Giới thiệu tổng quan về Spring Framework

Website chính thức của Spring

https://Spring.io

Mục tiêu của Spring Framework

Đơn giản hóa phát triển ứng dụng:

Spring Framework tập trung vào việc giảm bớt sự phức tạp của phát triển ứng dụng Java bằng cách cung cấp các khái niệm và công cụ giúp làm giảm boilerplate code, tạo điều kiện thuận lợi cho việc xây dựng các ứng dụng mạnh mẽ mà không cần đầu tư quá nhiều công sức.

Mục tiêu của Spring Framework

Hỗ trợ các nguyên tắc của lập trình hướng đối tượng (OOP)

Spring Framework khuyến khích việc sử dụng các nguyên tắc của lập trình hướng đối tượng như dependency injection (DI), inversion of control (IoC) và hợp thành (composition) để tạo ra mã nguồn dễ đọc, linh hoạt và dễ bảo trì.

Mục tiêu của Spring Framework

Tích hợp dễ dàng:

Spring Framework cung cấp các công cụ và tính năng để tương tác và tích hợp với các công nghệ và framework khác. Nó hỗ trợ việc tích hợp với các công nghệ như Java Enterprise Edition (Java EE), Hibernate, JPA, JDBC, RESTful Web Services, Thymeleaf, và nhiều hơn nữa.

Mục tiêu của Spring Framework

Hỗ trợ cho các khía cạnh phát triển ứng dụng:

Spring Framework cung cấp nhiều module và tính năng để hỗ trợ các khía cạnh phát triển ứng dụng như xử lý yêu cầu web, quản lý giao dịch, xác thực và phân quyền, xử lý dữ liệu, và nhiều khía cạnh khác.

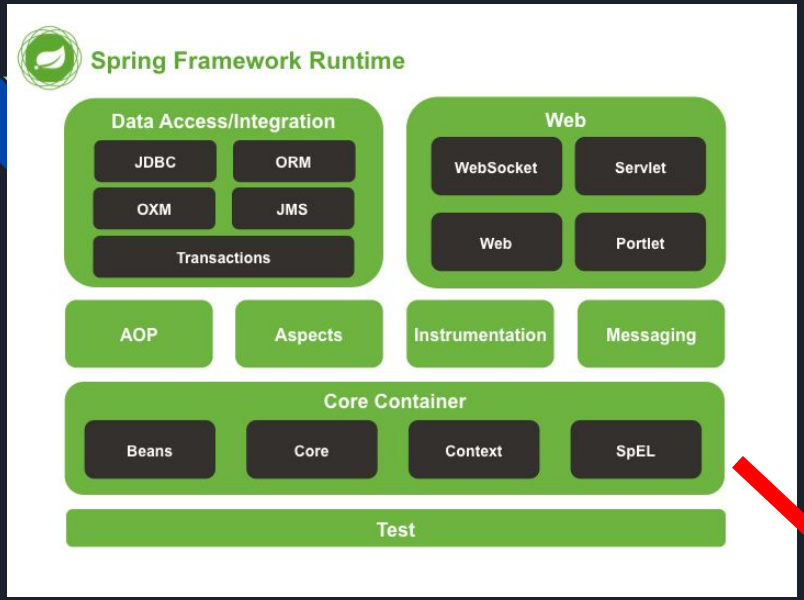
Mục tiêu của Spring Framework

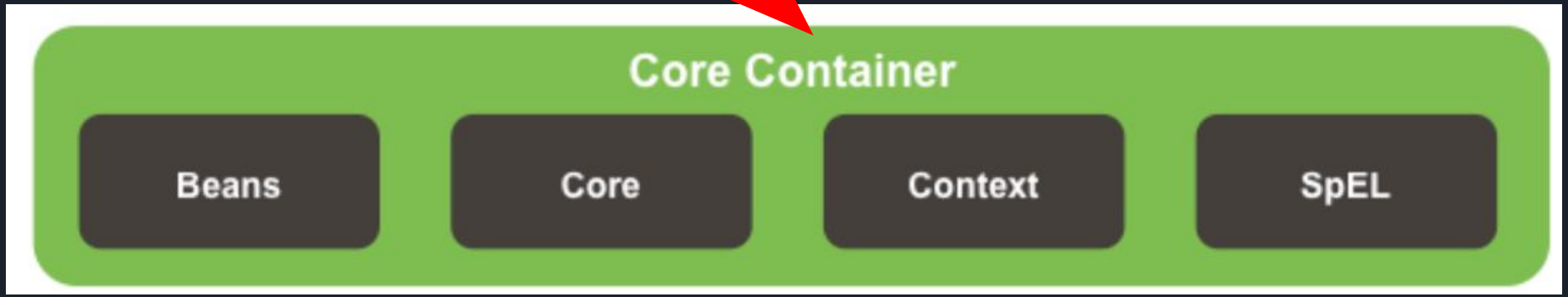
Tóm lại, mục tiêu của Spring Framework là đơn giản hóa và tối ưu hóa quá trình phát triển ứng dụng Java, giúp nhà phát triển xây dựng các ứng dụng linh hoạt, dễ dùng và dễ bảo trì.

