INTRODUCTION TO SPRING BOOT WITH JAVA

Simplifying Java Application Development

AGENDA

What is Spring Boot
Setting up Spring Boot
Creating Basic Spring Boot Application
REST APIs in Spring Boot
Running & Testing Applications

Spring Framework

- What is Spring?
  - Spring is a comprehensive framework for Java that provides a powerful ecosystem to develop complex applications, especially in web and enterprise environments
  - Initially focused on simplifying Java Enterprise Edition development, **Spring** now offers a range of projects, from **Spring Boot** to **Spring Cloud** for microservices and **Spring Security** for application security
- Spring Framework GitHub Repository
- The official website with documentation: Spring.io

Spring Boot

- What is Spring Boot?
  - A Java-based framework for creating stand-alone, production-ready applications
  - Built on top of the **Spring** framework
  - Streamlines application configuration and deployment

Why Use Spring Boot?

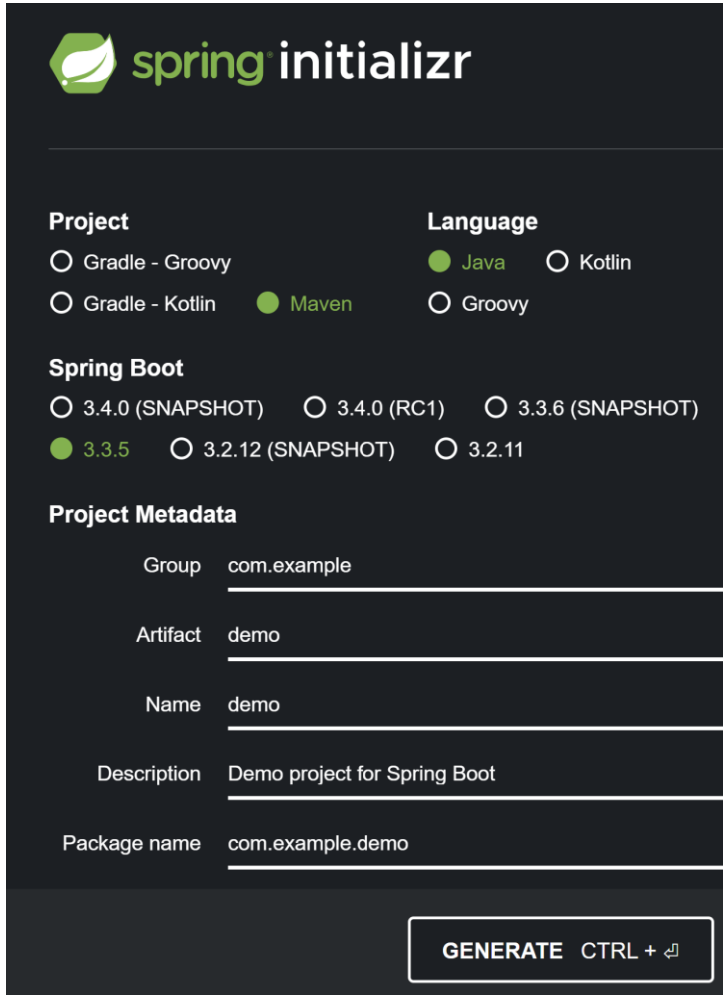
- Simplified dependency management with **auto-configuration**
- **Embedded servers** (Tomcat, Jetty) for easy deployment
- **Production-ready features** (e.g., health checks, metrics)
- Consistent, modern development approach for **microservices**

Key Features of Spring Boot

- **Auto-configuration**: Sets up common configurations automatically
- **Embedded Server**: Tomcat by default, but others are available
- **Spring Boot CLI**: Command-line tool for rapid prototyping
- **Spring Initializer**: Web-based tool for quick project setup

