Maven:

Maven has three phases. Each phase is called a life cycle. In each phase there are certain goals that are to be achieved. Lifecycle is a order of those goals.

Clean – removes temporary directories and files.

Default – where the most useful goals live.

Site- where the documentation is generated.

Clean phase:

Pre-clean: hook before cleaning

Clean: does the actual cleaning

Post-clean: hook for after cleaning

Default phase:  
mnvw default

Compile – compiles the code into bytecode

Test – runs unit tests

Package – creates a jar or a war file

Verify – runs checks and integration tests.

**Compile -> test- > package -> verify. All these happens in this order only.**

**Commands – mnvw clean, mnvw compile , mnvw test, mnvw package , mnvw verify**

**Now how to use these in the actual springboot project?**

**Maven Project Structure:**

**Src/main/resources src/test/resources target**

Note: Springboot actually allows us to override configuration inside of the test by placing the config files inside the src/test/resources.

Our build project and any files that are processed by maven are present in the target directory.

**Maven Workflow:**

Mvnw clean package

Now the target directory is created in the project directory. Enter into that by tyoing cd target

Cd target

Java – jar quickstart-0.0.1-SNAPSHOT.jar -> run this jar file to start the application.

This is how we run our application using maven.

**Maven SpringBoot Plugin:**

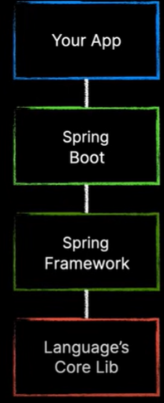
**To run our application using the springboot maven plugin.**

**Command: mvnw spring-boot:run -> This is going to start the application on the specified port.**

**Spring vs Springboot:**

Springboot is built on top of spring framework.

Springboot is just an another layer over spring. A layer which solves the problems in the spring which is auto-configuration.

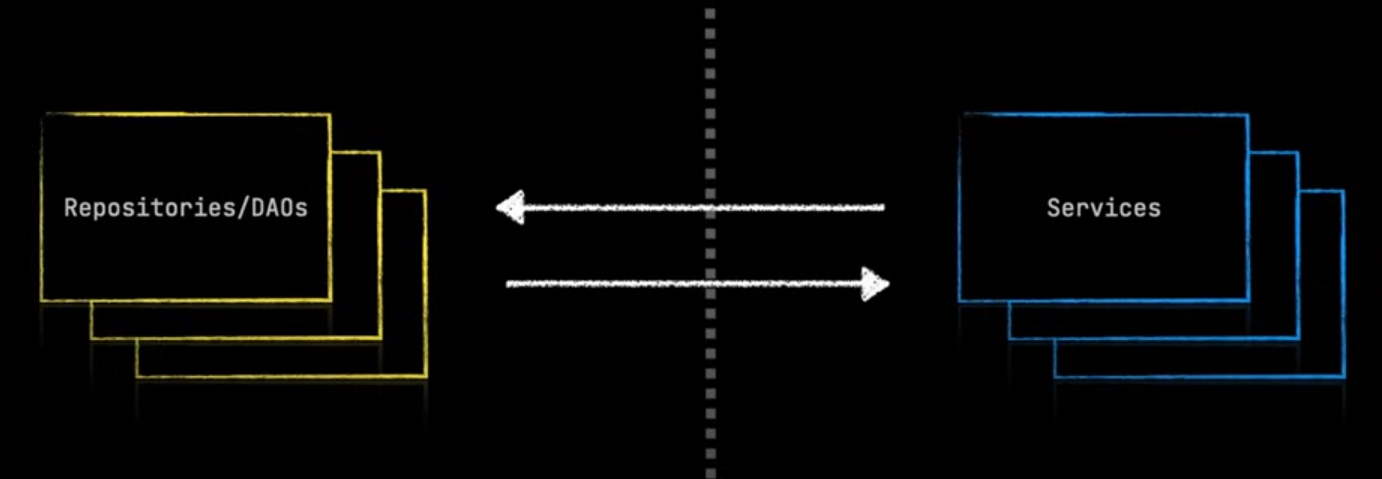
Before understanding how springboot simplifies our lives, we need to understand the basics of spring framework first.

