Spring Boot CMPY Learning

1. **What is Spring boot ?**
   - Spring Boot is a **framework** built on **Spring** that simplifies the development of **Java-based applications**. It removes the need for **complex configurations** and allows developers to build applications **quickly and efficiently**.

2. **What are the key features of spring boot ?**

   i) **Auto Configuration (Main Feature)**
   - **Automatically configures Spring components** based on project dependencies.
   - No need for **manual XML configuration**.
   - Example: If you include spring-boot-starter-web, Spring Boot **automatically configures** a web server (Tomcat).

   ii) Spring Boot Starters (Dependency Management)
   - Pre-configured **starter packages** simplify adding dependencies.
   - **Example:**
     - **spring-boot-starter-web** → Adds Spring MVC & Embedded Tomcat.
     - **spring-boot-starter-data-jpa** → Adds Hibernate & JPA.
     - **spring-boot-starter-test** → Adds JUnit & Mockito for testing.

   iii) Embedded Servers (No External Server Needed)
   - Comes with built-in servers like **Tomcat, Jetty, and Undertow**.
   - No need to manually install or configure a web server.
   - Just run the application, and it works like a **standalone** program!

   iv) Spring Boot CLI (Rapid Development with Command Line)
   - This command line tool is generally for managing dependencies, creating projects and running the applications.
   - Allows running **Spring Boot apps using Groovy** without full Java setup.
   - Reduces boilerplate code.
   - Example : ***spring run myapp.groovy***

   v) Actuator (Monitoring & Management)
   - Provides **ready-to-use endpoints** to check app health & metrics.
   - It also simplifies the troubleshooting management.
   - Example :
     - /actuator/health → Shows application health status.
     - /actuator/info → Displays app details.
   - **Great for production monitoring!**

vi) **Spring Boot DevTools (Auto Restart for Development)**
- Automatically reloads the application when code changes.
- Improves developer productivity.
- No need to manually restart the server after every change!

vii) **Microservices Support (Easily Build Microservices)**
- Perfect for building **microservices** using Spring Cloud.
- Supports:
  - **Service Discovery** (Eureka)
  - **Load Balancing** (Ribbon)
  - **Distributed Configuration** (Spring Cloud Config)
- **Spring Boot is widely used for microservices architecture.**

viii) **Security (Built-in Authentication & Authorization)**
- **Spring Security** integration for authentication & authorization.
- Supports **JWT, OAuth2, and LDAP** for securing applications.

ix) **Production-Ready (Optimized for Deployment)**
- Supports **Docker & Kubernetes** for cloud deployment.
- Provides **logging (Logback, Log4j)** and **monitoring tools**.
- Easily deployable as a **JAR** or **WAR** file.

3. What is pom.xml ?
- pom means Project Object Model is the configuration file for a Maven based spring boot project.
- It defines dependencies , plugins, build setting and metadata.
- It essential for managing libraries and automating the build process.

4. Why is pom.xml important ?
- **Manages Dependencies** → Automatically downloads required libraries.
- **Build & Packaging** → Defines how to compile, test, and package the project.
- **Version Control** → Ensures all developers use the same versions.
- **Plugins & Configurations** → Automates tasks like testing and deployment.

5. pom.xml ?
- Project Object Model
- What are the dependencies and plugins we can add it in the pom.xml file.
- We go to the Maven repository and we search for the what type of dependencies we want.
- There is no need to download the dependencies.

- Version controlling will control by the Maven because of the pom.xml.
- The dependencies will be automatically change its dependencies for the version.
- All the dependencies will be download to the location of ".m2" file because it will create local repository in our system itself.
- **If the pom.xml is not working for the database.**
    - **We remove the version in the pom.xml file**
- **If we try everything but our maven project is not working perfectly the delete ".m2" file**

6. Commands for Maven Project.

Command	Purpose
mvn clean	Deletes target/ directory (removes old builds).
mvn compile	Compiles the project.
mvn package	Packages the app into a .jar/.war file.
mvn install	Installs the package into the local repository.
mvn spring-boot:run	Runs the Spring Boot application.

