Day 12

URI - Uniform Resource Locator

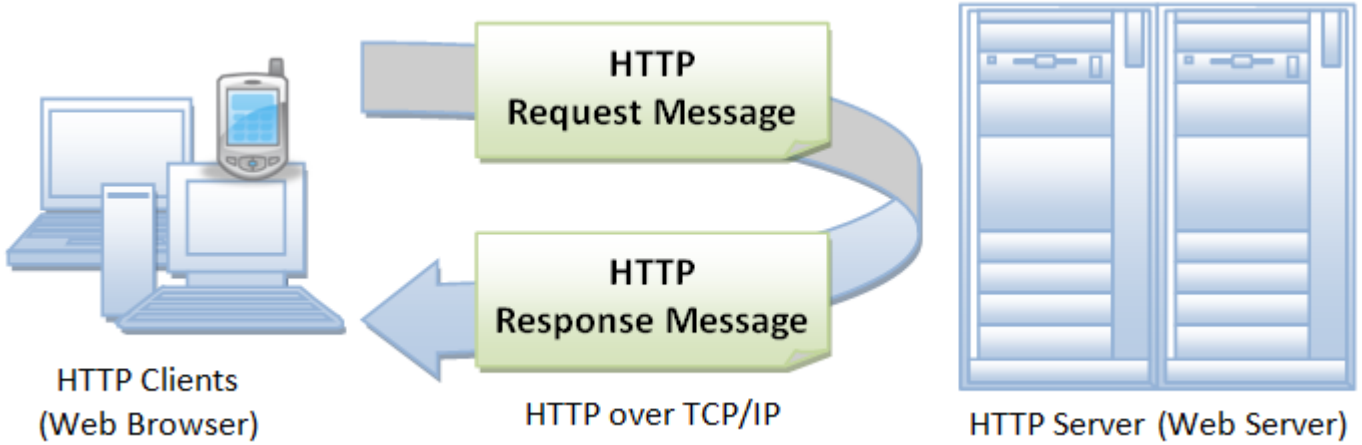
Protocol

Resource name

http://www.acme.corp/index.html

Server

HTTP Protocol



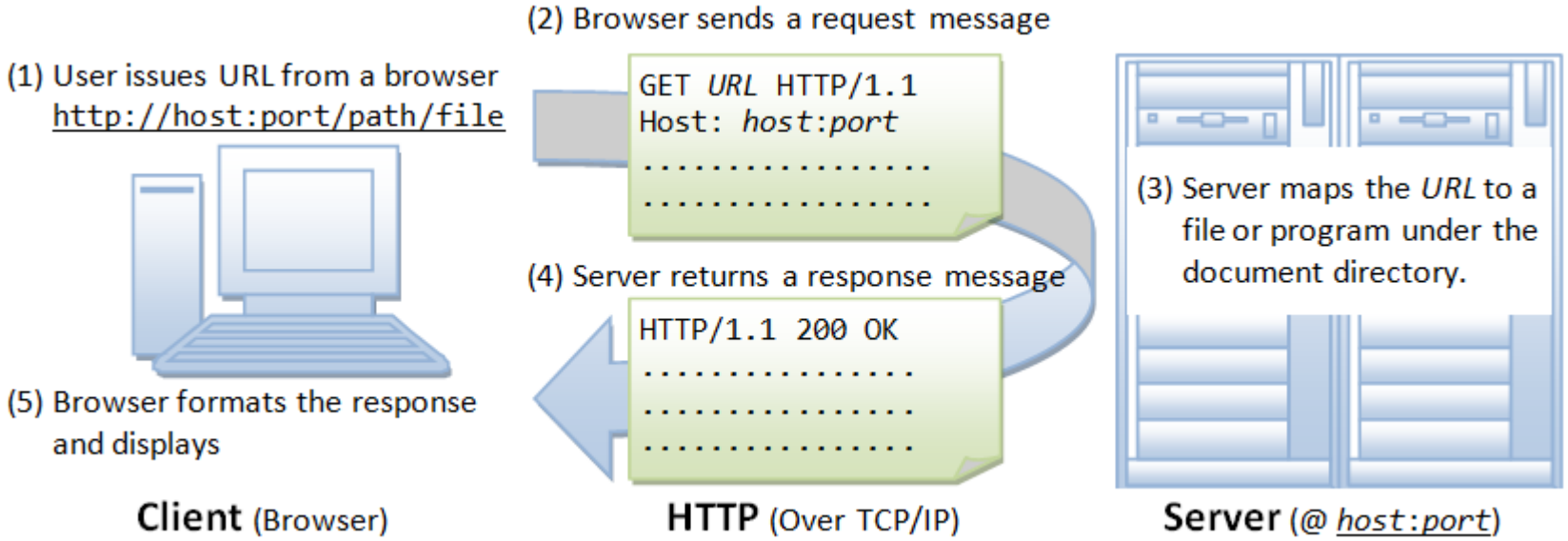
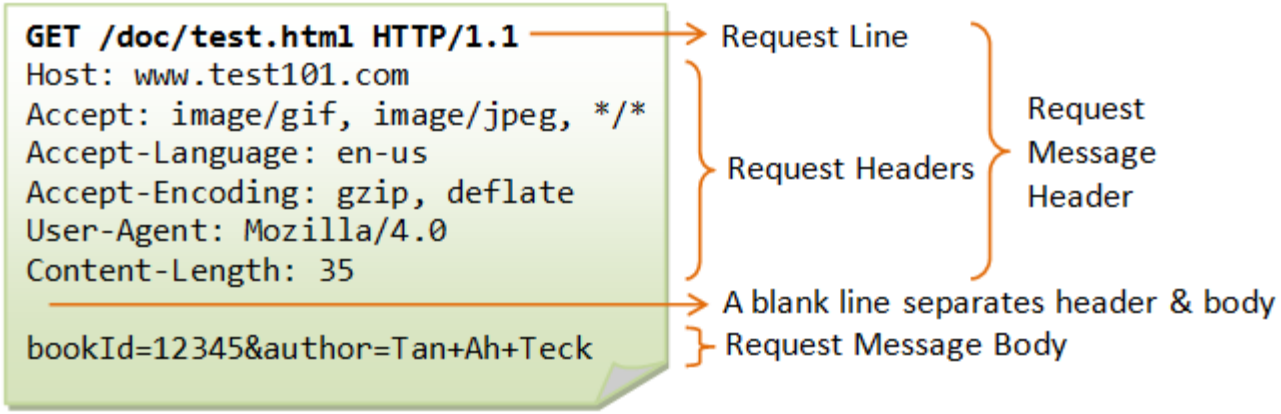


