SCHOOL OF ENGINEERING

Google Class Code:22u2lc7

SPRING BOOT

MODULE 1

Presentation Material

Department of Computer Science & Engineering

Course Code: **22CS3509**

Semester: V

Course Title: SPRING BOOT

Year: III

Faculty Name: Dr. Gokulakrishnan S

Course Learning Objectives:

SCHOOL OF ENGINEERING

This Course will enable students to:

1. Understand the concepts of Spring Boot, setting up a project, building RESTful APIs, data persistence with JPA, security integration, testing, deploying applications, and advanced features like microservices and Spring Cloud.

2. Develop and manage scalable Java-based web applications with Spring Boot, covering core concepts, RESTful APIs, data access, security, testing, and deployment.

3. Explain the role of spring boot and Extreme framework.

4. Recognize the importance of developing APIs.

5. Identify the features based on real world scenario.

Course Outcome:

Course Outcome	Description	Bloom's Taxonomy Level
At the end of the course the student will be able to:		
1	**Understand** the activities involved in spring boot and analyze the framework.	L1 & L2
2	**Apply** spring boot application development to develop restfull webservices.	L3
3	**Describe** Data access with spring data JPA and implement transaction management to ensure secure and consistent Spring applications.	L2
4	**Develop**, test, and secure reactive APIs using Spring WebFlux**.**	L2 & L3
5	**Build** and manage reactive data persistence using Spring Data for Cassandra and MongoDB.	L3

TEXT BOOKS:

1. Craig Walls, “Spring in Action”, Fifth Edition, Manning, ISBN 9781617294945

REFERENCE BOOKS:

1. Santosh Kumar K., “Spring and Hibernate”, Tata McGraw-Hill Publishing,2009,ISBN 978-0070680111
2. Paul Tepper Fisher and Brian D. Murphy, “Spring persistence with Hibernate”,Apress,2010, ISBN 978-1-4302-2632-1
3. Amritendu De, “Spring 4 and Hibernate 4: Agile Java Design and Development”,McGraw-Hill Education,2015, ISBN: 9780071845113
4. Chris Schaefer, Clarence Ho, and Rob Harrop ,Pro Spring. Apress

E-Resources:

1. https://www.udemy.com/course/spring-5-with-spring-boot-2/
2. https://www.youtube.com/playlist?list=PLYZhppjPNiP8u76RFIpN3oNtSIobuIF0Q

Activity Based Learning (Suggested Activities in Class)

- Setting up a project based on a real-world scenario
- Implementing features learned in previous modules
- Best practices in Spring Boot development

Exam	Total	Converted
CIA	60	60
SEE	100	40

Passing mark = CIA + SEE = 40

CIA Components		
CIA Components	**Total**	**Converted**
MSE1	40	**20**
MSE2	40	**20**
Assignment	10	**10**
Case Study	10	**10**

AAT (Alternate Assessment)

i. **First Component: Assignment for 10 Marks**
ii. **Second Component: Case Study for 10 Marks**

MODULE 1: Contents

Introduction to Spring Boot: Overview of Spring Framework and Spring Boot- History and evolution of Spring Boot, Benefits of using Spring Boot, Comparison with traditional Spring framework, Setting Up the Development Environment- Installation and setup of Java, Maven/Gradle, and IDE (IntelliJ/Eclipse), Creating a basic Spring Boot application Understanding Spring Boot starters and dependencies.

Spring Boot Project Structure: Explanation of key project files (pom.xml/build. Gradle, application. Properties/application),Main application class and @SpringBootApplication annotation.

Textbook 1: Chapter 1: 1.1 to 1.4

- ✓ **Spring Boot** is an open source Java-based framework used to create a micro Service.
- ✓ It is used to build stand-alone and production ready spring applications.

What is Micro Service?

- ✓ Micro Service is an architecture that allows the developers to develop and deploy services independently.
- ✓ Each service running has its own process and this achieves the lightweight model to support business applications.

