

**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP.HCM**  
**KHOA ĐÀO TẠO CHẤT LƯỢNG CAO**  
**NGÀNH CÔNG NGHỆ THÔNG TIN**



**Môn Học: Lập trình Web**

**GVHD:** *ThS.Nguyễn Minh Đạo*



**ĐỀ TÀI:**  
**QUẢN LÝ NGƯỜI DÙNG ĐĂNG BÀI VIẾT**

**Mã môn học:** WEPR330479

**Sinh Viên Thực Hiện:** Hồ Xuân Hiếu

**MSSV:** 20119339

**TP. Hồ Chí Minh, tháng 12 năm 2022**

## **LỜI CẢM ƠN**

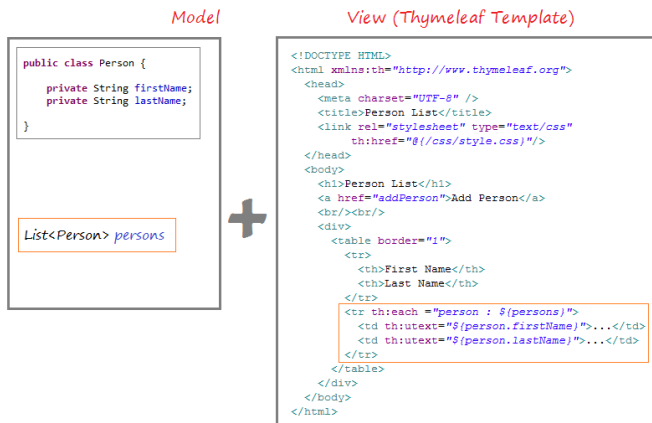
Lời đầu, em Hồ Xuân Hiếu xin chân thành cảm ơn quý thầy cô khoa Công Nghệ Thông Tin nói chung và thầy Nguyễn Minh Đạo nói riêng đã tận tình giảng dạy, trang bị cho em những kiến thức cần thiết và quý báu trong học kỳ vừa qua, giúp em có một nền tảng vững chắc để áp dụng và hoàn thành đề tài này.

Mặc dù em đã cố gắng hết sức và dành rất nhiều thời gian để hoàn thành đề tài trong khả năng nhưng chắc chắn sẽ không tránh khỏi những thiếu sót. Nhóm kính mong nhận được sự thông cảm và tận tình đóng góp ý kiến của quý thầy cô và các bạn.

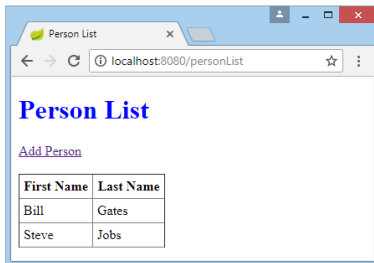
Lời cuối, em xin chân thành cảm ơn !



+



Thymeleaf Engine



# 1. Tìm hiểu về Thymeleaf và Java Spring Boot

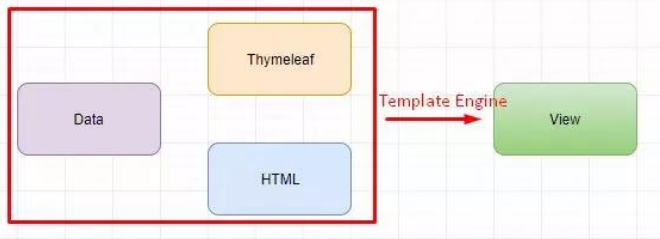
## 1.1. Thymeleaf là gì?

Thymeleaf là một Java Template Engine. Có nhiệm vụ xử lý và generate ra các file HTML, XML, v.v..

Mục tiêu chính của Thymeleaf là mang các mẫu tự nhiên trang nhã vào quy trình phát triển của bạn — HTML có thể được hiển thị chính xác trong trình duyệt và cũng hoạt động như các nguyên mẫu tĩnh, cho phép cộng tác mạnh mẽ hơn trong các nhóm phát triển.

Với các mô-đun dành cho Spring Framework, một loạt tích hợp với các công cụ yêu thích của bạn và khả năng bổ sung chức năng của riêng bạn, Thymeleaf là lựa chọn lý tưởng để phát triển web HTML5 JVM hiện đại — mặc dù nó có thể làm được nhiều hơn thế.

Thymeleaf có thể làm việc với cả hai môi trường Web và môi trường không phải Web. Nó phù hợp hơn khi được sử dụng để phục vụ XHTML/HTML5 trên tầng View (View Layer) của ứng dụng Web dựa trên kiến trúc MVC. Nó có thể xử lý bất kỳ một file XML nào, thậm chí trên các môi trường offline (Không trực tuyến). Nó hỗ trợ đầy đủ để tương tác với Spring Framework.

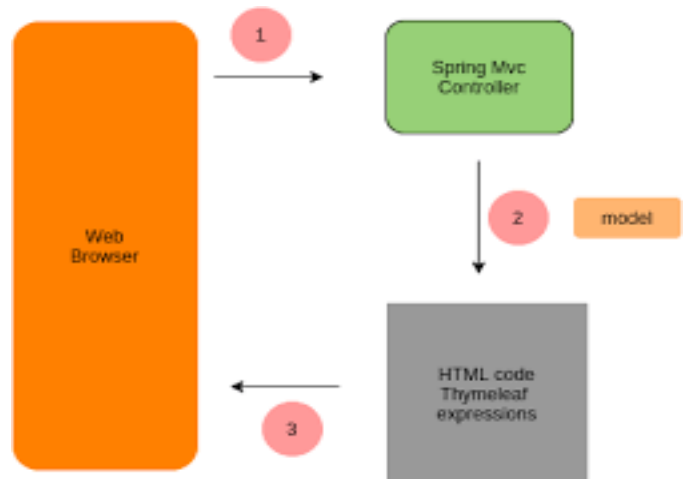
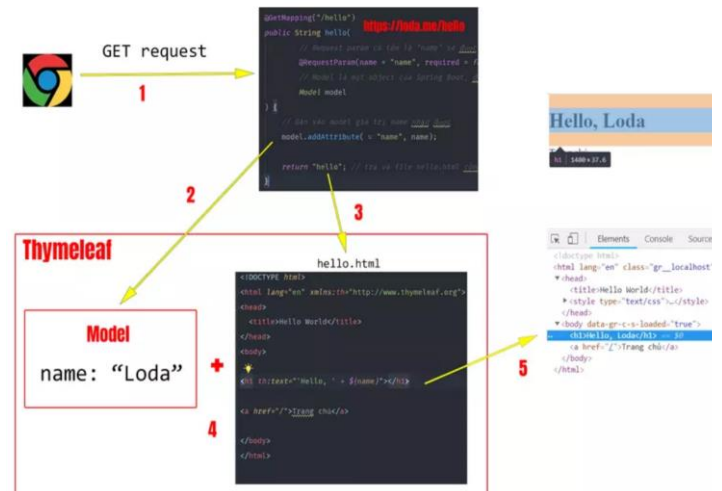


Thymeleaf cho phép bạn xử lý 6 loại template, mỗi loại được gọi là một chế độ template bao gồm:

- HTML
- XML
- TEXT
- JAVASCRIPT
- CSS
- RAW

### HTML Template

Template này cho phép bất kỳ loại đầu vào là các file **HTML** như **HTML5**, **HTML4** và **XHTML**. Việc kiểm tra tính hợp lệ hoặc tính đúng đắn sẽ được thực hiện và cấu trúc mẫu (hoặc code mẫu) sẽ được ưu tiên ở mức độ cao nhất trong đầu ra.



```

1 <table>
2   <thead>
3     <tr>
4       <th th:text="#{msgs.headers.name}">Name</th>
5       <th th:text="#{msgs.headers.price}">Price</th>
6     </tr>
7   </thead>
8   <tbody>
9     <tr th:each="prod: ${allProducts}">
10      <td th:text="${prod.name}">Oranges</td>
11      <td th:text="${#numbers.formatDecimal(prod.price, 1, 2)}">0.99</td>
12    </tr>
13  </tbody>
14 </table>
  
```

## **XML Template**

Cho phép đầu vào là các file XML. Trình phân tích cú pháp sẽ ném ra lỗi khi file XML vi phạm các lỗi như thiếu thẻ đóng/mở, thiếu dấu <>, thiếu thuộc tính v.v. Tuy nhiên trình phân tích cú pháp không áp dụng với DTD hoặc schema.

