Learning Consolidation **Implement Inversion of Control (IoC) Inside the Spring Application by Using Annotations**

Learning Objectives

- Explain the Bean Lifecycle
- Explore Spring MVC

Slide Note

Instantiate – The bean factory instantiates the beans in the beans.xml file.

Populate properties – It populates all properties as present in the beans.xml file.

Set bean name – The name of the bean is set in the bean factory that created it by passing bean id to the setBeanName() provided by the BeanNameAware interface.

Set bean factory – The bean is given a reference to the bean factory that manages it by using the setBeanFactory() of the BeanFactoryAware interface.

Pre initialization – Perform tasks before the bean is to be initialized by implementing BeanPostProcessor interface in the bean class and defining its postProcessBeforeInitialization() method.

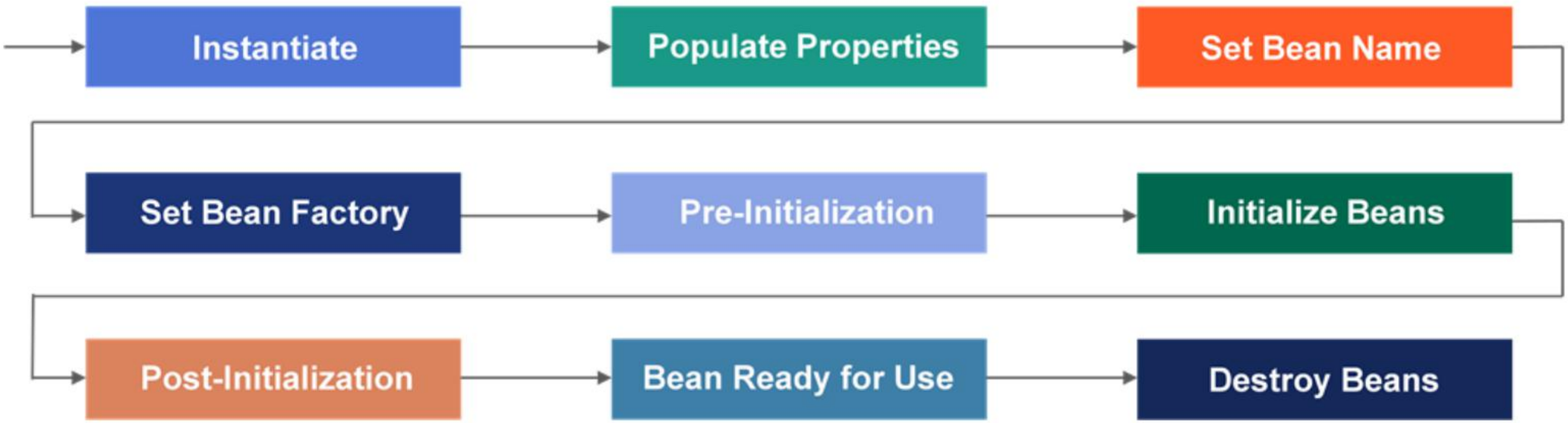
Initialize beans – Perform initialization tasks before the bean is ready to use, such as file opening, establish connection with db, allocating memory, etc. by implementing the InitializingBean interface and defining afterPropertiesSet() method.

Post initialization – Perform tasks after the bean is initialized by implementing BeanPostProcessor interface in the bean class and defining its postProcessAfterInitialization() method.

Bean ready for use – The bean is ready for use and will remain in the bean factory until it is no longer required.

Destroy beans – The bean is destroyed. If the bean implements DisposableBean interface the destroy() method is called.

The Lifecycle of a Bean in a Bean Factory



Slide Note

Instantiate – The bean factory instantiates the beans in the beans.xml file.

Populate properties – Populates all properties as present in the beans.xml file.

Set bean name – The name of the bean is set in the bean factory that created it by passing bean id to the setBeanName() provided by the BeanNameAware interface.

Set bean factory – The bean is given a reference to the bean factory that manages it by using the setBeanFactory() of the BeanFactoryAware interface.

Set Application context – The bean needs to be notified of the application context in which it resides. The setApplicationContext() method of the ApplicationContextAware interface is called.

Pre initialization – Perform tasks before the bean is to be initialized by implementing BeanPostProcessor interface in the bean class and defining its postProcessBeforeInitialization() method.

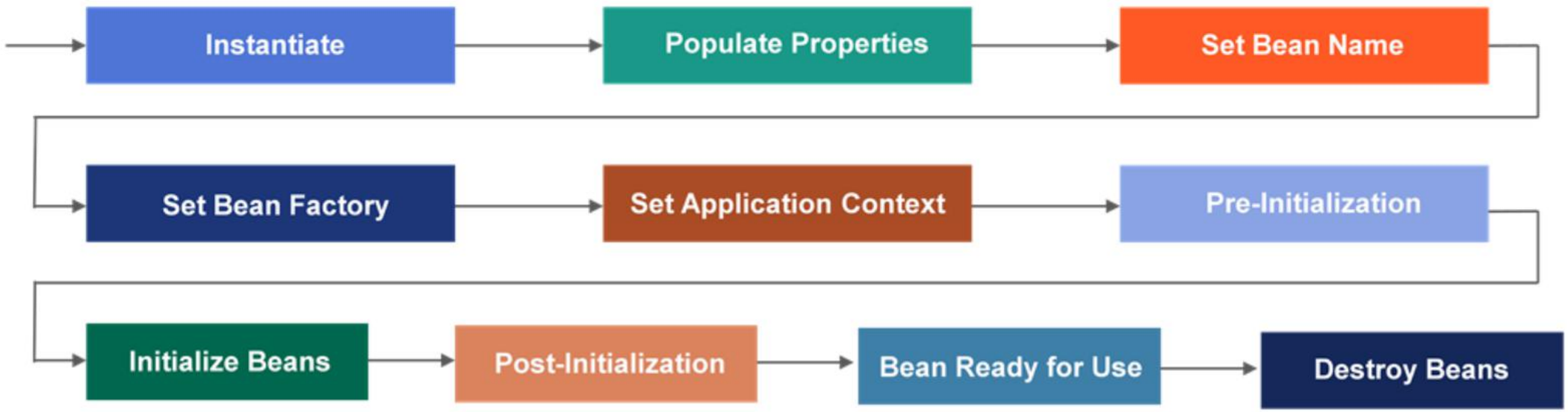
Initialize beans – Perform initialization tasks before the bean is ready to use, such as file opening, establish connection with db, allocating memory, etc. by implementing the InitializingBean interface and defining afterPropertiesSet() method.

Post initialization – Perform tasks after the bean is initialized by implementing BeanPostProcessor interface in the bean class and defining its postProcessAfterInitialization() method.

Bean ready for use – The bean is ready for use and will remain in the bean factory until it is no longer required.

Destroy beans – The bean is destroyed. If the bean implements DisposableBean interface the destroy() method is called.