- ✓ **What is Spring Boot?**
- ✓ Spring Boot provides a good platform for Java developers to develop a stand-alone and production-grade spring application that you can just run. You can get started with minimum configurations without the need for an entire Spring configuration setup.

- ✓ **Advantages**
- ✓ Easy to understand and develop spring applications
- ✓ Increases productivity
- ✓ Reduces the development time

Goals

- ✓ To avoid complex XML configuration in Spring
- ✓ To develop a production ready Spring applications in an easier way
- ✓ To reduce the development time and run the application independently
- ✓ Offer an easier way of getting started with the application

- ✓ **Why Spring Boot?**
- ✓ It provides a flexible way to configure Java Beans, XML configurations, and Database Transactions.
- ✓ It provides a powerful batch processing and manages REST endpoints.
- ✓ In Spring Boot, everything is auto configured; no manual configurations are needed.
- ✓ It offers annotation-based spring application
- ✓ Eases dependency management
- ✓ It includes Embedded Servlet Container

Evolution of Spring Boot

Spring Boot came into existence when in October 2012, a client, Mike Youngstrom made a Jira request asking for bootstrapping the spring framework so that it can be quickly started. And hence in early 2013, Spring Boot was made.

In April 2014, Spring Boot 1.0 was created followed by various versions.

Spring Boot 1.1 on June 2014,

1.2 in March 2015,

1.3 in December 2016,

1.4 in January 2017 and

Spring Boot 1.5 on February 2017.

As of September 2023, the latest stable version of Spring Boot is 3.1.3

https://endoflife.date/spring-boot

Spring Boot Architecture

To understand the architecture of Spring Boot, let us first see different layers and classes present in it.

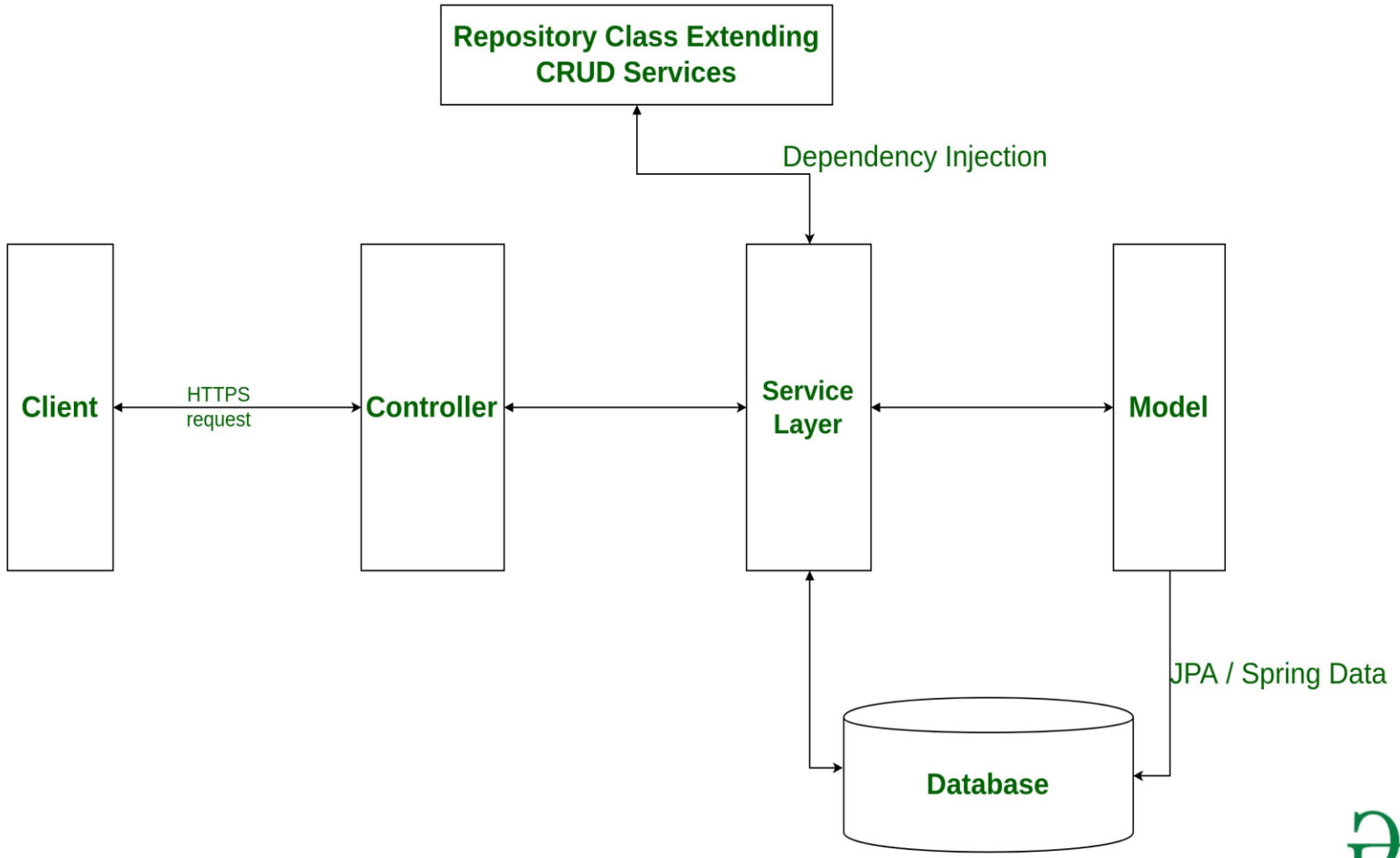
Layers in Spring Boot: There are four main layers in Spring Boot:

