Spring MVC
(Model-View-Controller)

LAHIRU
LIYANAPATHIRANA

What is Spring MVC?

Spring MVC is a powerful web framework that is part of the Spring ecosystem.

It is built on top of the **Model-View-Controller (MVC)** architectural pattern.

Spring MVC simplifies web development with its separation of concern and robust and scalable capabilities.

What is MVC Pattern?

MVC or **Model-View-Controller** pattern is an architectural design approach that organizes an application into three core components: Model, View, and Controller.

Each component is responsible for a distinct part of the application's functionality:

Model:
Handles the application's data and business logic. It manages tasks such as data storage, retrieval, manipulation, and enforcing business logic.

What is MVC Pattern?

View:
Represents the user interface, responsible for displaying data to the user. It generates the visual representation of the data managed by the Model, often using technologies like HTML, CSS, JavaScript, or formats like REST or SOAP.

Controller:
Serves as the mediator between the Model and the View. It processes incoming requests, manages user interactions, and updates both the Model and the View as needed. Additionally, it determines the appropriate View to present to the user.

Benefits of Spring MVC

Spring Support

Since Spring MVC is a part of the Spring ecosystem, it comes with all the benefits of the Spring framework.

Spring Integration

Easily integrates with other Spring projects like Spring Security and Spring Testing.

Flexible Architecture

Supports multiple view technologies, including JSP, Thymeleaf, and frontend frameworks like React and Angular.

Benefits of Spring MVC

Support for REST

Built-in JSON/XML serialization and deserialization for developing RESTful web applications.

Flexible Configuration

Spring MVC supports XML and Annotation based configuration for cleaner and more maintainable code.

Validation and Exception Handling

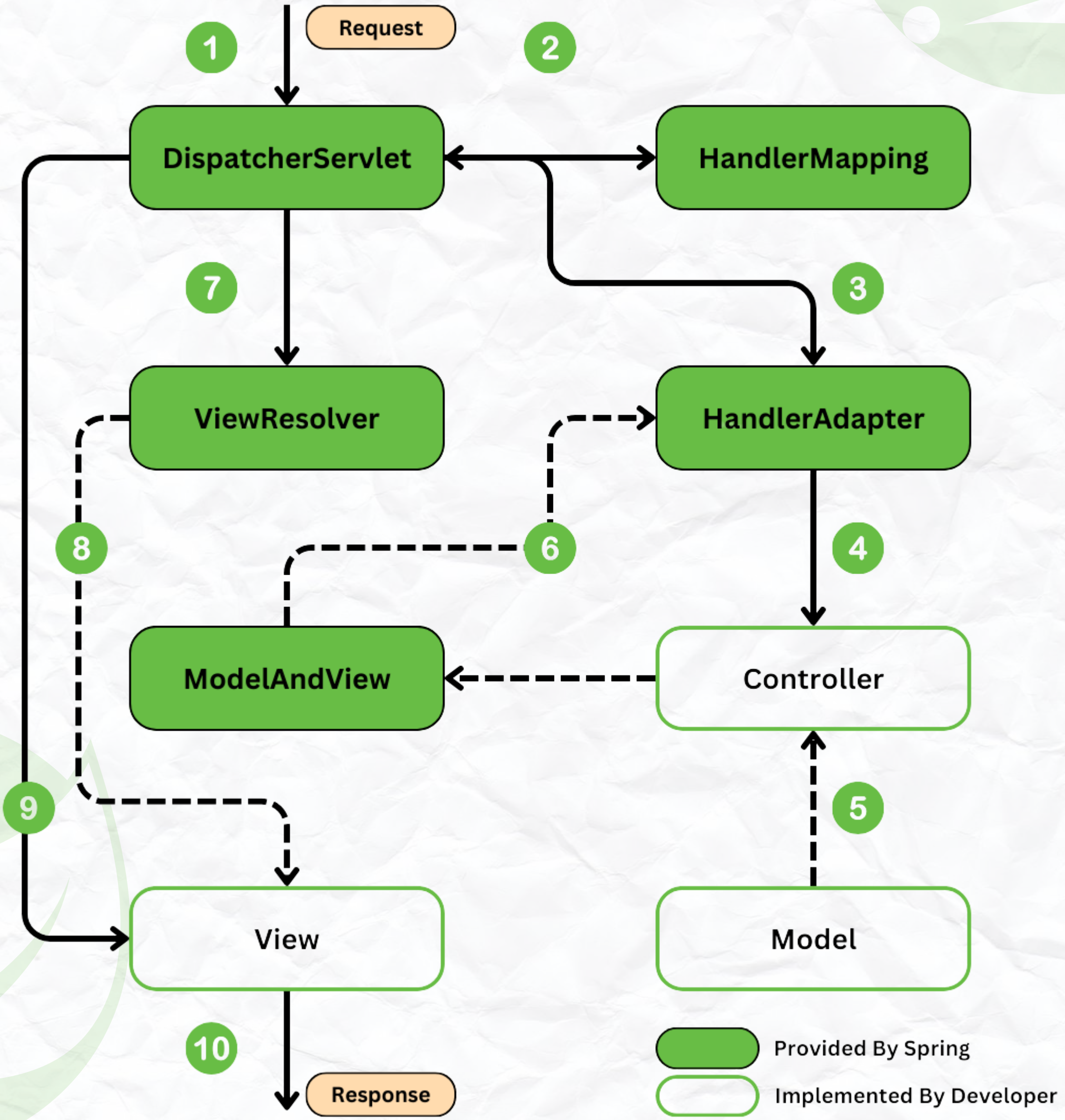
Includes robust validation mechanisms and customizable error handling for user-friendly experiences.