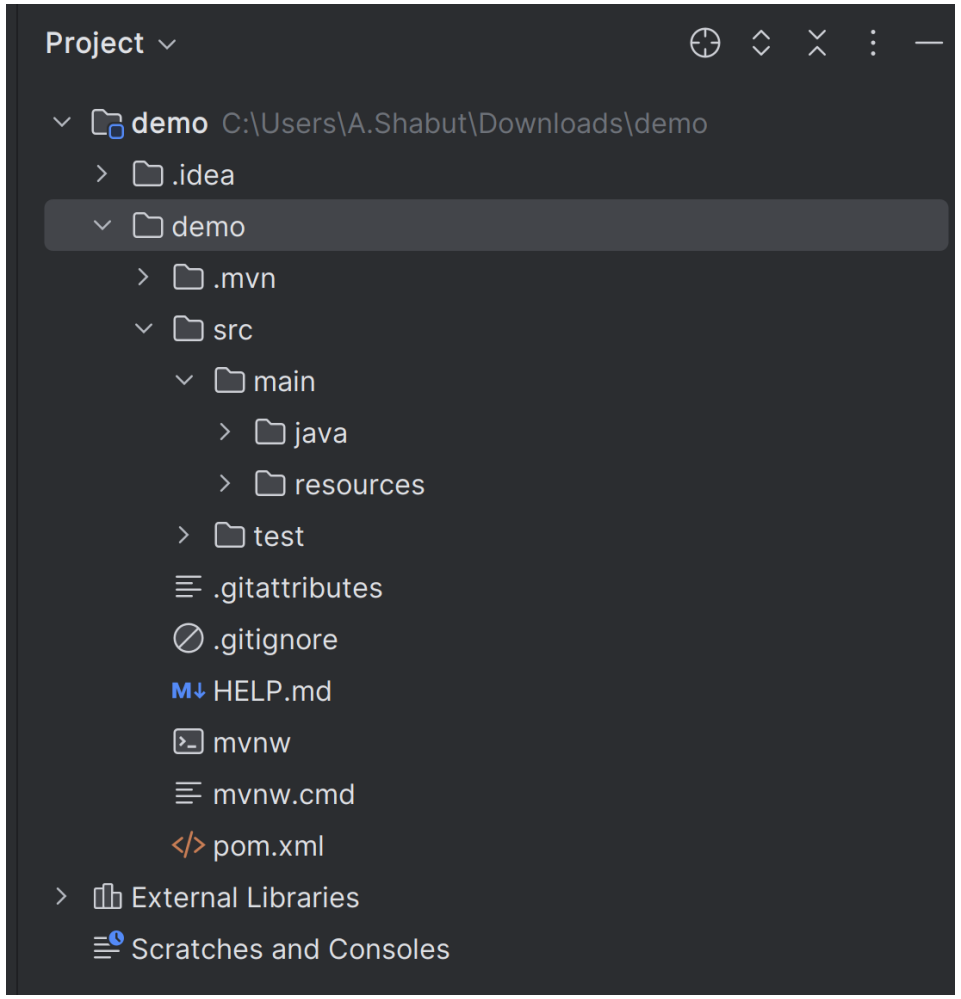
Setting Up Spring Boot

- Downloading and installing Java SDK (version 11+)
- Using Spring Initializer for project creation
  - Go to https://start.spring.io
  - Select Java, Maven or Gradle, and relevant dependencies
  - Example dependencies: Spring Web, Spring Data JPA, H2 Database



Project Structure Overview

- Overview of Maven/Gradle project layout
  - **src/main/java**: Contains Java source files
  - **src/main/resources**: Stores application properties, templates
  - **pom.xml** or **build.gradle**: Dependency management



Creating a Basic Spring Boot Application

- Demonstrate creating a main class with @SpringBootApplication
  - @SpringBootApplication annotation combines:
    - @Configuration
    - @EnableAutoConfiguration
    - @ComponentScan

Code Example

```
@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Dependency Injection in Spring Boot

- Explanation of Inversion of Control (IoC) and Dependency Injection
- Using @Autowired for dependency injection
- Benefits of loose coupling and testability

Creating REST APIs in Spring Boot

- Overview of @RestController and @RequestMapping annotations
- Code example

```
@RestController
public class HelloController {
    @GetMapping("/hello")
    public String hello() {
        return "Hello, Spring Boot!";
    }
}
```

Connecting to a Database with Spring Data JPA

- Spring Data JPA is used for database access
  - Spring Data JPA is a part of the larger Spring Data family,
  - Designed to make it easier to work with Java Persistence API (JPA)
  - Managing relational data in Java applications, providing an abstraction over data persistence
- Steps to configure JPA:
  - Add dependency: spring-boot-starter-data-jpa
  - Configure database settings in application.properties

```
@Entity
public class User {
    @Id
    @GeneratedValue
    private Long id;
    private String name;
}
```

Configuring Application Properties

- Overview of application.properties or application.yml
  - Common properties:
    - Server port (server.port)
    - Database connection settings (spring.datasource.*)
    - Logging levels (logging.level.*)

Testing in Spring Boot

- Unit testing with JUnit and Spring Boot's test utilities
- Writing simple tests for REST controllers
  - @SpringBootTest, @WebMvcTest, and @MockBean
- Example:

```
@SpringBootTest
public class HelloControllerTest {
    @Autowired
    private MockMvc mockMvc;

    @Test
    public void testHello() throws Exception {
        mockMvc.perform(get("/hello"))
            .andExpect(status().isOk())
            .andExpect(content().string("Hello, Spring Boot!"));
    }
}
```

Building and Running the Application

- Running Spring Boot from an IDE or command line
  - mvn spring-boot:run or ./mvnw spring-boot:run
  - Packaging the app as a JAR file with mvn package
- Deploying a JAR file to production environments

Monitoring and Metrics in Spring Boot

- Overview of Spring Boot Actuator
  - Adds health checks, metrics, and more
- Example endpoints:
  - /actuator/health
    - Shows the current health status of the application such as UP, DOWN, OUT_OF_SERVICE
  - /actuator/metrics
    - Provides a range of system and application metrics such as CPU usage, HTTP requests

Monitoring and Metrics in Spring Boot

- Importance of monitoring for production applications
  - Early Detection of Issues
  - Improving User Experience
  - Ensuring Security and Compliance
  - Capacity Planning and Scalability
  - Support for Continuous Improvement
  - Compliance and Accountability

Recap and Best Practices

- Key takeaways:
  - Spring Boot simplifies Java application development
  - Leverages Spring ecosystem for robust, scalable applications
  - Emphasize best practices in modular design, testing, and configuration management

Examples

- Follow the tutorial here to **Build your First Boot Spring Application:** [Getting Started | Building an Application with Spring Boot]