**Springboot & Hibernate**

Spring Boot is a microservice-based framework and making a production-ready application in it takes very less time.

**Features of Spring Boot**

Spring Boot is built on the top of the conventional spring framework. So, it provides all the features of spring and is yet easier to use than spring.

**It allows to avoid heavy configuration of XML which is present in spring**

**It provides easy maintenance and creation of REST end points**

**It includes embedded Tomcat-server**

**Deployment is very easy, war and jar file can be easily deployed in the tomcat server**

## **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 a **CLI** tool for developing and testing the Spring Boot application.
* It offers the number of **plug-ins**.

# **Creating a Spring Boot Project**

Following are the steps to create a simple Spring Boot Project.

**Step 1:** Open the Spring initializr [https://start.spring.io](https://start.spring.io/).

**Step 2:** Provide the **Group** and **Artifact** name. We have provided Group name **com.javatpoint** and Artifact **spring-boot-example**.

**Step 3:** Now click on the **Generate** button.

**Step 4:** Extract the **RAR** file.

**Step 5:** **Import** the folder.

File -> Import -> Existing Maven Project -> Next -> Browse -> Select the project -> Finish

**Step 6:** Run the **SpringBootExampleApplication.java** file.

## **Core Spring Framework Annotations**

**@Required:** It applies to the **bean** setter method.

**@Autowired:** Spring provides annotation-based auto-wiring by providing @Autowired annotation. It is used to autowire spring bean on setter methods, instance variable, and constructor

**@Configuration:** It is a class-level annotation. The class annotated with @Configuration used by Spring Containers as a source of bean definitions.

**@ComponentScan:** It is used when we want to scan a package for beans. It is used with the annotation @Configuration. We can also specify the base packages to scan for Spring Components.

**@Bean:** It is a method-level annotation. It is an alternative of XML <bean> tag. It tells the method to produce a bean to be managed by Spring Container.

**Spring Boot Architecture**

To understand the architecture of Spring Boot, let us first see different layers and classes present in it.

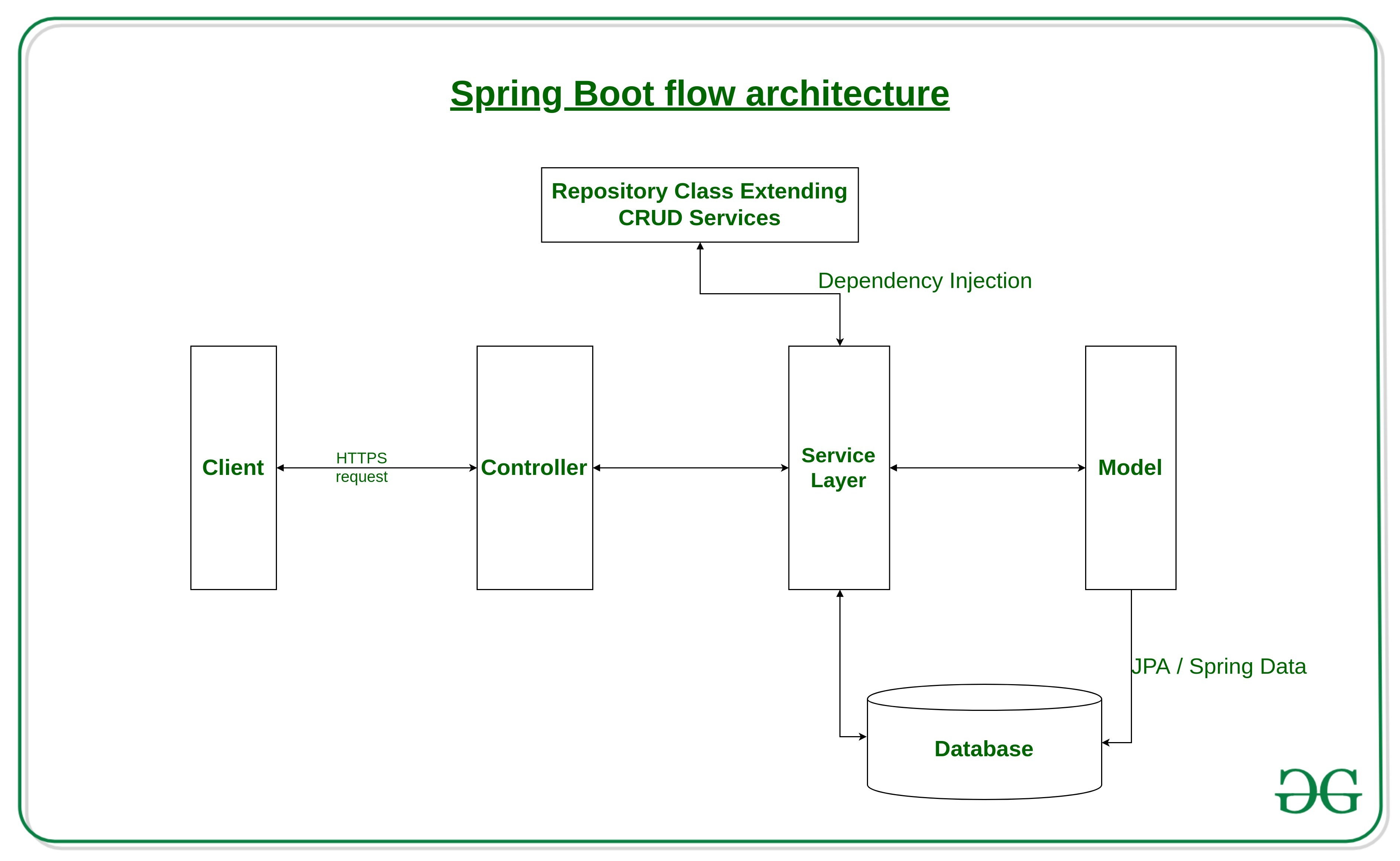
**Layers in Spring Boot:** There are four main layers in Spring Boot:

