SPRING FRAMEWORK

SESSION 1

TỔNG QUAN VỀ SPRING FRAMEWORK

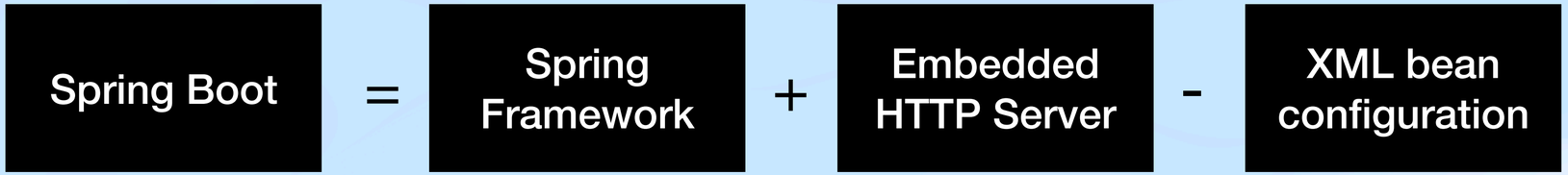
- Spring là một **Java framework** được sử dụng để phát triển các ứng dụng Java.
- Spring được chia làm nhiều module khác nhau, tuỳ theo mục đích phát triển ứng dụng mà ta dùng một trong các module đó.
  - Spring Core: là thành phần cốt lõi của Sprint Framework. Đây chính là nền tảng để xây dựng nên các thành phần khác trong hệ sinh thái của Spring Framework
  - Spring Bean: là trung tâm của Spring Core và là trái tim của một ứng dụng Spring
  - Spring MVC: được thiết kế dành cho việc xây dựng các ứng dụng nền tảng web theo pattern MVC.
  - Spring Data: cung cấp các phương thức để thao tác cd khác nhau (MySQL, Oracle v.v…).
  - Spring Security: cung cấp các cơ chế xác thực (Authentication) và phân quyền (authorization) cho ứng dụng.
  - Spring Boot: là một framework giúp chúng ta phát triển cũng như chạy ứng dụng một cách nhanh chóng.

TỔNG QUAN VỀ SPRING FRAMEWORK (II)

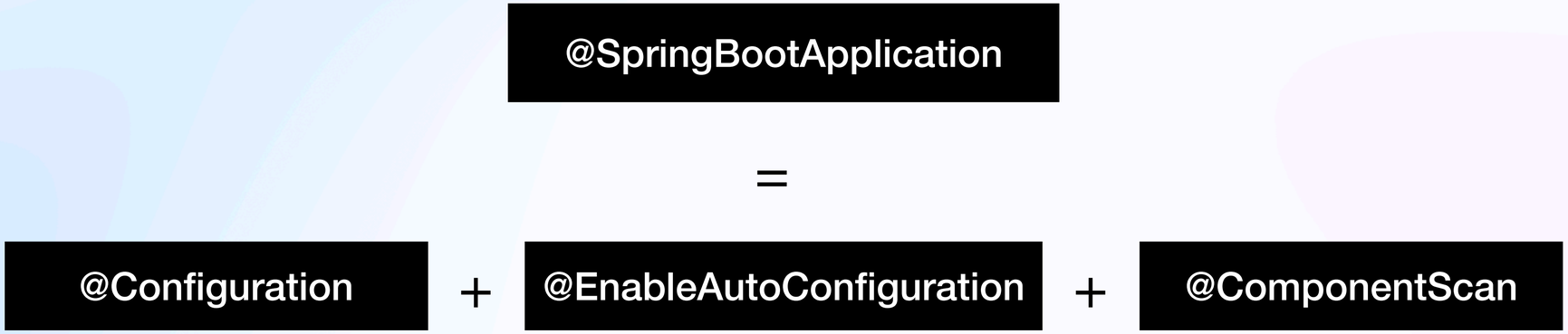
- Các lợi ích của Spring Framework
  - Spring Framework cung cấp rất nhiều module đủ để giúp chúng ta phát triển bất cứ thứ gì mà chúng ta muốn.
  - Spring cho phép lập trình viên sử dụng POJO object, đơn giản hoá trong lập trình.