- **Presentation Layer:** As the name suggests, it consists of views(i.e. frontend part)
- **Data Access Layer:** CRUD (create, retrieve, update, delete) operations on the database comes under this category.
- **Service Layer:** This consist of service classes and uses services provided by data access layers.
- **Integration Layer:** It consists of web different web services(any service available over the internet and uses XML messaging system).

Then we have utility classes, validator classes and view classes.

All the services provided by the classes are implemented in their corresponding classes and are retrieved by implementing the dependency on those interfaces.

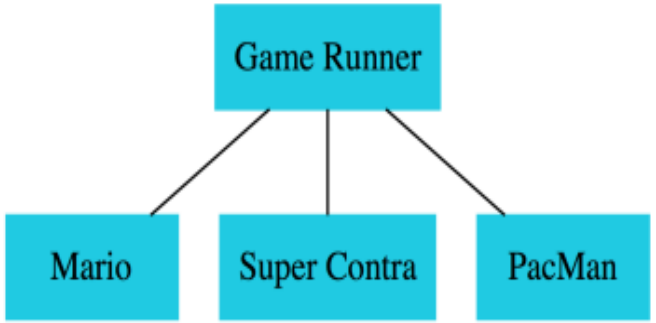
Spring Boot flow architecture



Loose Coupling with Spring Framework

In 28 Minutes

- Design Game Runner to run games:
  - Mario, Super Contra, PacMan etc
- **Iteration 1**: Tightly Coupled
  - GameRunner class
  - Game classes: Mario, Super Contra, PacMan etc
- **Iteration 2**: Loose Coupling - Interfaces
  - GameRunner class
  - GamingConsole interface
    - Game classes: Mario, Super Contra, PacMan etc
- **Iteration 3**: Loose Coupling - Spring
  - Spring framework will manage all our objects!
    - GameRunner class
    - GamingConsole interface
      - Game classes: Mario, Super Contra, PacMan etc



Iteration-1 Tightly Coupled

Demo1Application.java × MarioGame.java GameRunner.java

```
1  package com.example1.demo1;
2
3  import org.springframework.boot.SpringApplication;
8
9  @SpringBootApplication
10 public class Demo1Application {
11
12     public static void main(String[] args) {
13         SpringApplication.run(Demo1Application.class, args);
14         MarioGame game=new MarioGame();
15         GameRunner runner=new GameRunner(game);
16         runner.run();
17     }
18
19 }
20
```