The Lifecycle of a Bean in the Application Context



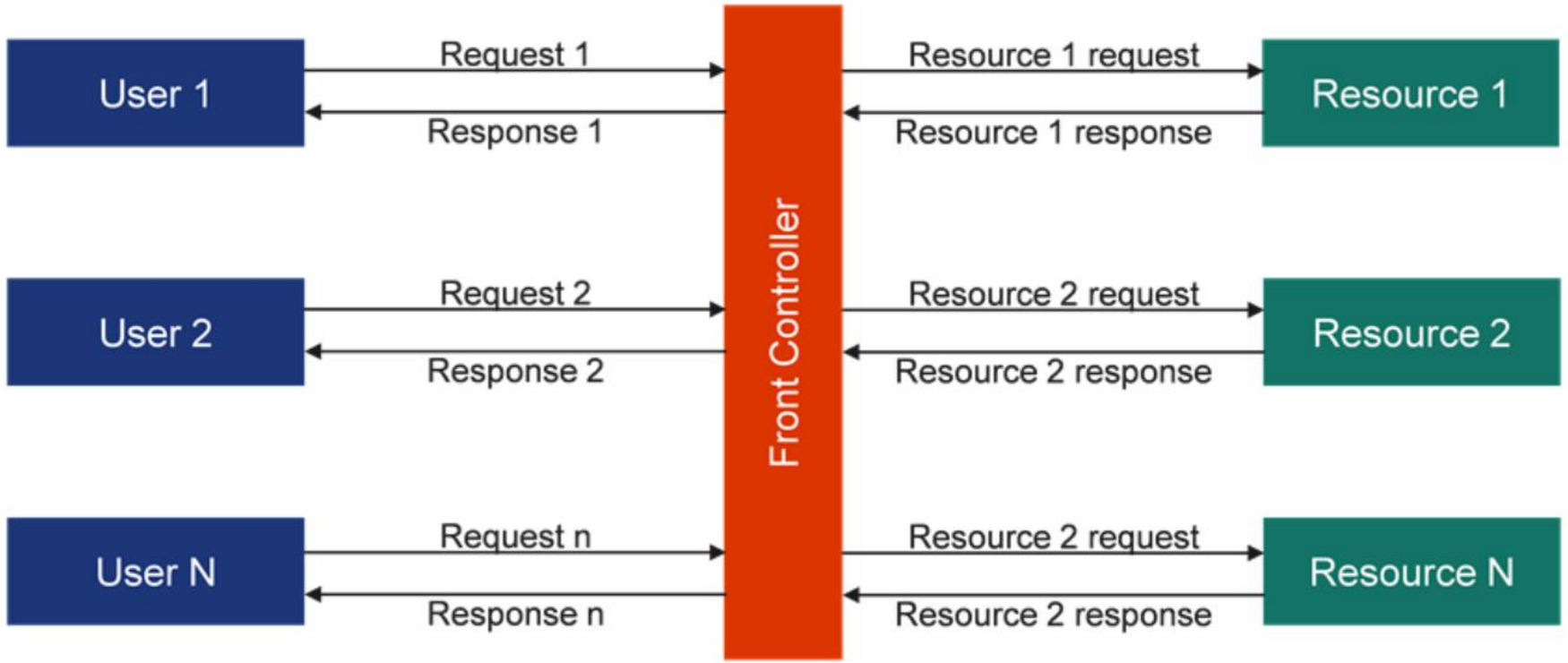
Slide Note

Design Patterns for Web Development

- Front Controller design pattern:
  - The design pattern enforces a single point of entry for all the incoming requests.
  - All requests are handled by a single piece of code, which further delegates the responsibility of processing the request to other objects in the application.
- MVC design pattern
  - MVC is a design pattern that provides a solution to layering an application by separating Business (Model), Presentation (View), and Control Flow (Controller).
  - The Model represents the application data.
  - The View represents the application's user interface.
  - The View takes the Model as the input and renders it appropriately for the end user.
  - The Controller is responsible for handling the request, generating the Model, and selecting the appropriate View for the request.

Slide Note

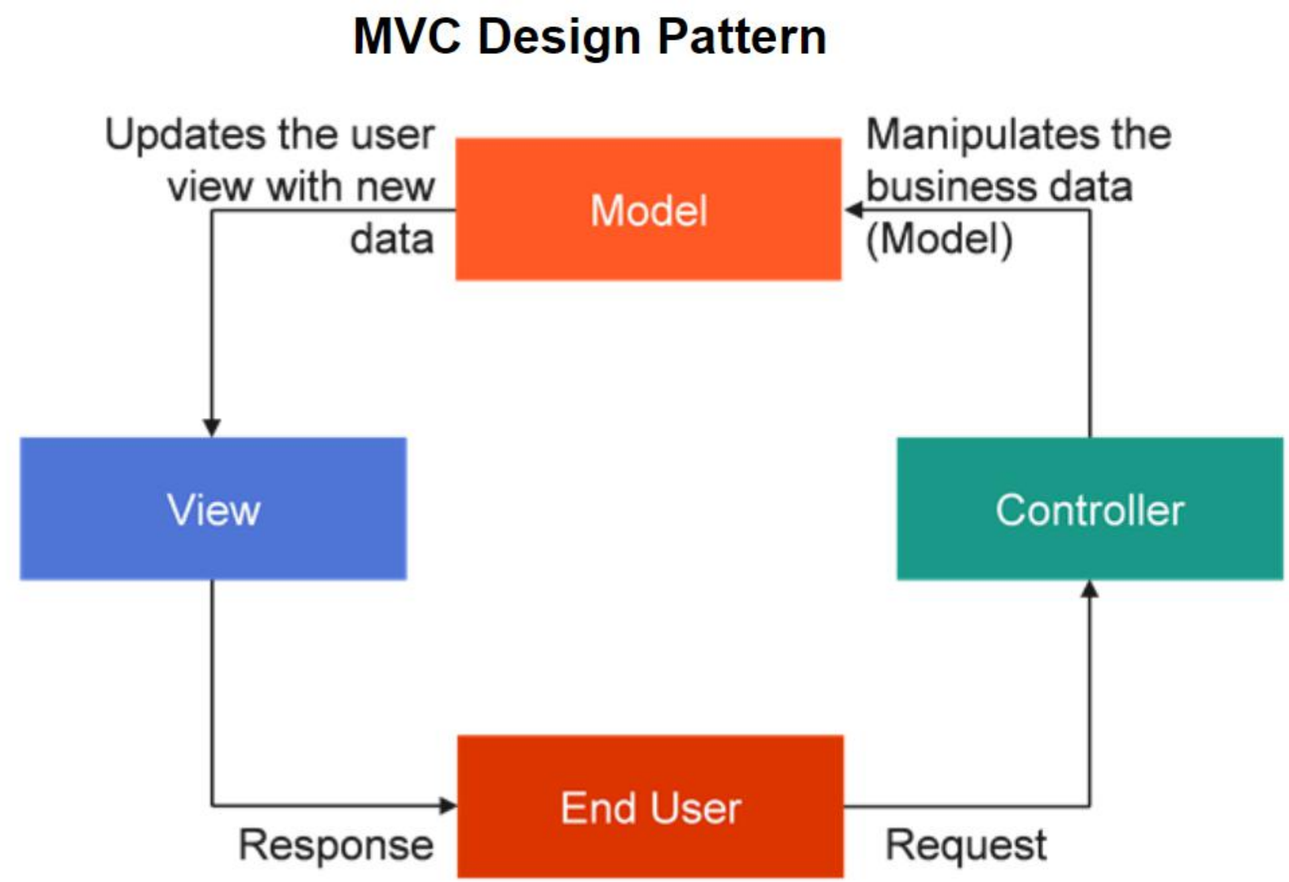
Front Controller Design Pattern



Check the reference.

Slide Note

MVC Design Pattern



Slide Note

MVC is a design pattern that provides a solution to layer an application by separating **Business (Model), Presentation (View)**, and **Control Flow (Controller)**. The **Model** contains the business logic, and the **Controller** is responsible for the redirection and interaction between the **View** and **Model** component contains the presentation part of the application.

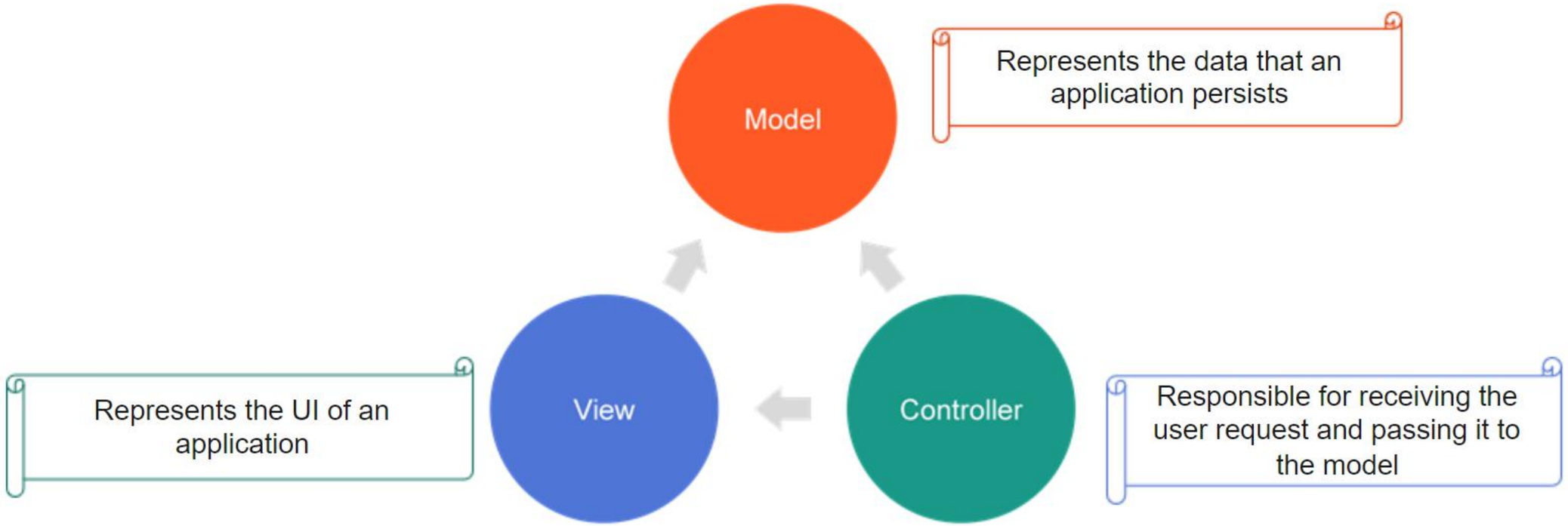
MVC Design Pattern (Cont'd)