TỔNG QUAN VỀ SPRING FRAMEWORK (II)

- Spring boot có thể giải thích một cách đơn giản bởi hình minh hoạ sau:



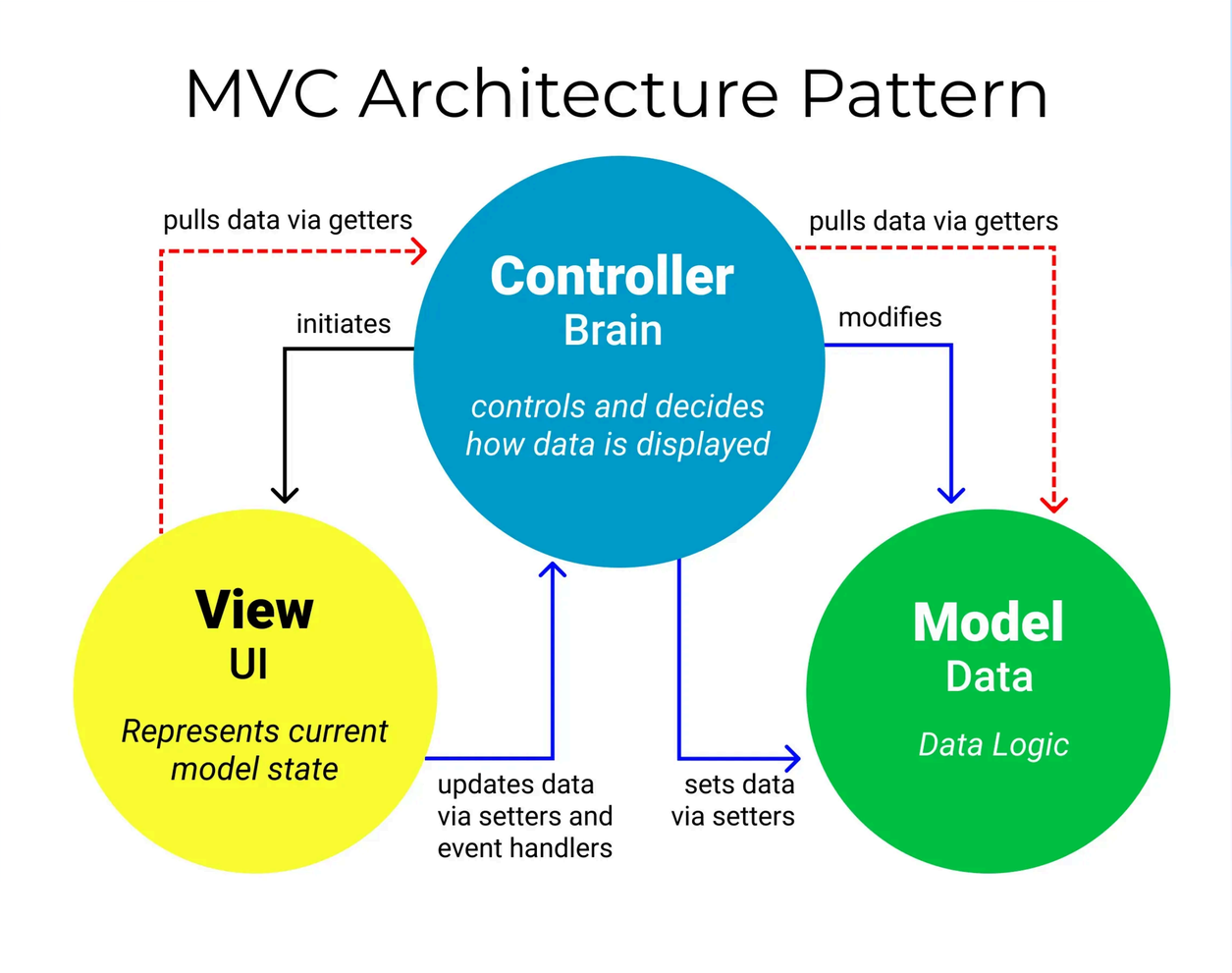
- Cách config để start một application trong Spring Boot cũng đơn giản hơn rất nhiều



MVC PATTERN

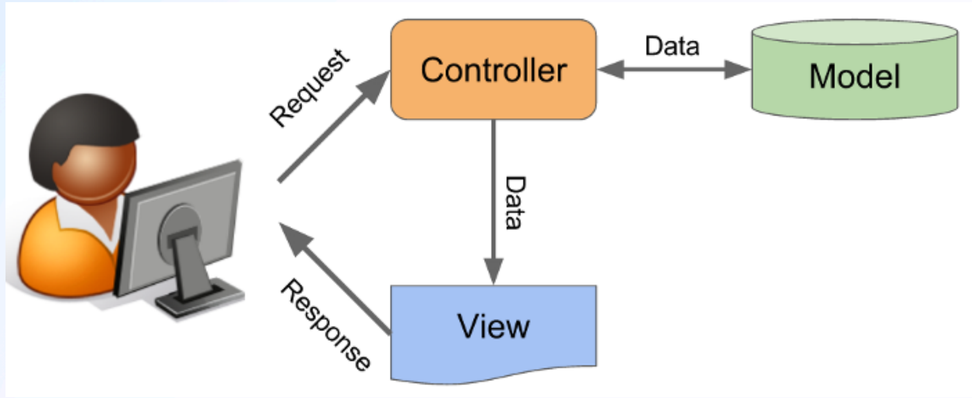
- MVC là viết tắt của “Model-View-Controller”. Đây là mô hình thiết kế được sử dụng trong kỹ thuật phần mềm để tạo lập giao diện người dùng và xử lý trên máy tính. MVC gồm 3 thành phần có nhiệm vụ riêng và độc lập kết nối với nhau
  - **Model** (dữ liệu) - Quản lý xử lý các dữ liệu (DB)
  - **View** (giao diện) - Nơi hiển thị dữ liệu cho người dùng
  - **Controller** (điều khiển) - Tiếp nhận yêu cầu người dùng, điều khiển tương tác giữa hai thành phần Model và View