7. What is application.properties?
    - Application properties file uses **key=value pair**.
    - No repeating key in the application properties.
    - It stores confidential property data, db related data, jpa configured data, server and logging data.
8. What is dependency Injection?
    - Dependency Injection (DI) is a design pattern that enables us to produce loosely coupled components. In DI, an object's ability to complete a task depends on another object. There three types of dependency Injections.
    - **Constructor injection:**
        - This is the most common type of DI in Spring Boot. In constructor injection, the dependency object is injected into the dependent object's constructor.
    - **Setter injection:**
        - In setter injection, the dependency object is injected into the dependent object's setter method.
    - **Field injection:**
        - In field injection, the dependency object is injected into the dependent object's field.
    - **Interface Injection:**
        - Spring does not support Interface Injection because it violates **separation of concerns**.

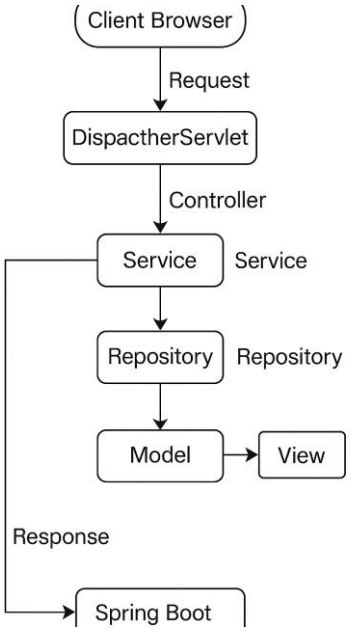
9. What is the advantage of spring framework, spring MVC framework and spring boot framework ?

Feature/Advantage	Spring	Spring MVC	Spring Boot
Dependency Injection & Aspect-Oriented Programming	✅	✅	✅
Web Support	❌	✅	✅
Annotation Support	✅	✅	✅
Auto Configuration	❌	❌	✅
Embedded Server	❌	❌	✅
Microservice Friendly	❌	⚠️ (can be)	✅
Fast Setup & Deployment	❌	❌	✅
Production Ready Tools	❌	❌	✅

10. What is Spring IOC Container ?
    - The Spring container is at the core of the Spring Framework. The container will create the objects, wire them together, configure them, and manage their complete life cycle from creation till destruction. The Spring container uses DI to manage the components that make up an application. These objects are called Spring Beans, which we will discuss in the next chapter.
    - The container gets its instructions on what objects to instantiate, configure, and assemble by reading the configuration metadata provided. The configuration metadata can be represented either by XML, Java annotations, or Java code.

11. What are the types of Container Spring provides ?
    - Spring provides two types of container, they are
        - Spring bean Factory Container
        - Spring Application Context Container

12. What is MVC architecture in the spring boot ?



- Spring Boot simplifies Spring MVC setup by auto-configuring most of the components (like DispatcherServlet, ViewResolver, etc.), letting you focus on business logic while still following the **MVC architecture**.
- Spring Boot follows the **Model-View-Controller (MVC)** design pattern. The idea is to **separate the responsibilities** of your application
- Model Layer (Data + Business Logic)
    - Represents application data and domain logic.
    - Contains:
        - POJOs
        - Repositories
        - Service classes
    - Handle data Processing and database interaction.
- Controller Layer (Presentation Logic)
    - Handles HTTP requests and responses.
    - Calls the Service/Model layer.
    - Passes model data to the view.
- View Layer (UI Layer)
    - Renders the final output using data from the model.
    - Supported view engines:
        - **Thymeleaf** (default in Spring Boot)
        - JSP (not recommended in Spring Boot for production)
        - FreeMarker
- Request work flow in spring boot +spring MVC
    1. **User sends request** to http://localhost:8080/users.
    2. **DispatcherServlet** receives the request
    3. **HandlerMapping** finds the appropriate controller method
    4. **UserController** is called → it calls UserService

