**Answer the following**

1. **Write down the steps for creating a simple Spring Boot application and how we can implement the same.**

**Ans –**

**Steps to Create a Simple Spring Boot Application**

1. **Setup Your Environment**:
    - Install Java Development Kit (JDK) (minimum version 8 or later).
    - Install an IDE like IntelliJ IDEA, Eclipse, or Visual Studio Code.
    - Install Maven or Gradle (build tools).
    - Ensure Spring Boot CLI is installed if you plan to use it.
2. **Create a Spring Boot Project**:
    - **Using Spring Initializr**:
        - Go to Spring Initializr .
        - Select the project type (Maven/Gradle), language (Java/Kotlin/Groovy), and Spring Boot version.
        - Add dependencies like Spring Web, Spring Data JPA, H2 Database, etc.
        - Download the generated project zip, extract it, and import it into your IDE.
    - **Using IDE Plugins**:
        - Many IDEs support Spring Boot project creation via integrated Spring Initializr tools.
3. **Set Up the Application Properties**:
    - Configure application-specific settings in src/main/resources/application.properties or application.yml (e.g., server port, database configuration).
4. **Create the Main Application Class**:

    Annotate the class with @SpringBootApplication.

```
package com.example.demo;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootAp plication;
@SpringBootApplication
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

5. **Create Your Controller**:

- Write a REST or MVC controller using @RestController or @Controller annotations.

```
package com.example.demo.controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
@RestController
@RequestMapping("/api")
public class HelloController {
    @GetMapping("/hello")
    public String sayHello() {
        return "Hello, World!";
    }
}
```

6. **Define Service and Repository Layers** (if needed):
    - Create a service class for business logic using @Service.
    - Create a repository interface for data persistence using @Repository or Spring Data JPA.

7. **Run the Application**:
    - Run the application using the main method in your application class or via the IDE.
    - Alternatively, use mvn spring-boot:run or gradle bootRun.
8. **Test the Application**:
    - Use a browser or tools like **Postman** to test endpoints.
    - For example: Navigate to http://localhost:8080/api/hello for the sample endpoint.

**2. Explain the Spring MVC framework with its components.**

**Ans –**

**Spring MVC**

Spring MVC (Model-View-Controller) is a module of the Spring Framework that provides a robust architecture to build web applications. It separates the application's concerns into three main components: Model, View, and Controller.

**Key Components of Spring MVC:**

1. **DispatcherServlet**:
    - Acts as the front controller.
    - Intercepts all incoming HTTP requests and routes them to the appropriate handlers.
    - Configured in web.xml or auto-configured in Spring Boot.
2. **Controller**:
    - Handles user input and business logic.
    - Annotated with @Controller or @RestController.
    - Maps incoming requests to handler methods using annotations like @RequestMapping, @GetMapping, @PostMapping, etc.
3. **Model**:
    - Represents application data.

- Passed between controller and view to display or process data.
- In Spring, Model, ModelMap, or ModelAndView are used to carry data.

4. **View**:
    - Represents the presentation layer.
    - Spring supports various view technologies like JSP, Thymeleaf, Freemarker, etc.
    - Views are resolved using View Resolvers, such as InternalResourceViewResolver.
5. **HandlerMapping**:
    - Maps incoming requests to the appropriate handler method or controller.
6. **HandlerAdapter**:
    - Executes the matched handler method.
7. **ViewResolver**:
    - Resolves logical view names returned by controllers into actual view files.
8. **ExceptionResolver**:
    - Handles exceptions globally and returns error responses.

**Flow of Spring MVC:**

1. A user sends an HTTP request to the server.
2. **DispatcherServlet** intercepts the request.
3. It consults **HandlerMapping** to determine the appropriate controller.
4. The controller processes the request and interacts with the **Model** layer (e.g., database operations).
5. The controller returns a **ModelAndView** object or response body.
6. **ViewResolver** resolves the view name to an actual view (e.g., JSP/Thymeleaf template).
7. The **View** is rendered and sent as an HTTP response to the user.