Image from https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html

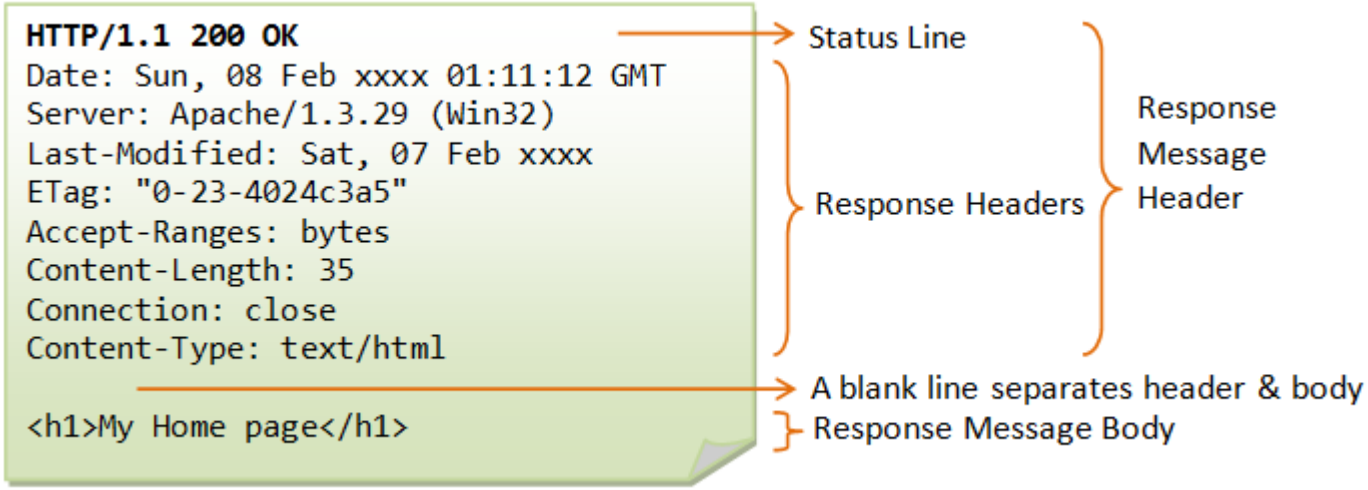
HTTP Request



- Request line
  - HTTP method (GET, POST, PUT, DELETE, etc) - VERB
  - Resource name - NOUN
- HTTP headers
  - Key value pairs
- Entity body/payload
  - Holds the request's content, if any

Image from https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html

HTTP Response



- Response line
  - Status code - 100, 200, 300, 400, 500
- HTTP headers
  - Key value pairs - similar to request
- Entity body/payload
  - Holds the response's content, if any

