**UNIT-1-Introduction to Spring Boot: Overview of Spring Framework and Spring Boot- History and evolution of Spring Boot, Benefits of using Spring Boot, Comparison with traditional Spring framework, Setting Up the Development Environment- Installation and setup of Java, Maven/Gradle, and IDE (IntelliJ/Eclipse), Creating a basic Spring Boot application Understanding Spring Boot starters and dependencies.**

**Spring Boot Project Structure: Explanation of key project files (pom.xml/build. Gradle, application. Properties/application), Main application class and @SpringBootApplication annotation.**

**Textbook 1: Chapter 1: 1.1 to 1.4**

**Introduction to Spring Boot and Overview of Spring Framework and Spring Boot**

**Spring Framework** is a powerful framework for building Java applications. It provides comprehensive infrastructure support for developing Java applications, allowing developers to focus on building their applications without getting bogged down by the underlying complexities. The core features include dependency injection, aspect-oriented programming, transaction management, and a vast array of modules for various purposes (e.g., Spring MVC for web applications).

**Spring Boot** is an extension of the Spring Framework that simplifies the process of setting up and developing Spring applications. It provides convention over configuration, allowing developers to create stand-alone, production-grade Spring applications quickly. Spring Boot eliminates the need for extensive configuration and boilerplate code, making it easier to get started with Spring.

**History and Evolution of Spring Boot**

Spring Boot was created by Pivotal Software (now part of VMware) and was first released in April 2014. It was developed to address the challenges developers faced when setting up Spring applications, particularly in terms of configuration and deployment. Over the years, Spring Boot has evolved significantly, adding features such as:

**- Auto-configuration:** Automatically configures Spring applications based on the dependencies on the classpath.

**- Embedded Servers:** Allows applications to run independently without needing an external server (e.g., Tomcat, Jetty).

**- Production-ready Features:** Includes metrics, health checks, and externalized configuration for building production-grade applications.

**Benefits of Using Spring Boot**

**1. Rapid Development:** Spring Boot enables quick setup and development of applications with minimal configuration.

**2. Convention over Configuration:** Reduces the need for extensive configuration, allowing developers to focus on business logic.

**3. Embedded Servers:** Simplifies deployment by allowing applications to run in an embedded server.

**4. Microservices Ready:** Designed with microservices architecture in mind, making it easy to build and deploy microservices.

**5. Rich Ecosystem:** Leverages the vast ecosystem of Spring projects and libraries.

**Comparison with Traditional Spring Framework**

**- Configuration:** Traditional Spring requires extensive XML or Java configuration, while Spring Boot uses sensible defaults and auto-configuration.

**- Setup Time:** Spring Boot reduces setup time significantly with starter dependencies and embedded servers.

**- Deployment:** Spring Boot applications can be packaged as executable JARs, whereas traditional Spring applications often require external servers for deployment.

**- Community and Support:** Both frameworks have strong community support, but Spring Boot has rapidly gained popularity due to its ease of use.

**Setting Up the Development Environment**

**1.Installation and Setup of Java:**

 - Ensure that you have JDK installed (Java Development Kit). You can download it from the [Oracle website](https://www.oracle.com/java/technologies/javase-downloads.html) or use OpenJDK.

**2. Maven/Gradle Setup:**

 **- Maven:** Download and install Maven from the [Apache Maven website](https://maven.apache.org/download.cgi). Set up the environment variable MAVEN\_HOME and add MAVEN\_HOME/bin to your PATH.

 **- Gradle:** Download and install Gradle from the [Gradle website](https://gradle.org/install/). Set up the environment variable GRADLE\_HOME and add GRADLE\_HOME/bin to your PATH.

**3. IDE Installation:**

 **- IntelliJ IDEA:** Download and install IntelliJ IDEA from the [JetBrains website](https://www.jetbrains.com/idea/download/). It has excellent support for Spring and Spring Boot.

