**Spring Boot**

It is module of spring core framework which speed up development

Spring Framework+Embedded Tomcat-Configuration.xml=Spring Boot

**Why should we use Spring Boot Framework?**

We should use Spring Boot Framework because:

* The dependency injection approach is used in Spring Boot.
* It contains powerful database transaction management capabilities.
* It simplifies integration with other Java frameworks like JPA/Hibernate ORM, Struts, etc.
* It reduces the cost and development time of the application.

Advantages of Spring Boot

* It creates **stand-alone** Spring applications that can be started using Java **-jar**.
* It tests web applications easily with the help of different **Embedded** HTTP servers such as **Tomcat, Jetty,** etc. We don't need to deploy WAR files.
* It provides opinionated '**starter**' POMs to simplify our Maven configuration.
* It provides **production-ready** features such as **metrics, health checks,** and **externalized configuration**.
* There is no requirement for **XML** configuration.
* It offers the number of **plug-ins**.
* It also minimizes writing multiple **boilerplate codes** (the code that has to be included in many places with little or no alteration), XML configuration, and annotations.

How to start with spring Boot :

1.Create maven project and add starter dependencies.

2.Using Spring Initializr –Tool

1.Creating spring boot project using STS

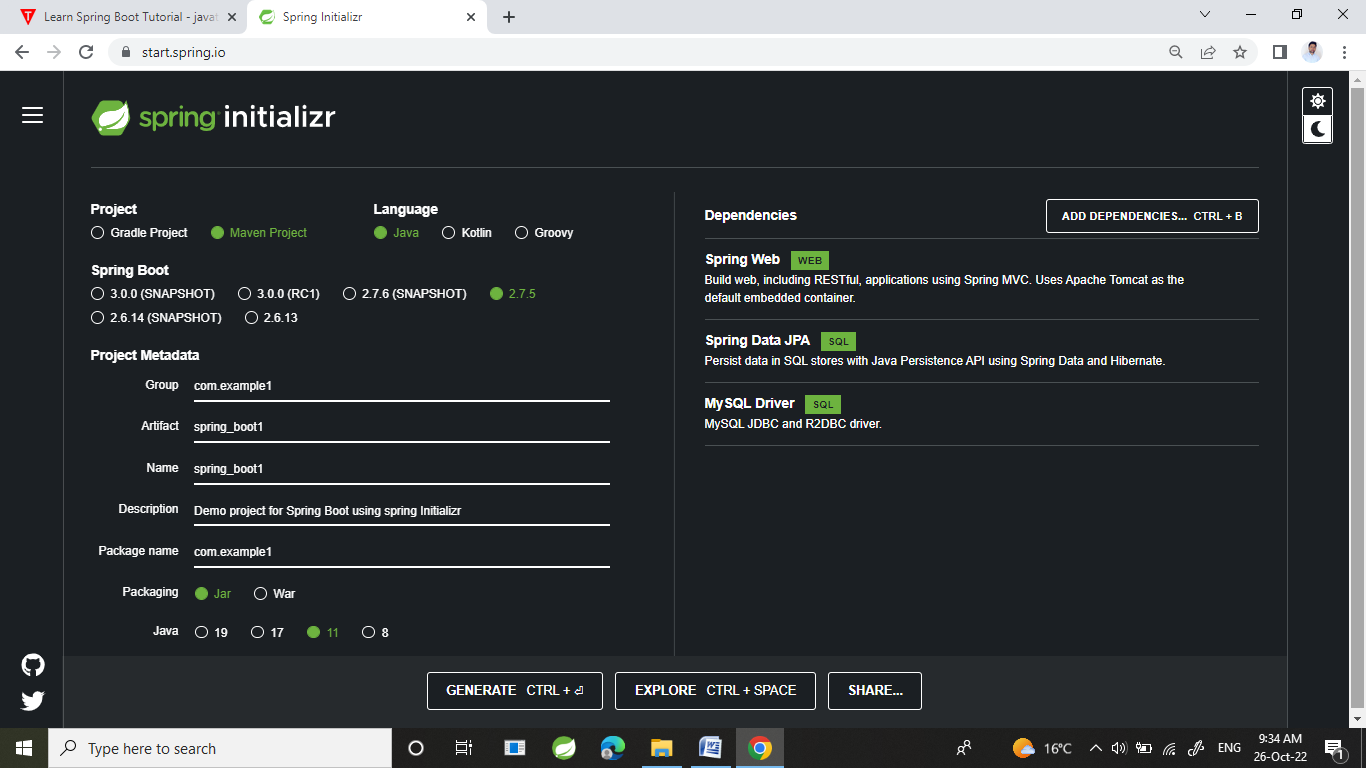
New—spring starter Project—fill the required detail group Id,artefact Id.Description,name,packaging

