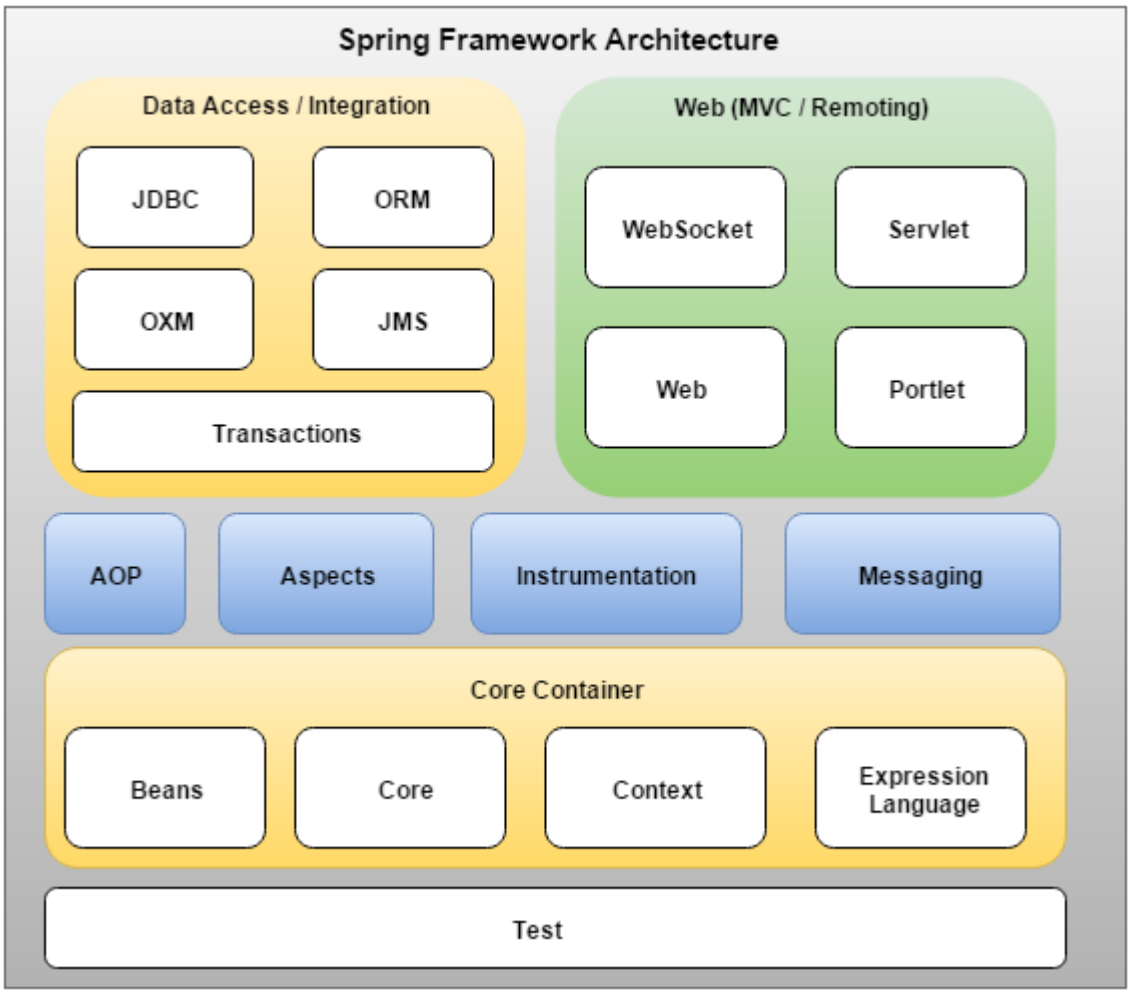
G.VINAY (LAB13)

1.Explain the architecture of Spring Framework

Spring framework architecture:



MODULES:

The Spring Framework consists of features organized into about 20 modules. These modules are grouped into Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation, and Test, as shown in the following diagram.

CORE CONTAINER:

The Core Container consists of the Core, Beans, Context, and Expression Language modules.

The Core and Beans modules provide the fundamental parts of the framework, including the IoC and Dependency Injection features. The BeanFactory is a sophisticated implementation of the factory pattern. It removes the need for programmatic singletons and allows you to decouple the configuration and specification of dependencies from your actual program logic.

The Context module builds on the solid base provided by the Core and Beans modules: it is a means to access objects in a framework-style manner that is similar to a JNDI registry. The Context module inherits its features from the Beans module and adds support for internationalization (using, for example, resource bundles), event-propagation, resource-loading, and the transparent creation of contexts by, for example, a servlet container. The Context module also supports Java EE features such as EJB, JMX ,and basic remoting. The ApplicationContext interface is the focal point of the Context module.

