R.Anusha(AF0311774)

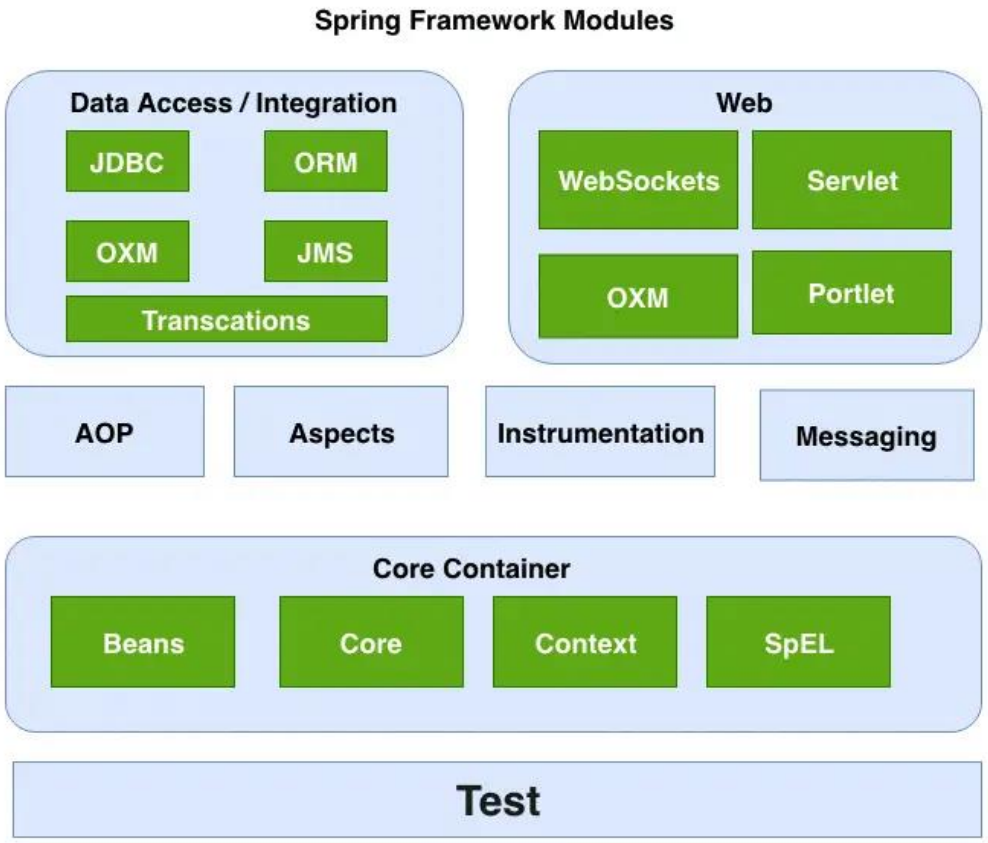
Q1. Explain the architecture of Spring Framework

The Spring framework is a widely-used open-source Java framework that provides a comprehensive programming and configuration model for building enterprise applications. Its architecture is designed around two core principles: Dependency Injection (DI) and Aspect-Oriented Programming (AOP).

The Spring framework consists of several modules, which can be categorized into four main areas: Core Container, Data Access/Integration, Web, and Miscellaneous. The Core Container provides the fundamental functionality of the Spring framework, including the IoC container and ApplicationContext. The Data Access/Integration area provides support for integrating with databases and other data sources. The Web area provides support for building web applications, including the Spring MVC and Spring WebFlux modules. The Miscellaneous area includes other modules that provide additional functionality, such as the Spring Security module for authentication and authorization features.