- This design pattern helps us develop a loosely coupled application by segregating various concerns into different layers.
- The MVC design pattern enforces the application to be divided into three layers: Model, View, and Controller.
- Model:
  - It represents the application data.
- View:
  - It is the application's user interface. The View takes the Model as an input and renders it appropriately for the end user.
- Controller:
  - It is responsible for handling the request, generating the Model, and selecting the appropriate View for the request.

Slide Note

Introduction to Spring MVC

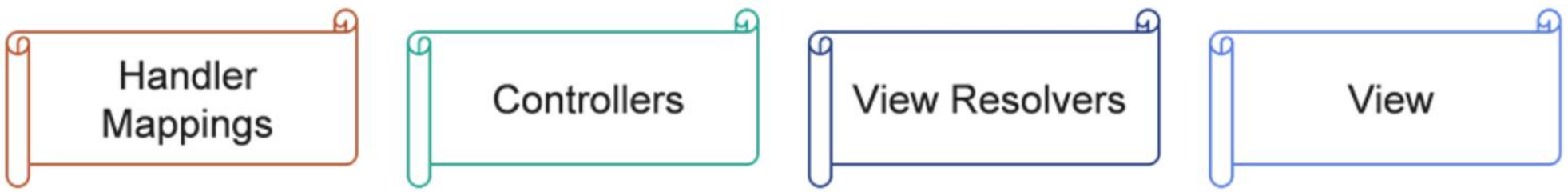
- The core purpose of the MVC pattern is to separate the business logic from UIs to allow them to change independently.
- The Spring Web MVC framework takes advantage of the AOP and DI features of the Spring framework to help create loosely coupled web applications.