Image from https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html

HTTP Response Status Code

- 2XX - Success
  - 200 OK
  - 201 Created
- 4XX - Client Error
  - 400 Bad Request
  - 401 Unauthorized
  - 404 Not Found
  - 406 Not Acceptable
- 5XX - Server Error
  - 500 Internal Server Error
  - 503 Service Unavailable

Image from https://itnext.io/api-calls-and-http-status-codes-e0240f78f585

Method, Resource and Status

Operation	Verb	Noun	Outcome
Read	GET	/customer/1	200 OK
Create	POST	/customer	201 Created
Update	PUT	/customer/1	200 OK
Delete	DELETE	/customer/1	200 OK

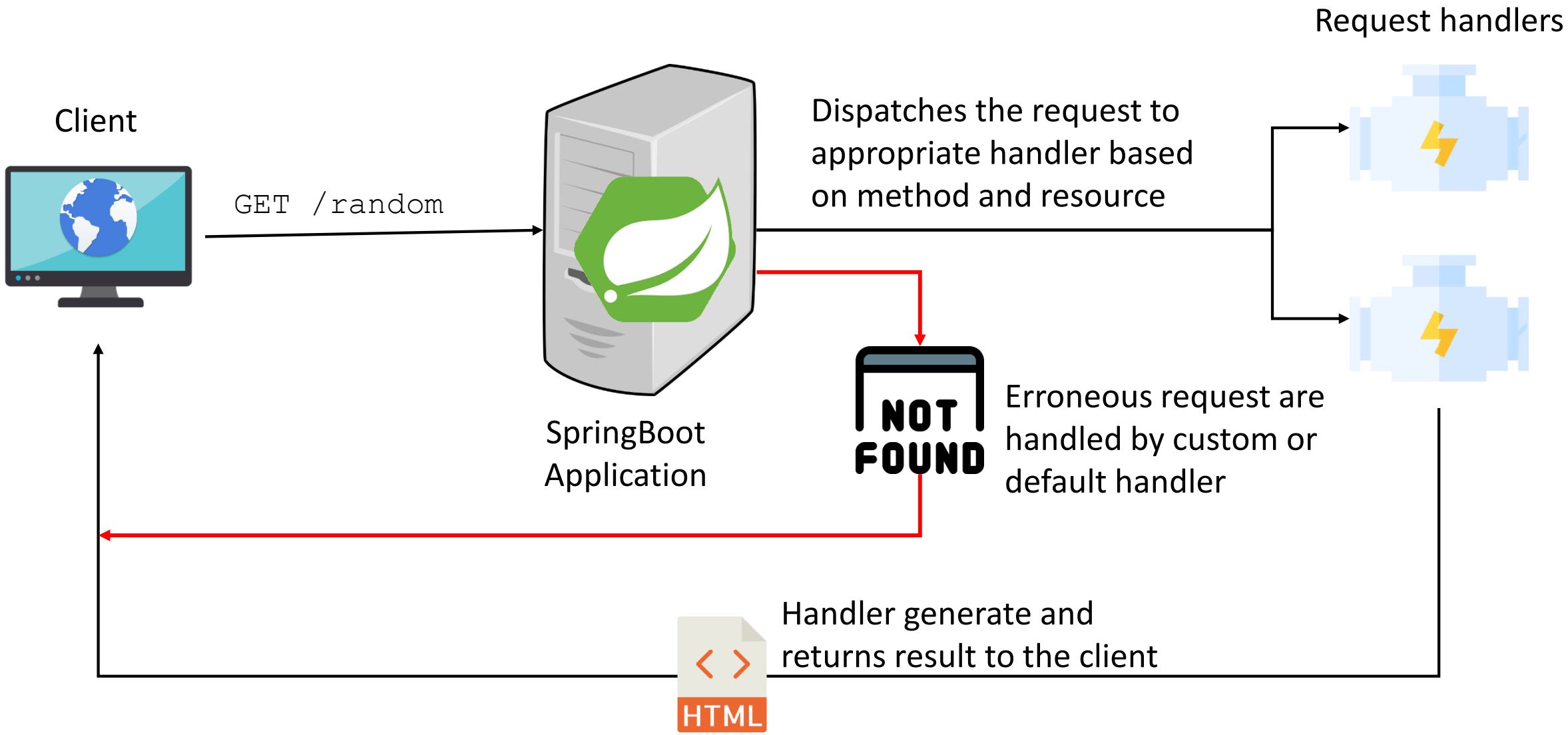
REQUEST

RESPONSE

HTTP `GET` Method

- GET method is used by the browser to GET a resource from a server
  - URL is typed in the address bar
  - Clicked on a “link”
  - Loading resources in a web page eg. images, JavaScript, CSS
- Originate from the browser
  - Includes the type of content it is requesting - HTML, image, CSS
    - Uses the HTTP header Accept
    - `Accept: text/html`
- GET method may include parameters called query string
  - `GET /weather?city=singapore`

Processing HTTP Request



Processing HTTP Request

Presentation tier

The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.

