**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**.