1. What is Spring?

Spring is a **comprehensive framework** for building **Java applications**, especially for enterprise-level development. It provides infrastructure support for developing Java apps, allowing developers to focus on **business logic** rather than boilerplate code.

It supports:

- Dependency Injection (DI)
- Inversion of Control (IoC)
- Aspect-Oriented Programming (AOP)
- Transaction management
- Integration with various technologies (JPA, JDBC, JMS, etc.)

```
@Service
public class EmailService {
    public void send(String msg) {
        System.out.println("Email sent: " + msg);
    }
}

@RestController
public class UserController {

    @Autowired
    private EmailService emailService;

    @PostMapping("/register")
    public void register() {
        emailService.send("Welcome!");
    }
}
```

2. What is Spring Boot?

Spring Boot is an **extension of the Spring Framework** that simplifies the process of building production-ready Spring applications.

It provides:

- Embedded servers (like Tomcat, Jetty)
- Auto-configuration
- Opinionated defaults
- Starter dependencies (spring-boot-starter-web, etc.)
- No need for web.xml or extensive applicationContext.xml

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

3. What is the relation between Spring Platform and Spring Boot?

The Spring Platform refers to the entire ecosystem of Spring projects — such as Spring Framework, Spring Boot, Spring Data, Spring Security, etc.

Spring Boot is one of the projects within the Spring Platform that helps you bootstrap and run Spring applications with minimal setup.

4. What is the relation between Spring Platform and Spring Framework?

The Spring Framework is the core project of the Spring Platform.

It includes the fundamental features like Dependency Injection, IoC container, AOP, and transaction support.

Other projects like Spring Boot, Spring Data, Spring Security depend on Spring Framework.

Example: You can use the Spring Framework directly in a traditional XML-based setup or annotation-based config:

```
<!-- applicationContext.xml -->
<bean id="emailService" class="com.example.EmailService" />
```

```
@Component
public class EmailService { }
```

5. What is Dependency Injection and how is it done in the Spring platform/framework?

Dependency Injection (DI) is a **design pattern** where the **dependencies are provided by the framework**, rather than the class creating them itself.

In Spring, DI can be done in 3 ways:

- Constructor Injection
- Setter Injection
- Field Injection (via @Autowired)

```
@Service
public class UserService {
   private final EmailService emailService;

   @Autowired
   public UserService(EmailService emailService) {
      this.emailService = emailService;
   }
}
```

6. What is Inversion of Control (IoC) and how is it related to Spring?

Inversion of Control (IoC) is a principle where the control of object creation and dependency management is inverted from the application code to the framework (like Spring).

Spring implements IoC using its IoC Container, which is responsible for:

Managing lifecycle of beans

Injecting dependencies

Controlling configuration

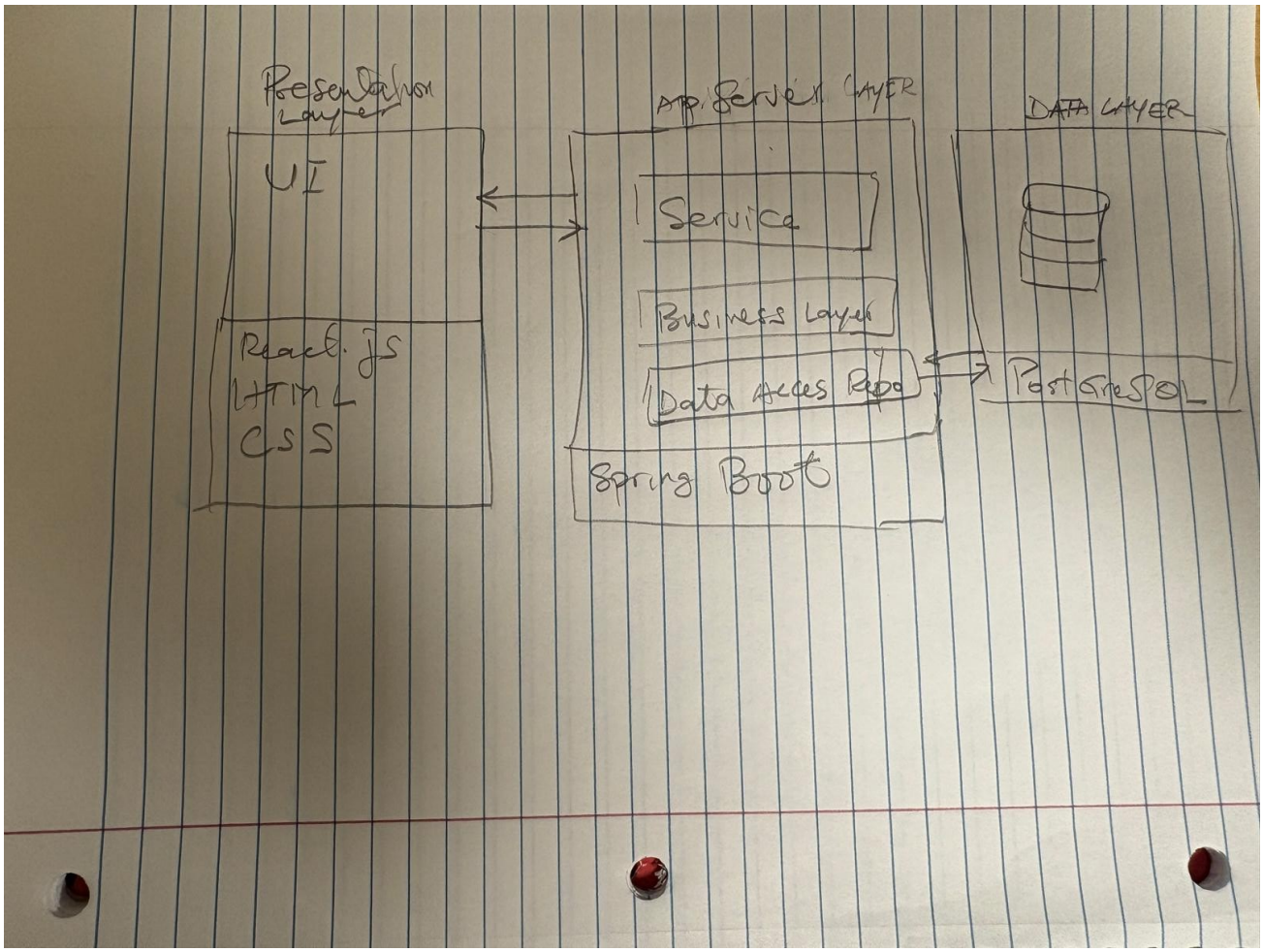
Example: Instead of doing this manually:

```
EmailService email = new EmailService();
UserService userService = new UserService(email);
```

You let Spring handle it:

```
@Configuration
public class AppConfig {
    @Bean
    public EmailService emailService() {
        return new EmailService();
    }

    @Bean
    public UserService userService() {
        return new UserService(emailService());
    }
}
```


Presentation Layer
UI
React.js
HTML
CSS
API Server Layer
Service
Business Layer
Data Access Repo
Spring Boot
DATA LAYER
PostgreSQL