NEXT----Add Required dependencies ---finish

2.Creating Spring Boot Project using Spring Initializr

1.Search Spring Initializr---open <https://start.spring.io/>

2.fill the required detail-----Genrate



**Spring:** Spring Framework is the most popular application development framework of Java. The main feature of the Spring Framework is **dependency Injection** or **Inversion of Control** (IoC). With the help of Spring Framework, we can develop a **loosely** coupled application. It is better to use if application type or characteristics are purely defined.

**Spring Boot:** Spring Boot is a module of Spring Framework. It allows us to build a stand-alone application with minimal or zero configurations. It is better to use if we want to develop a simple Spring-based application or RESTful services.

**Spring MVC:** Spring MVC is a Web MVC Framework for building web applications. It contains a lot of configuration files for various capabilities. It is an HTTP oriented web application development framework.

[**https://www.simplilearn.com/tutorials/spring-boot-tutorial/spring-vs-spring-boot**](https://www.simplilearn.com/tutorials/spring-boot-tutorial/spring-vs-spring-boot)

**Difference between Spring MVC and Spring Boot :**

|  |  |  |
| --- | --- | --- |
| S.No. | SPRING MVC | SPRING BOOT |
| 1. | Spring MVC is a Model View, and Controller based web framework widely used to develop web applications. | Spring Boot is built on top of the conventional spring framework, widely used to develop REST APIs. |
| 2. | If we are using Spring MVC, we need to build the configuration manually. | If we are using Spring Boot, there is no need to build the configuration manually. |
| 3. | In the Spring MVC, a deployment descriptor is required. | In the Spring Boot, there is no need for a deployment descriptor. |
| 4. | Spring MVC specifies each dependency separately. | It wraps the dependencies together in a single unit. |
| 5. | Spring MVC framework consists of four components : Model, View, Controller, and Front Controller. | There are four main layers in Spring Boot: Presentation Layer, Data Access Layer, Service Layer, and Integration Layer. |
| 6. | It takes more time in development. | It reduces development time and increases productivity. |
| 7. | Spring MVC do not provide powerful batch processing. | Powerful batch processing is provided by Spring Boot. |
| 8. | Ready to use feature are provided by it for building web applications. | Default configurations are provided by it for building a Spring powered framework. |

**SPRING BOOT ARCHITECTURE\**

The **Spring Boot** is built on top of the core [Spring](https://www.geeksforgeeks.org/introduction-to-spring-framework/) framework. It is a simplified and automated version of the spring framework. The spring boot follows a **layered architecture** in which each layer communicates to other layers(Above or below in hierarchical order

1. **Presentation Layer** – Authentication & Json Translation

The presentation layer is the top layer of the spring boot architecture. It consists of Views. i.e., the front-end part of the application. It handles the HTTP requests and performs authentication. It is responsible for converting the JSON field’s parameter to Java Objects and vice-versa. Once it performs the authentication of the request it passes it to the next layer. i.e., the business layer.

1. **Business Layer** – Business Logic, Validation & Authorization

The business layer contains all the business logic. It consists of services classes. It is responsible for validation and authorization.

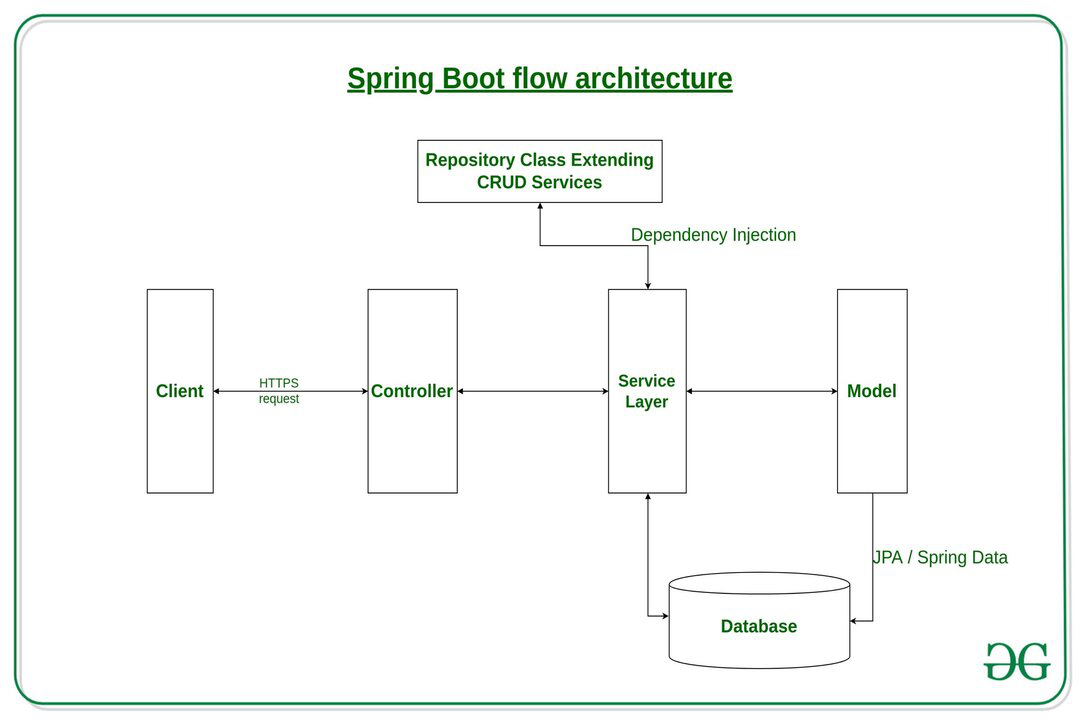
1. **Persistence Layer** – Storage Logic

The persistence layer contains all the database storage logic. It is responsible for converting business objects to the database row and vice-versa.

1. **Database Layer** – Actual Database

The database layer contains all the databases such as [MySql](https://www.geeksforgeeks.org/sql-tutorial/), [MongoDB](https://www.geeksforgeeks.org/mongodb-an-introduction/), etc. This layer can contain multiple databases. It is responsible for performing the [CRUD](https://www.geeksforgeeks.org/spring-boot-crud-operations/) operations.

### Spring Boot Flow Architecture



*Fig 2 – Spring boot flow architecture*

**Explanation:**

* The Client makes an **HTTP** request(GET, PUT, POST, etc.)
* The HTTP request is forwarded to the **Controller**. The controller maps the request. It processes the handles and calls the server logic.
* The business logic is performed in the **Service layer**. The spring boot performs all the logic over the data of the database which is mapped to the spring boot model class through [Java Persistence Library](https://www.geeksforgeeks.org/spring-boot-spring-data-jpa/)(**JPA**).
* The [JSP](https://www.geeksforgeeks.org/introduction-to-jsp/) page is returned as Response from the controller.

**What is use of application.properties**

This file contain different configuration related properties of your spring boot application in key-value pair

**application.properties** file is used to write the application-related property into that file. This file contains the different configuration which is required to run the application in a different environment, and each environment will have a different property defined by it. Inside the application properties file, we define every type of property like changing the port, database connectivity, connection to the eureka server, and many more.

#Tomcat server port number 8080

#server.port=7070------To change port number

#context path default value : empty

server.servlet.context-path=/spring\_boot

#DB properties--------------------------------------------------------

spring.datasource.url=jdbc:mysql://localhost:3306/springBoot?useSSL=false&allowPublicKeyRetrieval=true

spring.datasource.username=root

spring.datasource.password=sairam

#---------------------------------------------------------------------------

# JPA properties---------------------------------------------------------

spring.jpa.show-sql = true

spring.jpa.hibernate.ddl-auto = update

#spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect

#logging.level.org.springframework.orm.hibernate5=DEBUG

#-----------------------------------------------------------------------------

# Spring MVC ViewResolver related--------------------------------------

spring.mvc.view.prefix=/WEB-INF/views

spring.mvc.view.suffix=.jsp

#--------------------------------------------------------------------------------

#Disable open session in view

spring.jpa.open-in-view=false

Spring Boot Starters : Spring Boot Starters are dependency descriptors that can be added under the **<dependencies>** section in pom.xml. There are around 50+ Spring Boot Starters for different Spring and related technologies. These starters give all the dependencies under a single name. For example, if you want to use Spring Data JPA for database access, you can include **spring-boot-starter-data-jpa**dependency.