Logic tier

This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.

Data tier

Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.

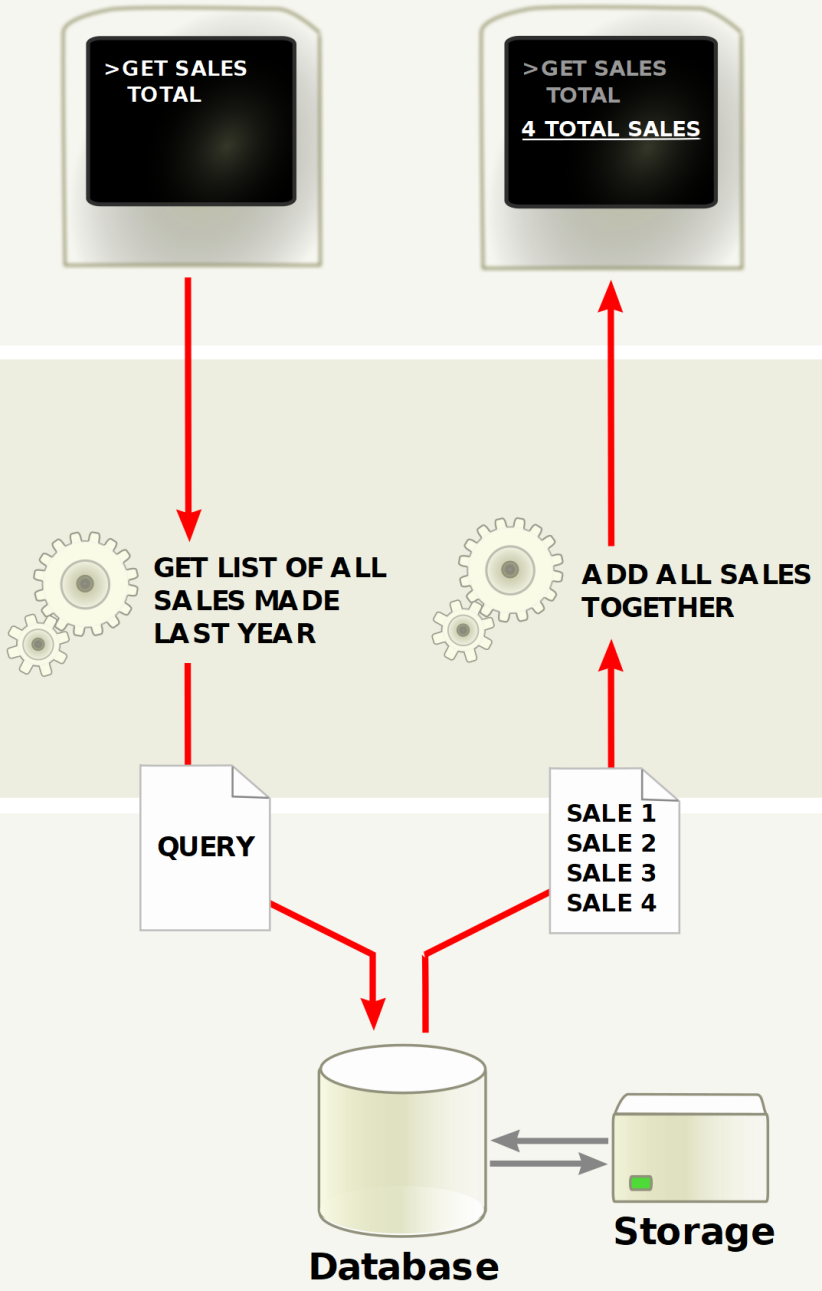


Image from https://en.wikipedia.org/wiki/Business_logic#/media/File:Overview_of_a_three-tier_application_vectorVersion.svg

Model View Controller Architecture

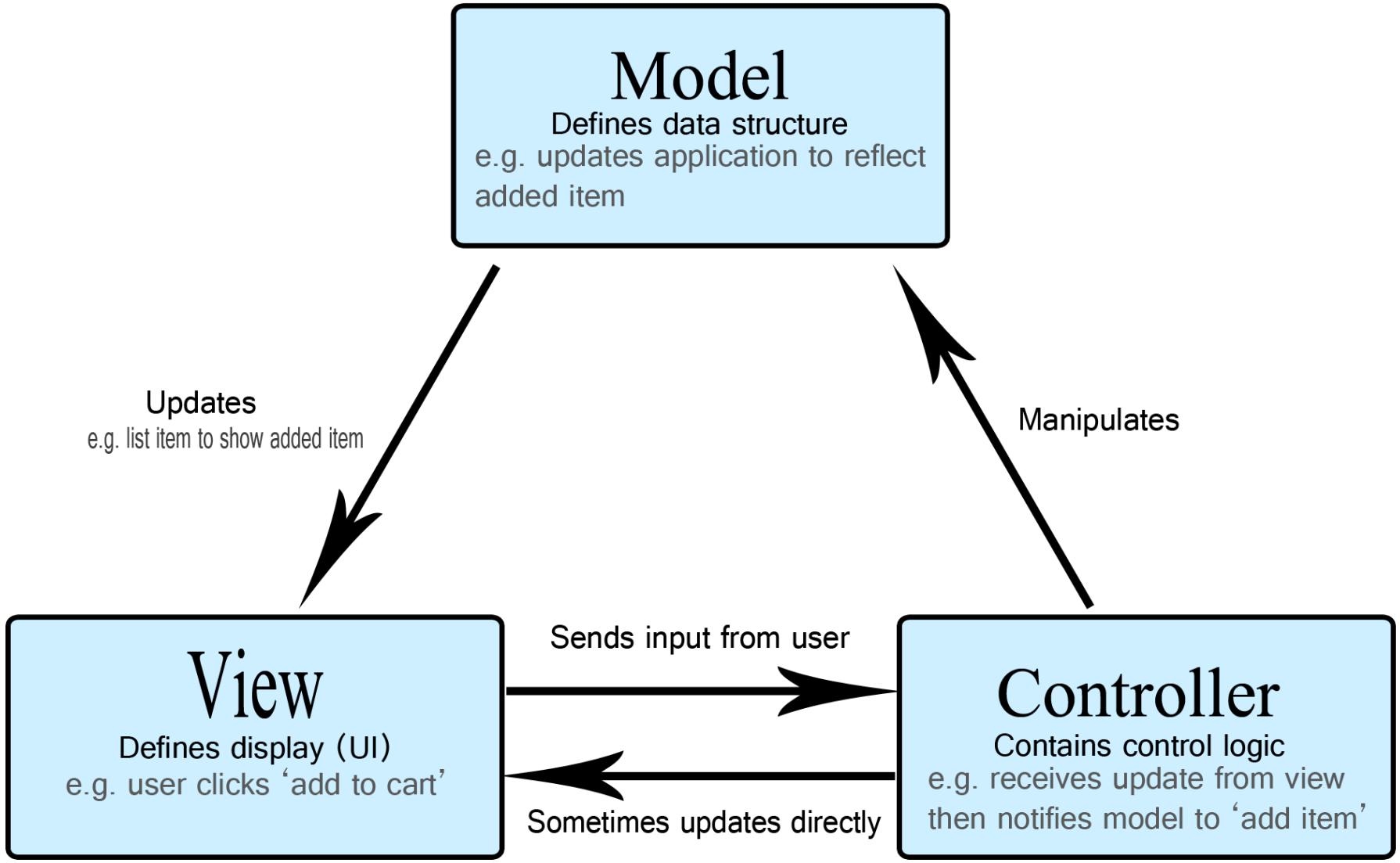


Image from https://developer.mozilla.org/en-US/docs/Glossary/MVC

Types of View

- Controller can return any type of view in an acceptable format
  - Eg HTML - text/html
  - Eg. JSON - application/json
- Some views are
  - Static eg images, videos, CSS, JavaScript
  - Dynamic eg. front page of news site, items in inventory
- Dynamic views are generated by the controller
  - Required data can be queried from database, other applications via API or RPC calls

Server Side Template Engine

- Template engine is used to generate dynamic HTML
  - Never use string concatenation to create HTML
- Many template engines - JSP, Velocity, Mustache
- Thymeleaf is a server side template engine
  - Good support for Spring
  - Templates are stored in `resources/templates` directory
  - Need to add Thymeleaf dependency when generating SpringBoot application

Thymeleaf - Static Content

Annotate this class as a controller

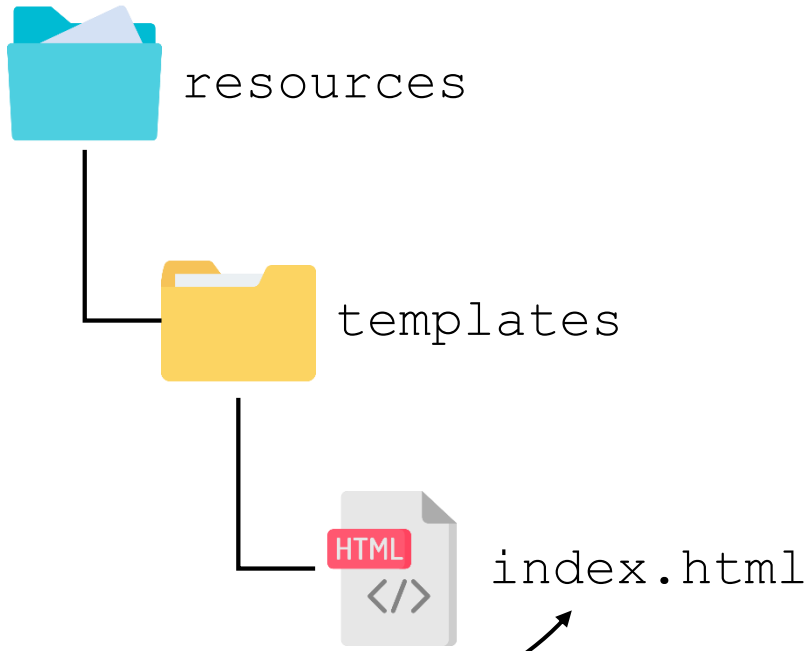
The following resources should be handled by this controller

```
@Controller
@RequestMapping(path = { "/", "/index.html" })
public class IndexResource {

    @GetMapping(produces = { "text/html" })
    public String index() {
        return "index"
    }
}
```

The result is HTML

Invoke this method when the HTTP method is `GET /` or `GET /index.html`



Process Request - Under the Hood

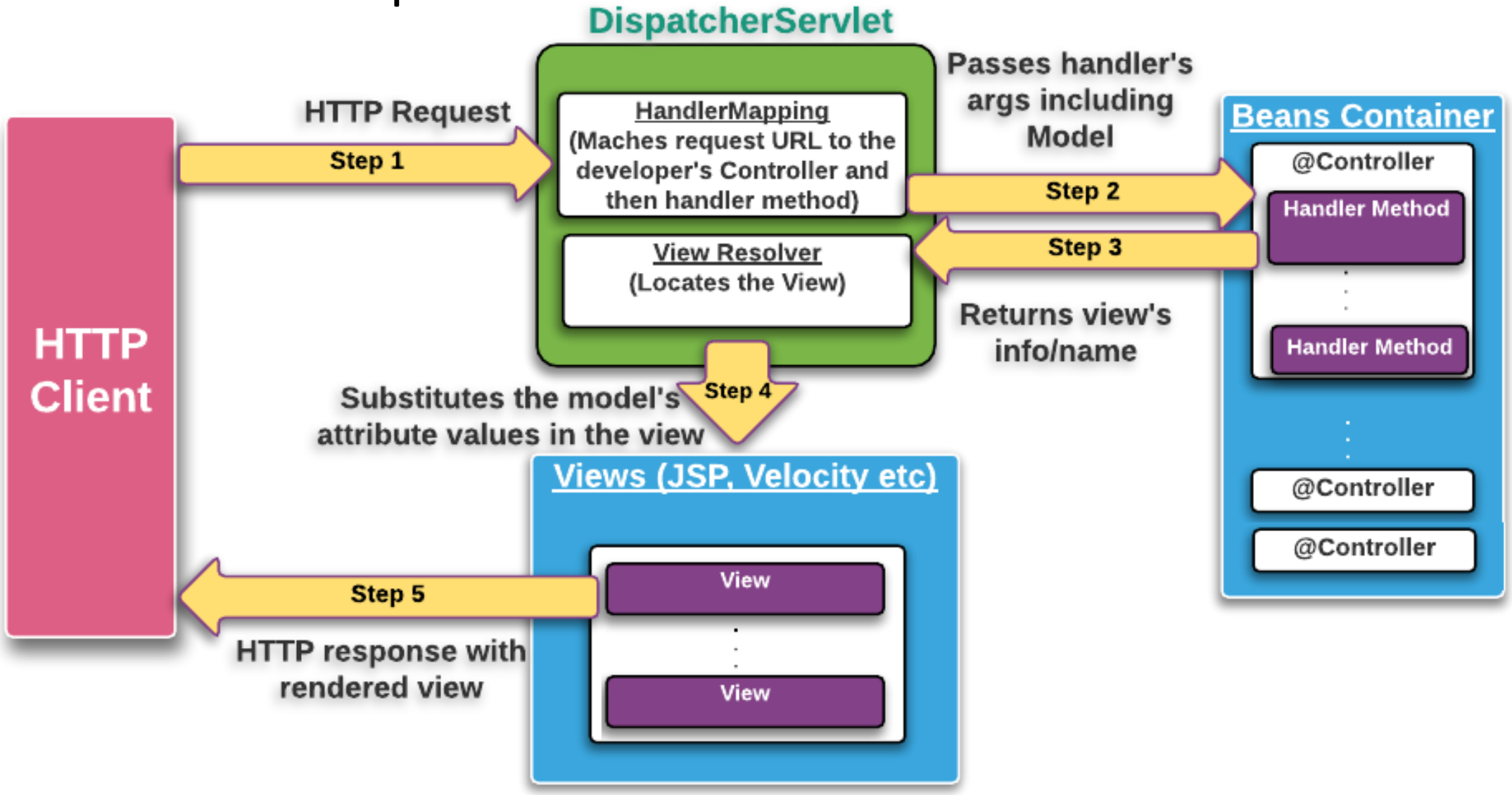


Image from https://www.logicbig.com/tutorials/spring-framework/spring-web-mvc/spring-mvc-intro.html

