Lernskript: Spring Boot & REST-APIs

Spring Boot ist eine Erweiterung des Spring Frameworks, die dessen Nutzung radikal vereinfacht. Sein Kernprinzip ist **"Convention over Configuration"** (Konvention vor Konfiguration).

Teil 1: Die grundlegende Analogie - Das "Warum"

- **Konzept:** Anstatt eine Anwendung aus hunderten Einzelteilen manuell zu konfigurieren (klassisches Spring), agiert Spring Boot wie ein **intelligenter Generalunternehmer**. Man sagt ihm nur, *was* man bauen will (z.B. eine Web-Anwendung), und es liefert ein fertig konfiguriertes Grundgerüst mit sinnvollen Standardeinstellungen.
- **Hauptvorteile:**
    1. **Autokonfiguration:** Spring Boot erkennt die verwendeten Bibliotheken ("Starter") und konfiguriert sie automatisch.
    2. **Eingebetteter Server:** Es verpackt die Anwendung in eine einzige, ausführbare `.jar`-Datei, die einen eigenen Webserver (z.B. Tomcat) bereits enthält.

Teil 2: Praktische Anwendungsfälle - Das "Wofür"

Use Case 1: Eine minimale Web-Anwendung

Mit Spring Boot kann eine lauffähige Web-Anwendung in nur zwei Klassen erstellt werden.

- **Die Hauptklasse:**

```
@SpringBootApplication // Aktiviert die Autokonfiguration und startet alles
public class MeineWebApp {
    public static void main(String[] args) {
        SpringApplication.run(MeineWebApp.class, args);
    }
}


  Der Controller: Eine Klasse, die auf Web-Anfragen lauscht.

  @RestController // Standard für REST-APIs
  public class HalloWeltController {
      @GetMapping("/hallo") // Ordnet diese Methode der URL "/hallo" zu
      public String sagHallo() {
          return "Hallo, JavaRoad!";
      }
  }
```

Use Case 2: Eine REST-API für Datenobjekte

```
Problem: Andere Systeme (z.B. ein Frontend) müssen Daten im standardisierten JSON-Format abfragen können.

Lösung: Ein @RestController, der direkt Java-Objekte (wie List<Benutzer>) zurückgibt. Spring Boot wandelt diese Objekte automatisch in JSON um.

Wichtige Annotationen:

    @RequestMapping("/api/basis-pfad"): Definiert eine gemeinsame Basis-URL für alle Methoden in der Klasse.

    @GetMapping("/{id}"): Reagiert auf HTTP GET-Anfragen an einen Pfad mit einer variablen ID.

    @PathVariable int id: Nimmt den Wert aus dem {id}-Platzhalter der URL und übergibt ihn als Methodenparameter.
```

Teil 3: Vertiefung (JavaMasta's Profi-Tipps)

```
@RestController vs. @Controller:

    @Controller: Klassisch, für Web-Anwendungen, die serverseitig gerendertes HTML zurückgeben (z.B. mit Thymeleaf).

    @RestController: Modern, für REST-APIs. Fügt automatisch @ResponseBody hinzu, was die Rückgabewerte direkt in den HTTP-Antwortkörper (meist als JSON) serialisiert.

Die HTTP-Verben: Für jede Standard-HTTP-Aktion gibt es eine passende Annotation:

    @GetMapping: Daten lesen (Read).

    @PostMapping: Neue Daten erstellen (Create). Daten werden oft mit @RequestBody aus dem Anfragekörper gelesen.

    @PutMapping: Bestehende Daten vollständig aktualisieren (Update).

    @DeleteMapping: Daten löschen (Delete).

Spring Boot "Starters": Das Geheimnis der Autokonfiguration liegt in den "Startern" in der pom.xml. Wenn man spring-boot-starter-web hinzufügt, weiß Spring Boot, dass es einen Webserver, JSON-Bibliotheken und alles für REST-Controller vorkonfigurieren muss.
```