 **- Eclipse:** Download and install Eclipse from the [Eclipse website](https://www.eclipse.org/downloads/). You may want to install the Spring Tools Suite (STS) plugin for better Spring support.

**Creating a Basic Spring Boot Application**

**1. Generate a Spring Boot Project:**

 - Use the [Spring Initializr](https://start.spring.io/) to generate a new Spring Boot project. Choose your preferred project metadata (Group, Artifact, Name, etc.), select dependencies (like Spring Web, Spring Data JPA), and click "Generate".

**2. Understanding Spring Boot Starters and Dependencies:**

 **- Starters:** They are a set of convenient dependency descriptors that you can include in your project. For example, spring-boot-starter-web includes everything needed to build a web application.

 **- Dependencies:** Spring Boot manages dependencies for you, allowing you to easily add functionality by including the appropriate starter or dependency in your pom.xml (for Maven) or build.gradle (for Gradle).

**Introduction to Spring Boot**

**1. Overview of Spring Framework and Spring Boot**

**Spring Framework:**

**- Core Concepts:**

 **- Dependency Injection (DI):** A design pattern that allows the creation of dependent objects outside of a class and provides those objects to a class through a constructor or setter.

 **- Aspect-Oriented Programming (AOP):** Allows separation of cross-cutting concerns (like logging and security) from the business logic.

 **- Spring MVC:** A web framework built on top of the Spring framework, providing a model-view-controller architecture.

**- Modules of Spring Framework:**

 - Core Container

 - Data Access/Integration

 - Web

 - AOP

 - Messaging

 - Testing

**Spring Boot:**

**- Purpose:** Designed to simplify the process of developing Spring applications by eliminating boilerplate code and configurations.

**- Key Features:**

 -Embedded Servers: Allows running applications as standalone Java applications with embedded servers like Tomcat or Jetty.

 - Spring Boot CLI: A command-line tool that enables you to run and test Spring Boot applications quickly.

 - Actuator Module: Provides production-ready features like monitoring, metrics, and health checks.

**2. History and Evolution of Spring Boot**

**- 2002:** Release of the Spring Framework, which changed the way Java developers approached application development.

**- 2013:** Introduction of Spring Boot, aimed at reducing the complexity of setting up a Spring application.

**- Key Releases:**

 - Spring Boot 1.x: Focused on simplifying the Spring application setup and introduced the concept of starters.

 - Spring Boot 2.x: Added support for reactive programming with WebFlux, enhanced actuator features, and improved compatibility with Spring Framework 5.

**- Current Trends:** Continuous improvement for cloud-native applications, integration with microservices architecture, and support for modern development practices like DevOps and CI/CD.

**3. Benefits of Using Spring Boot**

**- Rapid Development:** Streamlined setup process and reduced configuration time lead to quicker development cycles.

**- Microservices Ready:** Built with microservices architecture in mind, allowing easier deployment and scalability of applications.

**- Rich Ecosystem:** Leverages the existing Spring ecosystem, making it easier to integrate with other Spring projects like Spring Data, Spring Security, etc.

**- Community Support:** A large community of developers and extensive documentation provide robust support and resources for troubleshooting.

**- Customizable:** While it favors convention over configuration, Spring Boot still allows for extensive customization to meet specific project needs.

 **4. Comparison with Traditional Spring Framework**



**5. Setting Up the Development Environment**

**A. Installation and Setup:**

**1. Java:**

 - JDK Installation: Download the latest JDK from Oracle or OpenJDK.

 - Environment Variables: Set JAVA\_HOME to the JDK installation path and add the bin directory to the system PATH.

**2. Maven/Gradle:**

 **- Maven Installation:**

 - Download Apache Maven from the official website.

 - Set MAVEN\_HOME to the Maven installation path and add the bin directory to the system PATH.

 **- Gradle Installation:**

 - Download Gradle from the official website.

 - Set GRADLE\_HOME to the Gradle installation path and add the bin directory to the system PATH.