Benefits of Spring MVC

Testability

Enhances unit testing with easy dependency injection and mock integration.

Spring MVC Request Life Cycle



LAHIRU
LIYANAPATHIRANA

Spring MVC Request Life Cycle

1. The **DispatcherServlet** receives the HTTP request.

2. The **DispatcherServlet** dispatches the task of selecting an appropriate Controller to **HandlerMapping**. HandlerMapping selects the controller mapped to the incoming request URL and returns it to the **DispatcherServlet**.

3. The **DispatcherServlet** dispatches the task of executing the business logic of the Controller to **HandlerAdapter**.

4. The **HandlerAdapter** calls the business logic process of the Controller.

5. The Controller executes the business logic and can return **Model**, **ModelAndView**, **String** (view name), or **Response Body**.

LAHIRU
LIYANAPATHIRANA

Spring MVC Request Life Cycle

6. The **Controller** returns the above types, converts them to **ModelAndView**, and returns to the **HandlerAdapter**.

7. The **DispatcherServlet** dispatches the task of resolving the **View** corresponding to the View name to **ViewResolver**.

8. The **ViewResolver** returns the View mapped to the View name.

9. The **DispatcherServlet** dispatches the rendering process to the returned **View**.

10. The **View** renders **Model** data and writes the rendered output directly to the response object.

LAHIRU
LIYANAPATHIRANA

Key Components of Spring MVC

The following are the key components in the Spring MVC architecture.

- DispatcherServlet
- WebApplicationContext
- HandlerMapping
- Controller and RestController
- Model
- ModelAndView
- ViewResolver
- View
- HandlerInterceptors
- HandlerExceptionResolver

DispatcherServlet

The DispatcherServlet is the central component in Spring MVC architecture. It is designed around the **Front Controller pattern**.

It acts as the main entry point for all incoming requests to the Spring MVC application.

Following are the key responsibilities of the DispatcherServlet:

- Manages the complete request-handling workflow.
- Receive all incoming HTTP requests and parses information.

DispatcherServlet

- Collaborates with the HandlerMapping to find the relevant controller for the request.
- Work with the HandlerAdapter to invoke the selected controller.
- Collaborate with the ViewResolver to determine the appropriate view for the request.
- Coordinates the view resolution process and manages the view templates.
- Coordinates with model data transfer to the views.
- Provides the centralized exception and error handling process.

WebApplicationContext

The WebApplicationContext extends the plain ApplicationContext to support web applications.

HandlerMapping

The HandlerMapping maps the incoming requests to the specific controller methods. It determines which controller should be used for the given request URL.

Spring MVC provides several handler mapping implementations such as *RequestMappingHandlerMapping*.

HandlerAdapter

Acts as a bridge between the DispatcherServlet and the Controller. It invokes the Controller picked by the HandlerMapping. It allows the DispatcherServlet to invoke methods on the controller regardless of how the controller is implemented.

Controller

The Controllers are special classes annotated with the @Controller. The controller is responsible for processing user requests and returning the appropriate responses.

RestController

The *@RestController* is a specialized version of the @Controller annotation used for creating RESTful web services.

It combines *@Controller* and *@ResponseBody* and automatically serializes return objects in JSON or XML.

Model

The Model represents the data and business logic of the application.

It encapsulates the data (using POJOs or JavaBeans) used in the application. The model is populated by the controller and passed to the view for rendering.

ModelAndView

The ModelAndView is a holder for both the model and the view in a Spring MVC application and is used by the controller to return both model data and view name to the DispatcherServlet.

ViewResolver

The ViewResolver maps logical view names returned by controllers to actual view implementations such as JSPs. It can be configured via bean names or property files.

Spring MVC provides multiple implementations for ViewResolver such as *InternalResourceViewResolver*, *ThymeleafViewResolver*, etc.

View

The View is responsible for rendering the model data to the user. It receives data from the model and presents it in the given format. Spring MVC supports various view technologies like *JSP*, *Thymeleaf*, *and FreeMarker*.

HandlerInterceptor

The HandlerInterceptor performs operations before and after a request is processed by the controller. This is useful for cross-cutting concerns like logging, security checks, or modifying the model.

Spring MVC provides three main methods for intercepting, *preHandle()*, *postHandle()*, and *afterCompletion()*.

HandlerExceptionResolver

The HandlerExceptionResolver handles exceptions thrown during the request process. It provides a centralized exception-handling mechanism and allows developers to define custom responses or views when exceptions occur.

This enhances the applications' ability to gracefully handle and report errors.

MultipartResolver

The MultipartResolver handles the file uploads and parses multipart request data making it easier to handle file uploads. It supports various file upload libraries and configurations.

LocaleResolver

The LocaleResolver resolves the user's locale to provide internationalization support.

ThemeResolver

The ThemeResolver resolves the theme of the application, allowing for dynamic theme changes.

Did You Find This Post Useful?

Stay Tuned for More Spring Related Posts Like This

LAHIRU
LIYANAPATHIRANA