The Expression Language module provides a powerful expression language for querying and manipulating an object graph at runtime. It is an extension of the unified expression language (unified EL) as specified in the JSP 2.1 specification. The language supports setting and getting property values, property assignment, method invocation, accessing the context of arrays, collections and indexers, logical and arithmetic operators, named variables, and retrieval of objects by name from Spring's IoC container. It also supports list projection and selection as well as common list aggregations.

DATA ACCESS/INTEGRATION:

The Data Access/Integration layer consists of the JDBC, ORM, OXM, JMS and Transaction modules.

The JDBC module provides a JDBC-abstraction layer that removes the need to do tedious JDBC coding and parsing of database-vendor specific error codes.

The ORM module provides integration layers for popular object-relational mapping APIs, including JPA, JDO, Hibernate, and iBatis. Using the ORM package you can use all of these O/R-mapping frameworks in combination with all of the other features Spring offers, such as the simple declarative transaction management feature mentioned previously.

The OXM module provides an abstraction layer that supports Object/XML mapping implementations for JAXB, Castor, XMLBeans, JiBX and XStream.

The Java Messaging Service (JMS) module contains features for producing and consuming messages.

The Transaction module supports programmatic and declarative transaction management for classes that implement special interfaces and for all your POJOs (plain old Java objects).

WEB:

The Web layer consists of the Web, Web-Servlet, Web-Struts, and Web-Portlet modules.

Spring's *Web* module provides basic web-oriented integration features such as multipart file-upload functionality and the initialization of the IoC container using servlet listeners and a web-oriented application context. It also contains the web-related parts of Spring's remoting support.

The *Web-Servlet* module contains Spring's model-view-controller (MVC) implementation for web applications. Spring's MVC framework provides a clean separation between domain model code and web forms, and integrates with all the other features of the Spring Framework.

The *Web-Struts* module contains the support classes for integrating a classic Struts web tier within a Spring application. Note that this support is now deprecated as of Spring 3.0. Consider migrating your application to Struts 2.0 and its Spring integration or to a Spring MVC solution.

The *Web-Portlet* module provides the MVC implementation to be used in a portlet environment and mirrors the functionality of Web-Servlet module.

AOP AND INSTUMENTATION:

Spring's AOP module provides an AOP Alliance-compliant aspect-oriented programming implementation allowing you to define, for example, method-interceptors and pointcuts to cleanly decouple code that implements functionality that should be separated. Using source-level metadata functionality, you can also incorporate behavioral information into your code, in a manner similar to that of .NET attributes.

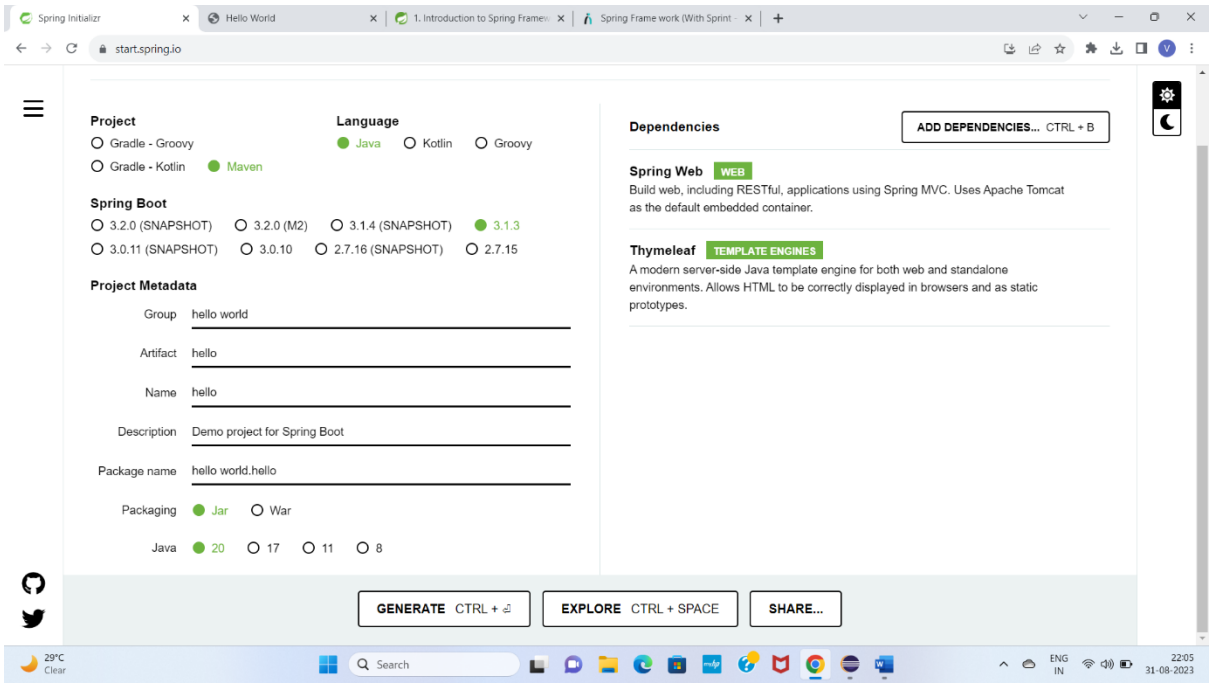
The separate *Aspects* module provides integration with AspectJ.

