CYDEO

Spring MVC

Date: 1st March 2022

What is a Web App?

- An app you access through your web browser is a web app.
- Accessing an app in a browser makes the app more comfortable to use. You do not have to install anything, and you can use it from any device that has access to the internet.

What is a Web App?

- A web app is composed of two parts:
  - The client-side: which is what the user directly interacts with. It is also been referred as frontend.
  - The server-side: which receives requests from the client and sends back data in response. It is also been referred as the backend.

Frontend

Backend

The web client (browser) directly interacts with the user.

The web client makes requests to the web server to get or process specific data.

The server-side-app sometimes needs to store, get, and change some data in a database.

The user

REQUEST: Give me the product list page I need to display.

RESPONSE: Here is what you need to display to the user.

The web client uses the data it gets from the server to display something.

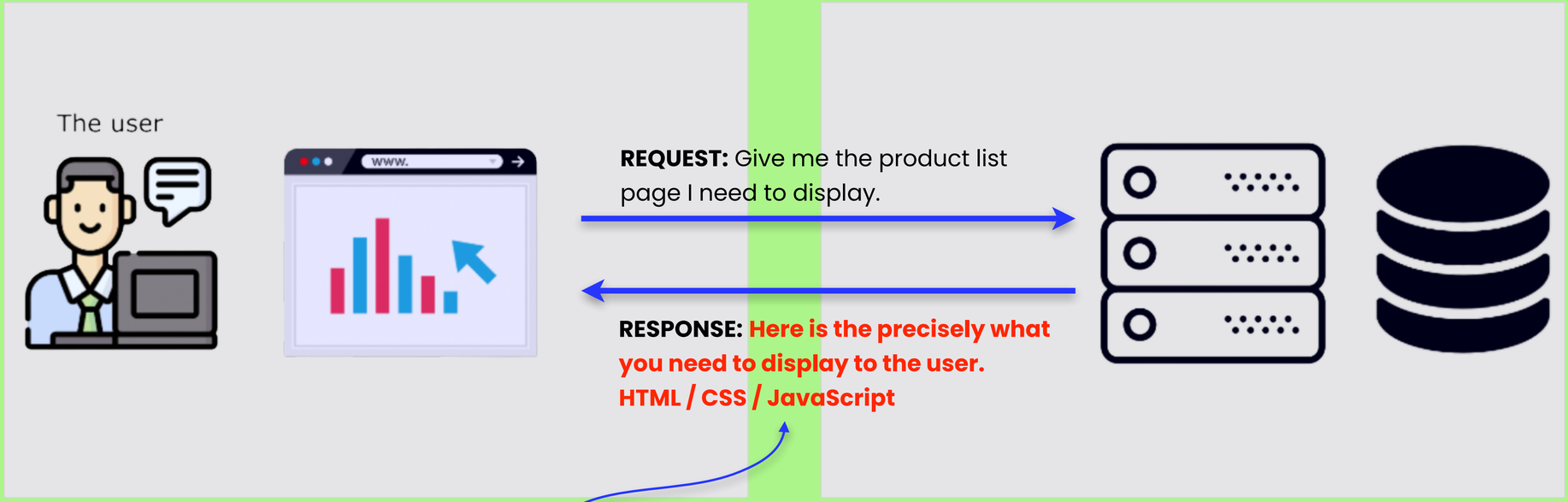
The web server gets the client requests and processes the data it receives from the client. In the end, the server responds to the client's request.

Different Fashions of Implementing a Web App

- Apps where the backend provides the fully baked view in response to a client's request.
- Apps using frontend-backend separation.

Frontend

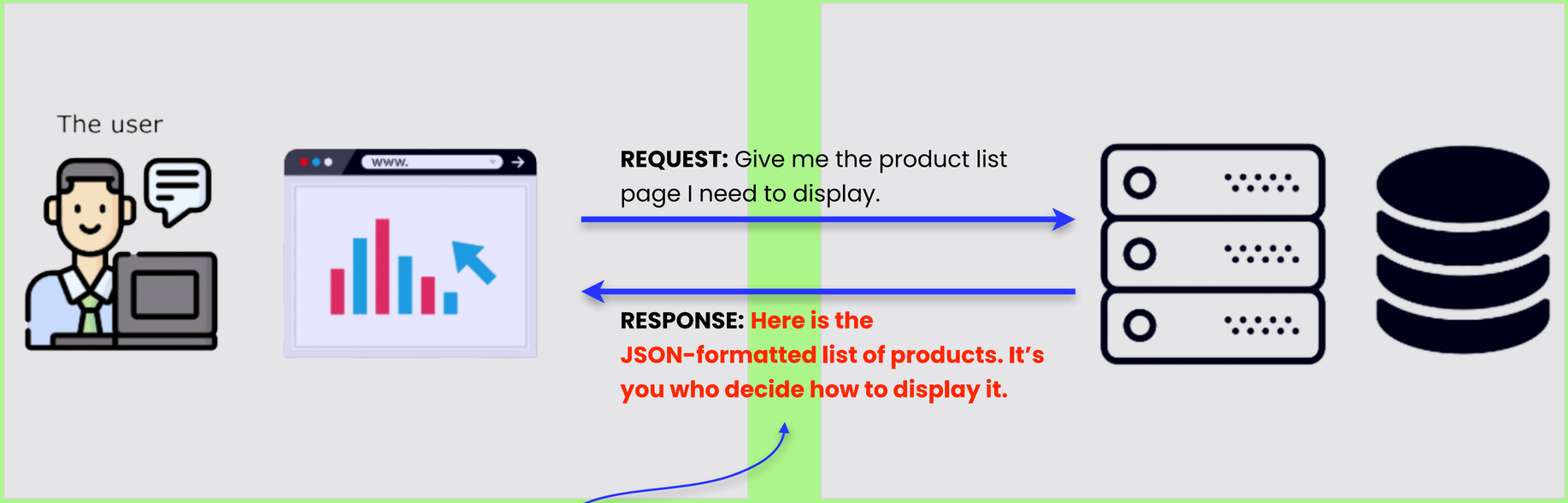
Backend



In a standart web app, the client receives from the server a response containing precisely what the browser needs to display. The server-side app sends data in HTML, CSS and JavaScript format that the browser interprets and displays.

Frontend

Backend



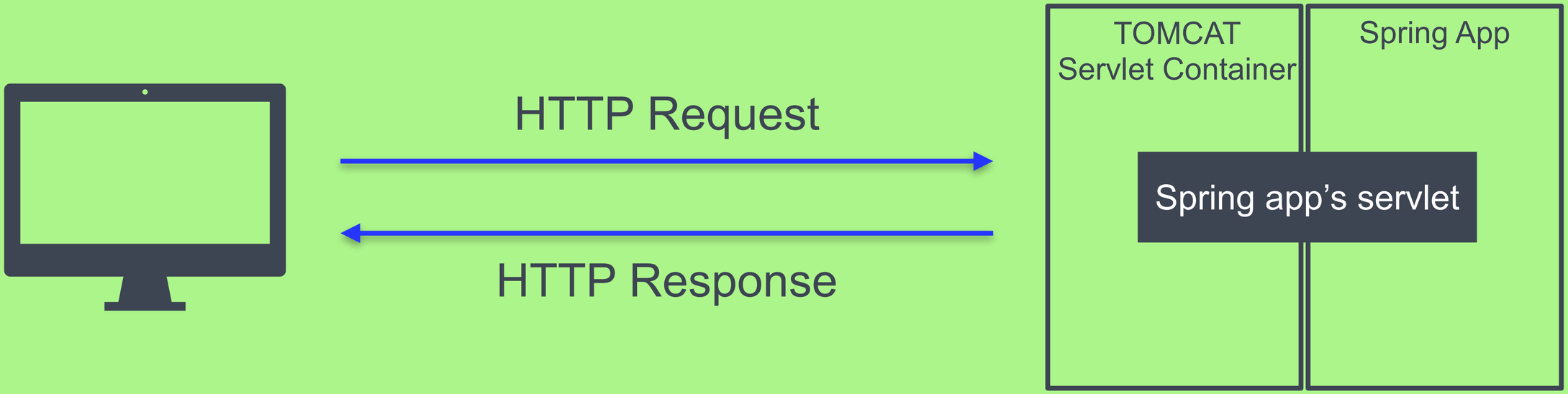
The web server gets the client requests and processes the data it receives from the client. In the end, the server responds to the client's request.

