Complete Spring Boot Guide: From Zero to CRUD API

1. Java Basics (What You Need to Know)

What is Java?

Java is a programming language that runs on any computer. Think of it like English - once you write something in Java, any computer that "speaks Java" can understand it.

Key Java Concepts:

Classes and Objects

```java
// A Class is like a blueprint for making things
public class Car {
    String color;
    String brand;

    // Constructor - how to create a new car
    public Car(String color, String brand) {
        this.color = color;
        this.brand = brand;
    }
}

// An Object is the actual thing made from the blueprint
Car myCar = new Car("Red", "Toyota");
```

Methods (Functions)

```java
// Methods do things - like actions
public String getCarInfo() {
    return "This is a " + color + " " + brand;
}
```

Annotations

```java
```

```
@Override  // This @ symbol means "annotation" - it gives instructions to Java
public String toString() {
    return "Car details";
}
```

2. What is Spring Framework?

Spring is like a toolbox that makes building Java applications much easier. Imagine you're building a house - instead of making every nail and screw yourself, Spring gives you pre-made tools.

Key Spring Concepts:

Dependency Injection: Spring automatically gives your code the things it needs

```
java

// Instead of creating objects manually:
UserService service = new UserService(new UserRepository());

// Spring does it automatically:
@Autowired
UserService service;  // Spring finds and injects this for you
```

IoC (Inversion of Control): You don't control object creation - Spring does it for you

3. What is Spring Boot?

Spring Boot is Spring made super easy. It's like having a smart assistant that sets up everything for you automatically.

Regular Spring: You have to configure everything manually (like setting up a stereo system wire by wire)
Spring Boot: Everything works out of the box (like a pre-configured smart speaker)

Spring Boot Features:

- **Auto-configuration**: Sets up everything automatically
- **Embedded server**: No need to install separate web servers
- **Starter dependencies**: Pre-packaged sets of libraries
- **Production-ready**: Built-in health checks, monitoring

4. Understanding REST APIs

API: Application Programming Interface - a way for programs to talk to each other **REST**: A style of building APIs using HTTP (web) requests

HTTP Methods:

- **GET**: Retrieve data (like asking for information)
- **POST**: Create new data (like submitting a form)
- **PUT**: Update existing data (like editing a document)
- **DELETE**: Remove data (like deleting a file)

Example:

- GET /users → Get all users
- GET /users/1 → Get user with ID 1
- POST /users → Create a new user
- PUT /users/1 → Update user with ID 1
- DELETE /users/1 → Delete user with ID 1

5. Setting Up Spring Boot Project

Project Structure:

```
my-project/
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   └── com/example/demo/
│   │   │       ├── DemoApplication.java
│   │   │       ├── controller/
│   │   │       ├── model/
│   │   │       ├── repository/
│   │   │       └── service/
│   │   └── resources/
│   │       └── application.properties
└── pom.xml (dependencies list)
```

6. Building a CRUD REST API - Step by Step

Let's build a simple User management system!

Step 1: Main Application Class

java

```java
@SpringBootApplication // This makes it a Spring Boot app
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

Step 2: Create User Model (Data Structure)

java

```java
@Entity // This means it's a database table
@Table(name = "users")
public class User {
    @Id // Primary key
    @GeneratedValue(strategy = GenerationType.IDENTITY) // Auto-increment
    private Long id;

    @Column(name = "name")
    private String name;

    @Column(name = "email")
    private String email;

    // Constructors
    public User() {} // Empty constructor (required)

    public User(String name, String email) {
        this.name = name;
        this.email = email;
    }

    // Getters and Setters (ways to access/modify data)
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
}
```

Step 3: Create Repository (Database Access)

java

```java
@Repository  // This handles database operations
public interface UserRepository extends JpaRepository<User, Long> {
    // JpaRepository gives you free methods like:
    // save(), findAll(), findById(), deleteById()

    // You can add custom methods:
    List<User> findByName(String name);
    Optional<User> findByEmail(String email);
}
```

Step 4: Create Service (Business Logic)

java

```java
@Service  // This contains business logic
public class UserService {

    @Autowired  // Spring automatically provides this
    private UserRepository userRepository;

    // Get all users
    public List<User> getAllUsers() {
        return userRepository.findAll();
    }

    // Get user by ID
    public Optional<User> getUserById(Long id) {
        return userRepository.findById(id);
    }

    // Create new user
    public User createUser(User user) {
        return userRepository.save(user);
    }

