Spring Boot

- 1. Introduction to Spring Framework and Core Modules
 - Overview of Spring Framework
 - The Spring Framework is a comprehensive, modular, and open-source framework for building enterprise-level Java applications. It was designed to simplify Java development by providing a comprehensive programming and configuration model, promoting best practices such as dependency injection, aspect-oriented programming, transaction management, and more.
 - Key Features of Spring Framework
 - Inversion of Control (IoC) / Dependency Injection (DI)
 - Facilitates loose coupling between components.
 - Aspect-Oriented Programming (AOP)
 - Enables modularization of cross-cutting concerns like logging and security.
 - Transaction Management
 - Simplifies transaction handling for database operations.
 - o Model-View-Controller (MVC) Framework
 - Supports building web applications.
 - o Integration Support
 - Provides integration with various data access, messaging, and web services.
 - Core Modules of Spring Framework
 - o Spring is modular, with each module providing specific functionality. The core modules are:
 - i Core Container
 - o Spring Core ('spring-core'): Provides fundamental parts of the framework, including the IoC and DI features.
 - Spring Beans ('spring-beans'): Contains the BeanFactory and ApplicationContext implementations for bean management.
 - Spring Context ('spring-context'): Extends the core container and provides context-aware features like resource loading and event propagation.
 - Spring Expression Language (SpEL) ('spring-expression'): Supports powerful expression language for querying and manipulating objects at runtime.
 - ii. Data Access/Integration
 - o JDBC ('spring-jdbc'): Simplifies JDBC operations and exception handling.
 - ORM modules ('spring-orm'): Integrates with ORM frameworks like Hibernate, JPA, JDO, etc.
 - o Transaction Management ('spring-tx'): Provides consistent programming model for transaction handling, abstracting underlying transaction APIs.
 - iii. Web
 - Web ('spring-web'): Basic web integration features.
 - Web MVC ('spring-webmvc'): Implements the Model-View-Controller pattern for building web applications.
 - o WebSocket, WebFlux: For reactive and real-time web applications.
 - iv. AOP and Instrumentation
 - o Spring AOP ('spring-aop'): Provides aspect-oriented programming support.
 - Aspects ('spring-aspects'): Supports AspectJ integration.
 - v. Testing
 - Spring Test ('spring-test'): Provides testing support with mock objects and testing utilities.

Summary

The Spring Framework's core modules provide a robust foundation for Java application development, emphasizing modularity, flexibility, and ease of integration. Whether you're building a simple standalone app or a complex enterprise system, Spring's core modules facilitate rapid development, maintainability, and scalability.

In brief

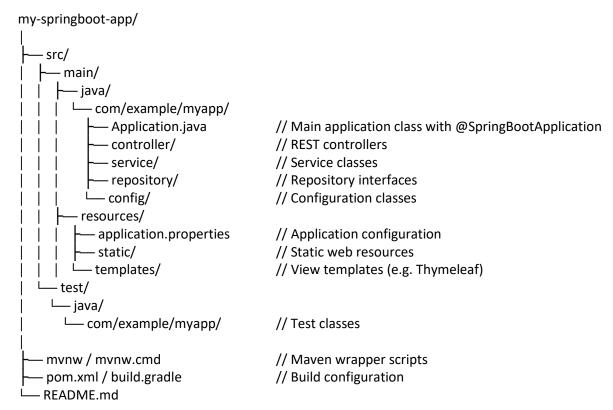
- Spring Core handles fundamental features like IoC and DI.
- Additional modules extend functionality for data access, web development, AOP, security, and more.
- The framework promotes best practices and simplifies complex enterprise development tasks.

2. Need of Spring Boot and its Advantages

- Spring Boot simplifies the development of Spring-based applications by providing a "starter" approach. It automatically configures common dependencies and eliminates much of the boilerplate code needed for a basic Spring application. This drastically reduces development time and effort.
- Advantages
 - Faster development: Reduced setup time and configuration.
 - Simplified dependency management: Automatic configuration of common dependencies.
 - Easy deployment: Out-of-the-box support for various deployment environments.
 - Small footprint: Lightweight and efficient.
 - Modern features: Integrates well with modern development practices (e.g., REST APIs, microservices).
 - Strong community support: Extensive documentation and readily available support resources.

3. Configuration: Spring Boot vs. traditional Spring

- Spring Boot simplifies Spring application development by providing auto-configuration, starter dependencies, and embedded servers, reducing the need for extensive manual setup. In contrast, traditional Spring configuration requires detailed XML or Java-based configuration, explicit bean definitions, and manual setup of components. Overall, Spring Boot streamlines development and accelerates setup, while traditional Spring offers more granular control but involves more boilerplate code.
- 4. Spring Boot project structure and auto-configuration
 - Spring Boot Project Structure
 - A typical Spring Boot project follows a standardized structure that promotes clarity and ease of development. While it's flexible, a common layout looks like this:



Key Components

- Main Application Class: Annotated with `@SpringBootApplication`, this is the entry point.
- Controllers: Handle HTTP requests.
- Services: Business logic.

- Repositories: Data access layer (e.g. JPA repositories).
- Configuration: Custom configuration classes.
- Resources: External configuration files, static assets, templates.

Auto-Configuration in Spring Boot

 Auto-configuration is a core feature of Spring Boot that automatically configures Spring application context based on the dependencies present on the classpath and the external configurations provided.

How it works

- i. *Conditional Configuration*: Uses `@Conditional` annotations internally to activate configurations only when certain conditions are met (e.g., presence of a class, bean, or property).
- ii. *Starters*: Spring Boot provides "starters" (e.g. `spring-boot-starter-web`, `spring-boot-starter-data-jpa`) that include dependencies and auto-configuration classes for common functionalities.
- iii. `@EnableAutoConfiguration`: Usually included via `@SpringBootApplication`, which combines `@Configuration`, `@EnableAutoConfiguration`, and `@ComponentScan`.
- iv. Auto-Configuration Classes:
 - Located in `META-INF/spring.factories` within Spring Boot modules.
 - These classes are annotated with `@Configuration` and define beans that are conditionally loaded.

o Example:

- If you include `spring-boot-starter-web`, Spring Boot auto-configures:
 - ✓ An embedded Tomcat server
 - ✓ Spring MVC infrastructure
 - ✓ Message converters, etc.
 - If you include `spring-boot-starter-data-jpa`:
 - ✓ Configures a DataSource, EntityManager, and JPA repositories

o Customization

- You can override auto-configuration by defining your own beans or properties in 'application.properties'.
- To disable specific auto-configurations, use `@SpringBootApplication(exclude = {AutoConfigClass.class})`.

Summary

- Project Structure: Organized into 'java/', 'resources/', with clear separation of controllers, services, repositories, and configs.
- Auto-Configuration: Simplifies setup by automatically configuring beans based on dependencies and environment, reducing boilerplate code and accelerating development.