**SPRING BOOT NOTE**

**What is Spring?**

The Spring Framework (Spring) is an [open-source](https://www.techtarget.com/whatis/definition/open-source) [application](https://www.techtarget.com/searchsoftwarequality/definition/application) [framework](https://www.techtarget.com/whatis/definition/framework) that provides [infrastructure](https://www.techtarget.com/searchdatacenter/definition/infrastructure) support for developing [Java](https://www.theserverside.com/definition/Java) applications. One of the most popular Java Enterprise Edition ([Java EE](https://www.theserverside.com/definition/J2EE-Java-2-Platform-Enterprise-Edition)) frameworks

Released in June 2003 by Rod Johnson under the [Apache 2.0 license](https://www.techtarget.com/whatis/definition/Apache-License)

Spring is a Java web framework build to create enterprise production ready application.

When we create a Java enterprise application, we need to do a lot of configurations, we need to add lot packages, jars and also we need to add database support as well.

Spring is Java web framework it will allows us to do lot of things. Spring provides a bunch of modules for various configurations, we can use those modules to build our application. But to use all these modules, we need to do lot of configurations to up and run the application.

Long story short, Spring framework helps us to create production ready application but we need to do lot of configuration.

**Why Spring?**

Java programs are complex and feature many heavyweight [components](https://www.techtarget.com/whatis/definition/component). Heavyweight means the components are dependent on the underlying operating system ([OS](https://www.techtarget.com/whatis/definition/operating-system-OS)) for their appearance and properties.

Spring is considered to be a secure, low-cost and flexible framework. Spring improves coding efficiency and reduces overall application development time because it is lightweight -- efficient at utilizing system resources -- and has a lot of [support](https://www.techtarget.com/whatis/definition/support).

Spring removes tedious configuration work so that developers can focus on writing [business logic](https://www.techtarget.com/whatis/definition/business-logic). Spring handles the infrastructure so developers can focus on the application.

**What is Spring Boot?**

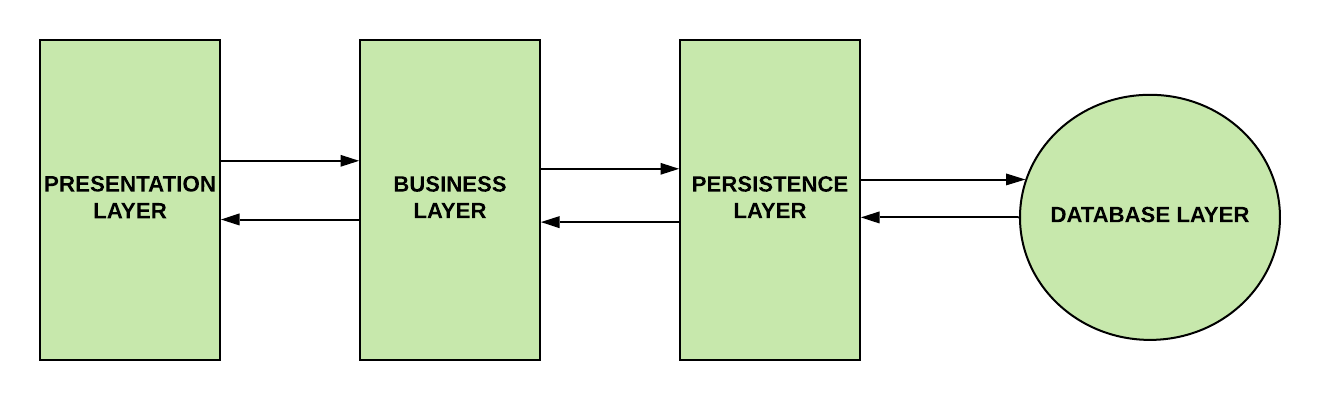
Spring boot is a tool introduced by Spring Team, to create Spring based applications quickly and easily. It provides zero configuration support. We can concentrate only on the actual business logic rather than concentrating on the configuration.

Spring boot is not a framework, it’s a tool or extension to create Spring based applications. Spring boot internally uses Spring.

Spring Boot Architecture

Spring Boot Architecture has four layers:

* **Presentation Layer**
* **Business Layer**
* **Persistence Layer**
* **Database Layer**



Presentation Layer

This layer is at the top of the architecture. This tier is responsible for:

✔️ Performing authentication.

✔️ Converting JSON data into an object (and vice versa).

✔️ Handling HTTP requests.

✔️ Transfering authentication to the business layer.

The presentation layer is the equivalent of the Controller class. The Controller class handles all the incoming REST API requests (GET, POST, PUT, DELETE, PATCH) from the Client.

Business Layer

The business layer is responsible for:

✔️ Performing validation.

✔️ Performing authorization.

✔️ Handling the business logic and rules.

This layer is the equivalent to the Service class. It's where we handle the business logic. If you're wondering what do we mean by "business logic", I found an interesting discussion on [StackExchange](https://softwareengineering.stackexchange.com/questions/234251/what-really-is-the-business-logic" \t "_blank). In short, the business logic in software engineering is where we decide what the software needs to do. An example of this is validation. If you are ever requested to validate something, this needs to happen inside the Service class.

The Business layer communicates with both the Presentation layer and the Persistence Layer.

