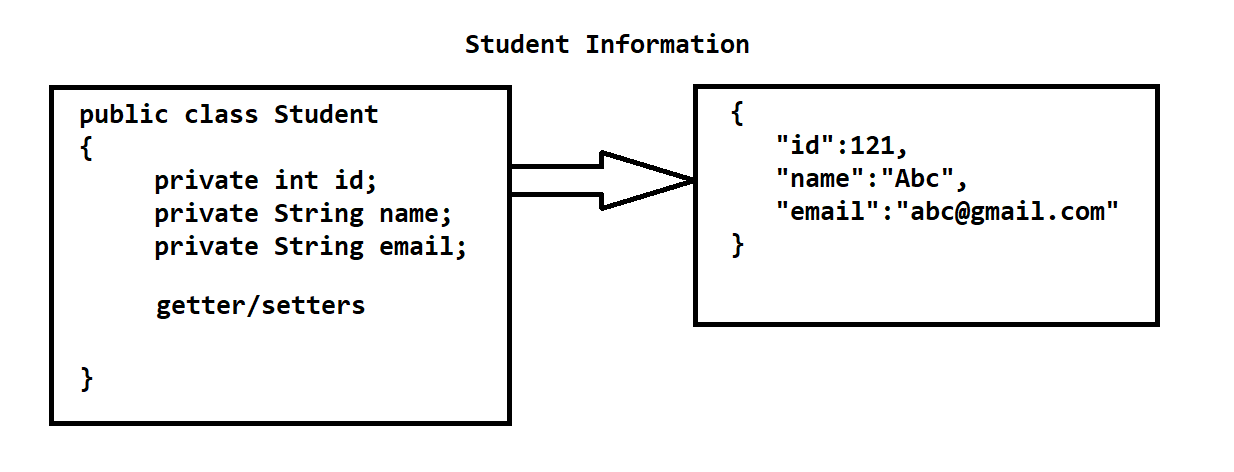
}

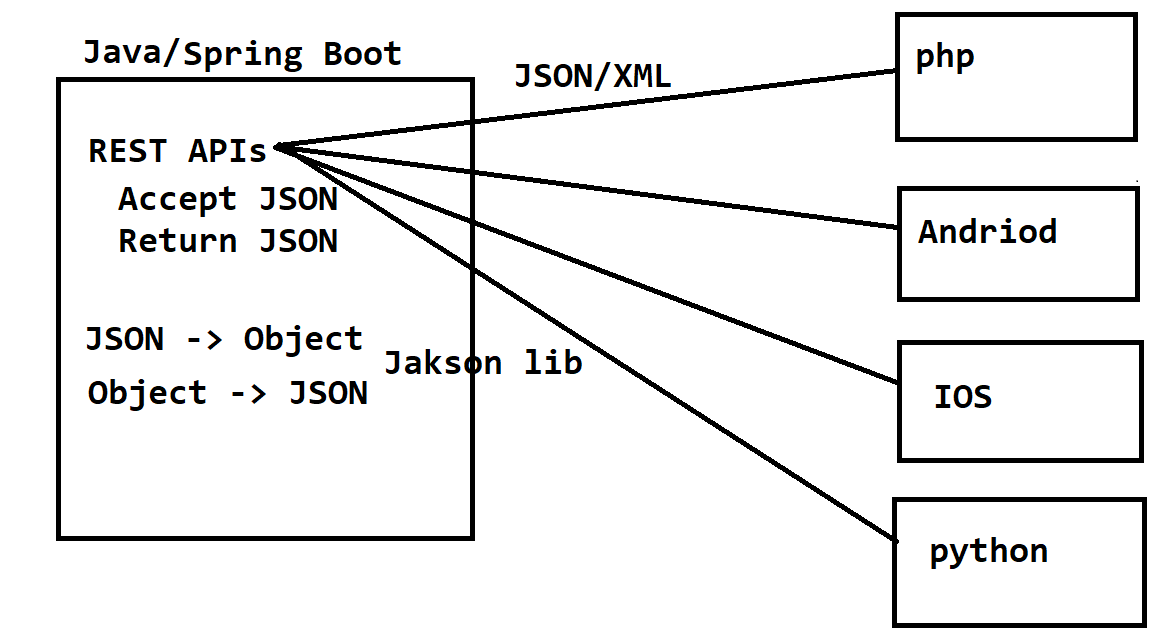
]