## **Text Template**

Cho phép sử dụng cú pháp đặc biệt cho các template không đánh dấu (markup). Ví dụ về các template như vậy có thể là email hoặc các tài liệu thông thường. Lưu ý rằng, các template HTML hoặc XML cũng có thể được xử lý dưới dạng TEXT, trong trường hợp này, chúng sẽ không được coi là đánh dấu (markup) và tất cả các thẻ, DOCTYPE, comment, v.v. sẽ được coi là văn bản thuần túy.

## **Javascript Template**

Cho phép xử lý các tệp JavaScript trong ứng dụng Thymeleaf. Điều này có nghĩa là có thể sử dụng dữ liệu mô hình bên trong các tệp JavaScript theo cùng cách thực hiện trong các tệp HTML. Chế độ template JAVASCRIPT được coi là văn bản và do đó, nó sử dụng cú pháp đặc biệt giống như chế độ template TEXT.

## **CSS Template**

Template này sẽ cho phép xử lý các tệp CSS liên quan đến ứng dụng Thymeleaf. Tương tự như chế độ template JAVASCRIPT, chế độ template CSS cũng là một chế độ văn bản và sử dụng cú pháp xử lý đặc biệt từ chế độ template TEXT.

## **RAW Template**

Template này đơn giản là không xử lý các template. Có nghĩa là, nó được sử dụng để chèn các tài nguyên không bị ảnh hưởng (tệp, phản hồi URL, v.v.) vào các template đang được xử lý. Ví dụ, các tài nguyên bên ngoài, không được kiểm soát ở định dạng HTML có thể được đưa vào các template ứng dụng, sẽ là an toàn khi biết rằng bất kỳ mã Thymeleaf nào mà các tài nguyên này có thể include sẽ không được thực hiện.

## Ưu và nhược điểm của Thymeleaf

| Ưu điểm  | Nhược điểm  |
|--|---|
| ThymeLeaf là một dự án nguồn mở lành mạnh: các tính năng mới được phát hành mỗi tháng, tài liệu tốt... | Thymeleaf không có hỗ trợ thư viện thẻ JSP            |
| Đây là công cụ template lý tưởng   | Thymeleaf chưa có thẻ tương đương với các thẻ custom. |
| Expression Language được sử dụng nhiều hơn JSP Expression Language.                                    |   |
| Không giống như các tệp JSP, Thymeleaf hoạt động tốt đối với các email Rich HTML.                      |   |

### Cú pháp

Cú pháp của **Thymeleaf** sẽ là một **attributes** (Thuộc tính) của thẻ HTML và bắt đầu bằng chữ th:.

Với cách tiếp cận này, bạn sẽ chỉ cần sử dụng các thẻ HTML cơ bản đã biết mà không cần bổ sung thêm syntax hay thẻ mới như JSP truyền thống.

#### Ví dụ:

Để truyền dữ liệu từ biến name trong Java vào một thẻ H1 của HTML.

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

Chúng ta viết thẻ H1 như bình thường, nhưng không chứa bất cứ text nào trong thẻ. Mà sử dụng cú pháp **th:text="\${name}"** để **Thymeleaf** lấy thông tin từ biến **name** và đưa vào thẻ **H1**.

Kết quả khi render ra:

