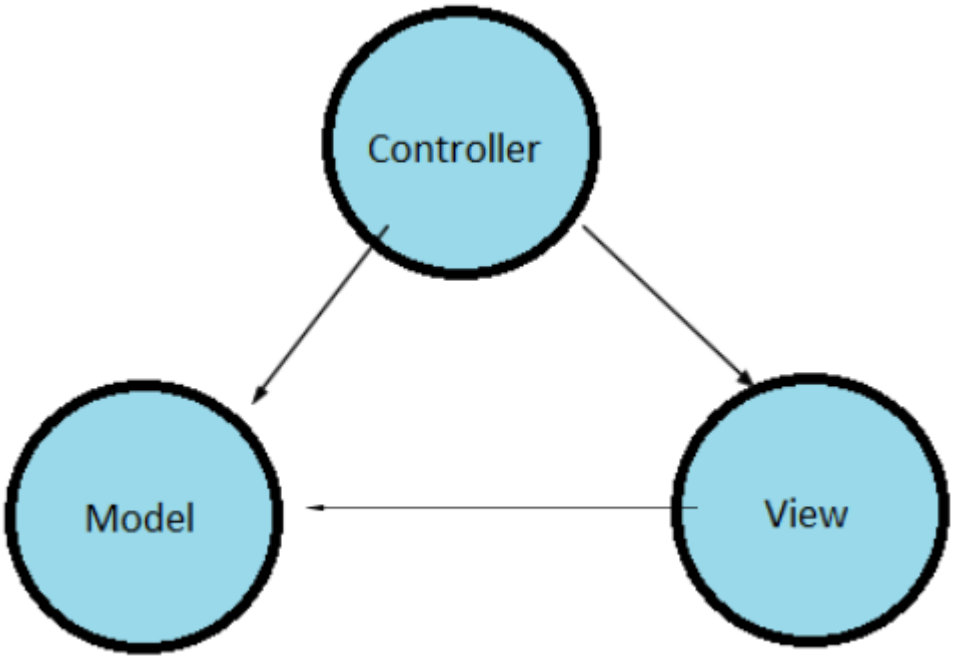
Session-34

Web App using spring boot

Spring Boot:

Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can “just run”.

We will use the concept of MVC, i.e., Model-View-Controller. MVC is an architecture that separates various components of an application like the Input Logic, Business Logic, and UI Logic.



Spring is a popular Java application framework. Spring Boot is an effort to create stand-alone, production-grade Spring based applications with minimal effort.

Spring Boot web application example

The application shows a message and today's date. The message is retrieved from an appplication's property.

```
pom.xml
src
├───main
│   ├───java
│   │   └───com
│   │       └───ep
│   │           │   Application.java
│   │           └───controller
│   │                   MyController.java
│   └───resources
│       │   application.properties
│       └───templates
│               index.pebble
└───test
    └───java
```

This is the project structure.

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
            http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.ep</groupId>
    <artifactId>springbootfirstweb</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
    </properties>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.2.2.RELEASE</version>
    </parent>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

    <dependency>
        <groupId>io.pebbletemplates</groupId>
```

```
        <artifactId>pebble-spring-boot-starter</artifactId>
        <version>3.1.2</version>
    </dependency>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

This is the Maven build file. The `spring-boot-starter-web` is starter for building web, including RESTful, applications using Spring MVC.

The `pebble-spring-boot-starter` contains the Pebble template engine. When Spring Boot detects this starter, it automatically configures Pebble for us.

The application is packaged into a JAR file, which contains an embedded Tomcat web server.

resources/application.properties

```
application.message: Hello there
```

The `application.properties` file contains various configuration settings of a Spring Boot application. We have one custom message option.

com/ep/controller/MyController.java

```
package com.ep.controller;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import java.time.LocalDate;
import java.util.Map;

@Controller
public class MyController {

    @Value("${application.message}")
    private String message = "Hi there";

    @GetMapping("/")
    public String index(Model model) {

        model.addAttribute("now", LocalDate.now());
        model.addAttribute("message", this.message);

        return "index";
    }
}
```

This is the controller class for the Spring Boot web application. A controller is decorated with the `@Controller` annotation. The controller has one mapping. The mapping resolves to the `index.pebble`, which is located in the `resources/templates` directory.

```
@Value("${application.message}")
private String message = "Hi there";
```

We inject a value from the `application.properties` into the `message` variable.

```
@GetMapping("/")
public String index(Model model) {

    model.addAttribute("now", LocalDate.now());
    model.addAttribute("message", this.message);

    return "index";
}
```

The `@GetMapping` annotation maps a GET request with the / path to the index method handler. A model is created and filled with data. Spring Boot resolves the `index` view to the `index.pebble` template file, to which it also sends the model data.

resources/templates/index.pebble

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Home page</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
<body>

<p>
Today: {{ now }}
</p>

<p>
Message: {{ message }}
</p>

</body>
</html>
```

The `index.pebble` displays two values: the current date and the received message. Both values are passed to the template via the controller.

```
<p>
Today: {{ now }}
</p>
```

Pebble uses the `{{}}` syntax to display the variable.

com/ep/Application.java

```
package com.ep;

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

@SpringBootApplication
public class Application {

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

The `Application` sets up the Spring Boot application.

```
$ mvn spring-boot:run
```

We run the application. Now we can navigate to `localhost:8080` to see the application message.

OUTPUT: