Programming in Java

8. Spring MVC Web

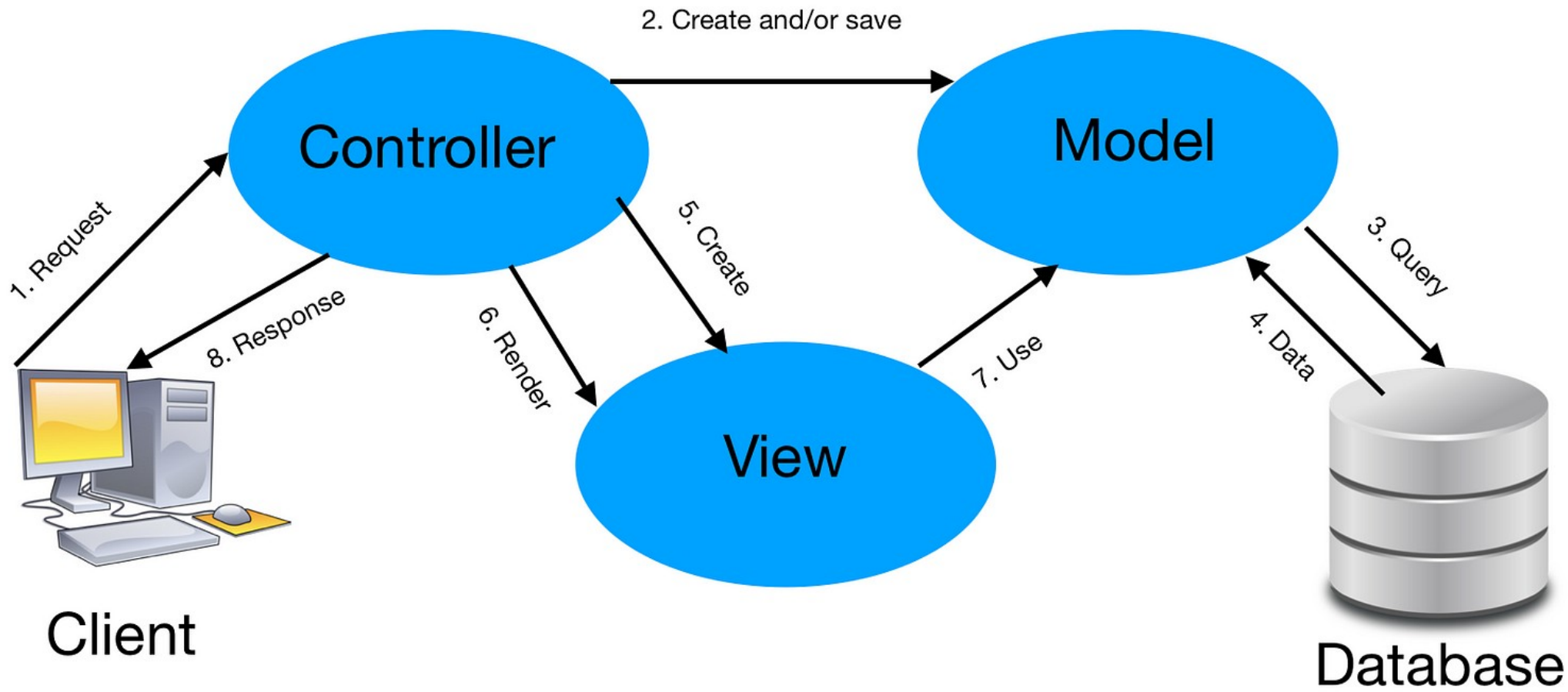
MVC Architecture

- Early design pattern
  - For designing interactive systems
  - Originally proposed in 1979
- MVC stands for Model View Controller
  - The most popular interaction architecture
  - Used to break up a large application into smaller functional sections.
  - Most of the popular frameworks follow the MVC design pattern
  - Including the Spring Web MVC framework
  - MVC is generally the go to for designing RESTful web applications

MVC Architecture

- Early design pattern
  - For designing interactive systems
  - Originally proposed in 1979
- MVC stands for Model View Controller
  - The most popular interaction architecture
  - Used to break up a large application into smaller functional sections.
  - Most of the popular frameworks follow the MVC design pattern
  - Including the Spring Web MVC framework
  - MVC is generally the go to for designing RESTful web applications