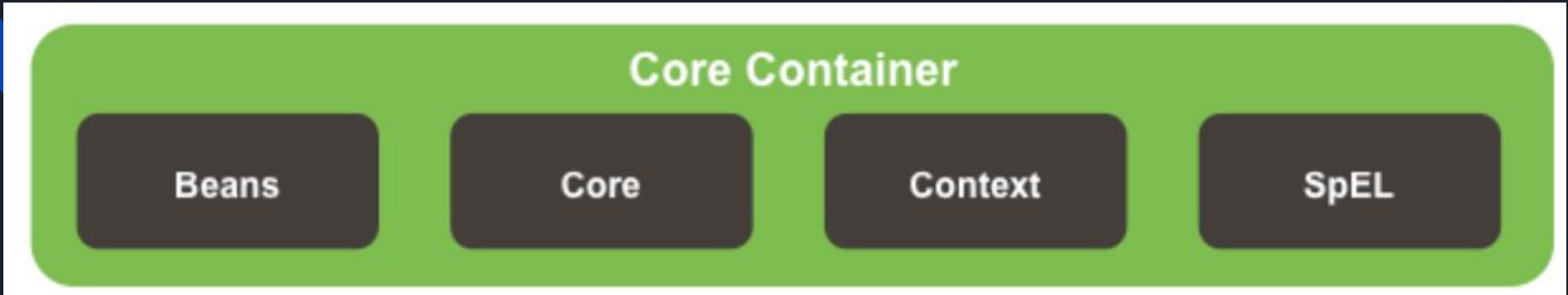
Mục tiêu của Spring Framework

Tích hợp dễ dàng với container:

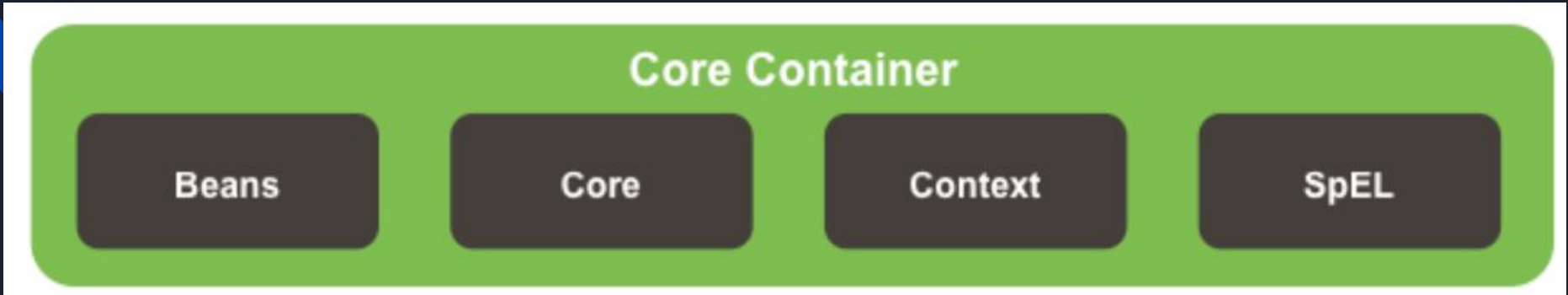
Spring Framework hỗ trợ việc triển khai ứng dụng trên các container như Apache Tomcat, Jetty, IBM WebSphere, và môi trường Java Standalone.


Spring Framework Runtime
Data Access/Integration
JDBC
ORM
OXM
JMS
Transactions
Web
WebSocket
Servlet
Web
Portlet
AOP
Aspects
Instrumentation
Messaging
Core Container
Beans
Core
Context
SpEL
Test

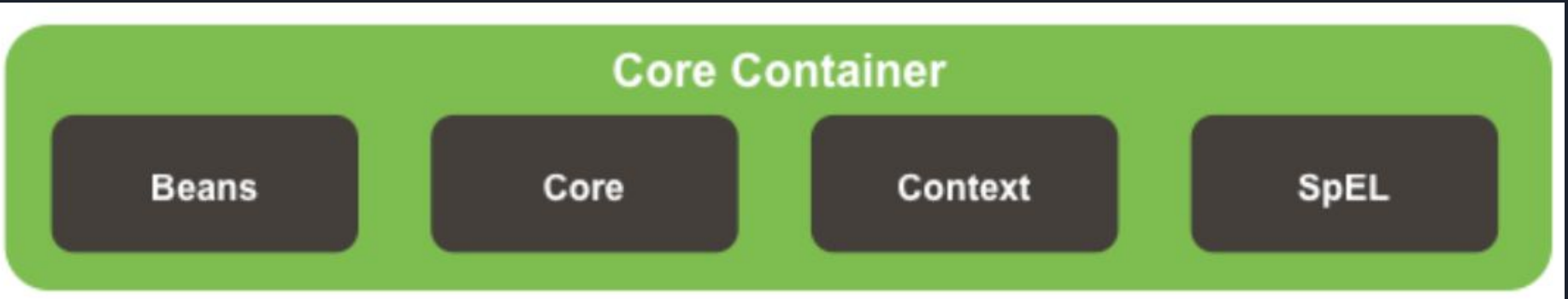

Core Container
Beans
Core
Context
SpEL



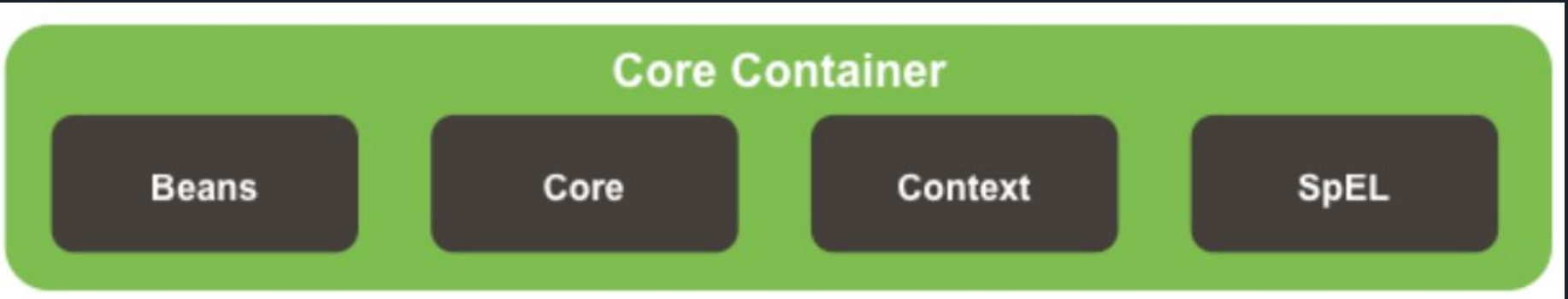
Bean: Bean là thành phần cốt lõi trong Core Container của Spring và là trách nhiệm chính của Spring Container. Bean đại diện cho các đối tượng trong ứng dụng được quản lý bởi Spring. Spring Container sẽ tạo, cấu hình và quản lý các Bean. Các Bean được khai báo và cấu hình thông qua các tệp cấu hình XML, Annotations hoặc Java-based configuration.



Core: Core là một thành phần quan trọng khác trong Core Container. Nó chứa các lớp và giao diện cung cấp các tính năng cốt lõi của Spring như dependency injection (DI) và inversion of control (IoC). Core cung cấp các lớp và giao diện như ApplicationContext, BeanFactory, ResourceLoader, và các lớp tiện ích khác để hỗ trợ việc quản lý và tạo ra các Bean.

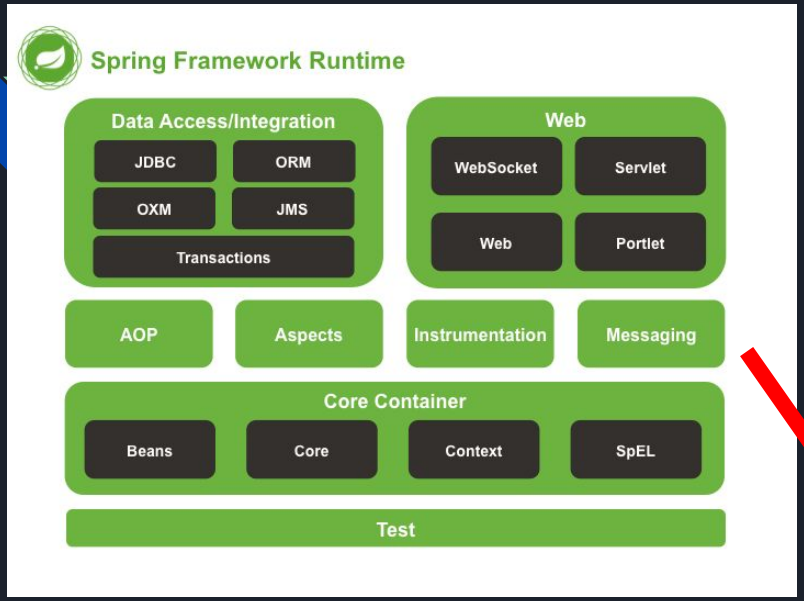


Context: Context là thành phần thứ ba trong Core Container của Spring. Nó cung cấp các tính năng phục vụ cho việc tạo và quản lý các Bean, bao gồm cả xử lý sự kiện, quản lý giao dịch và quản lý chuỗi thông báo. Context được xây dựng dựa trên Core và cung cấp các chức năng mở rộng để giúp quản lý các Bean trong quá trình thực thi ứng dụng.



SpEL (Spring Expression Language) là một ngôn ngữ biểu thức mạnh trong Core Container của Spring. SpEL được sử dụng trong các tệp cấu hình XML, Annotations và Java-based configuration để thực hiện các tác vụ như:

- Truy cập và đặt giá trị thuộc tính của đối tượng.
- Gọi các phương thức của đối tượng.
- Thực hiện các phép tính và toán tử.
- Xử lý biểu thức điều kiện và vòng lặp.
- Tương tác với các tài nguyên bên ngoài như properties file, environment variables, hoặc các bean khác trong Spring Container.


Spring Framework Runtime
Data Access/Integration
JDBC
ORM
OXM
JMS
Transactions
Web
WebSocket
Servlet
Web
Portlet
AOP
Aspects
Instrumentation
Messaging
Core Container
Beans
Core
Context
SpEL
Test

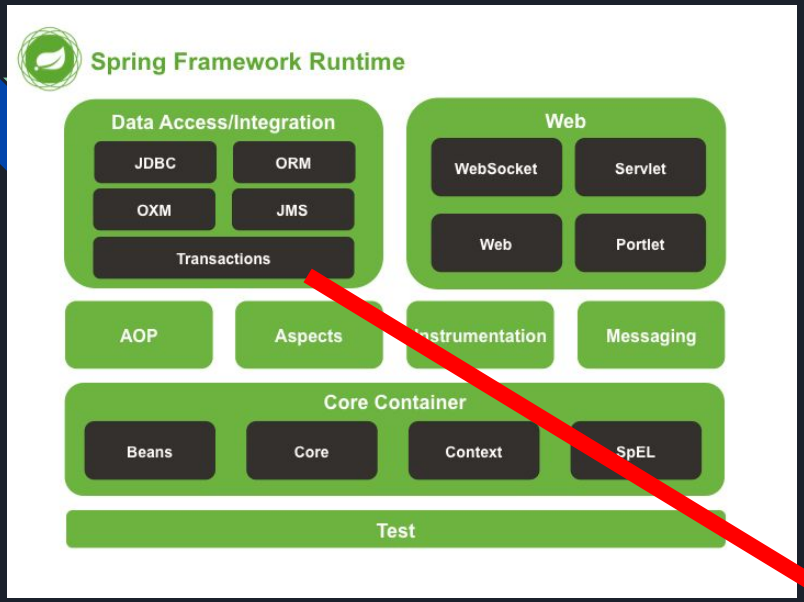

AOP
Aspects
Instrumentation
Messaging

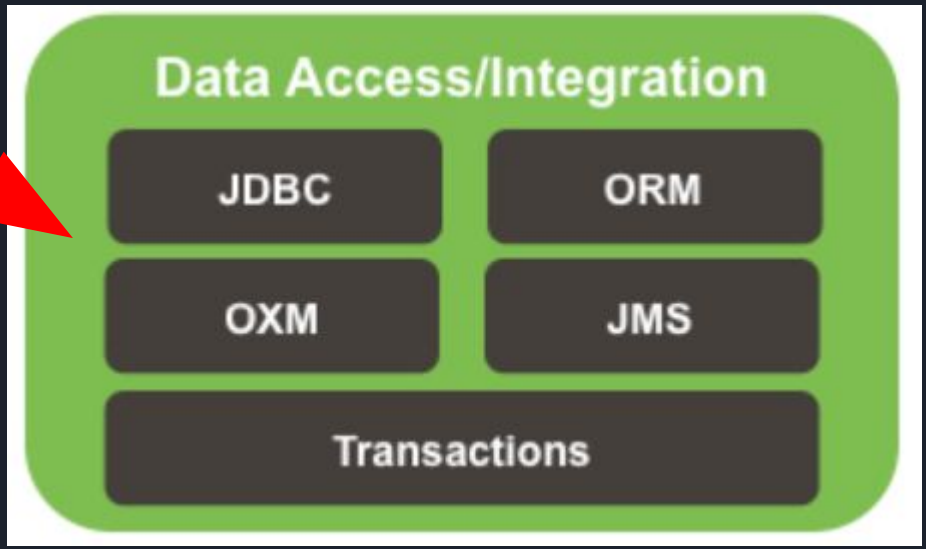
AOP (Aspect-Oriented Programming) là một mô hình lập trình được sử dụng trong Spring Boot để tách biệt các khía cạnh (concerns) khác nhau trong ứng dụng. Nó cho phép bạn tách riêng các mục đích của mã như logging, giao dịch, bảo mật, caching và xử lý ngoại lệ, và áp dụng chúng vào mã của bạn mà không cần thay đổi trực tiếp mã đó.

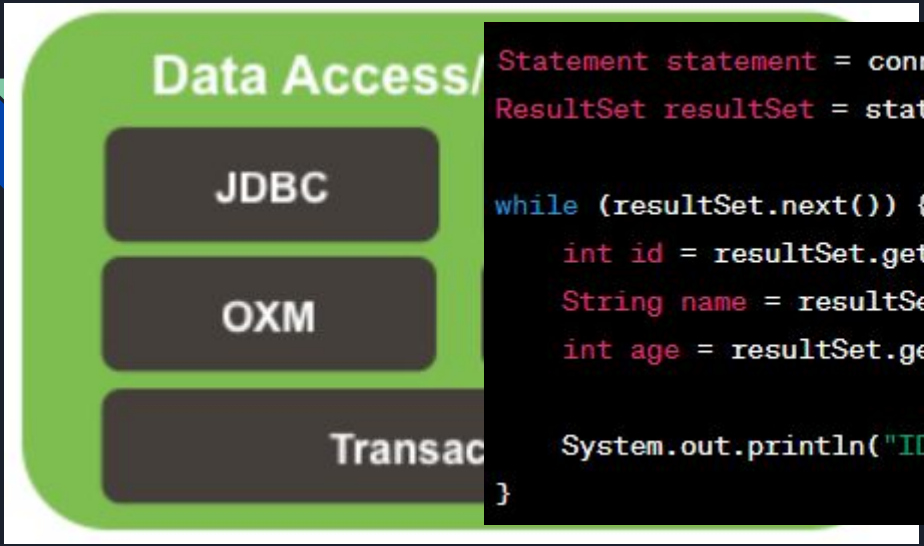
Aspect: Aspect là một đối tượng chứa mã thực hiện các khía cạnh chung như logging, giao dịch, bảo mật, và nhiều hơn nữa. Aspect được xác định bằng cách sử dụng annotations hoặc XML configuration.

"Instrumentation" có thể được hiểu như việc mở rộng hoặc tăng cường chức năng của một ứng dụng, đặc biệt là các khía cạnh liên quan đến giám sát (monitoring), phân tích (analytics) và ghi lại (logging).

"Message" thường liên quan đến việc truyền thông tin hoặc dữ liệu giữa các thành phần của hệ thống. Nó có thể ám chỉ việc gửi và nhận các thông điệp (message) qua các giao thức như HTTP, JMS (Java Message Service), AMQP (Advanced Message Queuing Protocol), Kafka, v.v. Spring Boot cung cấp nhiều cơ chế để hỗ trợ giao tiếp qua tin nhắn, bao gồm Spring Integration, Spring Cloud Stream và Spring AMQP.

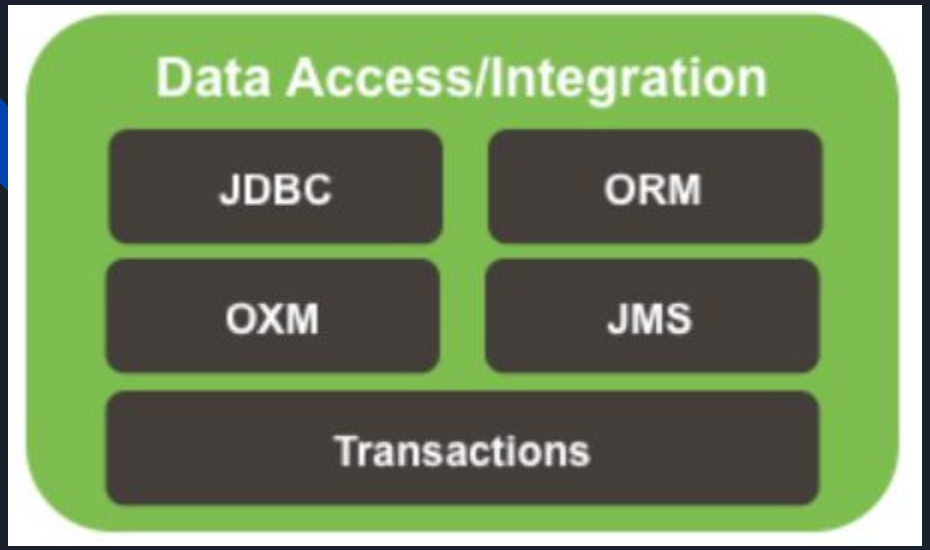

Spring Framework Runtime
Data Access/Integration
JDBC
ORM
OXM
JMS
Transactions
Web
WebSocket
Servlet
Web
Portlet
AOP
Aspects
Instrumentation
Messaging
Core Container
Beans
Core
Context
SpEL
Test


Data Access/Integration
JDBC
ORM
OXM
JMS
Transactions



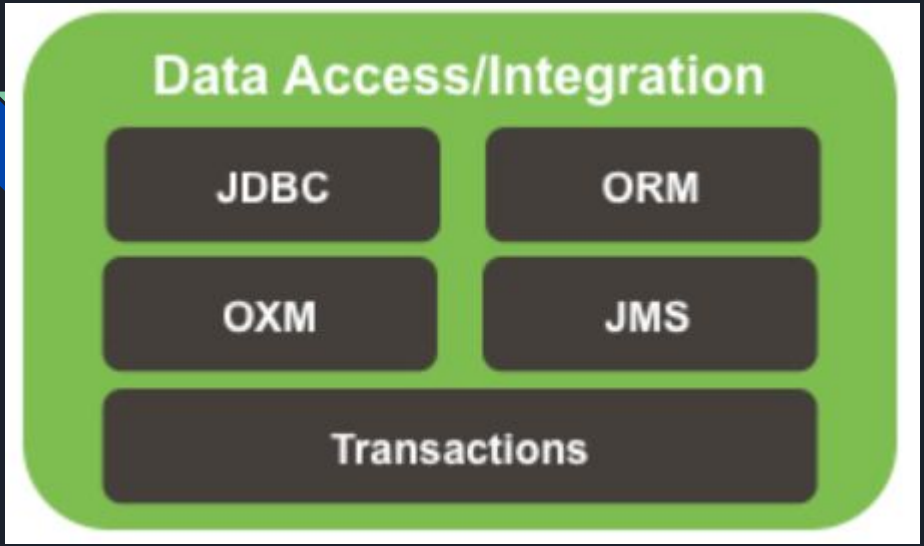

```
Statement statement = connection.createStatement();
ResultSet resultSet = statement.executeQuery("SELECT * FROM employees");

while (resultSet.next()) {
    int id = resultSet.getInt("id");
    String name = resultSet.getString("name");
    int age = resultSet.getInt("age");

    System.out.println("ID: " + id + ", Name: " + name + ", Age: " + age);
}
```

- JDBC (Java Database Connectivity) là một API (Application Programming Interface) trong Java cho phép các ứng dụng Java tương tác với các cơ sở dữ liệu quan hệ.
- Nó cung cấp một tập hợp các lớp và giao diện để thực hiện các thao tác như kết nối đến cơ sở dữ liệu, truy vấn dữ liệu, thêm, sửa, xóa dữ liệu, và quản lý các tài nguyên liên quan đến cơ sở dữ liệu.
- JDBC là một tiêu chuẩn trong ngôn ngữ Java để giao tiếp với các cơ sở dữ liệu quan hệ phổ biến như MySQL, Oracle, SQL Server, v.v. Điều này cho phép phát triển ứng dụng Java kết nối và làm việc với dữ liệu trong cơ sở dữ liệu một cách dễ dàng và linh hoạt.



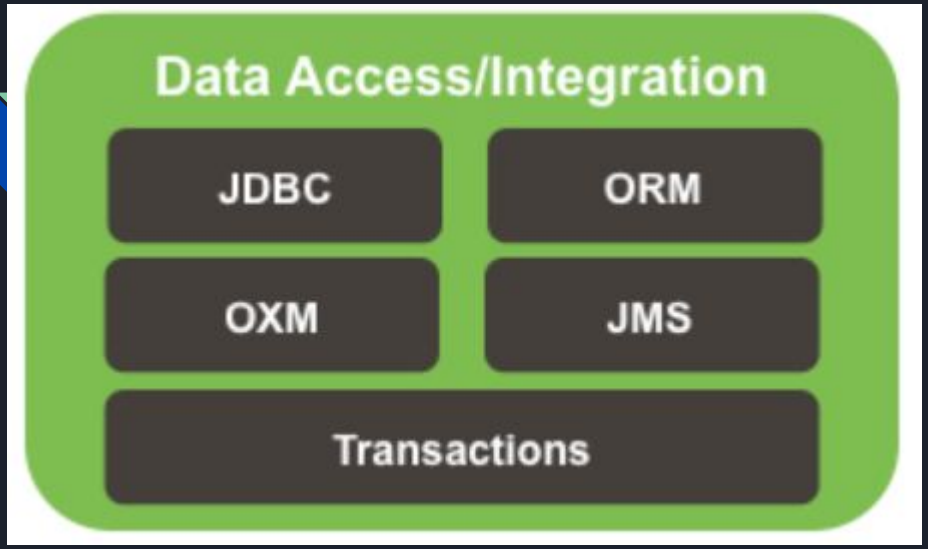

```java
import javax.persistence.*;

@Entity
@Table(name = "employees")
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "name")
    private String name;

    // Getters and setters
}
```

- ORM (Object-Relational Mapping) là một kỹ thuật trong lập trình phần mềm để ánh xạ dữ liệu giữa các đối tượng trong mã lập trình và cơ sở dữ liệu quan hệ. Nó giúp đơn giản hóa và tự động hóa quá trình truy xuất, lưu trữ và tương tác với dữ liệu trong cơ sở dữ liệu từ mã Java (hoặc các ngôn ngữ lập trình khác) mà không cần viết các truy vấn SQL chi tiết.



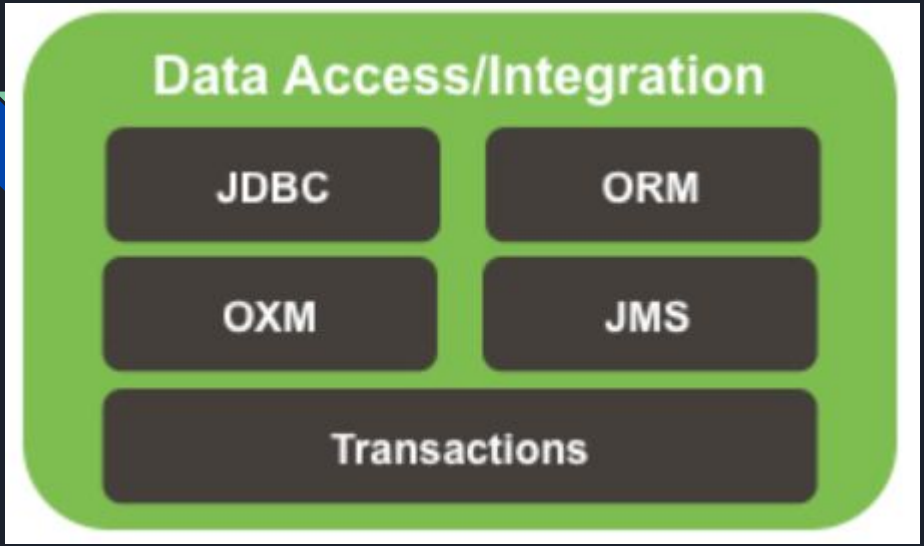

```java
import javax.xml.bind.*;

public class XmlToJavaExample {
    public static void main(String[] args) throws JAXBException {
        String xml = "<person><name>John</name><age>30</age></person>";

        JAXBContext jaxbContext = JAXBContext.newInstance(Person.class);
        Unmarshaller unmarshaller = jaxbContext.createUnmarshaller();

        Person person = (Person) unmarshaller.unmarshal(new StringReader(xml
        System.out.println("Name: " + person.getName());
        System.out.println("Age: " + person.getAge());
    }
}
```

- OXM (Object-XML Mapping) là một kỹ thuật trong lập trình phần mềm để ánh xạ dữ liệu giữa các đối tượng trong mã lập trình và dữ liệu XML. Nó cho phép chuyển đổi (mapping) dữ liệu giữa các đối tượng Java và các tài liệu XML một cách dễ dàng và tự động.



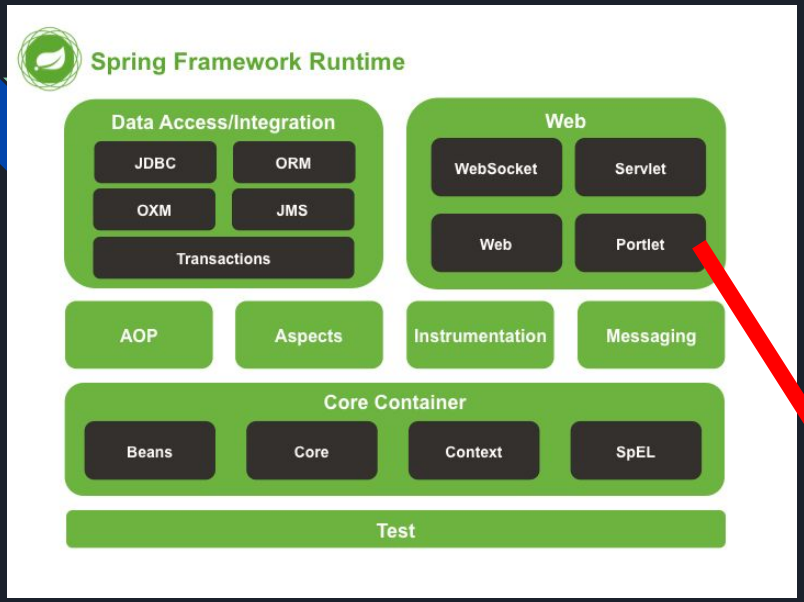

```java
// Chờ và nhận một tin nhắn
Message receivedMessage = consumer.receive();
if (receivedMessage instanceof TextMessage) {
    TextMessage textMessage = (TextMessage) receivedMessage;
    String messageText = textMessage.getText();
    System.out.println("Received message: " + messageText);
}
```

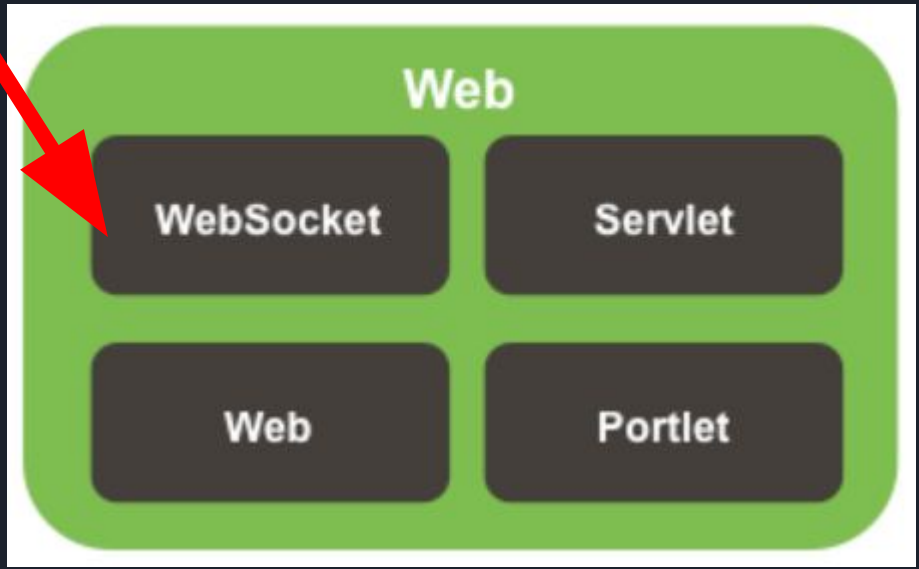
- JMS is a standard Java API that allows a Java application to send messages to another application. It is highly scalable and allows us to loosely couple applications using asynchronous messaging. Using JMS we can read, send and read messages.

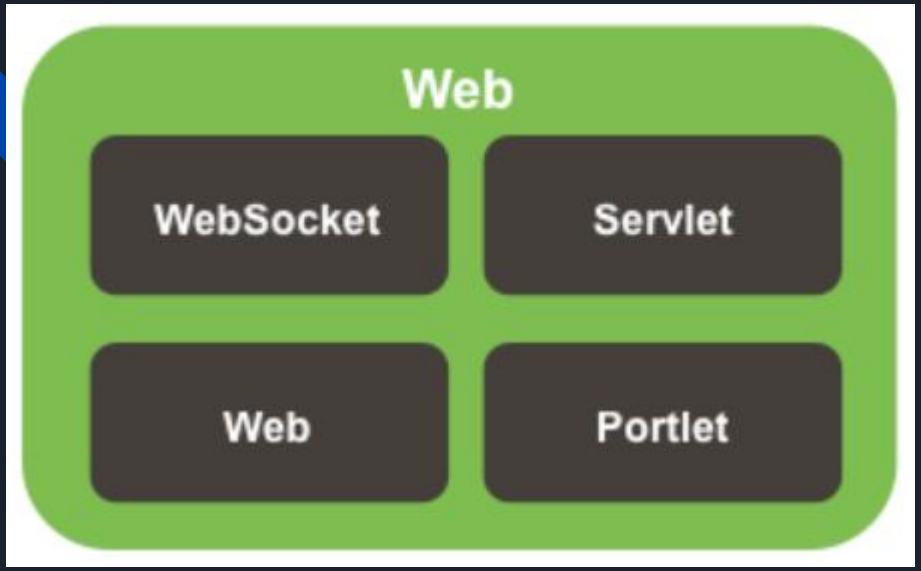



```java
@Service
@Transactional
public class UserServiceImpl implements UserService {
    @Autowired
    private UserRepository userRepository;

    public void saveUser(User user) {
        userRepository.save(user);
    }

    public User getUserById(int id) {
        return userRepository.findById(id);
    }
}
```

- Transactions (giao dịch) là một cơ chế cho phép thực hiện các hoạt động cơ bản của cơ sở dữ liệu một cách an toàn và đáng tin cậy. Giao dịch giúp đảm bảo tính toàn vẹn của dữ liệu bằng cách đảm bảo rằng các thao tác cơ sở dữ liệu được thực hiện hoàn toàn hoặc không thực hiện hoàn toàn.

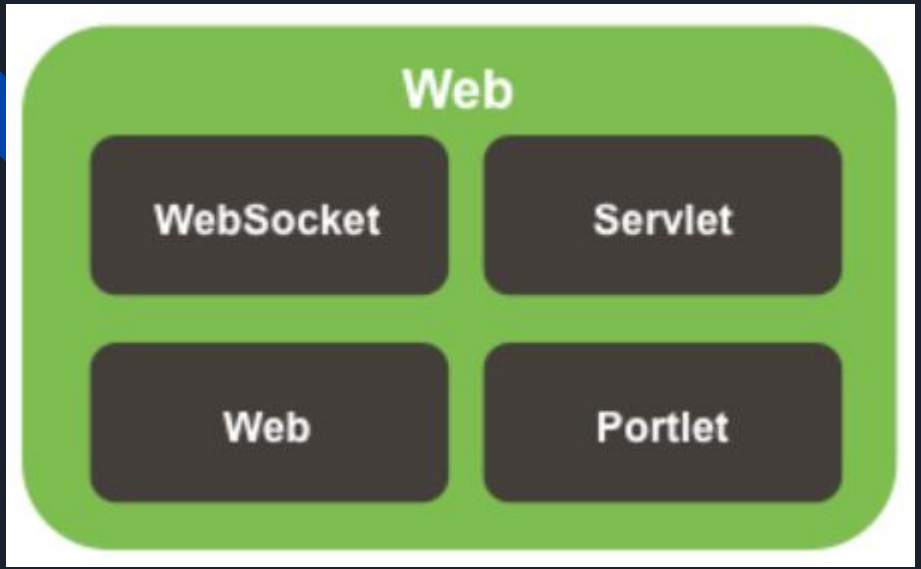

Spring Framework Runtime
Data Access/Integration
JDBC
ORM
OXM
JMS
Transactions
Web
WebSocket
Servlet
Web
Portlet
AOP
Aspects
Instrumentation
Messaging
Core Container
Beans
Core
Context
SpEL
Test


Web
WebSocket
Servlet
Web
Portlet



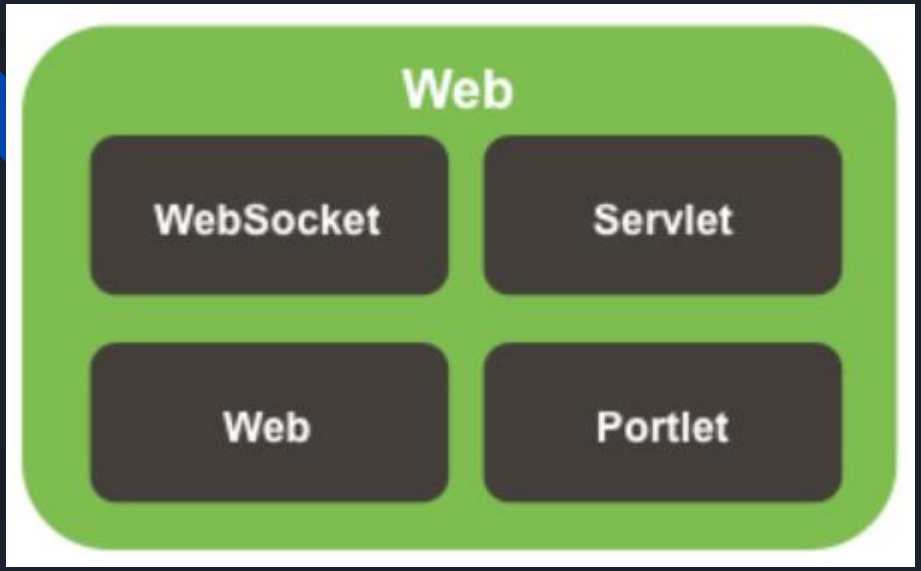

```
const socket = new WebSocket('ws://localhost:8080');

socket.onopen = function() {
  console.log('Kết nối WebSocket đã được thiết lập.');
};

socket.onmessage = function(event) {
  const message = event.data;
  console.log('Nhận dữ liệu từ máy chủ: ' + message);
};

socket.onclose = function() {
  console.log('Kết nối WebSocket đã bị đóng.');
};

const data = 'Dữ liệu cần gửi';
socket.send(data);
```