**3. IDE (IntelliJ/Eclipse):**

 **- IntelliJ IDEA:**

 - Download and install IntelliJ IDEA (Community or Ultimate).

 - Install the Spring Boot plugin via the Plugin Marketplace for enhanced Spring support.

 **- Eclipse:**

 - Download Spring Tools Suite (STS), which is an Eclipse-based IDE tailored for Spring development.

 - Ensure Maven or Gradle plugins are installed for dependency management.

**6. Creating a Basic Spring Boot Application**

**1. Project Setup:**

 - Using Spring Initializr:

 - Visit Spring Initializr (https://start.spring.io).

 - Choose project metadata (Group, Artifact, Name, etc.) and select dependencies like Spring Web, Spring Data JPA, etc.

 - Generate the project and import it into your IDE.

**2. Understanding Spring Boot Starters and Dependencies:**

 **- Starters:** Simplified dependency management. For example, spring-boot-starter-web includes everything needed for web applications (Spring MVC, embedded Tomcat, etc.).

 - Dependencies in pom.xml (for Maven) or build.gradle (for Gradle): Manage libraries required for the project. Spring Boot automatically configures beans based on the libraries present.

**3. Running the Application:**

 - The main application class is annotated with @SpringBootApplication, which is a combination of @Configuration, @EnableAutoConfiguration, and @ComponentScan.

 - Use mvn spring-boot:run (for Maven) or ./gradlew bootRun (for Gradle) to run the application.

 - Access the application via http://localhost:8080/ or the specified port.

**1. Overview of Spring Boot Project Structure**

A typical Spring Boot project has a well-defined structure that enhances organization and maintainability. The structure mainly revolves around the build tool chosen (Maven or Gradle) and follows conventions that make it easy to navigate.

**2. Key Project Files**

**A. pom.xml (for Maven projects)**

**- Purpose:** This is the Project Object Model file used by Maven to manage project dependencies, build configuration, and project information.

 **- Key Sections:**

 **- Project Coordinates:**

 *<groupId>com.example</groupId>*

 *<artifactId>demo</artifactId>*

 *<version>0.0.1-SNAPSHOT</version>*

 **- groupId:** Unique identifier for the group of the project.

  **- artifactId:** Name of the project.

  **- version:** Version of the artifact.

 **- Dependencies:**

 <dependencies>

 <dependency>

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

 <artifactId>spring-boot-starter-web</artifactId>

 </dependency>

 <dependency>

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

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

 </dependency>

 <dependency>

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

 <artifactId>spring-boot-starter-test</artifactId>

 <scope>test</scope>

 </dependency>

 </dependencies>

 **- Dependency Management:** Manages libraries required for the project. Spring Boot starters simplify this by including a set of dependencies needed for common functionalities.

 **- Build Configuration:**

 <build>

 <plugins>

 <plugin>

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

 <artifactId>spring-boot-maven-plugin</artifactId>

 <version>2.5.4</version>

 <executions>

 <execution>

 <goals>

 <goal>repackage</goal>

 </goals>

 </execution>

 </executions>

 </plugin>

 </plugins>

 </build>

 **- Plugins:** Specifies how to build the project. The spring-boot-maven-plugin allows packaging the application as an executable JAR.

**B. build.gradle (for Gradle projects)**

**- Purpose:** This file is used by Gradle as a build configuration file where dependencies and plugins are defined.

**- Key Sections:**

 **- Plugins:**

 *plugins {*

 *id 'org.springframework.boot' version '2.5.4'*

 *id 'io.spring.dependency-management' version '1.0.11.RELEASE'*

 *id 'java'*

 *}*

 **- Spring Boot Plugin:** Enables Spring Boot support in the Gradle project.

 **- Dependencies:**

 *dependencies {*

 *implementation 'org.springframework.boot:spring-boot-starter-web'*

 *implementation 'org.springframework.boot:spring-boot-starter-data-jpa'*

 *testImplementation 'org.springframework.boot:spring-boot-starter-test'*

 *}*

 - Similar to Maven, this section defines the libraries required for the project.