How the Client and the Server Side Communicate?

- A web browser uses a protocol named Hypertext Transfer Protocol (HTTP) to communicate with the server over the network.
- Your app does not need to know to process the HTTP messages. It will use a component already designed to understand the HTTP.
- Understanding the HTTP is not only thing we need. We need something that can translate the HTTP request and response to a Java app.

Servlet Container

- A servlet is nothing more than a Java Object which directly interacts with the Servlet Container.
- Servlet Container is a translator of the HTTP messages for our Java app.
- Servlet Container has basically a context of servlet instances it controls, just as Spring does with its beans.
- Tomcat is called a ‘Servlet Container’
- We do not create servlet instances. We only need to remember the servlet is the entry point to our app’s logic. It is the component the Servlet Container (Tomcat) directly interacts with.

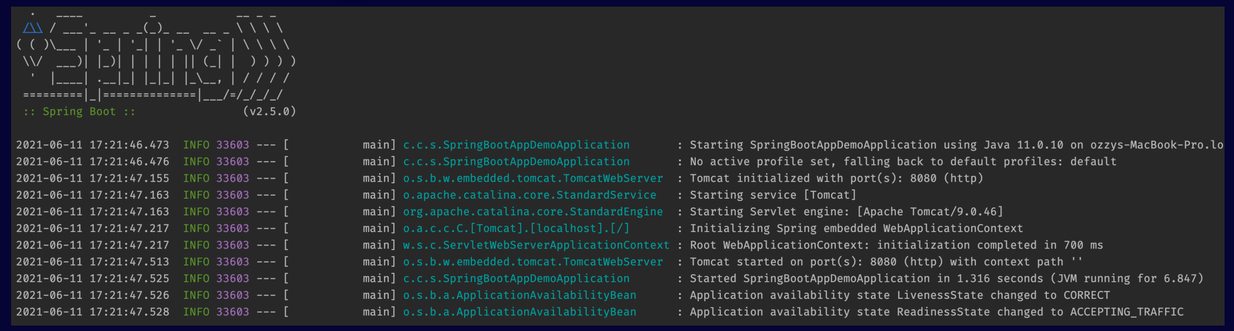


The Spring app defines a servlet object and registers it into the Servlet Container. Now both Spring and the Servlet Container know this object and can manage it. Servlet Container calls this object for any client request, allowing the servlet to manage the request and the response.