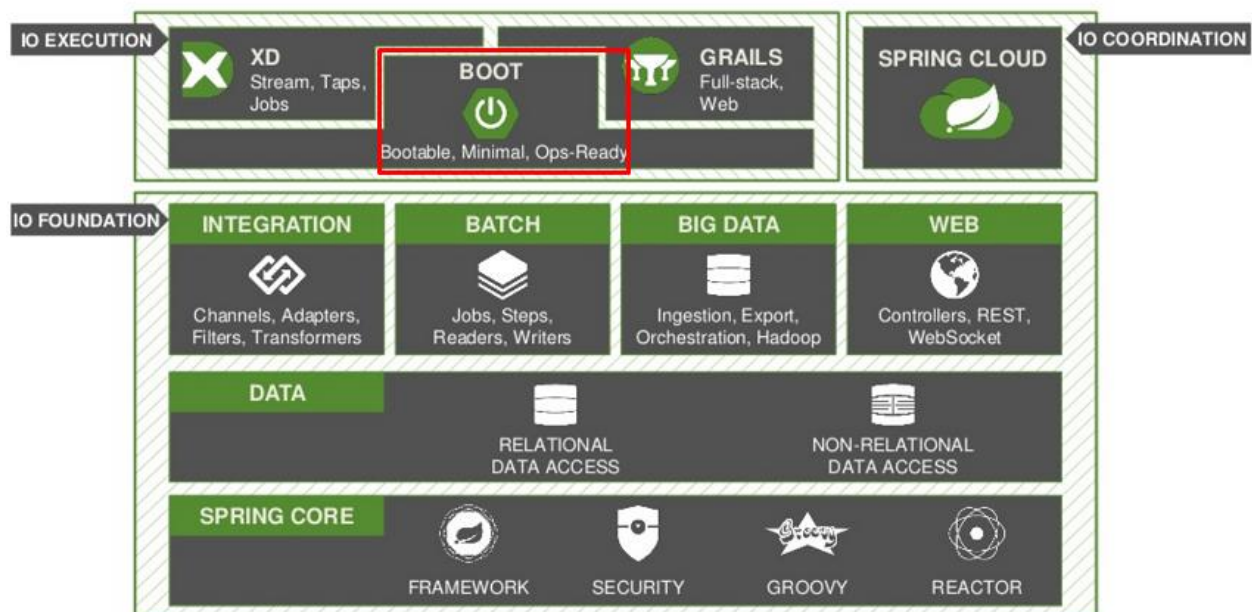
```
// Giả sử String name = "loda"
<h1>Loda</h1>
```

Thuộc tính th:text biến mất và giá trị biến name được đưa vào trong thẻ H1.

Đó là cách **Thymeleaf** hoạt động.



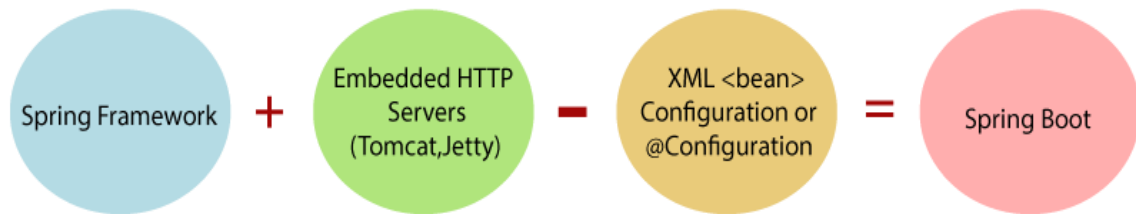
## 1.2. Java Spring Boot



Spring Boot là một module nằm trong Spring Framework, nó cung cấp giao diện và khả năng phát triển các ứng dụng độc lập với rất ít các bước cấu hình rườm rà hoặc gần như bằng không.

Spring Boot được đóng gói với rất nhiều thư viện phụ thuộc các module nền tảng của Spring Framework nhưng được giảm thiểu đi các mã nguồn dài dòng, phức tạp nhằm cung cấp sự thuận tiện và phù hợp với từng mục đích khi phát triển ứng dụng.

## Một số tính năng nổi bật của Spring Boot



- Tạo các ứng dụng Spring độc lập
- Nhúng trực tiếp Tomcat, Jetty hoặc Undertow (không cần phải deploy ra file WAR)
- Các starter dependency giúp việc cấu hình Maven đơn giản hơn
- Tự động cấu hình Spring khi cần thiết
- Không sinh code cấu hình và không yêu cầu phải cấu hình bằng XML ...

**Để phát triển một ứng dụng web cơ bản HelloWorld sử dụng Spring framework sẽ cần ít nhất 5 công đoạn sau:**

Tạo một project sử dụng Maven với các dependency cần thiết của Spring MVC và Servlet API.

Một tập tin web.xml để khai báo DispatcherServlet của Spring MVC.

Một tập tin cấu hình của Spring MVC.

Một class Controller trả về một trang “Hello World” khi có request đến.

Cuối cùng là phải có một web server dùng để triển khai ứng dụng lên chạy.

Trong các công đoạn này, chỉ có công đoạn tạo một class Controller thì có thể khác cho các ứng dụng khác nhau vì mỗi ứng dụng có một yêu cầu khác nhau. Còn các công đoạn khác thì như nhau.