 **- Spring Boot Configuration:**

 *bootJar {*

 *mainClassName = 'com.example.demo.DemoApplication'*

 *}*

 - Specifies the main class for the Spring Boot application.

**C. application.properties or application.yml**

**- Purpose:** Configuration file for Spring Boot applications. It is used to set various application properties.

**- Common Properties:**

 **- Server Port:**

 server.port=8081

 - Changes the default server port from 8080 to 8081.

 **- Database Configuration:**

 *spring.datasource.url=jdbc:mysql://localhost:3306/mydb*

 *spring.datasource.username=root*

 *spring.datasource.password=secret*

 *spring.jpa.hibernate.ddl-auto=update*

 - Configures the database connection settings.

 **- Logging Configuration:**

 *logging.level.root=INFO*

 *logging.file.name=app.log*

 - Sets logging levels and log file names.

**- YAML Format:**

 - Alternatively, you can use application.yml for a more structured format:

 *server:*

 *port: 8081*

 *spring:*

 *datasource:*

 *url: jdbc:mysql://localhost:3306/mydb*

 *username: root*

 *password: secret*

 *jpa:*

 *hibernate:*

 *ddl-auto: update*

 *logging:*

 *level:*

 *root: INFO*

 *file:*

 *name: app.log*

**3. Main Application Class and @SpringBootApplication Annotation**

**A. Main Application Class**

**- Purpose:** The entry point for the Spring Boot application. It contains the main method which is executed to start the application.

**- Structure:**

 *package com.example.demo;*

 *import org.springframework.boot.SpringApplication;*

 *import org.springframework.boot.autoconfigure.SpringBootApplication;*

 *@SpringBootApplication*

 *public class DemoApplication {*

 *public static void main(String[] args) {*

 *SpringApplication.run(DemoApplication.class, args);*

 *}*

 *}*

**B. @SpringBootApplication Annotation**

**- Description:** A convenience annotation that combines three key annotations:

 **- @Configuration:** Indicates that the class can be used by the Spring IoC container as a source of bean definitions.

 **- @EnableAutoConfiguration:** Enables Spring Boot’s auto-configuration mechanism, which attempts to automatically configure your Spring application based on the dependencies present on the classpath.

 **- @ComponentScan:** Enables component scanning, allowing Spring to discover and register beans in the specified package and its sub-packages.

**- Benefits:**

 - Reduces boilerplate code and configuration.

 - Simplifies the setup process for new Spring applications.

 - Ensures that the application is configured correctly based on the included dependencies.

**Suggested Reading/Resources**

**- Books:**

 - "Spring in Action" by Craig Walls

 - "Spring Boot in Action" by Craig Walls

 **- Online Resources:**

 - Official Spring Boot Documentation:

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

- Online tutorials on Maven and Gradle for deeper understanding.

- "Spring Boot in Action" by Craig Walls for practical insights into project structure and configuration.

**- Courses:**

 - Udemy: Spring & Hibernate for Beginners (includes Spring Boot)

 - Coursera: Spring Framework Specialization

**Lecture Plan: Teaching Spring Boot REST API Development**

This lecture is designed for students who are **new to Spring Boot** and REST API development. The goal is to make them understand **what Spring Boot is, how it works, and how to create their first REST API**. The session will include both **theory and hands-on exercises**.

**📌 1: Introduction to Spring Boot**

**🔹 What is Spring Boot?**

* Spring Boot is a **framework** that simplifies Java-based web application development.
* It is built on top of the **Spring Framework** and provides **default configurations**.
* It helps developers **create stand-alone, production-ready applications** with minimal setup.
* **No need for XML configuration!** Spring Boot uses **annotations** instead.