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

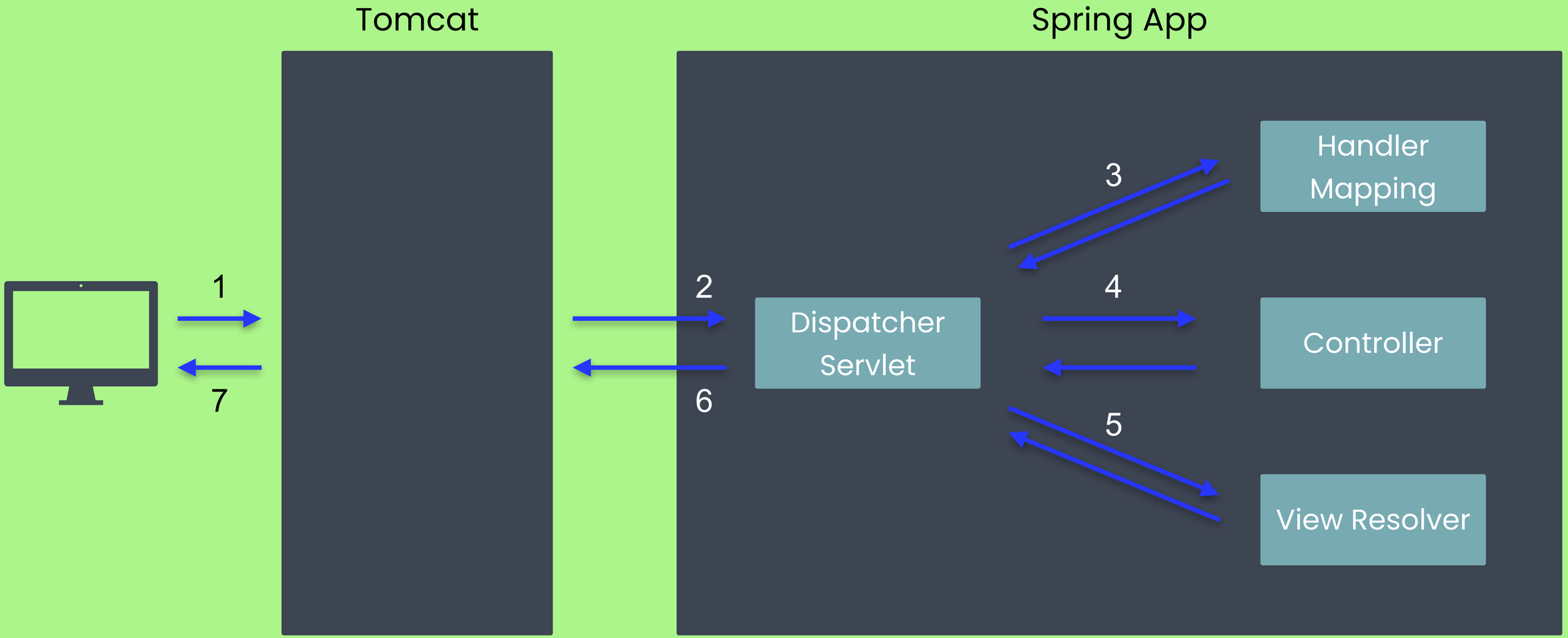
Spring knows we need a servlet container and it will configure a Tomcat instance.

Run the Spring Boot Application



Starting the Tomcat server

Spring MVC Architecture



1. The client makes an HTTP request

2. Tomcat gets the client's HTTP request. Tomcat has to call a servlet component for the HTTP request. Tomcat calls a servlet Spring Boot configured. This servlet is called as Dispatcher Servlet or Front Controller.

3. Dispatcher Servlet's responsibility is to manage the request further inside the Spring App. It has to find what **controller** action to call for the request and what to send back in response to the client.

4. Dispatcher Servlet need to find the controller action to call for the request. To find out which controller action to call, the Dispatcher Servlet delegates to a component named Handler Mapping.

5. After finding out which controller action to call, the Dispatcher Servlet calls that specific controller action. The controller returns to the Dispatcher servlet the page name it needs to render for the response. We refer to this HTML page also as “**view”**

6. Dispatcher servlet delegates the responsibility of getting the view content to a component named View Resolver.

7. Dispatcher Servlet returns in the HTTP response the rendered view.

Implementing a web app with Spring MVC

- To add a web page to your app, you follow two steps :