Demo1Application.java | MarioGame.java | GameRunner.java

```java
package com.example1.demo1.game;

public class MarioGame {
    void up()
    {
        System.out.println("Mario UP");
    }

}
```

Demo1Application.java | MarioGame.java | GameRunner.java

```java
package com.example1.demo1.game;

public class GameRunner {
    private MarioGame game;
    public GameRunner(MarioGame game)
    {
        this.game=game;
    }

    public void run()
    {
        game.up();
    }
}
```

Problems | Servers | Terminal | Data Source Explorer | Properties | Console

<terminated> Demo1Application [Java Application] C:\Users\HP\OneDrive\DSU\Subjects Related\SpringBoot\eclipse-jee-2024-09-R-win32-x86_64\eclips

```
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/

 :: Spring Boot ::                (v3.3.3)

2024-09-19T10:34:55.947+05:30  INFO 6928 --- [demo1] [           main] com.example1.demo1.Demo1Application
2024-09-19T10:34:55.951+05:30  INFO 6928 --- [demo1] [           main] com.example1.demo1.Demo1Application
2024-09-19T10:34:56.696+05:30  INFO 6928 --- [demo1] [           main] com.example1.demo1.Demo1Application
Mario UP
```

SCHOOL OF ENGINEERING

Iteration-2 Loosely Coupled- Level 2 - Java Interfaces

Demo1Application.java × | MarioGame.java | GameRunner.java | Gamingconsole.java | Pacman.java

```java
package com.example1.demo1;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class Demo1Application {

    public static void main(String[] args) {
        SpringApplication.run(Demo1Application.class, args);
        //MarioGame game=new MarioGame();
        Pacman game=new Pacman();
        GameRunner runner=new GameRunner(game);
        runner.run();
    }

}
```

Demo1Application.java | MarioGame.java | GameRunner.java

```
1  package com.example1.demo1.game;
2
3  public class MarioGame implements Gamingconsole {
4      public void up()
5      {
6          System.out.println("DSU Mario UP");
7      }
8
9  }
10
```

Demo1Application.java | MarioGame.java | GameRunner.java | Gamingconsole.java | Pacman.java

```
1  package com.example1.demo1.game;
2
3  public class Pacman implements Gamingconsole {
4      public void up()
5      {
6          System.out.println("DSU Pacman UP");
7      }
8
9  }
10
```

Demo1Application.java | MarioGame.java | GameRunner.java

```
1  package com.example1.demo1.game;
2
3  public class GameRunner {
4      private Gamingconsole game;
5      public GameRunner(Gamingconsole game)
6      {
7          this.game=game;
8      }
9
10     public void run()
11     {
12         game.up();
13     }
14 }
15
```

Demo1Application.java | MarioGame.java | GameRunner.java | Gamingconsole.java

```java
package com.example1.demo1.game;

public interface Gamingconsole {
    void up();
}

```

```
:: Spring Boot ::                (v3.3.3)

2024-09-19T11:14:36.506+05:30  INFO 9176 --- [demo1] [           main] com.example1.demo1.Demo1Application
2024-09-19T11:14:36.512+05:30  INFO 9176 --- [demo1] [           main] com.example1.demo1.Demo1Application
2024-09-19T11:14:37.558+05:30  INFO 9176 --- [demo1] [           main] com.example1.demo1.Demo1Application
DSU Pacman UP
```

Demo1Application.java | MarioGame.java | GameRunner.java | Gamingconsole.java | Pacman.java

```java
package com.example1.demo1;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

import com.example1.demo1.game.GameRunner;
import com.example1.demo1.game.MarioGame;
import com.example1.demo1.game.Pacman;
import com.example1.demo1.game.Gamingconsole;

@SpringBootApplication
public class Demo1Application {

    public static void main(String[] args) {
        SpringApplication.run(Demo1Application.class, args);
        //MarioGame game=new MarioGame();
        Gamingconsole game=new Pacman();
        GameRunner runner=new GameRunner(game);
        runner.run();
    }