**Presentation Layer:** As the name suggests, it consists of views(i.e. frontend part)

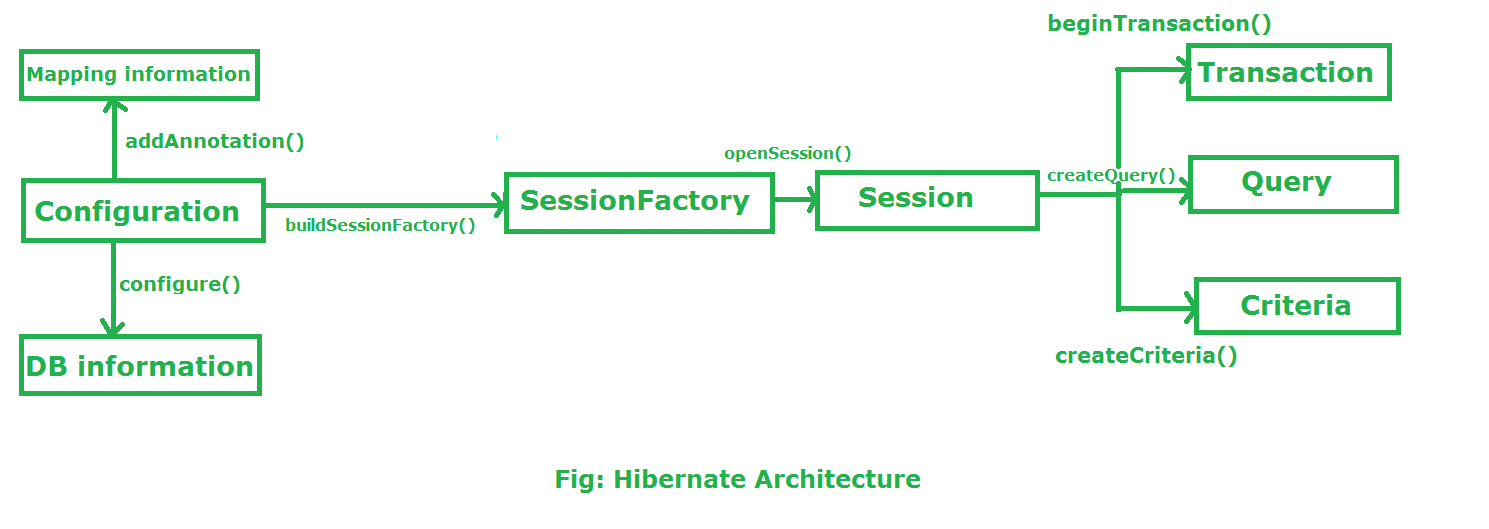
**Data Access Layer:** CRUD (create, retrieve, update, delete) operations on the database comes under this category.