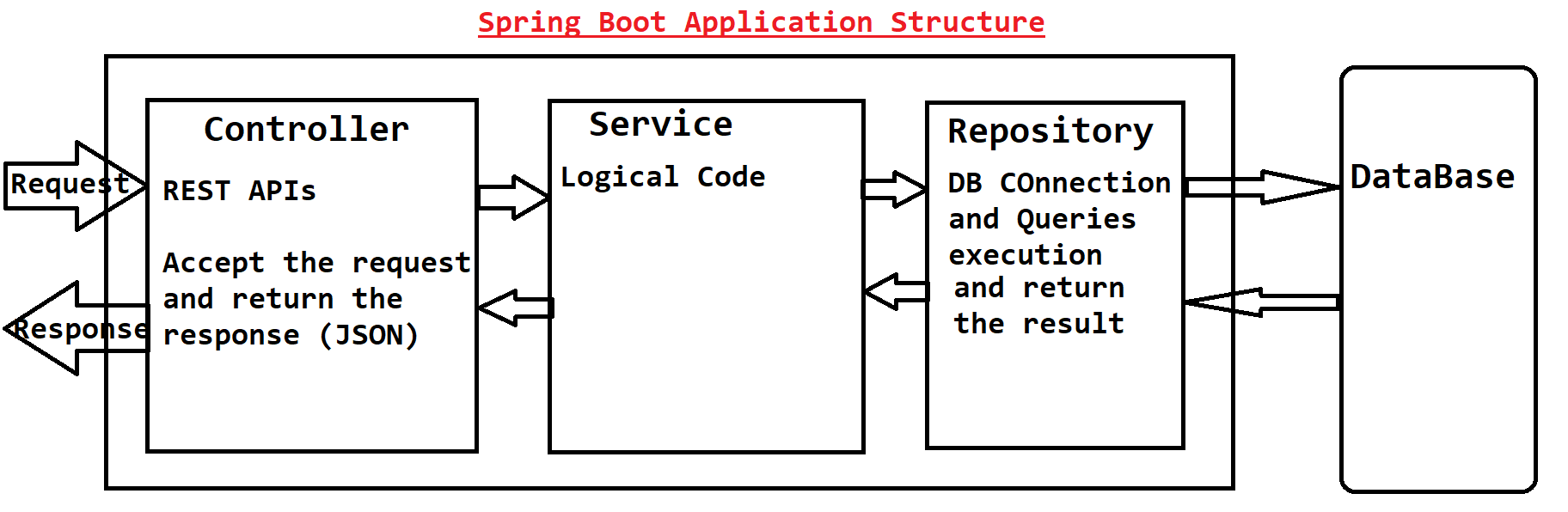
**JSON To Java Class**

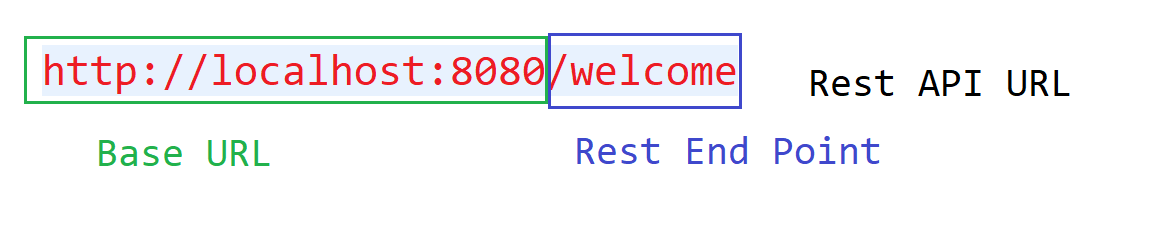


**Java Class to JSON**









Postman tool:

1. Postman tool is use to test the REST APIs.
2. It can be use to send request and return the response from the REST APIs
3. This tool can be use in the Testing of the Rest API
4. Download and install Postman tool

[**https://www.postman.com/downloads/**](https://www.postman.com/downloads/)

**HTTP Method**

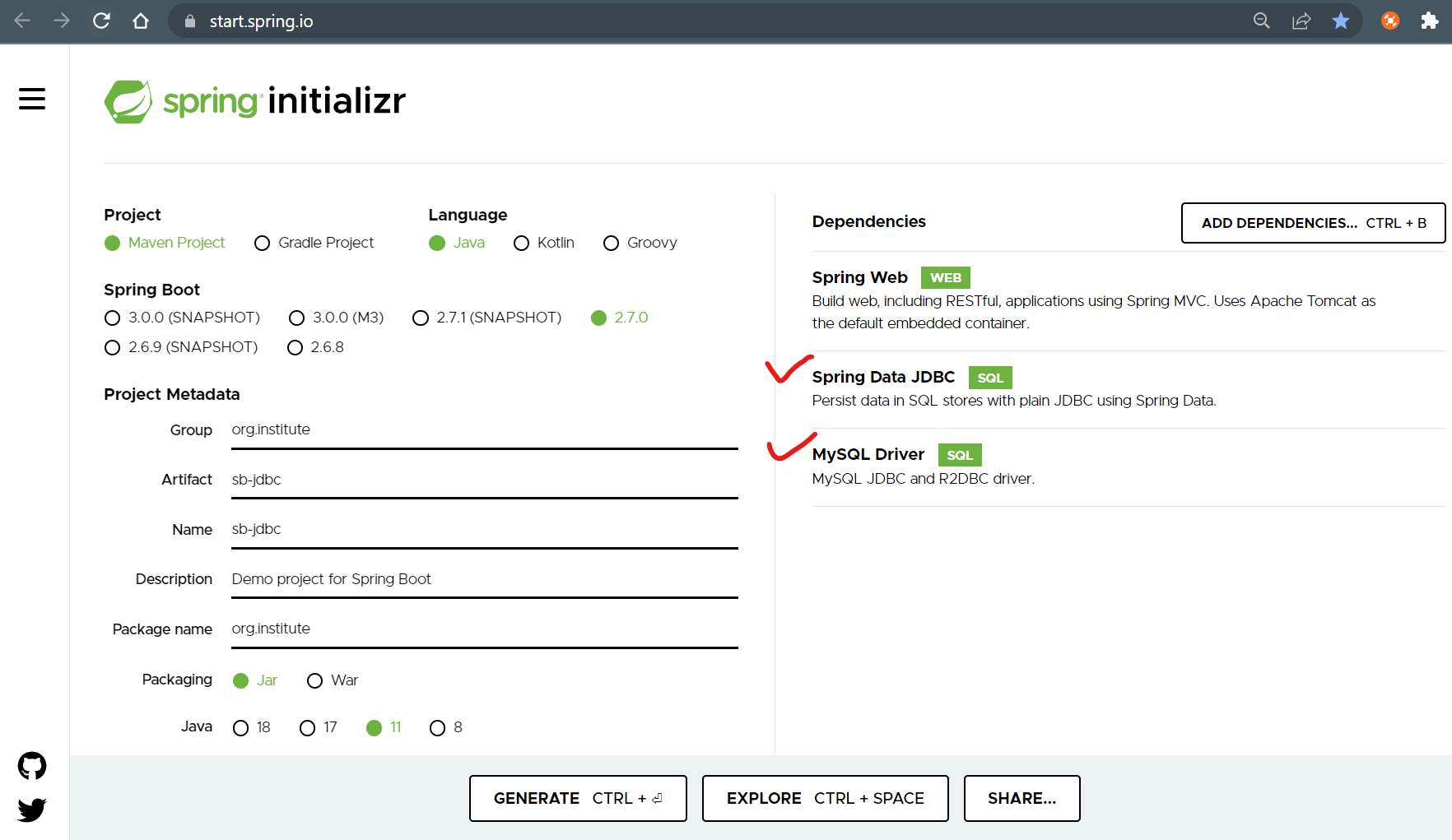
1. Http (Hyper Text Transfer Protocol)
2. Use to connect two parties and also carry a data from one location to another.
3. There are multiple methods of HTTP Protocol such as **GET, POST, PUT, DELETE**, OPTION, HEAD, PATCH etc..
   1. **GET:** Is use to retrieve the information. Can pass the Data in form of parameters or path variable.
   2. **POST:** Is use to create new entry or accept the value from the request. Can pass data in form of JSON (Request Body)
   3. **PUT:** is use to Update the existing entry. Accept data from the user. Can pass data in form of JSON (Request Body)
   4. **DELETE:** Is use to delete the existing entry. Can pass the Data in form of parameters or path variable.