MVC Architecture



Components of MVC

- There are three components
- View
  - The view is where the user interacts
  - In a strict definition of the model, it displays the result of performing an operation
  - This doesn’t have to be a screen but could be an output file or a printer
- Controller
  - Where the client requests are processed
  - Includes routing the request to the model component that should handle it
  - Possible controller interactions are often presented in the View
- Model
  - The actual application that processes the request

Interfaces and MVC

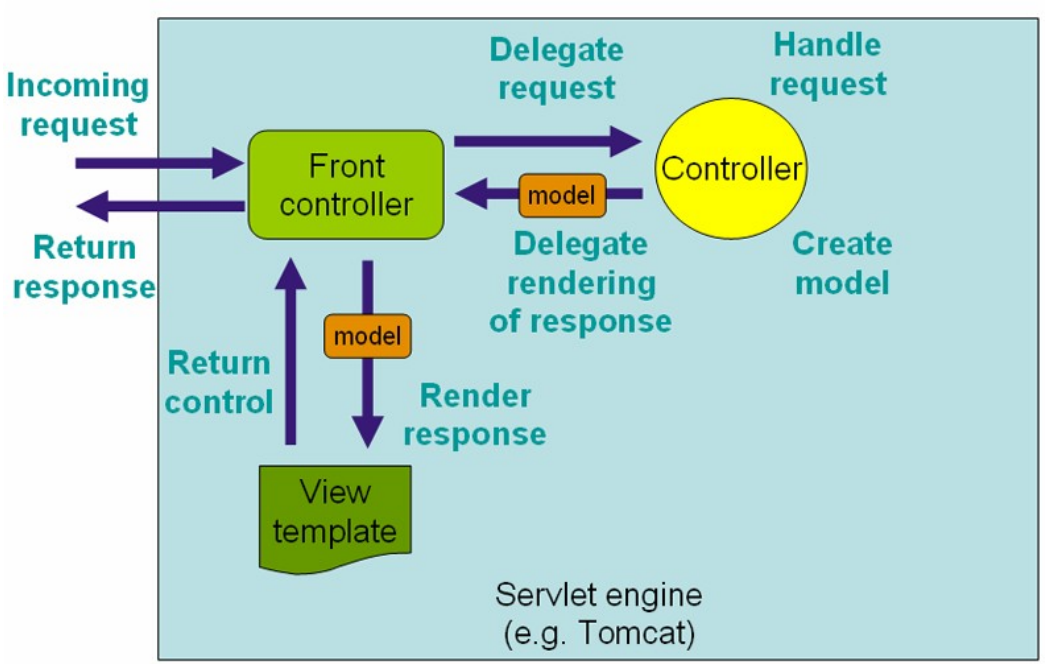
- There is a correlation between MVC and Interface/Implementation
- The view+controller is analogous to the interface
- The model is analogous to the implementation
- Reasons for separating these concerns
  - Prevents tight coupling between a client and a server application
  - Essentially the Bridge design pattern
  - Multiple view/controllers can be used with the same model
  - Multiple models can be used with the same view/controller

Spring Web MVC

- A specific implementation of the MVC architecture
- Intended for web services
  - Controller processes REST requests
  - The view is the returned HTML page
  - The model is the application accessed via the web interface
- Build around a dispatcher servelet
  - Takes a REST request
  - Defines a method for each request like GET:/myapp/sale/897
  - These methods call the appropriate model operations
- Works in a similar way to
  - STRUTS, Java Server Faces, Jave EE Servlets
  - Intended to simplify deploying this common architecture