Persistence Layer

This layer is responsible for:

✔️ Containing storage logic.

✔️ Fetching objects and translating them into database rows (and vice versa).

This layer is the equivalent of the Repository interface. We write database queries inside this interface.

The Persistence layer is the only layer that communicates with the Business layer and the Database layer.

Database Layer

This layer is responsible for:

✔️ Performing database operations (mainly CRUD operations).

This layer is simply the actual database that you decide to use to build your application.

Spring Boot Workflow



Spring Boot workflow acts like this:

1. The Client makes an HTTP request.
2. The Controller class receives the HTTP request.
3. The Controller understands what type of request will process, and then it deals with it.
4. If it is needed, it calls the service class.
5. The Service Class is going to handle the business logic. It does this on the data from the database.
6. If everything goes well, we return a JSP page.

* Presentation Layer - **controller**package
* Business Logic Layer - **service**package
* Data Access Layer - **repository**package
* Persistence layer -use the **JPARepository** Interface

**Advantages of using Spring boot**

* Rapid application development
* Managing the dependencies
* Auto configuration support
* Support for embedded servers
* Support for creating microservices

**What is tightly coubled and loose coubling:**

Without a Spring Framework, application code tends to be tightly coupled (interdependent), which is not considered good coding practice. Loose coupling is ideal because loosely coupled components are independent, meaning changes in one will not affect the operation of others.

**Dependency Injection**

In typical Java applications, we as a developer, create Java objects. But when we are working with enterprise production grade applications, this is not a good idea. Because creating the objects, destroying object and managing their lifecycle is not easy. So instead of we create the objects, we give a control over to the framework to do manage the lifecycle of objects. This is called Inversion of Control.

To use Inversion of Control, we need to use Dependency Injection, it a design pattern it will allows us to create Java objects without using new keyword.

When the spring boot application runs, spring has a factory or container, it creates the Beans for us, next whenever we ask for the particular bean, first it will check in the container, if its present then it gives the already created object otherwise it creates new objects for us.

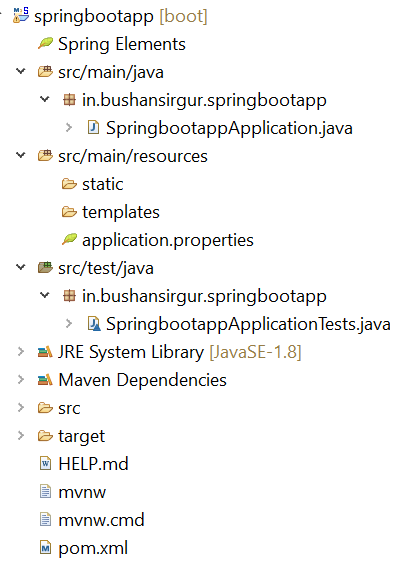
BEAN:A Spring bean is an object that is instantiated, created, and managed by the IoC container. Beans are the backbone of an application.

[Dependency injection](https://www.techtarget.com/searchapparchitecture/definition/dependency-injection) - A programming design pattern that makes code loosely coupled, meaning that any change in the application of one, will not affect the other.

[Inversion of control](https://www.theserverside.com/feature/Meaning-of-inversion-of-control-in-Spring-and-Java-IoC-explained) (IoC) - Taking control away from the class and giving it to the Spring Framework.

Inversion of control container - This is the core of the Spring Framework where objects are created, wired together, configured, and managed throughout their life cycle.

**Project Structure**

If you create a spring boot project either using STS IDE or Spring Initializr, it follows the standard **maven project sturcture**  
[](https://imgbb.com/)

It creates so many files and folders. Let’s discuss these one by one in detail

* ***src/main/java***: Contains the java source code. It contains the base package by default. The base package contains the base class with main() method.
* ***in.bushansirgur.springbootapp***: This is the base package which is created by spring initializer. The package name is given at the time creating the project.
* ***SpringbootappApplication.java***: This is the base class, which contains the main() method with @SpringBootApplication annotation. This is the starting point to our spring boot application.

package in.bushansirgur.springbootapp;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringbootappApplication {

public static void main(String[] args) {

SpringApplication.run(SpringbootappApplication.class, args);

}

}

* ***src/main/resources***: It contains static folder, templates folder and property file.
* ***static***: This folder contains the static files such as html, javascript, css files. If we are creating the spring boot web application then this folder may contain the static files.
* ***templates***: This folder contains the templates for our application. For example, if we are creating web application, if we use thymeleaf template engine then this folder contains all the UI templates.
* ***src/test/java***: It contains the java source files for Unit testing. This also contains the base package and base class for Unit testing.
* ***JRE System library***: This contains the JDK and JRE system related jars.
* ***Maven Dependencies***: This contains the dependency jars for our application.
* ***src***: This folder is same as the src/main/java
* ***target***: This folder contains the war/jar and build related files after the application is successfully build.
* ***HELP.md***: This file contains the reference links for spring boot and apache tomcat
* ***mvnw***: These are the Maven wrapper files, which allows us to run the maven project on Linux/mac
* ***mvnw***.cmd: This is also Maven wrapper files, which allows us to run the maven project on Windows platform
* ***pom.xml***: It contains the project related information and dependencies used in the application.