    // Update user
    public User updateUser(Long id, User userDetails) {
        User user = userRepository.findById(id)
            .orElseThrow(() -> new RuntimeException("User not found"));

        user.setName(userDetails.getName());
        user.setEmail(userDetails.getEmail());

        return userRepository.save(user);
    }

    // Delete user
    public void deleteUser(Long id) {
        userRepository.deleteById(id);
    }
}
```

Step 5: Create Controller (API Endpoints)

```java
```

```java
@RestController  // This handles HTTP requests
@RequestMapping("/api/users")  // Base URL path
public class UserController {

    @Autowired
    private UserService userService;

    // GET /api/users - Get all users
    @GetMapping
    public List<User> getAllUsers() {
        return userService.getAllUsers();
    }

    // GET /api/users/{id} - Get user by ID
    @GetMapping("/{id}")
    public ResponseEntity<User> getUserById(@PathVariable Long id) {
        Optional<User> user = userService.getUserById(id);
        return user.map(ResponseEntity::ok)
                .orElse(ResponseEntity.notFound().build());
    }

    // POST /api/users - Create new user
    @PostMapping
    public User createUser(@RequestBody User user) {
        return userService.createUser(user);
    }

    // PUT /api/users/{id} - Update user
    @PutMapping("/{id}")
    public ResponseEntity<User> updateUser(@PathVariable Long id,
                                           @RequestBody User userDetails) {
        try {
            User updatedUser = userService.updateUser(id, userDetails);
            return ResponseEntity.ok(updatedUser);
        } catch (RuntimeException e) {
            return ResponseEntity.notFound().build();
        }
    }

    // DELETE /api/users/{id} - Delete user
    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteUser(@PathVariable Long id) {
        userService.deleteUser(id);
        return ResponseEntity.ok().build();
```

```
    }
}
```

Step 6: Configuration (application.properties)

properties

```
# Database configuration (using H2 in-memory database for simplicity)
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.username=sa
spring.datasource.password=password
spring.h2.console.enabled=true

# JPA configuration
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
```

7. Key Spring Boot Annotations Explained

Annotation	Purpose	Where to Use
@SpringBootApplication	Makes it a Spring Boot app	Main class
@RestController	Handles HTTP requests	Controller classes
@Service	Contains business logic	Service classes
@Repository	Handles database operations	Repository classes
@Entity	Marks as database table	Model classes
@Autowired	Automatic dependency injection	Any class
@GetMapping	Handle GET requests	Controller methods
@PostMapping	Handle POST requests	Controller methods
@PutMapping	Handle PUT requests	Controller methods
@DeleteMapping	Handle DELETE requests	Controller methods
@PathVariable	Extract URL parameters	Method parameters
@RequestBody	Extract request body data	Method parameters

8. Testing Your API

Once your application runs (usually on http://localhost:8080), you can test:

Get all users: GET http://localhost:8080/api/users **Create user**: POST http://localhost:8080/api/users

json

```
{
  "name": "John Doe",
  "email": "john@example.com"
}
```

9. Common Interview Questions & Answers

Q: What is Spring Boot? A: Spring Boot is a framework that makes building Spring applications easier by providing auto-configuration, embedded servers, and starter dependencies.

Q: What is dependency injection? A: It's when Spring automatically provides objects that your code needs, instead of you creating them manually.

Q: Difference between @Component, @Service, @Repository? A: They're all similar - they tell Spring to manage these classes. @Service is for business logic, @Repository is for data access, @Component is generic.

Q: What is @RestController? A: It combines @Controller and @ResponseBody - it handles HTTP requests and automatically converts responses to JSON.

Q: What is JPA? A: Java Persistence API - it's a way to work with databases using Java objects instead of SQL queries.

10. Key Concepts Summary

- **Spring Boot**: Framework for easy Java web applications
- **REST API**: Web service that uses HTTP methods (GET, POST, PUT, DELETE)
- **CRUD**: Create, Read, Update, Delete operations
- **MVC Pattern**: Model (data), View (presentation), Controller (handles requests)
- **Dependency Injection**: Spring automatically provides dependencies
- **Annotations**: Instructions that tell Spring how to handle your classes
- **JPA/Hibernate**: Tools for working with databases

11. Project Dependencies (pom.xml)

```xml
```

```
<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>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
  </dependency>
</dependencies>
```

This guide covers everything you need to understand Spring Boot and build a CRUD API. Practice building this example and you'll be ready for your interview!