MVC PATTERN

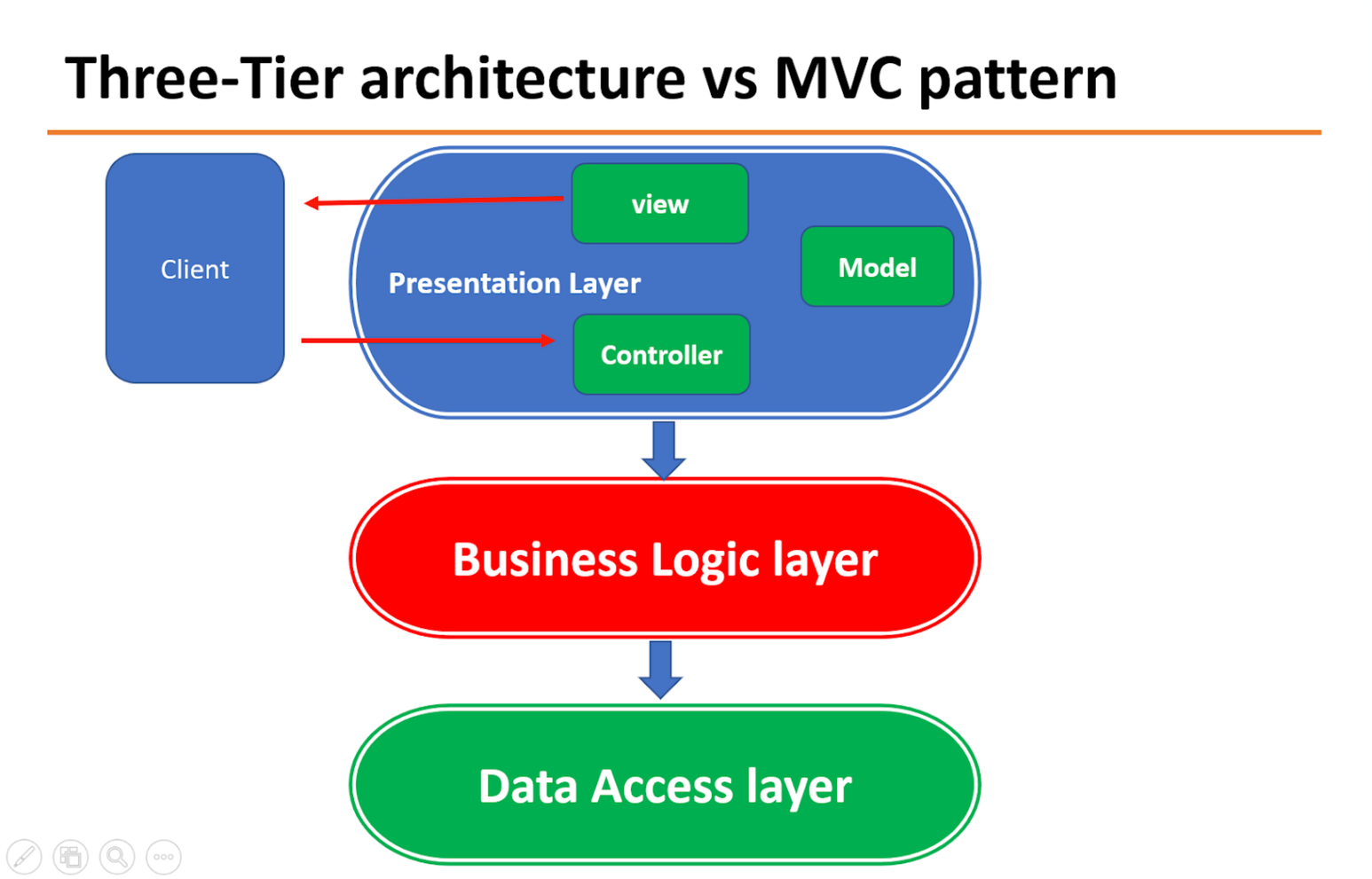

MVC Architecture Pattern
pulls data via getters
Controller
Brain
controls and decides how data is displayed
pulls data via getters
initiates
modifies
View
UI
Represents current model state
Model
Data
Data Logic
updates data via setters and event handlers
sets data via setters