Thymeleaf - Dynamic Content

```
@Controller
@RequestMapping(path = { "/", "/index.html" })
public class IndexResource {

    @GetMapping(produces = { "text/html" })
    public String index(Model model) {
        model.addAttribute("currTime", (new Date()).toString());
        return "index"
    }
}
```

Inject model for binding data to view attributes

```
index.html
<h1>
    The current time is <span data-th-text="${currTime}"></span>
</h1>
```

Thymeleaf - Conditions

```
@Controller
@RequestMapping(path = { "/", "/index.html" })
public class IndexResource {

    @GetMapping(produces = { "text/html" })
    public String index(Model model) {
        Calendar cal = Calendar.getInstance();
        model.addAttribute("currHour", cal.get(Calendar.HOUR_OF_DAY);
        return "index"
    }
}
```

```
<div data-th-if="${currHour le 11}">
    Good Morning
</div>
<div data-th-unless="${currHour le 11}">
    Good Afternoon
</div>
```

Thymeleaf - Iteration

- Iterating over collections
  - Array, `java.util.List`, `java.util.Map.Entry` (for Maps)
- Iteration status provides the following information
  - `index` - the current index
  - `count` - collection size
  - `odd`, `even`, `first`, `last` - boolean properties

Thymeleaf - Iteration

```
@Controller
@RequestMapping(path = { "/cart" })
public class CartResource {

    @GetMapping(produces = { "text/html" })
    public String getCart(Model model) {
        List<Item> cart = getShoppingCart());
        model.addAttribute("cart", cart);
        return "cart"
    }
}
```

```
public class Item {
    private String itemName;
    private Integer quantity;
    // getter and setters
    public String getItemName() { return itemName; }
    public void setItemName(String n) { itemName = n; }
    ...
}
```

Thymeleaf - Iteration

```
<table>
   <tr data-th-each="item, status :${cart}"
         data-th-class="${status.odd? light-grey: light-blue}">
      <td data-th-text="${item.itemName}"></td>
      <td data-th-text="${item.quantity}"></td>
   </tr>
</table>
```

Each item is Item instance. Properties are accessed using the getter eg. `getItemName()`

GET Request with Query String

Parameters associated with a GET request

```
GET /weather?city=singapore&units=metrics
```

Parameter - key/value pair

Start of query string

Delimiter

Reading Query String

Mandatory parameter. Parameter name is inferred from the name of the formal parameter

```
@GetMapping
public String weather(
    @RequestParam(required=true) String city,
    @RequestParam(name="units", defaultValue="metrics") String units,
    Model model) {
    ...

}
```

Explicitly provide the parameter name. This parameter has a default value if it is not set

Reading Query String

Get all query strings in a MultiValueMap

```
@GetMapping
public String weather(
    @RequestParam MultiValueMap<String, String> queryParams,
    Model model) {

    final Collection<String> cities =
          (Collection<String>)queryParams.get("cities");
    final String units = queryParam.getFirst("units");
    ...

}
```

Get multi valued parameters

Get single value parameter

GET Request with Path Variable

Weather of a city is part of the resource name

Query string

```
GET /weather/singapore&units=metrics
```

```
GET /weather/singapore
GET /weather/tokyo
GET /weather/kuala%20lumpur
GET /weather/london
GET /weather/san%20francisco
```

This part of the resource's name is a variable

Reading Path Variable

Inject the value from the resource name and bind it to the parameter

```
@GetMapping="{city}")
public String weather(
    @PathVariable(name="city", required=true) String city,
    @RequestParam(name="units", defaultValue="metrics") String units,
    Model model) {
    ...

}
```

Query String vs Path Variable

Query String

- Used in traditional web application
- Used to provide additional information on the resource
  - Eg Weather in metric unit

Path Variable

- Used in RESTful API
- Resource name with the variable portion is a unique identifier of a resource
  - Eg. Singapore's weather