**🔹 Why Do We Use Spring Boot for REST APIs?**

* **Simplifies API development** with built-in tools.
* Comes with an **embedded server** (Tomcat/Jetty).
* Has built-in support for **JSON** responses.
* **Automatic dependency management** (using Maven or Gradle).
* **Microservices-friendly** (can be used with Spring Cloud).

**💡 Real-World Example**

Imagine building a **weather app** that fetches live weather data from a server. The **backend** (Spring Boot) provides REST APIs like:

**GET /api/weather?city=Bangalore**

**Response:**

{

 "city": "Bangalore",

 "temperature": "30°C",

 "humidity": "65%"

}

**REST API, or Representational State Transfer API, is a type of application programming interface (API) that allows different applications to communicate with each other. REST APIs are often used for mobile app development and the Internet of Things (IoT).**

**How does REST API work?**

* REST APIs use HTTP requests to communicate with web services.
* They follow a stateless protocol, where each request is treated independently.
* Clients can access and manipulate resources using standard HTTP methods like GET, POST, PUT, and DELETE.
* REST APIs are designed to support high-performing and reliable communication at scale.

**Why is REST API popular?**

* REST APIs are considered easier to use than other protocols like SOAP (Simple Object Access Protocol).
* They are faster and more lightweight, with increased scalablity.
* They provide a flexible, lightweight way to integrate applications.

**Examples of REST APIs in use**

* Twitter's REST API provides developers with access to tweet data.
* Instagram's API provides endpoints for interaction with photos, users, and analytics.

**📌 2: Understanding the Project Structure**

When we create a **Spring Boot project**, it has a **default folder structure**:

****

****

**📌 3: Understanding SpringBootDemoApplication.java**





**📌 4: Understanding HelloController.java**

****

****

**🔹 Hands-On Exercise**

* Run the program and **test the API** in a **browser**:

http://localhost:8080/api/hello

* Expected Output:

**Hello, Spring Boot!**

**📌 5: Hands-On - Running the Project in Eclipse**

**🔹 Step 1: Install Required Software**

* **Install Java JDK 17 or later**
* **Download Eclipse IDE**
* **Install Spring Boot Plugin (STS - Spring Tool Suite)**

**🔹 Step 2: Create a New Spring Boot Project**

1. **Go to File → New → Spring Starter Project**.
2. Enter:
	* **Project Name:** SpringBootDemo
	* **Group:** com.example
	* **Artifact:** demo
3. Select **Spring Web**, then click **Finish**.

**🔹 Step 3: Add Code**

* Inside src/main/java/com/example/demo/controller/, create a new Java class HelloController.java and add the code.

**🔹 Step 4: Run the Application**

* Right-click SpringBootDemoApplication.java → **Run As → Spring Boot App**.

**📌 6: Assign Practical Tasks**

**Task 1: Change the Endpoint**

* Modify the controller to use /api/greet instead of /api/hello.
* **Expected URL:** http://localhost:8080/api/greet
* **Expected Output:** "Hello, Spring Boot!"

**Task 2: Return a JSON Response**

* Modify HelloController.java to return a **JSON response** instead of plain text.



**Expected Output (JSON)**:

**{**

 **"message": "Hello, Spring Boot!"**

**}**

**Task 3: Introduce Query Parameters**

* Modify the API to accept **a user’s name** and return a personalized message.

**@GetMapping("/hello")**

**public Map<String, String> sayHello(@RequestParam(defaultValue = "Guest") String name) {**

 **return Map.of("message", "Hello, " + name + "!");**

**}**

* **Test with URL**:

http://localhost:8080/api/hello?name=DSU

* **Expected Output**:

{

 "message": "Hello, DSU!"

}

**🎯 Conclusion**

✅ Students will now understand:

1. What **Spring Boot** is and why we use it.
2. How to create a **REST API**.
3. How to run the project and **test endpoints**.
4. How to **modify APIs** to return JSON responses.
5. How to **handle query parameters**.