Slide Note

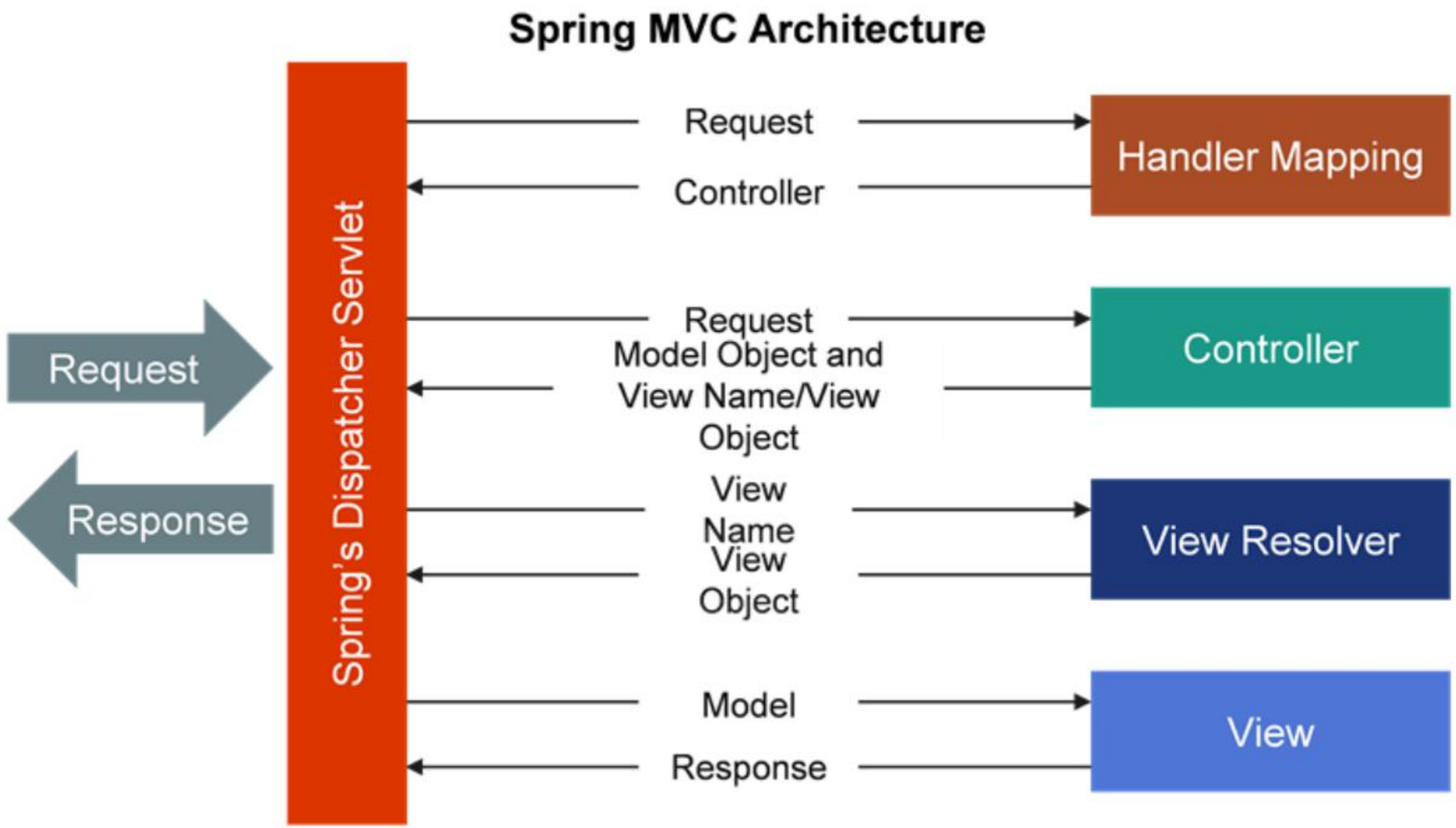
The Spring MVC Framework

- The Spring Web MVC framework is built around a front controller servlet called `DispatcherServlet`.
- Servlets are simple Java classes used to create a web application. They reside on the server side.
- The `DispatcherServlet` intercepts all user requests before passing them to a controller class.
- The `DispatcherServlet` is responsible for delegating the user request to various components of the application while executing a user request.
- The Spring MVC framework makes use of the following components while processing a user request.



Slide Note

Spring MVC Architecture



Slide Note

Spring MVC Architecture (cont'd)

- The front controller design pattern, which comes after the MVC design pattern, is the foundation of Spring's MVC module.
- All the incoming requests are handled by the single servlet named `DispatcherServlet`, which acts as the front controller in Spring's MVC module.
- The `DispatcherServlet` then refers to the `HandlerMapping` to find a controller object that can handle the request.
- `DispatcherServlet` then dispatches the request to the controller object to perform the business logic to fulfill the user's request. (The controller may delegate the responsibility to further application objects known as service objects)

Slide Note

Spring MVC Architecture (cont'd)

- The controller returns an encapsulated object containing the Model object and the View object (or a logical name for the View).
- `ModelAndView` is the class that represents this encapsulated object in Spring's MVC.
- In case where `ModelAndView` contains the logical name of the View, the `DispatcherServlet` refers to the `ViewResolver` to find the actual View object based on the logical name.
- `DispatcherServlet` then passes the Model object to the View object to render it to the end user.

Slide Note

Spring MVC Architecture (cont'd)