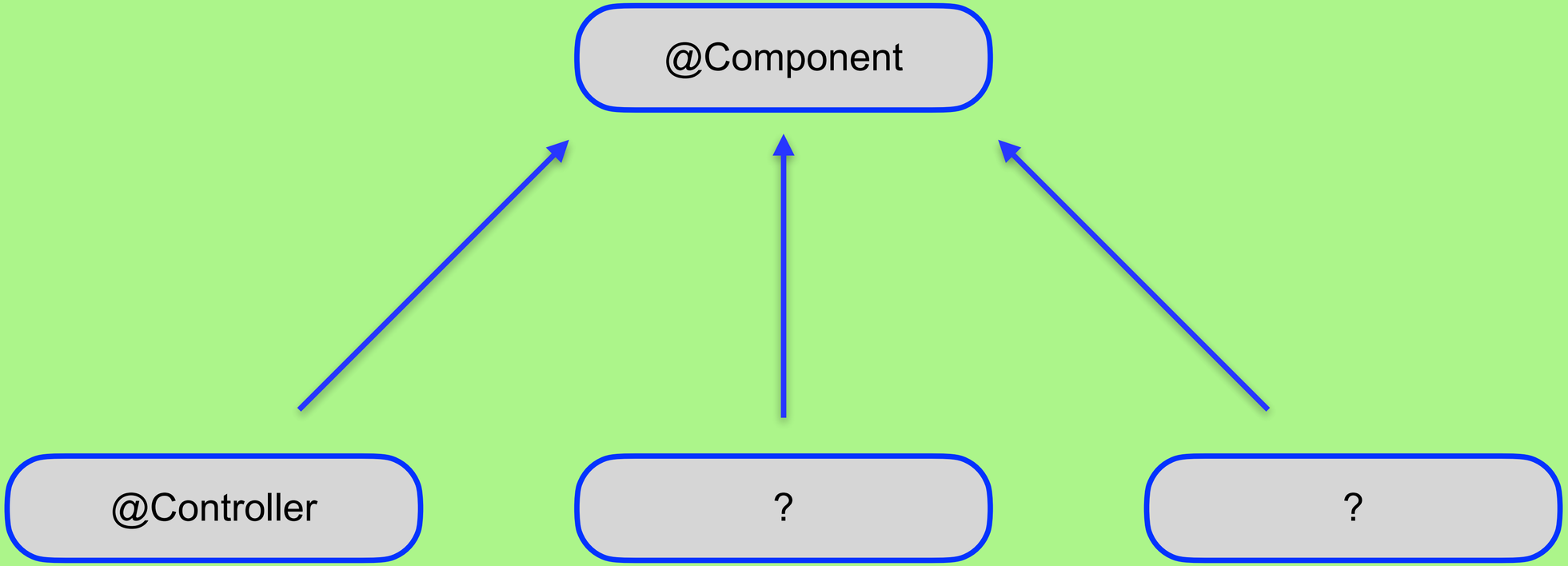
1. Write an HTML document with the content you want to be displayed by the browser.

2. Write a controller with an action for the web page created at point 1.

Controller

- **@Controller** annotation indicates that the annotated class is controller.
- The controller actions are methods associated with specific HTTP requests.
- It is specialization of @Component and is autodetected through classpath scanning.
- It is typically used in combination with annotated handler methods based on the **@RequestMapping** annotation.

@Component is used to mark a class from which a bean will be created.



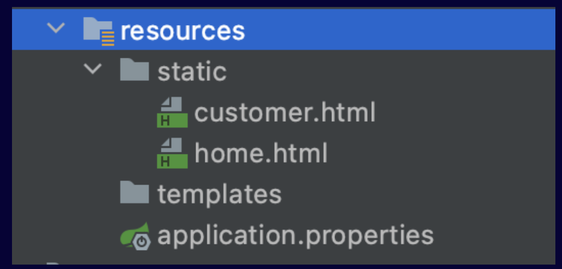
@RequestMapping

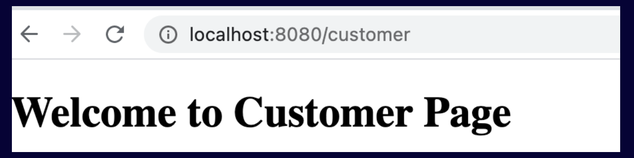
- This annotation maps HTTP requests to handler methods of MVC and REST controllers
- It can be applied to class level and/or method level.

```java
@Controller
public class HomeController {

	@RequestMapping("/home")
	public String home() {
		return "home.html";
	}

	@RequestMapping("/customer")
	public String customer() {
		return "customer.html";
	}
}
```