**Get Data from the Request**

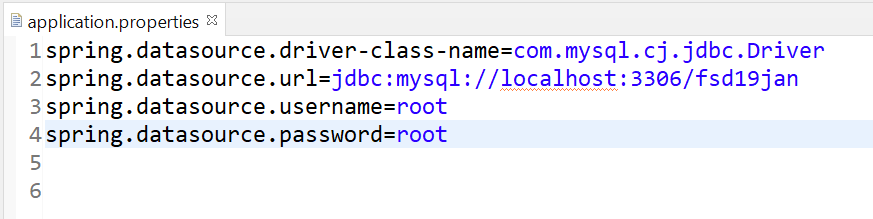
1. **Getting Data in JSON**
   1. Create a DTO class and use it as an input parameter of the method.
   2. Make use of **@ReuqestBody** annotation to convert JSON into Java Object
2. **Getting data in form of parameter**
   1. You can receive the data in the form of parameter.
   2. To accept parameter at java code use **@RequestParam** annotation before the method input parameter.
3. **Getting data in form of Path variable**
   1. You can receive the user data in the form of path variable which is a dynamic part of the path.
   2. To accept the value from the path use **@Pathvariable** annotation before method input parameter.
   3. By using this option you can values mandatory for the user.

**Spring JDBC**

1. Spring Java Database Connectivity
2. It is a set of APIs using which you can connect with DB more efficiently.
3. There are majorly 2 APIs which can be used for DB Connection
   1. **DriverManagerDataSource** : This class is use to established or provide the Db connections.
   2. **JdbcTemplate** : This class is used to execute SQL queries.
4. Steps to use Spring JDBC in an application
   1. Add a Spring JDBC Dependency



* 1. Provide DB connection configuration/properties inside application.properties file.



* 1. Use JdbcTemplate class by autowiring it to execute the query and get result back.
     1. There are 2 major methods you can used from the JdbcTemplate class to execute query
        1. query(): Is use to execute Select type of query
        2. update(): is used to execute other than select query

Task:

Create Student Inquiry module of an application. Create Following functionalities by REST API.

1. Create New Inquiry (Id(auto-generated), Name, contact, email, City, highQulification, passing Year, inqFor(Java, ASP.NET, Python, Testing), inqDate) -> RequestBody ->JSON
2. Get all the inquires
3. Delete The Inquiry by ID -> RequestParam or Path Param
4. Get Inquires by City
5. Get Inquires by inqFor

Create an Swagger API for all the rest endpoints

**Swagger Rest API documentation**

1. Can get a REST API documentation using Swagger.
2. Swagger is also known as Open API
3. Along with API doc you can also test the rest API like postman.
4. Add Swagger Documentation in to Application
   1. Add a dependency