5. UserService gets data from UserRepository
6. Controller adds data to Model and returns view name
7. **ViewResolver** resolves user-list to user-list.html
8. Thymeleaf renders the page with model data
9. Response sent back to the user

13. What is spring security ?

- Spring Security is a **security framework** for **authentication**, **authorization**, and **protection against common attacks** in Java applications (especially web apps using Spring Boot).
- The **key features** of spring security is **Authentication**, **Authorization**, **password Encoding**, **CSRF protection**, **Session management**, **OAuth2/OpenID connect** and **security Filters chain**.

a. It automatically secures all HTTP endpoints by default.
b. Client → Filter Chain → AuthenticationManager → ProviderManager → AuthenticationProvider → UserDetailsService → Authentication Success or Failure → Response.
c. **The Work flow:**

i) HTTP Request Comes In:
- When a request hits your application, it goes through the **Servlet Filter Chain**.

ii) Security Filter Chain Activation:
- Spring Security registers multiple filters in a **predefined order**. Key filters include:
  - SecurityContextPersistenceFilter - Restores security context from the session.
  - UsernamePasswordAuthenticationFilter - Handles login requests.
  - BasicAuthenticationFilter - Handles basic HTTP authentication.
  - ExceptionTranslationFilter - Handles exceptions like 401/403.
  - FilterSecurityInterceptor - Makes authorization decisions.
  - **Only one SecurityFilterChain is active per request, and it is matched by URL patterns.**

iii) Authentication Begins:
- If a request hits a protected resource, Spring Security checks:
  - Does the request have an **authentication token** (JWT, session cookie, etc.)?

- If yes → try to validate it **else** trigger authentication (e.g., login page or 401 error).

iv) **UsernamePasswordAuthenticationFilter** (During Login):

- This filter captures the login form's POST request (usually at /login).
- Extracts username and password.
- Creates a UsernamePasswordAuthenticationToken.

v) AuthenticationManager & ProviderManager:

- AuthenticationManager (typically a ProviderManager) tries one or more AuthenticationProvider 's.
- By default, it uses:
  - DaoAuthenticationProvider → Uses UserDetailsService to load user.
  - PasswordEncoder (e.g., BCrypt) to check passwords.

vi) UserDetailsService:

- Your custom implementation returns a UserDetails object.
- This contains the user's:
  - Username
  - Password
  - Authorities/Roles
  - ```
    public class MyUserDetailsService implements UserDetailsService {
       public UserDetails loadUserByUsername(String username) throws
    UsernameNotFoundException {
          return new User(username, passwordEncoder.encode("123"), List.of(new
    SimpleGrantedAuthority("ROLE_USER")));
       }
    }
    ```

vii) Successful Authentication:

- Spring Security stores authentication in SecurityContextHolder.
- The context is saved in the HttpSession by SecurityContextPersistenceFilter.
- The user is redirected to the originally requested page or a default page

viii) Failure:

- Error is caught by ExceptionTransulationFilter.
- Redirects to login page or sends HTTP 401.

d. Authorization:

- Once authenticated, for every secured request
  - FilterSecurityInterceptor checks whether the authenticated user has permission.
  - Compares user roles/authorities to those required by your annotations or DSL config
  - ```
    http.authorizeHttpRequests(authz -> authz
      .requestMatchers("/admin/**").hasRole("ADMIN")
      .anyRequest().authenticated()
    );
    ```

e. Session vs Stateless Authentication:
    - **Session-based (default)**: Stores Authentication object in session.
    - **Stateless (e.g., with JWT)**: No session; each request must carry a valid token.

- Summary Diagram:
    - [Client Request]
      ↓
      [Spring Security Filter Chain]
      ↓
      [AuthenticationManager] —→ [AuthenticationProvider] → [UserDetailsService]
      ↓
      [SecurityContextHolder]
      ↓
      [Controller/Service Method]