<dependencies>

<dependency>

<groupId>***org.springframework.boot***</groupId>

<artifactId>***spring-boot-starter-data-jpa***</artifactId>---------spring boot starter

</dependency>

</dependencies>

**Spring Boot Starter Web** : There are two important features of spring-boot-starter-web:

* It is compatible for web development
* Auto configuration

Starter of Spring web uses Spring MVC, REST and Tomcat as a default embedded server. The single spring-boot-starter-web dependency transitively pulls in all dependencies related to web development. It also reduces the build dependency count.

**Spring Data JPA :**

The java persistence API provides a specification for persisting, reading, and managing data from your java object to your relational tables in the database. JPA specifies the set of rules and guidelines for developing interfaces that follow standards. Straight to the point: JPA is just guidelines to implement ORM and there is no underlying code for the implementation. Spring Data JPA is part of the spring framework. The goal of spring data repository abstraction is to significantly reduce the amount of boilerplate code required to implement a data access layer for various persistence stores. Spring Data JPA is not a JPA provider, it is a library/framework that adds an extra layer of abstraction on the top of our JPA provider line Hibernate.

**Spring Boot AutoConfiguration :**

* The @EnableAutoConfiguration annotation enables the auto-configuration of Spring ApplicationContext by scanning the classpath components and registering the beans.
* This annotation is wrapped inside the @SpringBootApplication annotation along with @ComponentScan and @SpringBootConfiguration annotations.
* When running main() method, this annotation initiates auto-configuration.

***@SpringBootApplication=@ComponentScan+@EnableAutoConfiguration+@Configuration***

CRUD on Database :

**CRUDRepository Interface** : There is an interface available in Spring Boot named as CrudRepository that contains methods for CRUD operations. It provides generic Crud operation on a repository. It is defined in the package ***org.springframework.data.repository*** and It extends the Spring Data **Repository**interface. If someone wants to use CrudRepository in the spring boot application he/she has to create an interface and extend the CrudRepository interface.

*public interface CrudRepository<T, ID> extends Repository<T, ID>*

**Method 1: save():**Saves a given entity. Use the returned instance for further operations as the save operation

**Method 2: findById():** Retrieves an entity by its id.

**Method 3: findAll():**Returns all instances of the type.

**Method 4: count():** Returns the number of entities available.

**Method 5: delete(int id):**Deletes the entity with the given id.

**Custom Finder Method :**

By default finder method from CrudRepository-findAll(),findById() and so on…

But if you want to retrieve data by name or any other field then need to implement custom finder method

KEYWORD custom\_finder\_method Query

|  |  |  |
| --- | --- | --- |
| Distinct | findDistinctByLastnameAndFirstname | select distinct …​ where x.lastname = ?1 and x.firstname = ?2 |
| And | findByLastnameAndFirstname | … where x.lastname = ?1 and x.firstname = ?2 |
| Or | findByLastnameOrFirstname | … where x.lastname = ?1 or x.firstname = ?2 |
| Is, Equals | findByFirstname,findByFirstnameIs,findByFirstnameEquals | … where x.firstname = ?1 |

List<Person> **findByNameStartsWith** (String prefix);

*// Enables the distinct flag for the query*

List<Person> **findDistinctPeopleByLastnameOrFirstname**(String lastname, String firstname);

//select \* from Person where email=”emailAddress” and lastname=”lastname”

List<Person> **findByEmailAddressAndLastname**(EmailAddress emailAddress, String lastname);

//select \* from Person where age>:?

List<Person> **findByAgeGreaterThan** (int age);

**On Top Of that still if you want to execute complex Query based on some other criteria then need to execute JPQL(HQL) or native Query(SQL)**

**So whatever Query you have bind that query with custom method in @Query annotation**

**@Query = This annotation used simply to execute complex query based on diff criteria with respect to that method,it is method level annotation**

**JPQL :**

@Query(“select u from User u where u.name=:nm”)

public User getByName(@Param(“nm”) String name)

**By Native Query**

@Query(value="select \* from Book b WHERE b.author\_name = ?",nativeQuery=true )

public Book getByAuthor(String name);