WebSocket là một giao thức truyền thông hai chiều (full-duplex) dựa trên TCP, cho phép thiết lập kết nối liên tục và thời gian thực giữa một máy khách (client) và một máy chủ (server) trên web. Nó cho phép truyền thông tin hai chiều đồng thời, tức là cả client và server có thể gửi và nhận dữ liệu một cách đồng thời trong một kết nối duy nhất.



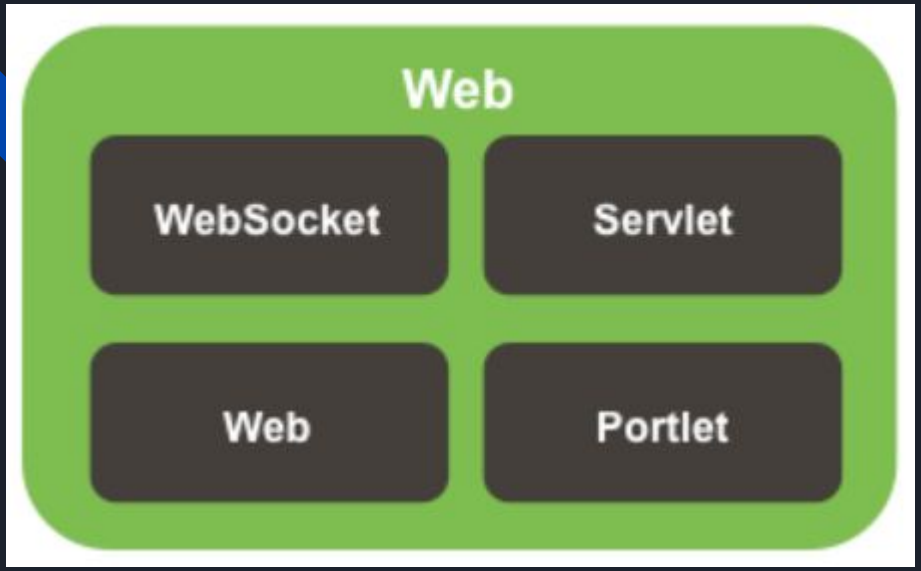
Web Trong Spring Framework, để xây dựng ứng dụng web, bạn có thể sử dụng module Spring Web MVC

```java
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
@RequestMapping("/hello")
public class HelloController {

    @GetMapping
    @ResponseBody
    public String sayHello() {
        return "Xin chào từ Spring MVC!";
    }
}
```



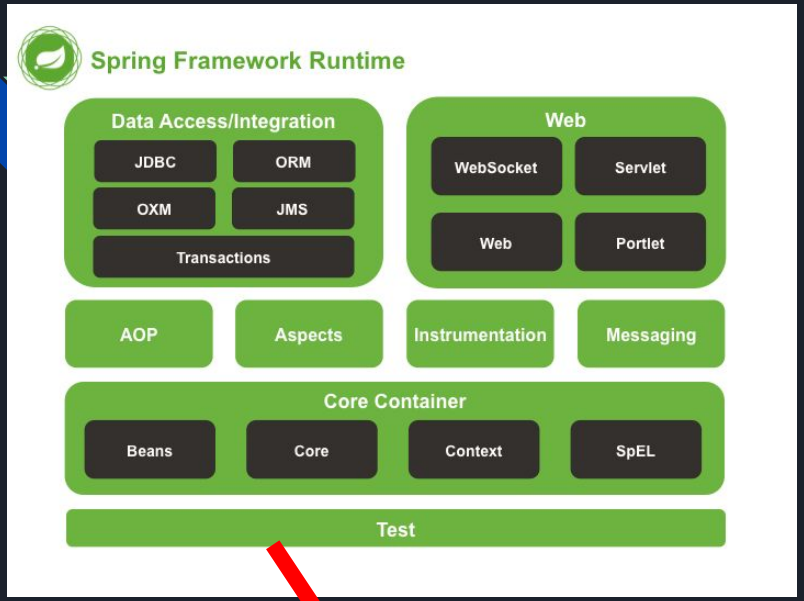

```java
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse
        throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html><body>");
    out.println("<h1>Hello, World!</h1>");
    out.println("</body></html>");
    out.close();
}
```

Servlet là một thành phần của Java Enterprise Edition (Java EE) được sử dụng để xây dựng ứng dụng web. Nó là một lớp Java được viết để xử lý yêu cầu (request) từ client và trả về phản hồi (response) tương ứng.




```
import org.springframework.context.annotation.*;
import org.springframework.web.portlet.config.*;

@Configuration
@EnableWebMvc
public class MyPortletConfig implements WebMvcConfigurer {

    // Cấu hình khác (nếu có)
}
```

Portlet là một khái niệm trong Java Portlet Specification (JSR-286) và được sử dụng để xây dựng các ứng dụng web phù hợp với mô hình portal.



Unit
Integration
Mock