LUỒNG XỬ LÝ TRONG MVC

- Khi một yêu cầu gửi từ máy Client đến Server, Controller sẽ tiếp nhận để phân tích lựa chọn Action phù hợp (dựa vào Route).
- Action method được chọn trên controller sẽ giao tiếp với Model để yêu cầu các dữ liệu tương ứng với yêu cầu người dùng.
- Model chuẩn bị data và gửi lại cho Controller.
- Controller sử dụng dữ liệu xử lý theo nghiệp vụ rồi gửi lại dữ liệu kết quả về View.
- View sẽ hiển thị kết quả cho người dùng trên trình duyệt.



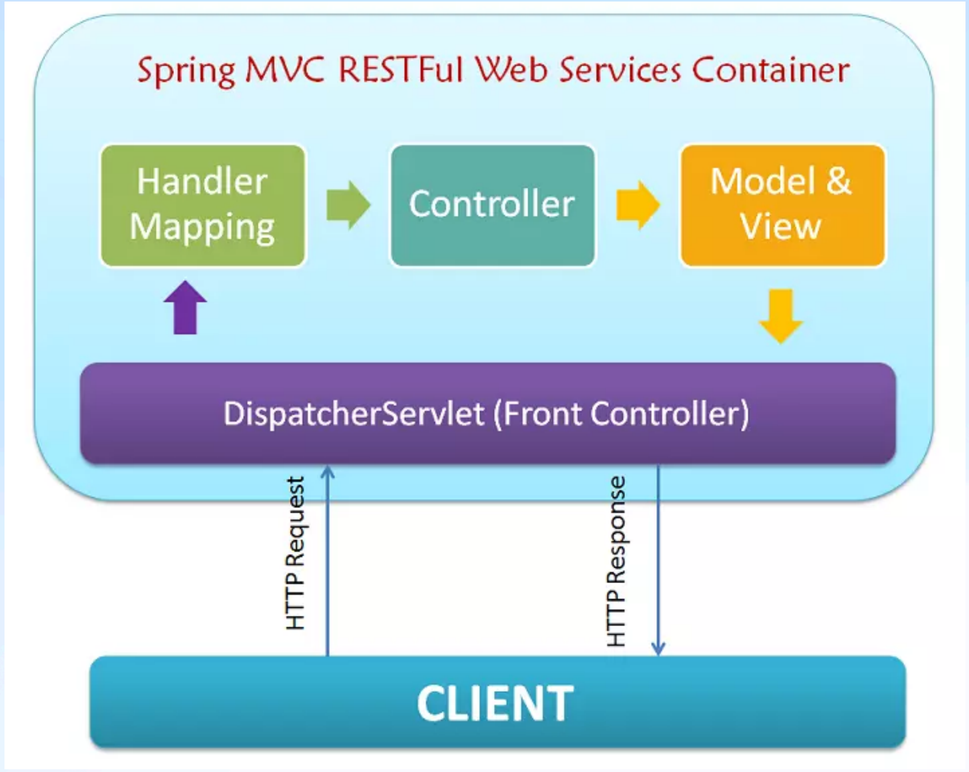
TỔ CHỨC SOURCE CODE



CONTROLLER TRONG SPRING BOOT

- Controller là một bean class được đánh dấu với Annotation là @controller hoặc @RestController.
    - @Controller có thể trả về View qua một String hoặc JSON data trong response body (nếu được chỉ định). Thích hợp cho các controller có routing, chuyển trang.
    - @RestControler chỉ có thể trả về data trong response body. Thích hợp cho các controller cung cấp các API

CONTROLLER TRONG SPRING BOOT



CONTROLLER MAPPING

- Controller sử dụng hai thông tin của HTTP Request để mapping với các action tương ứng.
  - URL - Xác định nơi mà Request muốn đến
  - HTTP Method - Thể hiện hành động (GET, POST, PUT, DELETE…) của tương tác.
- Cú pháp: @RequestMapping(value="/", method= RequestMethod.*GET*)
- Các annotation phổ biến như: @GetMapping, @PostMapping, @PutMapping, v.v.. hoặc sử dụng @RequestMapping với các thuộc tính method (nếu không chỉ định thì được hiểu mặc định là GET).