Workflow

```java
@Controller
public class StudentController {

	@RequestMapping("/s1")
	public String student() {
		return "student.html";
	}

	@RequestMapping("/s2")
	public String mentor() {
		return "mentor.html";
	}
}
@Controller
public class TeacherController {

	@RequestMapping("/t1")
	public String teacher() {
		return "teacher.html";
	}

	@RequestMapping("/t2")
	public String parent() {
		return "parent.html";
	}
}
```

mentor.html

```html
<!Doctype html>

<html>

<head></head>

<body>
    <h1>Welcome Mike</h1>
</body>

</html>
```

parent.html

```html
<!Doctype html>

<html>

<head></head>

<body>
    <h1>Welcome Emmy</h1>
</body>

</html>
```

student.html

```html
<!Doctype html>

<html>

<head></head>

<body>
    <h1>Welcome Ozzy</h1>
</body>

</html>
```

teacher.html

```html
<!Doctype html>

<html>

<head></head>

<body>
    <h1>Welcome Adam</h1>
</body>

</html>
```

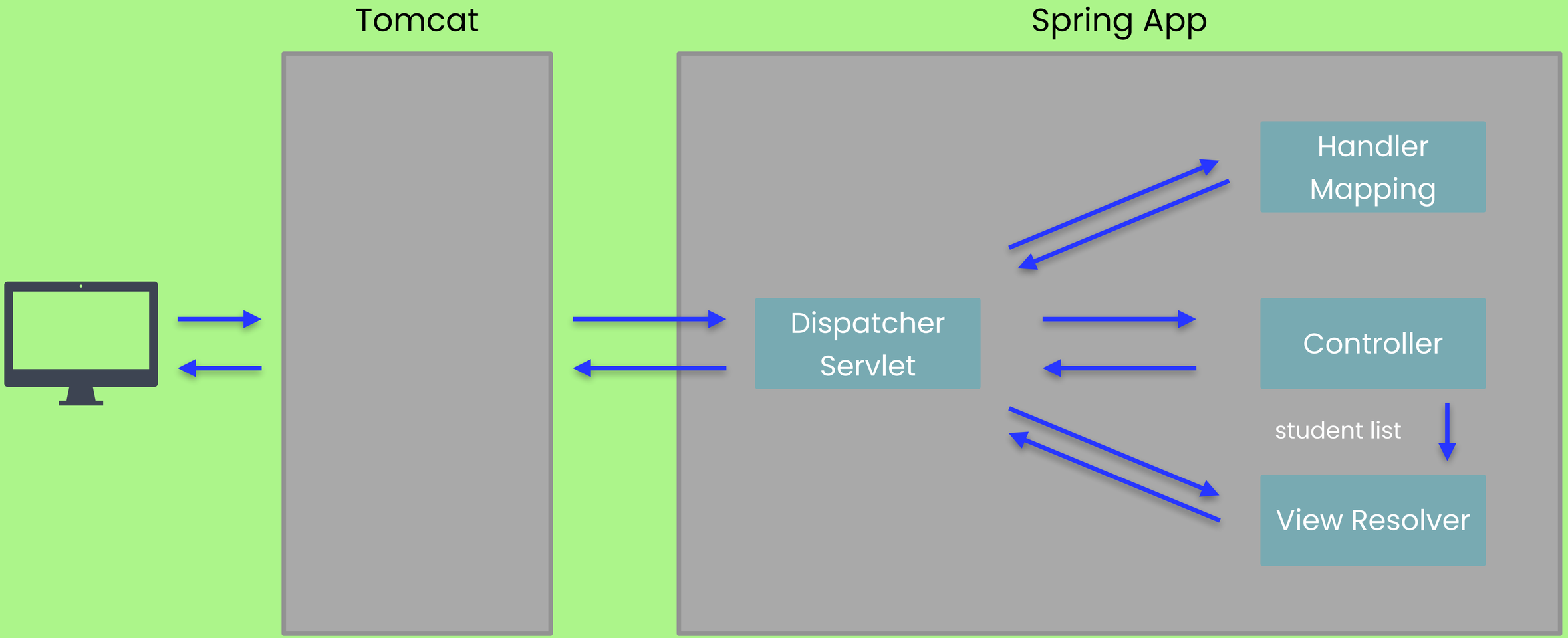
http://localhost:8080/s1 → ?

http://localhost:8080/t2 →

- How about the pages whose content changes according to how your app processes the data for specific requests?