- Dependency Injection (DI) is a design pattern that helps to reduce the coupling between the components of an application. By using DI, the Spring framework enables loose coupling between components, making the application more modular and easier to maintain. The Spring framework provides an Inversion of Control (IoC) container, which is responsible for creating and managing instances of JavaBeans, and the ApplicationContext, which provides a unified view of the entire application configuration.
- Aspect-Oriented Programming (AOP) allows developers to modularize cross-cutting concerns, such as logging, security, and transaction management, and apply them to multiple components of an application. This results in a more modular and reusable codebase. The Spring framework provides an AOP framework, which is responsible for managing cross-cutting concerns in the application, such as logging, security, and transaction management.
- Overall, the Spring framework architecture is based on the principles of modularity, separation of concerns, and flexibility, providing developers with a powerful set of tools to build robust, scalable, and maintainable enterprise applications. The framework's modular architecture allows developers to select only the necessary modules for their specific needs, reducing unnecessary overhead and complexity in the application. Additionally, the Spring framework's flexible configuration model allows developers to configure the application using various approaches, such as XML-based configuration, Java-based configuration, and annotation-based configuration

Spring Modules



The Spring framework is modular and consists of several modules that provide different functionalities to help build enterprise applications. The modules can be broadly categorized into four main areas: **Core Container, Data Access/Integration, Web, and Miscellaneous.** Let's take a closer look at each area and its corresponding modules:

Core Container

- The Core Container provides the fundamental functionality of the Spring framework, including the Inversion of Control (IoC) container and the ApplicationContext. It includes the following modules:
- **Spring Core:** This module provides the fundamental functionality of the Spring framework, including IoC and DI. The IoC container is the heart of the Spring Framework, responsible for creating and managing instances of JavaBeans. It uses dependency injection to wire the beans together.
- **Spring Beans:** This module provides the BeanFactory, which is the basic building block of the IoC container, and the BeanWrapper, which is responsible for managing the lifecycle of a bean. The Bean Factory is

the core interface for accessing the IoC container. It provides methods for retrieving beans.

- **Spring Context:** This module provides the ApplicationContext, which is an advanced version of the BeanFactory and provides additional features, such as internationalization and resource loading, and the ability to publish and consume events.
- **Spring Expression Language (SpEL):** This module provides a powerful expression language for querying and manipulating objects during runtime. SpEL supports a wide range of features, including property access, method invocation, conditionals, loops, and type conversion. It also provides support for accessing variables and functions defined in the application context, as well as support for defining custom functions and variables.

Data Access/Integration:

The Data Access/Integration area provides support for integrating with databases and other data sources. It includes the following modules:

- **Spring JDBC:** This module provides a simple JDBC abstraction layer that reduces the amount of boilerplate code required to work with JDBC. Spring JDBC provides support for transaction management, allowing developers to manage database transactions declaratively using Spring's transaction management.
- **Spring ORM:** This module provides integration with Object-Relational Mapping (ORM) frameworks, such as Hibernate and JPA. Spring ORM provides a higher-level abstraction layer on top of ORM frameworks, allowing developers to write less boilerplate code and more easily integrate ORM technologies with other Spring features, such as transaction management and caching.
- **Spring Data:** This module provides a consistent and easy-to-use programming model for working with data access technologies, including databases, NoSQL, and cloud-based data services. Spring Data provides a wide range of features, including automatic CRUD (Create, Read, Update, Delete) operations, query generation from method names, support for pagination and sorting, integration with Spring's transaction management, and more. Additionally, Spring Data provides support for common data access patterns, such as repositories and data access objects (DAOs).
- **Spring Transaction:** This module provides support for declarative transaction management in Spring applications. Spring Transaction provides support for various transaction propagation and isolation levels, allowing developers to manage transactions at different levels of granularity. Additionally, Spring Transaction provides support for different transaction management strategies, such as using a JTA transaction manager or a simple JDBC transaction manager.