<dependency>

<groupId>org.springdoc</groupId>

<artifactId>springdoc-openapi-ui</artifactId>

<version>1.6.9</version>

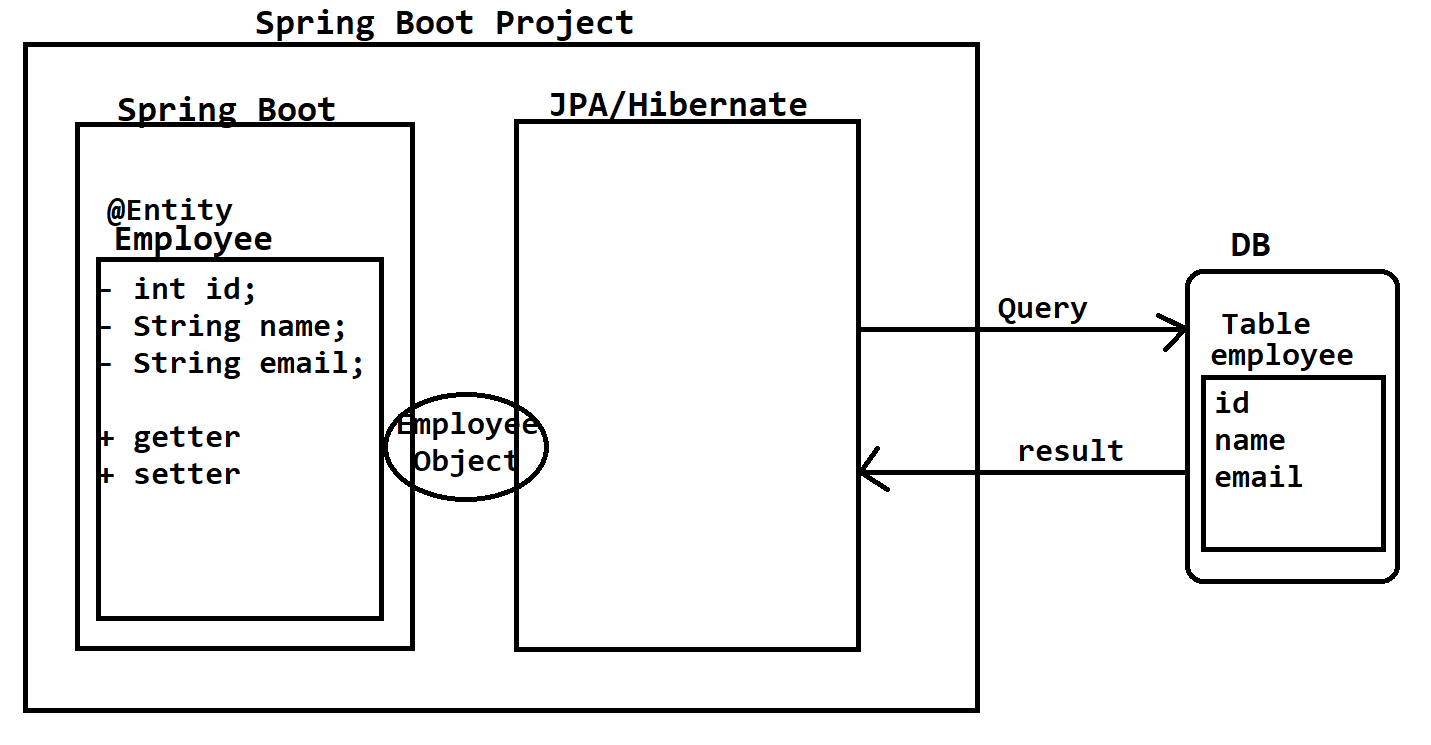
</dependency>

* 1. Access Document By URL

<http://localhost:8080/swagger-ui/index.html#/>

**Spring ORM (Object Relational Mapping)/JPA (Java Persistence API)**

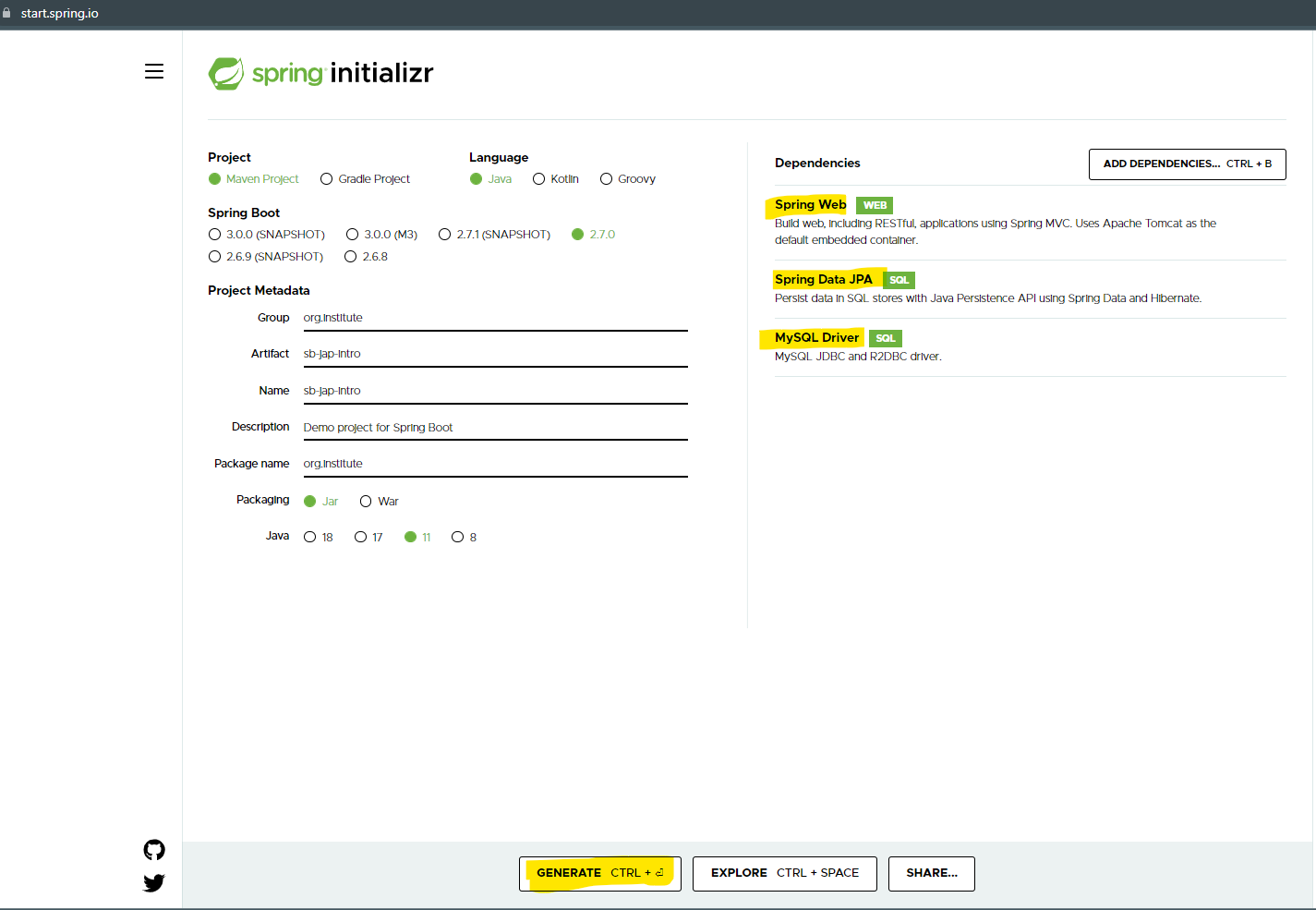
1. ORM is the Object Relational Mapping tool.
2. In this tool java objects will be mapped with the tables from the database using a framework.
3. There are multiple ORM tools available such as JPA, Hibernate, EJB etc.
4. Features of ORM tool
   1. Can develop application DB agnostic.
   2. ORM toll can generate and execute DDL and DML queries.
   3. ORM tool handles DB exception internally, and all the DB checked exception are converted into an Unchecked exception.
   4. ORM tool provide caching techniques using which you can improve the application performance.
   5. Hibernate provide us primary cache by default, you can also implement any third-party cache techniques as a part of secondary cache feature.
   6. ORM also supports the collection Mapping.
5. In the ORM tool you have to use Entity classes which connects with the table.
6. The Object of entity classes will be used to perform all DB operation.
7. Every entity class represent a table from the data base. And every table must have a primary key as per ORM rules.
8. You can convert your Encapsulated class (DTO classes) as an Entity class using @Entity annotation.