REQUEST DATA

- Controller nhận dữ liệu từ request, tuỳ vào dữ liệu nằm ở đâu mà có cách lấy khác nhau:
  - Request param (query string)
  - Path variable
  - Request Body
  - Header

REQUEST PARAM (QUERY STRING)

- Ví dụ ta có request sau: GET /users?dob=1990&name=Dũng thì chúng ta có 2 request param là dob=1990 và name=Dũng.
- Để lấy giá trị của 2 param này sử dụng @RequestParam("dob") int dob, @RequestParam("name") String name
- Trường hợp param là không bắt buộc thì sửa dụng thuộc tính required=false. Mặc định request param được định nghĩa là bắt buộc khi mapping.

```
@RequestParam(value = "age", required = false, defaultValue = "0") int age
```

PATH VARIABLE

- Path variable là một phần trong đường dẫn URL, ví dụ GET /users/123/info thì 123 là path variable. Path variable được định nghĩa trong annotation bằng cặp dấu { và }. Ví dụ: @GetMapping(“/users/{id}/info”)
- Sử dụng annotation @PathVariable để get giá trị của tham số.

```
@PathVariable(value = "id", required = false) int userId
```

- Có thể sử dụng Optional cho PathVariable như

```
@PathVariable Optional<String> id
```

REQUEST BODY

- Chỉ request method POST, PUT mới có request body, đây là nơi chứa data chính của request gửi lên. Thường thì request body sẽ ở dạng JSON hoặc form-data. Khi mapping vào controller sẽ tự đụng parse ra thành các object.

```
@PostMapping(“/login")
public ResponseEntity<?> login(@RequestBody LoginDto
loginDto) {
    // Dữ liệu trong request body có thể là JSON, form-
data,...
    // Tuy nhiên khi vào controller sẽ bị parse thành
object hết
}
```

HEADER

```
@PostMapping("/login")
public ResponseEntity<?>
login(@RequestHeader("Authorization") String authHeader) {
    // Biến authHeader sẽ có giá trị là giá trị của
Authorization header
}
```

CONTROLLER RESPONSE

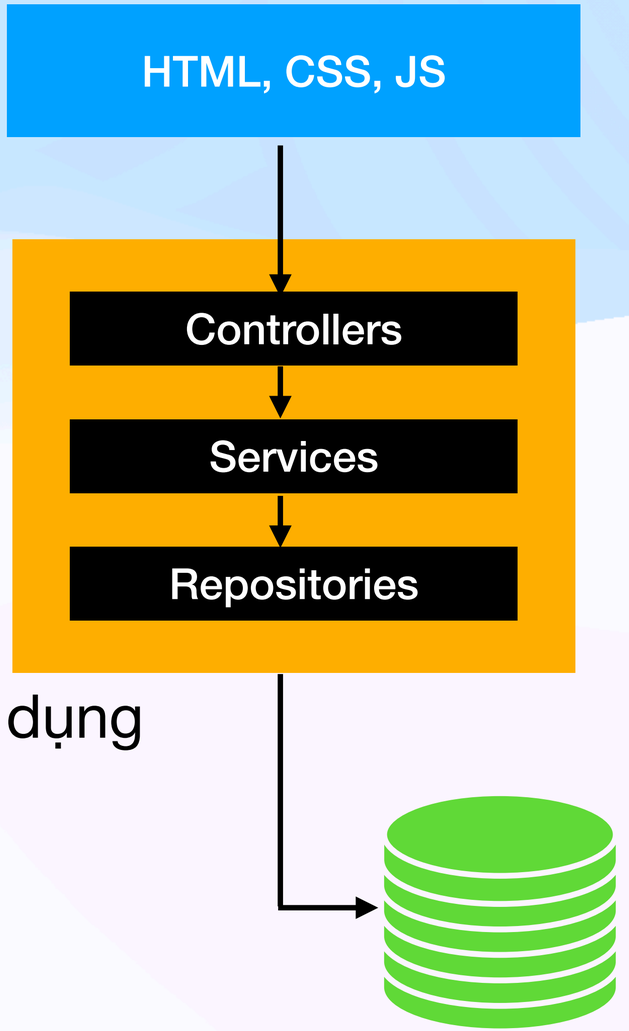
- Mặc định Spring Controller sẽ trả về một view có tên được chỉ định. Trong trường hợp muốn trả về một chuỗi dạng JSON thì sử dụng annotation @ResponseBody, nếu muốn thay JSON bằng XML thì có thể sử dụng thông qua thuộc tính produces trong @RequestMapping, hoặc @PostMapping, @GetMapping

```
@PostMapping(value = "/content", produces =
MediaType.APPLICATION_JSON_VALUE)
@ResponseBody
public ResponseTransfer postResponseJsonContent(
        @RequestBody LoginForm loginForm) {
    return new ResponseTransfer("JSON Content!");
}
```

- Dùng class ResponseEntity<T>. Đây là cách dùng đc đề xuất khi làm việc với các controller không sử dụng View.