```

Iteration-3 Loosely Coupled- Level 2 -Spring Framework

Demo1Applic... | GameRunner.java | Pacman.java | Game3.java | GamingConso... | component.java

```
1  package com.example1.demo1;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5  import org.springframework.context.ConfigurableApplicationContext;
6
7  import com.example1.demo1.game.Game3;
8  import com.example1.demo1.game.GameRunner;
9  import com.example1.demo1.game.MarioGame;
10 import com.example1.demo1.game.Pacman;
11 import com.example1.demo1.game.Gamingconsole;
12
13 @SpringBootApplication
14 public class Demo1Application {
15
16     public static void main(String[] args) {
17         ConfigurableApplicationContext context = SpringApplication.run(Demo1Application.class, args);
18
19     //  MarioGame game=new MarioGame();
20     //  Pacman game=new Pacman();
21         //Game3 game=new Game3();
22         //GameRunner runner=new GameRunner(game);
23         GameRunner runner=context.getBean(GameRunner.class);
24         runner.run();
25     }
26
27 }
28
```

Demo1Applic... | GameRunner.java | Pacman.java | Game3.java | GamingConso... | component.java

```java
package com.example1.demo1.game;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import com.example1.demo1.gamee.GamingConsole;
@Component
public class GameRunner {
    @Autowired
    private GamingConsole game;
    //private Pacman game;
    //private Game3 game;
    public GameRunner(GamingConsole game)
    {
        this.game=game;
    }

    public void run()
    {
        game.up();
    }
}
```

Demo1Applic... GameRunner.java Pacman.java Game3.

```
1  package com.example1.demo1.game;
2
3  import org.springframework.stereotype.Component;
4
5  import com.example1.demo1.gamee.GamingConsole;
6  @Component
7  public class Pacman implements GamingConsole {
8      public void up()
9      {
10         System.out.println("DSU Pacman UP");
11     }
12
13 }
14
```

Demo1Applic... GameRunner.java Pacman.java Game3.java

```
1  package com.example1.demo1.game;
2
3  import org.springframework.context.annotation.Primary;
4  import org.springframework.stereotype.Component;
5
6  import com.example1.demo1.gamee.GamingConsole;
7  @Component
8  @Primary
9  public class Game3 implements GamingConsole{
10     public void up()
11     {
12         System.out.println("GAME 3 LEFT");
13     }
14
15 }
16
```

Demo1Applic... | MarioGame.java | GameRunner.java | Pacman.java | Game3.java | GamingConso...

```
1 package com.example1.demo1;
2
3 public interface GamingConsole {
4 void up();
5 }
6
```

Demo1Applic... | MarioGame.java | GameRunner.java | Pacman.java | Game3.java | GamingConso...

```java
package com.example1.demo1.game;

import org.springframework.context.annotation.Primary;
import org.springframework.stereotype.Component;

import com.example1.demo1.gamee.GamingConsole;
@Component
@Primary
public class Game3 implements GamingConsole{
    public void up()
    {
        System.out.println("GAME 3 LEFT");
    }

}

```

Problems | Servers | Terminal | Data Source Explorer | Properties | Console

<terminated> Demo1Application [Java Application] C:\Users\HP\OneDrive\DSU\Subjects Related\SpringBoot\eclipse-jee-2024-09-R-win32-

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/

 :: Spring Boot ::                (v3.3.3)

2024-09-27T13:02:22.457+05:30  INFO 8664 --- [demo1] [           main] com.example1.demo1.Demo1Appli
2024-09-27T13:02:22.459+05:30  INFO 8664 --- [demo1] [           main] com.example1.demo1.Demo1Appli
2024-09-27T13:02:23.244+05:30  INFO 8664 --- [demo1] [           main] com.example1.demo1.Demo1Appli
GAME 3 LEFT
```