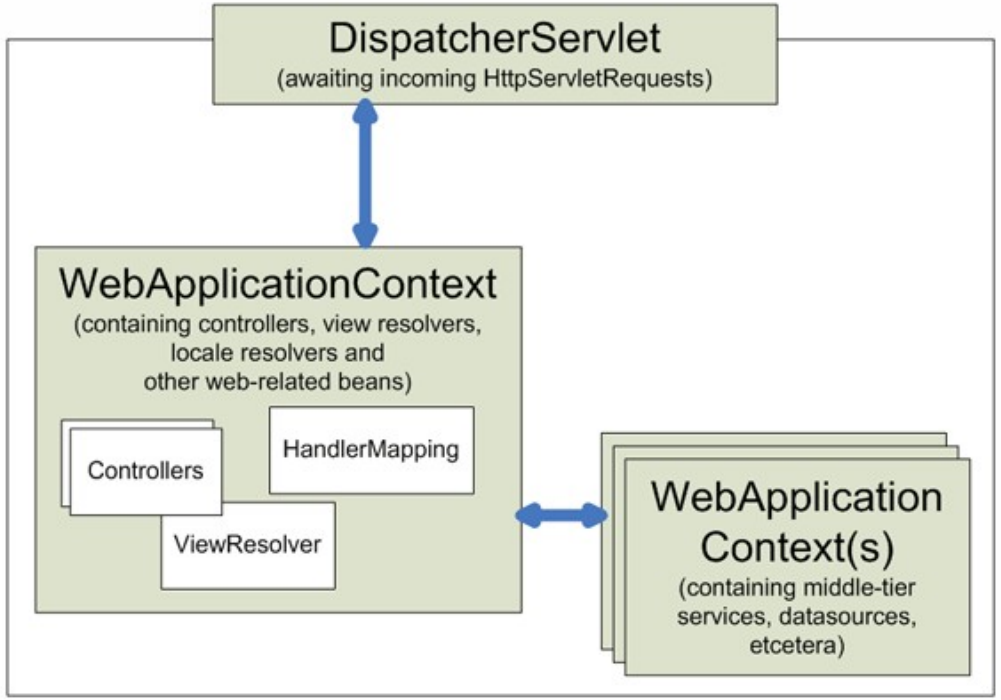
DispatcherServlet

- The incoming request is received
  - The appropriate request is made to the model
  - The response is the used to populate a view template
  - The rendered template with values is then returned to the dispatch servelet
  - And rendered response is returned to the client
- The view template can be very simple or very complex
  - In its simplest form, it could just be a string with interpolated values



Contexts

- In an earlier Spring module
  - We introduced the idea of a context
- In Spring MVC
  - There is a WebApplicationContext
  - This manages all the components used in the MVC architecture
  - Including wiring them all together
  - And managing the routing of requests



@Controller

- The @Controller annotation
  - Define a controller class that handles HTTP requests and returns view names
  - The controller processes user input and returns the name of a view
  - Spring resolves the view name to a template (e.g., an HTML file)
  - Data is passed to the view using a Model object
- When Spring Boot starts:
  - It scans for classes annotated with @Controller.
  - Registers them as Spring beans in the Spring context.
  - Routes HTTP requests to controller methods using mappings like @GetMapping, @PostMapping
  - Controller methods return logical view names, not raw data.
  - Spring resolves the view name via a ViewResolver like Thymeleaf or Java Server Pages
  - The view is rendered with any data added to the Model.

@Controller Example

- The controller code is shown on the right
  - User accesses /hello
  - Spring calls showHelloPage()
  - Adds "Welcome to Spring MVC!" to the model under key message
  - Returns "hello" as the view name
  - Spring looks for hello.html in the templates/ folder
  - View is rendered with the message inserted

```
src/
 └── main/
      └── resources/
           └── templates/
                └── hello.html
```

```java
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class HelloController {

    @GetMapping("/hello")
    public String showHelloPage(Model model) {
        model.addAttribute("message", "Welcome to Spring MVC!");
        return "hello"; // resolves to hello.html
    }
}
```

```html
<!DOCTYPE html>
<html>
<head>
    <title>Hello Page</title>
</head>
<body>
    <h1 th:text="${message}">Default Message</h1>
</body>
</html>
```

@Controller Example

- The controller code is shown on the right
  - User accesses /hello
  - Spring calls showHelloPage()
  - Adds "Welcome to Spring MVC!" to the model under key message
  - Returns "hello" as the view name
  - Spring looks for hello.html in the templates/ folder
  - View is rendered with the message inserted

```
src/
 └── main/
      └── resources/
           └── templates/
                └── hello.html
```

```java
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class HelloController {

    @GetMapping("/hello")
    public String showHelloPage(Model model) {
        model.addAttribute("message", "Welcome to Spring MVC!");
        return "hello"; // resolves to hello.html
    }
}
```

```html
<!DOCTYPE html>
<html>
<head>
    <title>Hello Page</title>
</head>
<body>
    <h1 th:text="${message}">Default Message</h1>
</body>
</html>
```

@RestController

- A specialized version of @Controller used in RESTful web services.
  - Combines @Controller + @ResponseBody.
  - Every method in a class annotated with @RestController returns data (usually JSON or XML), not a view.
  - Returns objects/data, not templates.
  - Automatically serializes return values into the HTTP response body using HTTP message converters (like Jackson for JSON).
  - Used to build REST APIs.

```java
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import java.util.Map;

@RestController
public class ApiController {

    @GetMapping("/api/greeting")
    public Map<String, String> greet() {
        return Map.of("message", "Hello from REST API");
    }
}
```


Lab 8-1
Spring MVC


Java™