**Spring JPA Application**

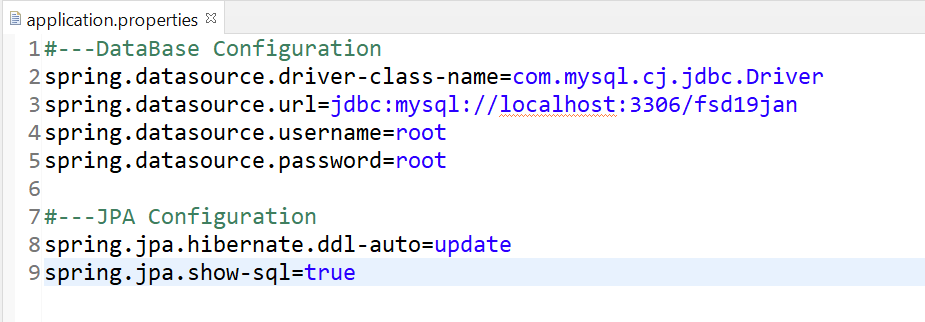
1. **Create Application**

Add the Spring JPA dependency while creating application.

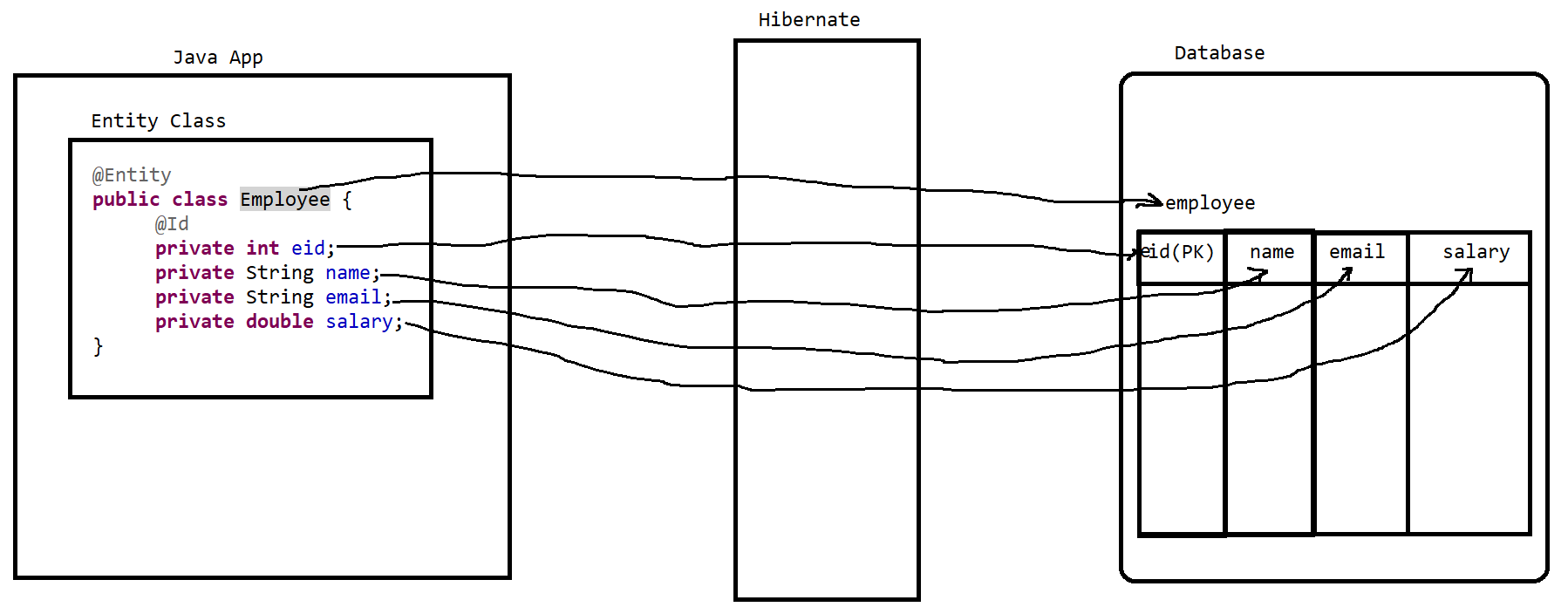


1. **Add Configurations**

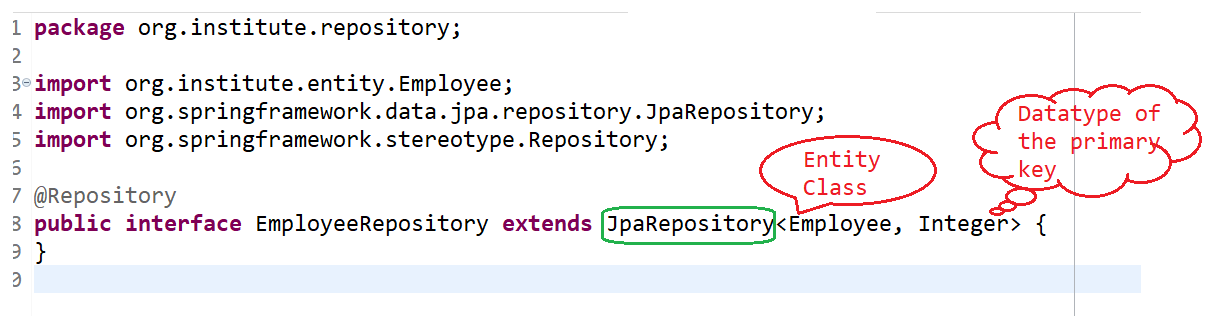
Add configurations related to DB and some configuration for the JPA inside application.properties file



1. Create Entity Classes
   1. These classes are the normal java classes which private variable and getters and setter.
   2. Entity classes are annotated with **@Entity** annotation.
   3. Every entity class connects with the table through hibernate.
   4. As per JPA/Hibernate every table must have a primary key. So you also have to create a variable for Primary key inside entity class.
   5. Any variable of the Entity class can be mark as primary key using **@Id** annotation.
   6. All annotations has to import from **javax.persistence** package



1. Create Repository to execute queries
   1. Repository is used to execute the DML (Insert, update, delete, select(DQL)) operations.
   2. To Create Repository spring JPA provided one interface called as **JpaRepository**
   3. To Create repository you have to create interface and extends your interface by JpaRepository interface.
   4. In the JpaRepository the basic DML operation are readily provided which can be directly used by extending the repository on custom interface.



* 1. Using this Repository you can perform following Db operation
     1. save(EntityClassObject) : save or update the record (insert and update query).
     2. delete(EntityClassObj) : delete the record by ID (delete query)
     3. findAll() : to get all the records from the DB (select \* query)
     4. findById(ID) : to get the records by ID(PK) (select by ID query with where clause)

**Entity Class Annotations**

|  |  |
| --- | --- |
| **Annotation** | **Description** |
| @Id | Is to mark any column as a primary key into DB |
| @Entity | To mark class as an entity class and in further execution hibernate will create a table for these classes |
| @Table | To customize the name of the table. By default table name is same as class name |
| @Column | To manage the column propertied like column name, size, nullable, unique etc.. |
| @GeneratedValue | This annotation is used to automatically generate the value for the primary key column |
| @Temporal | This annotation is used for Date type of variable to define the type of values (date, time, timestamp) to be store inside this column. |