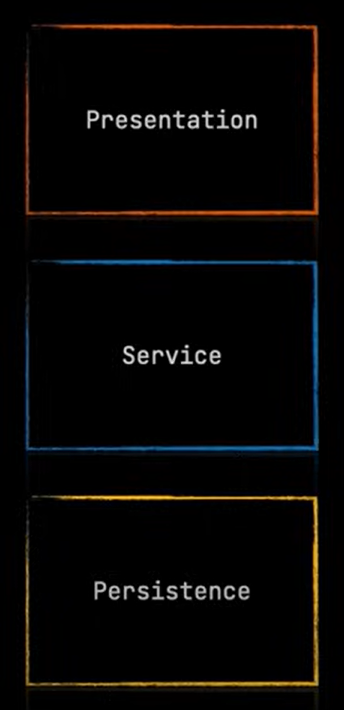
**Layers:**

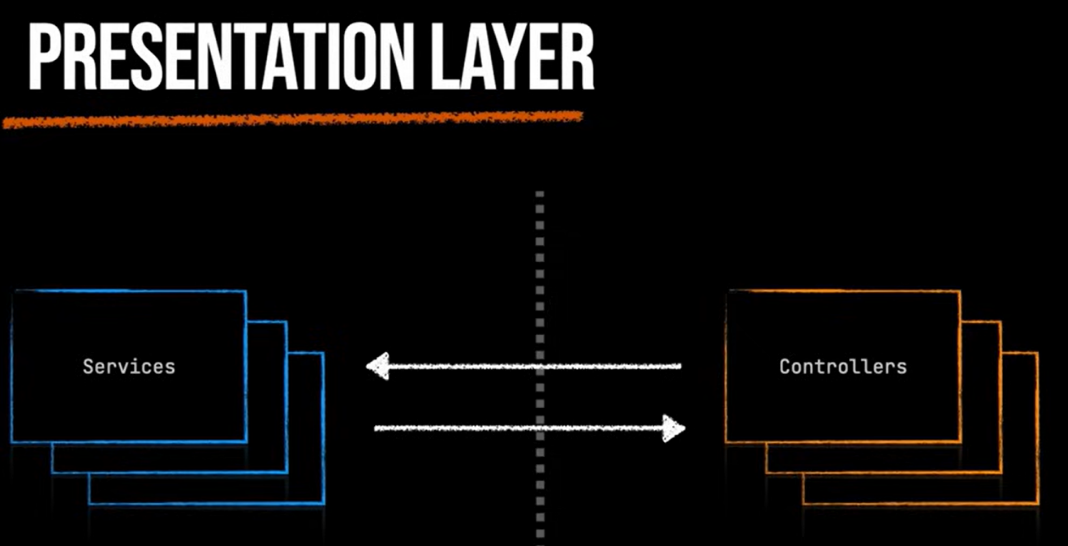
**The application code is broken up into various layers.**

Persistance layer: It is the layer which handles interactions with the database. The functionalities which are exposed by this layer are typically termed as CRUD.

We typically uses entities in this layer to interact with the database in one of the patterns. We have 2 patterns, which are repositories and DAO.

 Service Layer: The goal of this layer is to use all the functionalities exposed by the persistence layer to meet the business requirement. We use interfaces and classes in this layer. The functionalities in this layer can be as complicated as they can be. We never allow our presentation layer directly talk to the persistence layer.



The goal of this layer is to take all the data which comes as a result of using the service layer and present it to the user. This can be done by REST API for which we use the concept of controllers as an implementation of that.

Lets say we are moving from REST API to the GraphQL API, then we do not need to change anything in the service or persistence layer. Just making changes in the presentation layer is sufficient.

**Modularity:**

**Various dependencies which are used in spring and springboot:**

If we use spring, we need to create several beans manually including entity manager to manage the entites and transaction manager to manage the transactions and handful of other beans.

If we use springboot, then simply include the dependency and some configurations in the properties file which says how to connect to the database, and it will create all the needed beans for us automatically.

**Inversion of Control:**

**Spring IOC container is the core of spring framework. It creates the objects, configures and assemble their dependencies, manages the entire life cycle.**

Dependency Injection is the main functionality provided by Spring IOC(Inversion of Control). The Spring-Core module is responsible for injecting dependencies through either Constructor or Setter methods.

We can say that the spring framework is a dependency injection framework.

**Beans:**

As we came to know that the coding of interface instead of concrete classes is the way to do things.

And we also know that we should leave it upto the framework to supply the concrete classes wherever we declare our interfaces via dependency injection.

We can create beans using the config file in which we use @Bean annotation above a method to make it as a bean.

**@Component Annotation:**

This is the annotation which is to be added on top of a class which tells spring that the class is a bean which can be used by spring.

**@Service:**

This is the annotation which is used in the service layer which is used to label a particular class as a service as a bean and be able to inject this and have dependencies injected into this class.

**Component Scanning:**

**This is the annotation which tells the spring the start point to look for beans. From this point in the project hierarchy, look for beans and places that need beans.**

**@SpringBootApplication annotation has @ComponentScan+@Configuration+@EnableAutoConfiguration**

**@Configuration – Says spring that this class is a configuration class that is to be looked for beans during the component scan phase.**

**@EnableAutoConfiguration – IT IS THE PROCESS WHICH SPRINGBOOT USES TO PROVIDE THOSE SENSIBLE DEFAULTS.**

**SPRINGBOOT STARTERS ARE COLLECTION OF THOSE DEPENDENCIES WHICH ARE THERE IN ORDER TO SOLVE A PARTICULAR PROBLEM,** A starter dependency is nothing but a special dependency that aggregates commonly used dependencies for a particular feature.