APPLICATION LAYERS TRONG SPRING BOOT

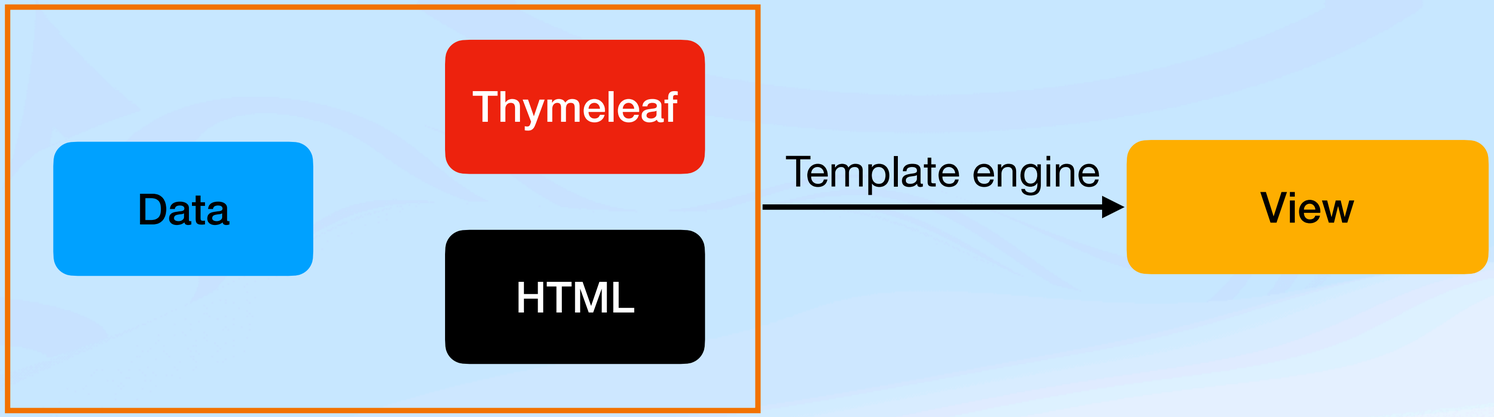
- Trong Project thường có 2 phần: Frontend và Backend
  - Frontend: là những file html, javascript, css v.v..
  - Backend: là java code xử lý yêu cầu từ frontend
- Trong Backend được tổ chức source code thành các layer
  - Controller: Nhận và xử lý request, sau đó gọi tiếp service
  - Service: Chứa các cài đặt nghiệp vụ của ứng dụng
  - Repositories: Implement các thao tác với dữ liệu của. ứng dụng



THYMELEAF

- Thymeleaf là một XML/XHTML/HTML5 template Engine, nó được sử dụng thay thế cho JSP trên tầng View trong các ứng dụng Java Web.
- Các lợi ích của Thymeleaf
  - Với Thymeleaf, ta chỉ cần sử dụng file HTML là có thể hiển thị mọi thứ.
  - Thymeleaf sẽ tham gia vào render các file HTML dưới dạng các thuộc tính có trong thẻ HTML-> do đó không cần thêm bất kỳ thẻ non-HTML nào.
  - Thymeleaf hỗ trợ cơ chế cache, do đó ta có thể cache dữ liệu hoặc custom để hiển thị view khi có thay đổi mà không cần restart server.

THYMELEAF VIEW



- Data được truyền từ controller xuống View thông qua một Model object.

```
@RequestMapping(“/")
public String index(final Model model){
    model.addAttribute("message", "Hello spring boot");
    return "index";
}
```

THYMELEAF VIEW

```
<html>
<head>
    <title>Welcome to my page</title>
</head>
<body>
    <h1>Welcome to Spring boot</h1>
    <p th:text="${message}">Default message</p>
</body>
</html>
```

THYMELEAF VIEW BINDING SYNTAX

- Loop: `th:each="obj : ${model}"` -> bên trong vòng lặp muốn truy cập tới thuộc tính nào thì sử dụng cú pháp `th:text="${obj.attribute}"` hoặc dùng `th:text="*{attribute}"` nếu đặt `th:object="${obj}"`
- Access to object attribute: `th:text="${obj.Id}"`
- Create link href: `th:href="@{/task}"`, sử dụng tương tự cho hyperlink
- Set source image: `th:src="@{path}"`, sử dụng chung cho các thuộc tính location
- Kiểm tra điều kiện: `th:if="${errorMessage}"`
- `th:action="@{/addPerson}"` sử dụng để định nghĩa action cho form.
- `th:field="*{fistName}"` sử dụng để định nghĩa cho các field trong form

SPRING BOOT JPA

- **Spring Boot JPA** là một phần trong hệ sinh thái Spring Data, giúp tạo ra một layer ở giữa tầng Service và Database, giúp chúng ta thao tác với Database một cách dễ dàng.
- Spring Boot JPA đã wrapper Hibernate và tạo ra một interface mạnh mẽ, giúp giải quyết các khó khăn khi làm việc với Hibernate.
- Để thêm Spring JPA vào project, ta cần thêm dependency spring-boot-starter-data-jpa.
- Để sử dụng CSDL, ta cần thêm các driver tương ứng bằng các dependency tương ứng. ví dụ: với MySQL thì thêm dependency mysql-connector-java

WHY JPA?