Web:

The Web area provides support for building web applications. It includes the following modules:

- **Spring MVC:** This module provides a Model-View-Controller (MVC) framework for building web applications. Spring MVC provides a range of features, including support for handling HTTP requests and responses, form handling, data binding, validation, and more. It also provides support for different view technologies, such as JSP (JavaServer Pages), Thymeleaf, and Velocity, allowing developers to choose the view technology that best suits their needs.
- **Spring WebFlux:** This module provides a reactive programming model for building web applications that require high concurrency and scalability. Spring WebFlux provides support for building reactive web applications using a range of technologies, such as Netty, Undertow, and Servlet 3.1+ containers. It also provides a range of features, including support for reactive data access, reactive stream processing, and reactive HTTP clients.
- **Spring Web Services:** This module provides support for building SOAP-based and RESTful web services. Spring Web Services provides support for generating WSDL (Web Services Description Language) from Java classes, and for generating Java classes from WSDL. This allows developers to define the contract (i.e., the interface) of their web service using WSDL, and to generate the Java classes that implement the web service from the WSDL.

Miscellaneous:

The Miscellaneous area includes other modules that provide additional functionality, such as:

- **Spring Security:** This module provides authentication and authorization features for Spring applications. Spring Security provides a range of authorization mechanisms, such as role-based access control and expression-based access control. It also provides support for securing different parts of the application using different security configurations, allowing developers to apply fine-grained security policies.
- **Spring Integration:** This module provides support for building message-driven and event-driven architectures. Spring Integration provides a range of integration patterns, such as messaging, routing, and transformation. It provides support for a range of messaging systems, such as JMS, AMQP, and Apache Kafka. It also provides support for integrating with different protocols, such as FTP, HTTP, and TCP.
- **Spring Batch:** This module provides support for batch processing and integration with enterprise systems. Spring Batch provides a range of

tools and utilities for building and managing batch processing applications, such as support for testing and debugging batch jobs, logging and monitoring, and integration with other Spring modules, such as Spring Data and Spring Integration.

- **Spring Cloud:** This module provides support for building cloud-native applications using Spring technologies. Spring Cloud provides a range of features for building cloud-native applications, such as service discovery, configuration management, and load balancing. It provides support for integrating with different cloud platforms, such as AWS and GCP, and for using different cloud-native technologies, such as containers and serverless computing.

Q2. 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.

HelloWorldApplication.java:

```
package com.example.HelloWorld;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class HelloWorldApplication {

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

HelloWorldController:

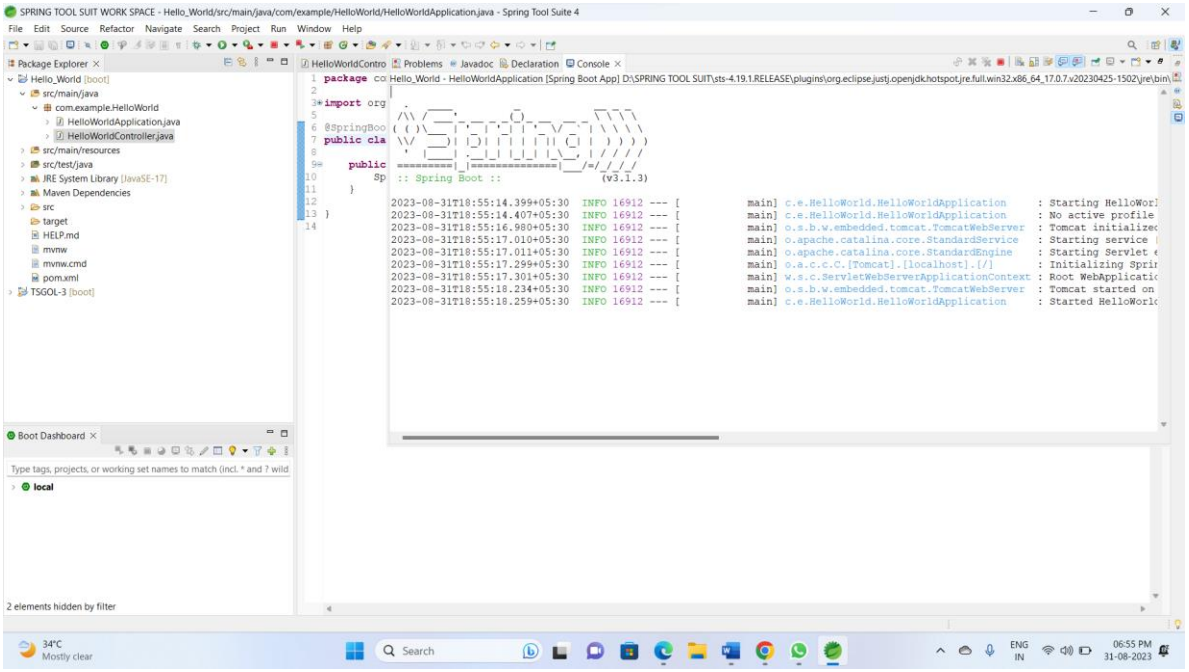
```
package com.example.HelloWorld;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class HelloWorldController {