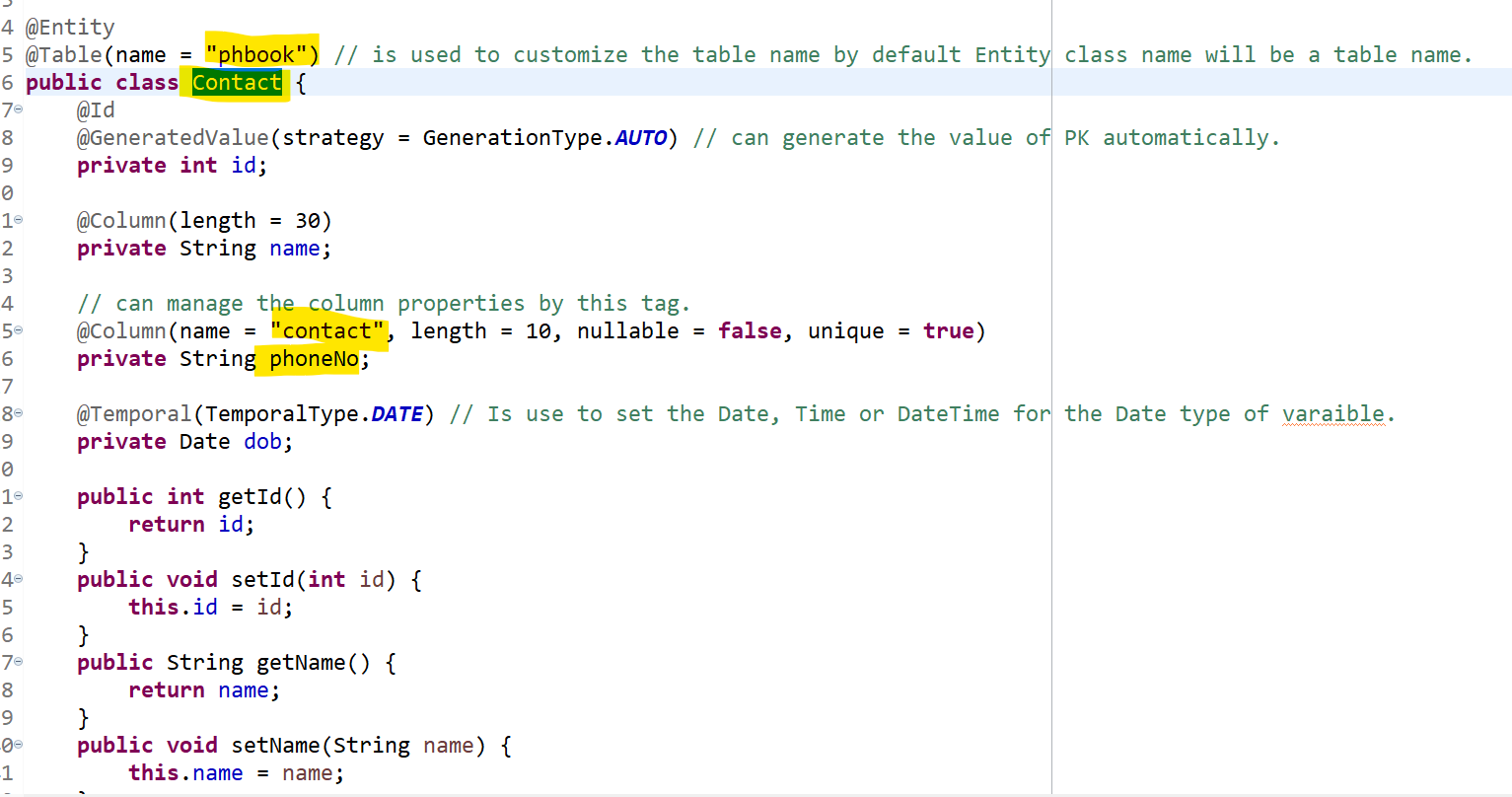
**Using Custom Query in SpringBoot JPA**

1. You can execute your own queried instead of getting generated from the JPA.
2. Spring JPA divided queries into two types:
   1. SQL
      1. You can write your SQL query directly and get the result.
      2. In the SQL the table name and the column name will be used inside query.
      3. This query is database dependent query.
      4. Example:

**select \* from phbook where contact='7709878987';**

* 1. HQL/JPQL
     1. HQL stands for Hibernate Query Language and JPQL stands for JPA Persistent Query Language.
     2. In these queried instead of table and column name you can use an entity class name and variable name.
     3. The queries will e a database independent.
     4. Example:

**From Contact where phoneNo=’ 7709878987’**



1. To execute SQL or HQL you can use **@Query** annotation on the method.
2. To Execute any DML query such as Insert, update, delete you have to use **@Modifying** and **@Transactional** annotation along with @Query annotation

**Task: Use Spring Boot JPA**

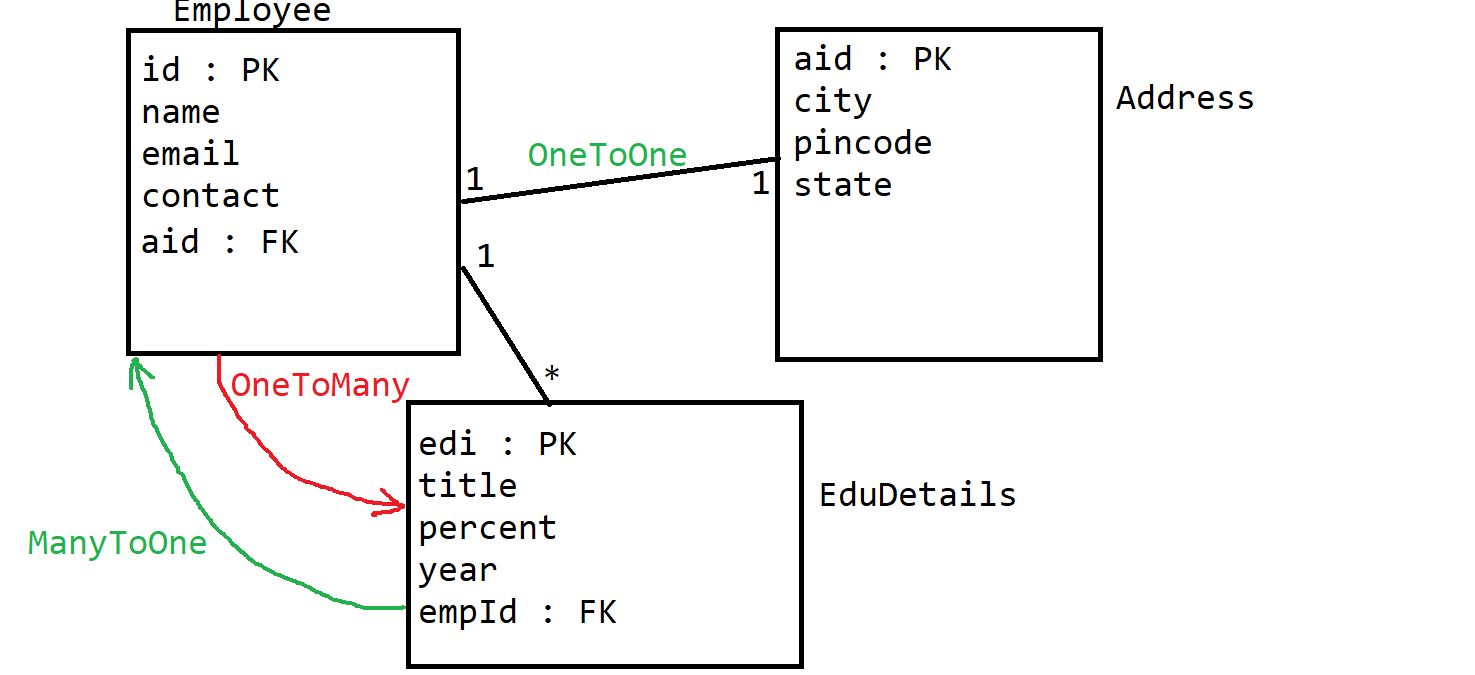
Create Student Inquiry module of an application. Create Following functionalities by REST API.