**Service Layer:** This consist of service classes and uses services provided by data access layers.

**Integration Layer:** It consists of web different web services



**Hibernate:** Hibernate is a framework which is used to develop persistence logic which is independent of Database software. In JDBC to develop persistence logic we deal with primitive types. Whereas Hibernate framework we use Objects to develop persistence logic which are independent of database software.



**Functionalities supported by Hibernate framework**

* Hibernate framework support **Auto DDL** operations. In JDBC manually we have to create table and declare the data-type for each and every column. But Hibernate can do **DDL operations** for you internally like creation of table,drop a table,alter a table etc.
* Hibernate supports **Auto Primary key generation**. It means in JDBC we have to manually set a primary key for a table. But Hibernate can this task for you.
* Hibernate framework is independent of Database because it supports **HQL (Hibernate Query Language)** which is not specific to any database, whereas JDBC is database dependent.
* In Hibernate, **Exception Handling is not mandatory**, whereas In JDBC exception handling is mandatory.
* Hibernate supports **Cache Memory** whereas JDBC does not support cache memory.
* Hibernate is a **ORM tool** means it support Object relational mapping. Whereas JDBC is not object oriented moreover we are dealing with values means primitive data. In hibernate each record is represented as a Object but in JDBC each record is nothing but a data which is nothing but primitive values.

**Configuration:**

* Configuration is a class which is present in org.hibernate.cfg package. It activates Hibernate framework. It reads both configuration file and mapping files.
* It activate Hibernate Framework
* **Configuration cfg=new Configuration();**
* It read both cfg file and mapping files
* **cfg.configure();**
* It checks whether the config file is syntactically correct or not.
* If the config file is not valid then it will throw an exception. If it is valid then it creates a meta-data in memory and returns the meta-data to object to represent the config file.

**SessionFactory:**

* SessionFactory is an Interface which is present in org.hibernate package and it is used to create Session Object.
* It is immutable and thread-safe in nature.
* buildSessionFactory() method gathers the meta-data which is in the cfg Object.
* From cfg object it takes the JDBC information and create a JDBC Connection.

**SessionFactory factory=cfg.buildSessionFactory();**

**Session:**

* Session is an interface which is present in org.hibernate package. Session object is created based upon SessionFactory object i.e. factory.
* It opens the Connection/Session with Database software through Hibernate Framework.
* It is a light-weight object and it is not thread-safe.
* Session object is used to perform CRUD operations.

**Session session=factory.buildSession();**

**Transaction:**

* Transaction object is used whenever we perform any operation and based upon that operation there is some change in database.
* Transaction object is used to give the instruction to the database to make the changes that happen because of operation as a permanent by using commit() method.
* **Transaction tx=session.beginTransaction();**

**tx.commit();**

**Query:**

* Query is an interface that present inside org.hibernate package.
* A Query instance is obtained by calling Session.createQuery().
* This interface exposes some extra functionality beyond that provided by Session.iterate() and Session.find():
  1. A particular page of the result set may be selected by calling setMaxResults(), setFirstResult().
  2. Named query parameters may be used.

**Query query=session.createQuery();**

**Criteria:**

* Criteria is a simplified API for retrieving entities by composing Criterion objects.
* The Session is a factory for Criteria. Criterion instances are usually obtained via the factory methods on Restrictions.

**Criteria criteria=session.createCriteria();**