JPA là một công nghệ quan trọng trong việc phát triển ứng dụng Java vì:

- Viết code ít hơn: Sử dụng JPA, chúng ta có thể sử dụng các đối tượng Java thay vì phải viết các câu truy vấn SQL hoặc sử dụng JDBC trực tiếp. Điều này làm cho code trở nên ngắn gọn hơn và dễ đọc hơn.
- Dễ bảo trì: Với JPA, việc thay đổi cấu trúc CSDL trở nên dễ dàng hơn. Chúng ta chỉ cần thay đổi định nghĩa của đối tượng Java và JPA sẽ tự động áp dụng các thay đổi này vào CSDL tương ứng.
- Hiệu suất cao: JPA giúp tối ưu hoá việc truy xuất và lưu trữ dữ liệu trong CSDL. Nó sử dụng các kỹ thuật như lazy loading và caching để giảm thiểu số lần truy cập vào CSDL.

DATABASE CONNECTION

- **MySQL**

```
spring.datasource.url=jdbc:mysql://localhost:3306/micro_db?useSSL=false
spring.datasource.username=root
spring.datasource.password=root


## Hibernate Properties
# The SQL dialect makes Hibernate generate better SQL for the chosen database
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect
# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto = update
```

- **SQL Server**

```
spring.datasource.url= jdbc:sqlserver://
localhost:1433;encrypt=true;trustServerCertificate=true;databaseName=Demo
spring.datasource.username= sa
spring.datasource.password= @someThingComplicated1234

spring.jpa.properties.hibernate.dialect= org.hibernate.dialect.SQLServerDialect
spring.jpa.hibernate.ddl-auto= update
```

SPRING JPA ENTITY

- Trong JPA, một Entity là một đối tượng Java ánh xạ vào một bảng trong CSDL. Đối tượng Entity chứa thông tin về cấu trúc và dữ liệu của một table.
- Để định nghĩa một Entity trong JPA, chúng ta có thể sử dụng các annotation như: @Entity, @Table, @Column

```
@Entity
@Table(name = "employees")
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "name")
    private String name;

    // getters and setters
}
```

ENTITY (1-1)

- Để tạo quan hệ one-one (1-1) trong Spring JPA, ta sử dụng annotation @OneToOne

```
@Entity
@Data
@Builder
public class Address { // Table address
    @Id
    @GeneratedValue
    private Long id;

    private String city;
    private String province;

    @OneToOne // Đánh dấu có mỗi quan hệ 1-1 với Person ở phía dưới
    @JoinColumn(name = "person_id") // Liên kết với nhau qua khóa ngoại person_id
    private Person person;
}
```

ENTITY RELATION(1-n)

- Quan hệ 1 - nhiều: Phía entity 1 có Primary Key trở thành Foreign Key bên Entity nhiều.
  - Unidirection: Từ Entity A có thể truy ra Entity B nhưng ngược lại không được
  - Bidirection: Từ Entity A truy ra B và từ B truy ra A.
- Ví dụ:
  - Có nhiều phân loại Category. Mỗi sản phẩm có một phân loại.
  - Một Post có nhiều Comment. Một comment chỉ gắn vào một Post: bidirection one-many

ENTITY RELATION(1-n)

- Để tạo quan hệ 1-n, ta sử dụng annotation @OneToMany (Unidirection), hoặc thêm @ManyToOne để tạo 1 -1 Bidirection.

```
@Entity
public class Department {

    @Id
    private Long id;

    @OneToMany
    @JoinColumn(name = "department_id")
    private List<Employee> employees;
}

@Entity
public class Employee {

    @Id
    private Long id;
}
```

ENTITY (n-n)

- Quan hệ 1-1 sử dụng @ManyToMany annotation

```
@Entity
public class Book {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;

    @ManyToMany
    @JoinTable(name = "book_author",
            joinColumns = @JoinColumn(name = "book_id"),
            inverseJoinColumns = @JoinColumn(name = "author_id"))
    private Set<Author> authors;

}

@Entity
public class Author {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
}
```

SPRING SESSION

- Spring Session là một module của Spring Framework, giúp quản lý session của user khi sử dụng web bằng một hệ thống lưu trữ khác thay vì quản lý session sử dụng server runtime. Ví dụ: MongoDB, Redis hoặc Hazelcast.
- Để sử dụng Spring Session sử dụng dependency Spring Session JDBC

```
<dependency>
<groupId>org.springframework.session</groupId>
<artifactId>spring-session-jdbc</artifactId>
</dependency>
```