1. Create New Inquiry (Id(auto-generated), Name, contact, email, City, highQulification, passing Year, inqFor(Java, ASP.NET, Python, Testing), inqDate) -> RequestBody ->JSON
2. Get all the inquires
3. Delete The Inquiry by ID -> RequestParam or Path Param
4. Get Inquires by City
5. Get Inquires by inqFor

Create an Swagger API for all the rest endpoints

**Relational Mapping in JPA**

1. In the Database there can be a multiple table having primary and foreign key relations between then.
2. This type of structure also have to follow for the entity classes and this can achieve by the relation mapping.
3. In JPA relational mapping can be done using HAS-A relation.
4. In this there can be 4 types of relation between the table
   1. One To One (@OneToOne)
   2. One To Many (@OneToMany)
   3. Many To One (@ManyToOne)
   4. Many To Many (@ManyToMany)



1. If you wanted to use a Primary of one table as a foreign key into another then, while creating entity class you have to create a reference of one class (Primary Key class) into another class (Foreign Key class).
2. Create a reference of a class into a class in which you want to create a foreign key. And use any one annotation on the reference variable out of 4 mentioned above.
3. By default, hibernate pull the details of the current table along with the tables which is related with the current table. This can affect the application performance. It can be avoid using FetchType Property.
4. The FetchType property has 2 values
   1. **EAGER**
      1. This property is use to get the record from the current table as well as related tables.
      2. This is the default setting for @OneToOne and @ManyToOne.
   2. **LAZY**
      1. This property is use to get the records from the current table only and it will not by default get the details from the related tables.
      2. This is the default setting for the @OneToMany and @ManyToMany
5. You can set the Fetch type for the relational mapping as follows.

@ManyToOne(fetch = FetchType.***EAGER***)

**Bidirectional Relation**

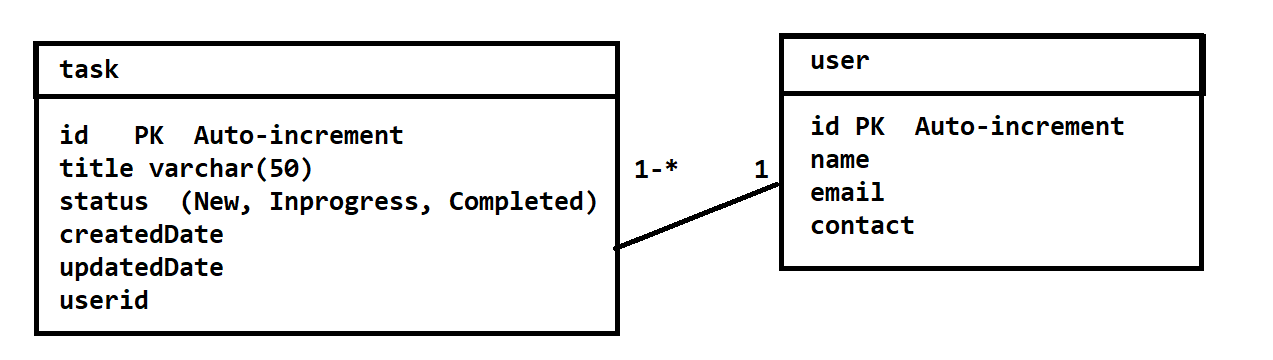
1. In the Bi-directional relation both the tables will be refer each other and due to this you will get the data from both the tables.
2. In the bi-directional relation there will be only one foreign key present and another entity will just points to foreign key present inside first table.
3. To mapped to a foreign key from another table you can use the mappedBy property.
4. Example:

@OneToMany(mappedBy = "user")

1. The value inside mappedBy property must be a variable name created into another table which denotes the foreign key.
2. Bi-directional Relation will create a issues while generating JSON response in REST API. This error is called as Nested Looping issue.
3. To Resolve this issue you have to use following annotations
   1. @JsonManagedReference : This annotation has to use on the field which is used for bi-direction (field with mappedBy properry)
   2. @JsonBackReference : This annotation is used on the field which is referred for the foreign key.

Task –

ToDoApplication



**Create Following REST APIs/Endpoints and Swagger REST API Document:**

|  |  |  |
| --- | --- | --- |
| **HTTP Method** | **Endpoint** | **Description** |
| **User Endpoint** | | |
| POST | /user/save | Save The user details |
| PUT | /user/update | Update user details |
| GET | /user/get-all | Get List of all available users |
| **Task Endpoint** | | |
| POST | /task/save | Add User Task |
| PUT | /task/update | Update Task Details |
| PUT | /task/update/{id}/{status} | Update Task status by ID Use HQL |
| GET | /task/task-by-id?id=<val> | Get Task by task ID |
| GET | /task/task-by-user?id=<val> | Get List of tasks by User Id |
| GET | /task/task-by-status?id=<val>&status=<val> | Get list of tasks by status and user Id (All task for user by status) |
| DELETE | /task/delete-by-id/{id} | Delete Task by ID |