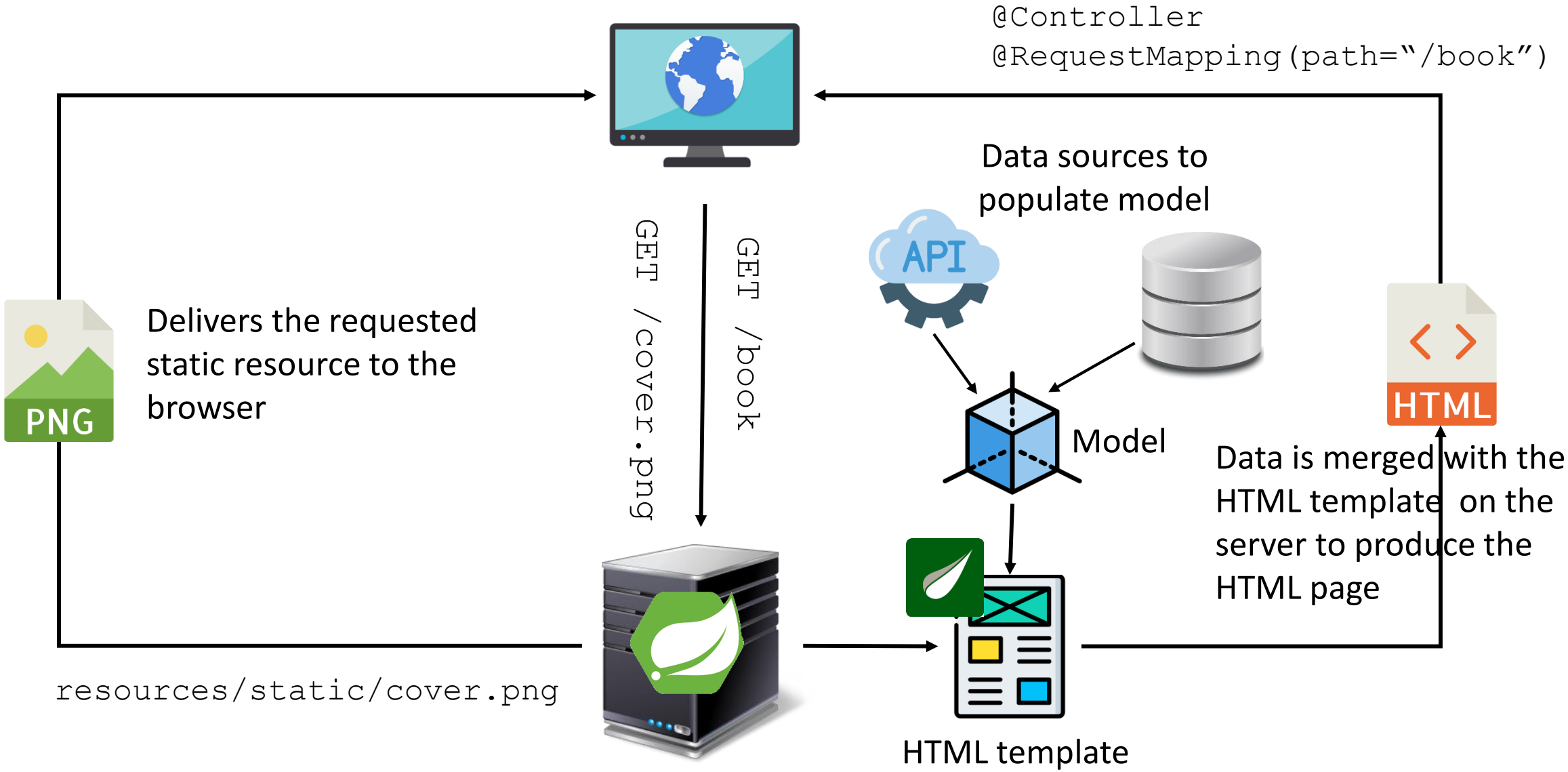
Customizing Error Page

- Default whitelabel error page can be overwritten with custom error page
  - Place in resources/templates/error.html
- The following attributes are available from the model
  - timestamp
  - status - HTTP status code
  - error
  - path - resource path
  - exception - exception class that caused the error
  - message - exception message
  - trace - exception stack trace

Example - Custom Error Page

```
<body>
   <h1>An error has occurred</h1>
   <h2 data-th-text="|${error} - ${message}|"></h2>
   <div>
      <img data-th-src="@{/404.png}">
   </div>
   <p>
      There seems to be an error with the following request
      <span data-th-text="${path}"></span>
   </p>
</body>
```

Static vs Server Rendered Pages



Static vs Server Rendered Pages

Static Resources

- Content do not change
  - Eg. About page, images
- Sends the same content for every request
- Content only changes if it is edited
- Fast loading time, content can be cached

Dynamic Pages

- Content of page changes depending on time, topic, etc
  - Eg. product search page
- Pages are generated on the server before sending the result back to the client
- May be slower depending on the complexity of the page
- Server side rendering