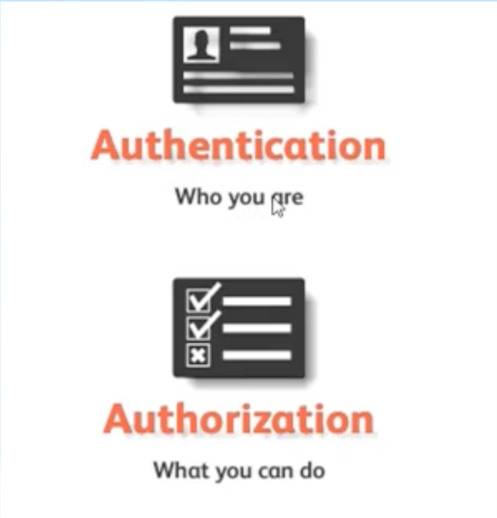
- Setting session store type là jdbc

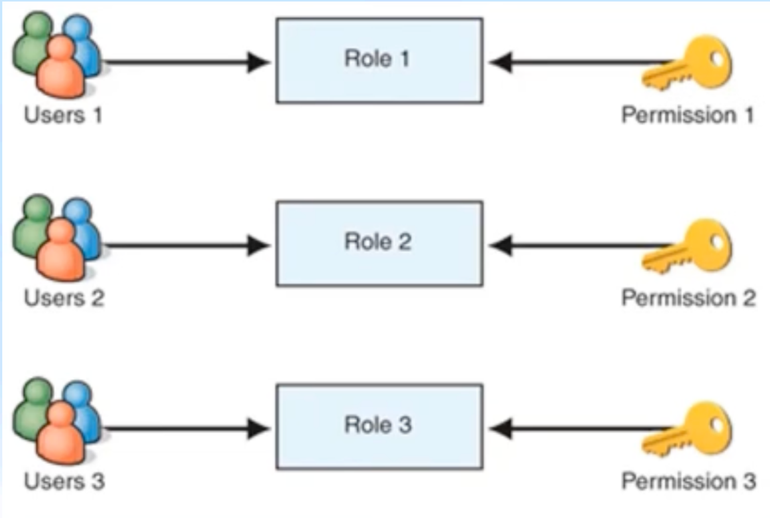
```
spring.session.store-type=jdbc
spring.session.jdbc.initialize-schema=always
spring.session.timeout=180
```

Authentication và Authorization

- Spring Security cung cấp các dịch vụ bảo mật toàn diện cho các ứng dụng phát triển dựa trên nền tảng Spring Framework.
- Spring Security cung cấp 2 cơ chế cơ bản:
  - Authentication (xác thực): là tiến trình thiết lập một principal. Principal có thể hiểu là một người, hoặc một thiết bị, hoặc một hệ thống nào đó có thể thực hiện một hành động trong ứng dụng.
  - Authorization (phân quyền) hay Access-control: là tiến trình quyết định xem một principal có được phép thực hiện một hành động trong ứng dụng của bạn hay không? Trước khi diễn tiến tới Authorization, principal cần phải được thiết lập bởi Authentication.

Authentication và Authorization





Các thành phần cốt lõi trong Spring Security

- SecurityContext: là interface cốt lõi của Spring Security, lưu trữ tất cả chi tiết liên quan đến bảo mật trong ứng dụng. Khi chúng ta kích hoạt Spring Security trong ứng dụng thì SecurityContext cũng sẽ được kích hoạt theo, để access đến SecurityContext thì chúng ta thông qua SecurityContextHolder.
- Principal: biểu diễn thông tin của User đăng nhập vào hệ thống sau khi được xác thực (Authentication), từ đây có thể extract được một vài thông tin của User như username.
- UserDetails: là một interface cốt lõi của SpringSecurity, nó đại diện cho một principal nhưng theo một cách mỡ rộng và cụ thể hơn.

  UserDetails bao gồm các method sau:

Các thành phần cốt lõi trong Spring Security

- getAuthorities(): trả về danh sách các quyền của người dùng
- getPassword(): trả về password đã dùng trong quá trình xác thực.
- getUsername(): trả về username đã dùng trong quá trình xác thực.
- isAccountNonExpired() trả về true nếu tài khoản của người dùng chưa hết hạn.
- isAccountNonLocked() trả về true nếu người dùng chưa bị khoá
- isCredentialsNonExpired() trả về true nếu chứng thực (mật khẩu) của người dùng chưa hết hạn.
- isEnable() trả về true nếu người dùng đã được kích hoạt.