The *Instrumentation* module provides class instrumentation support and classloader implementations to be used in certain application servers.

TEST:

The *Test* module supports the testing of Spring components with JUnit or TestNG. It provides consistent loading of Spring ApplicationContexts and caching of those contexts. It also provides mock objects that you can use to test your code in isolation.

2.Create a Simple spring boot application to print hello world message to user in browser when the spring spoot app is up on your server.



Hello/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
https://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>3.1.3</version>
		<relativePath/> <!-- lookup parent from repository -->
	</parent>
	<groupId>hello.world</groupId>
	<artifactId>hello</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<name>hello</name>
	<description>Demo project for Spring Boot</description>
	<properties>
		<java.version>20</java.version>
```

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

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

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

</project>
```

HelloApplication.java:

```
package hello.world.hello;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class HelloApplication {

public static void main(String[] args) {

	SpringApplication.run(HelloApplication.class, args);

}}
```

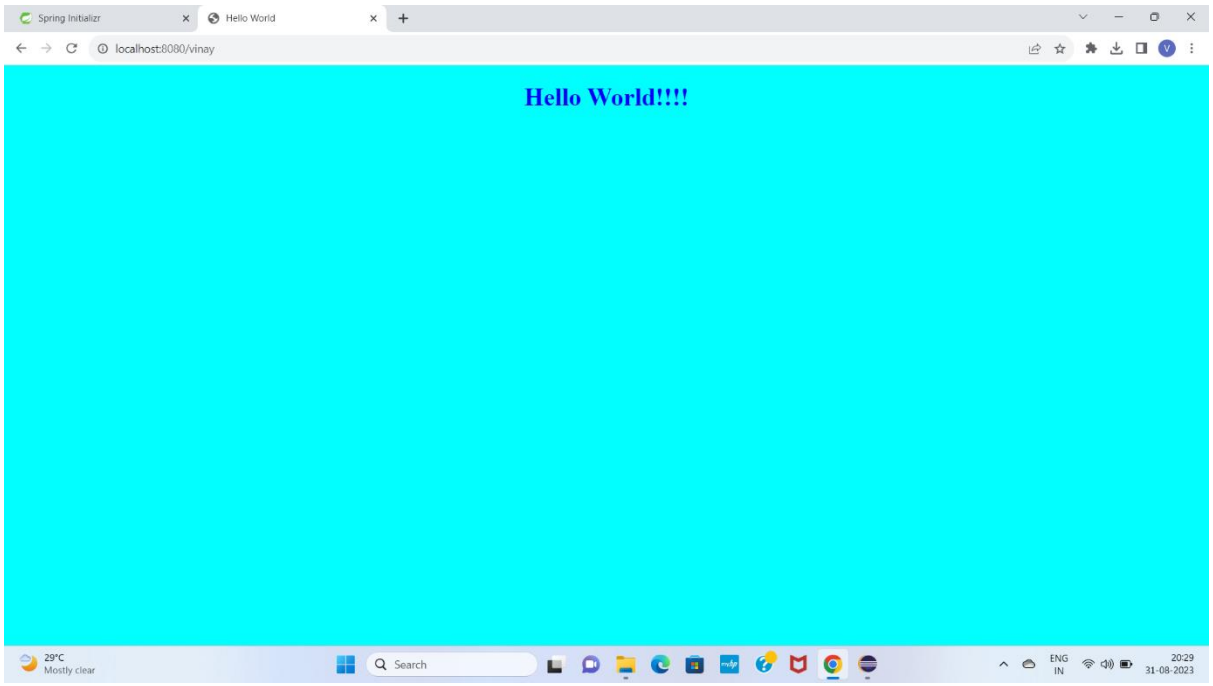
HelloController.java:

```
package hello.world.hello;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class Hellocontroller {
	@GetMapping("/vinay")
	public String hello() {
		return "hello";
	}

}
```

Hello.html:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Hello World</title>
</head>
<body bgcolor="aqua">
	<h1 align="center"><font color="blue">Hello World!!!!</font></h1>

</body>
</html>
```

Output:


Spring Initializr
Hello World
localhost:8080/vinay
Hello World!!!!
29°C
Mostly clear
Search
ENG
IN
20:29
31-08-2023