Spring MVC Flow

- The client sends an HTTP request to the web server.
- The dispatcher servlet uses the handler mapping to find out what controller action to call.
- The dispatcher servlet calls the controller's action.
- After executing the action associated with the HTTP request, the controller returns the view name **(and data to view)** the dispatcher servlet needs to render into the HTTP response.
- The response sent back to the client.


Tomcat
Spring App
Handler Mapping
Dispatcher Servlet
Controller
student list
View Resolver

Model

- The model is a container for application data.
- In controller, any data can be added (strings, objects, from database, etc...)
- View page can access data from model.

Template Engine

- A template engine is a dependency that allows you to easily get and display in the view variable data that the controller sends.
- Spring supports many view templates:
  - JSP (Java Server Pages)
  - **Thymeleaf**
  - Groovy
  - Jade

```xml
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

Dependency need to add to the pom.xml

```xml
<html lang="en" xmlns:th="http://www.thymeleaf.org">
```

This definition is equivalent to an import in Java.
It allows us further to use the prefix “**th**” to refer to specific features provided by thymeleaf in the view.

Model Interface

Method	Description
Model addAllAttributes(Collection<?> attributeValues)	It copies all the attributes in the provided collection into this Map.
Model addAllAttributes(Map<String, ?> attributes)	It copies all the attributes in the provided Map into this Map.
Model addAttribute(Object attributeValue)	It adds the given attribute to this Map through a generated name.
Boolean containsAttribute(String attributeName)	It searches whether the model contains the attribute of the given name.
Model mergeAttributes(Map<String, ?> attributes)	It copies all the attributes in the given Map into this Map, with the existing objects of the same name.

```java
@Controller
public class StudentController {

	@RequestMapping("/student")
	public String getStudentList(Model model) {

		model.addAttribute("name", "Mike");

		return "student.html";
	}
}
```

```
<!Doctype html>

<html lang="en" xmlns:th="http://www.thymeleaf.org">

<head></head>

<body>

    <h1 th:text="${name}"></h1>

</body>

</html>
```

${attribute-name} syntax, you refer to any of the attributes you send from the controller using the Model instance.

```java
@Controller
public class StudentController {

	@RequestMapping("/student")
	public String getStudentList(Model model) {

		String name = "Mike";

		String color = "red";

		model.addAttribute("name", name);

		model.addAttribute("color", color);

		return "student.html";
	}
}
```

```html
<body>

    <h1>Student's Name:

        <span
            th:style = " 'color:' + ${color} "
            th:text = " ${name} ">
        </span>

    </h1>

</body>
```

Task

1. Create a Mentor class with the FirstName, LastName, Age and Gender fields.
2. Gender should be an enum.
3. Create MentorController.
4. Create sample Mentor class objects.
5. Add those objects to the Model.
6. Create an HTML page.
7. Pass the Mentor objects added to Model earlier, to View and show them in the table on browser, as shown in the picture.

First Name	Last Name	Age	Gender
Mike	Smith	45	MALE
Tom	Hanks	65	MALE
Ammy	Bryan	25	FEMALE

You can use the HTML code at the side to create the below HTML table as a sample/starting point.

First Name	Last Name	Age	Gender
Mike	Smith	45	MALE
Tom	Hanks	65	MALE
Ammy	Bryan	25	FEMALE

```
<table>
    <thead>
        <tr>
            <td>First Name</td>
            <td>Last Name</td>
            <td>Age</td>
            <td>Gender</td>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>Mike</td>
            <td>Smith</td>
            <td>45</td>
            <td>MALE</td>
        </tr>
        <tr>
            <td>Tom</td>
            <td>Hanks</td>
            <td>65</td>
            <td>MALE</td>
        </tr>
        <tr>
            <td>Ammy</td>
            <td>Bryan</td>
            <td>25</td>
            <td>FEMALE</td>
        </tr>
    </tbody>
</table>
```