@GetMapping("/")
@ResponseBody
public String helloWorld() {
```

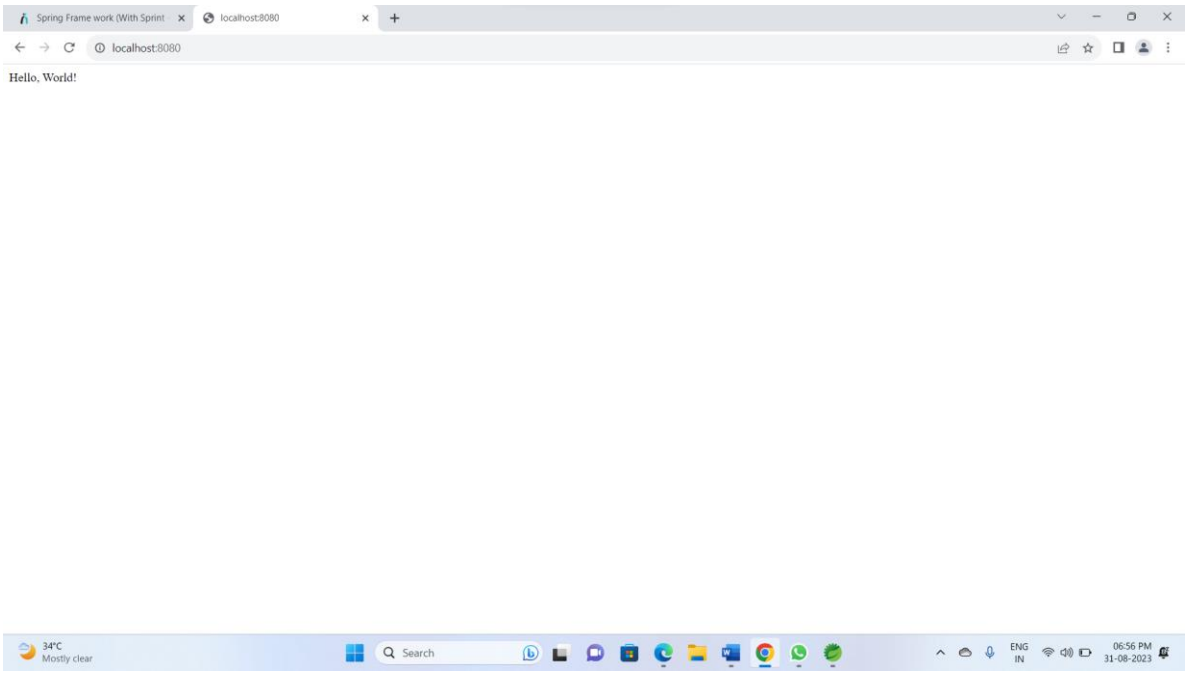
```
return "Hello, World!";
}
}
```

Hello.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Hello World</title>
</head>
<body>
<h1>Hello World</h1>
</body>
</html>
```

Output:




Spring Frame work (With Sprint
localhost:8080
localhost:8080
Hello, World!
34°C
Mostly clear
Search
ENG
IN
06:56 PM
31-08-2023