## Sự khác nhau giữa Spring và Spring Boot

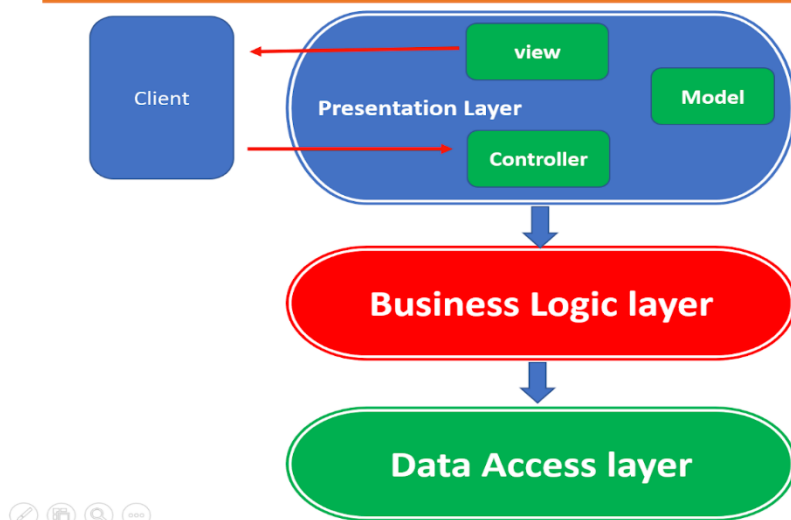


| Spring  | Spring Boot  |
|---|--|
| <b>Spring Framework</b> là một khung Java EE được sử dụng rộng rãi để xây dựng các ứng dụng.                      | Spring Boot Framework được sử dụng rộng rãi để phát triển các <b>REST APIs</b> .   |
| Nó nhằm mục đích đơn giản hóa việc phát triển Java EE giúp các nhà phát triển làm việc hiệu quả hơn.              | Nó nhằm mục đích rút ngắn độ dài mã và cung cấp cách dễ dàng nhất để phát triển <b>Web Application</b> .   |
| Tính năng chính của Spring Framework là <b>dependency injection</b> .   | Tính năng chính của Spring Boot là <b>Autoconfiguration</b> . Nó tự động cấu hình các lớp dựa trên yêu cầu.                                      |
| Nó giúp làm cho mọi thứ đơn giản hơn bằng cách cho phép chúng ta phát triển <b>loosely coupled applications</b> . | Nó giúp tạo một ứng dụng độc lập với ít cấu hình hơn.  |
| Lập trình viên cần viết rất nhiều mã (mã viết sẵn) để thực hiện nhiệm vụ tối thiểu.                               | Nó làm giảm đi mã viết sẵn.  |
| Để kiểm tra dự án Spring, chúng ta cần thiết lập sever một cách rõ ràng.  | Spring Boot cung cấp máy chủ nhúng như Jetty và Tomcat, v.v.   |
| Các nhà phát triển xác định thủ công các dependencies cho dự án Spring trong pom.xml.                             | Spring Boot đi kèm với khái niệm khởi động trong tệp pom.xml, bên trong xử lý việc tải xuống các JAR phụ thuộc dựa trên Spring Boot Requirement. |

## Luồng đi trong Spring Boot

Cấu trúc source code của Spring Boot được dựa trên hai mô hình là **mô hình MVC** và **mô hình 3 lớp**.

### Three-Tier architecture vs MVC pattern



Kết hợp hai mô hình lại, chúng ta có được ứng dụng Spring Boot hoàn chỉnh, gồm các thành phần sau:

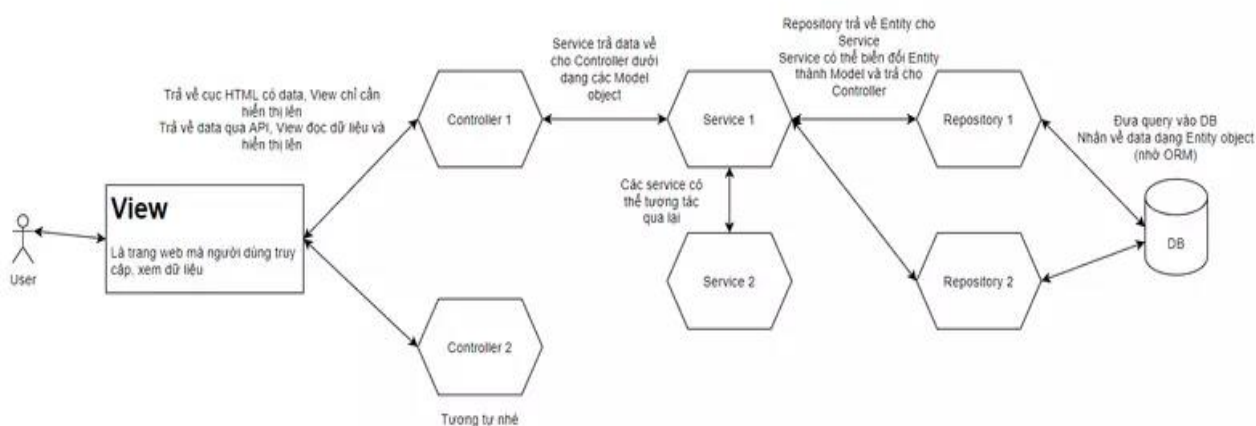
- Controller: trả về View (có chứa data sẵn, dạng trang HTML), hoặc Model thể hiện dưới dạng API cho View (View viết riêng bằng React, Vue, hoặc Angular).
- Service: chứa các code tính toán, xử lý. Khi Controller yêu cầu, thì Service tương ứng sẽ tiếp nhận và cho ra dữ liệu trả cho Controller (trả về Model). Controller sẽ gửi về View như trên.
- Repository: Service còn có thể tương tác với service khác, hoặc dùng Repository để gọi DB. Repository là thành phần trực tiếp tương tác, đọc ghi dữ liệu trong DB và trả cho service.

### Model và View trong Spring Boot

- Model chỉ đơn giản là các đối tượng được Service tính toán xong trả về cho Controller.
- View thì có 2 loại, một là dạng truyền thống là trả về 1 cục HTML có data rồi. Lúc này Controller sẽ pass dữ liệu vào View và return về (Spring MVC có JSP hoặc template engine như Thymeleaf làm điều đó).

- View dạng 2 là dạng View tách riêng (không thuộc về project Spring boot). Thường có trong các hệ thống dùng API. View sẽ được viết riêng bằng React, Angular,... Controller sẽ đưa dữ liệu Model thông qua API cho View, và cũng nhận lại các yêu cầu qua API luôn.

### Sơ đồ luồng đi



Xét sơ đồ theo chiều kim đồng hồ

- Đầu tiên, user sẽ vào View để xem, tương tác
- Khi user bắt đầu load dữ liệu (ví dụ click nút Reload), thì 1 request từ View gửi cho Controller.
- Controller nhận được yêu cầu, bắt đầu đi hỏi ông Service (trong code là gọi method của Service).
- Service nhận được yêu cầu từ Controller, đối với các code đơn giản có thể tính toán và trả về luôn. Nhưng các thao tác cần đụng tới database thì Service phải gọi Repository để lấy dữ liệu trong DB
- Repository nhận được yêu cầu từ Service, sẽ thao tác với DB. Data lấy ra trong DB được hệ thống ORM (như JPA hoặc Hibernate) mapping thành các object (trong Java). Các object này gọi là Entity.

Và bây giờ sẽ là đi ngược lại trả về cho user:

- Service nhận các Entity được Repository trả về, biến đổi nó. Biến đổi ở đây là có thể thực hiện tính toán, thêm bớt các field,... và cuối cùng biến Entity thành Model. Model sẽ được trả lại cho Controller.

- Controller nhận được Model, nó sẽ return cho View. Có 2 cách, một là dùng template engine pass dữ liệu Model vào trang HTML, rồi trả về cục HTML (đã có data) cho client. Cách 2 là gửi qua API, View tự parse ra và hiển thị tương ứng (hiển thị thế nào tùy View).

- Khi View hiển thị xong, user sẽ thấy danh sách user hiện lên trang web.

Một số mẹo hay để tổ chức luồng đi cho tốt:

- Giữ cho Controller càng ít code càng tốt. Vì Controller chỉ là trung gian kết nối thông, nên không nên chứa nhiều code, thay vào đó nên bỏ vào Service.

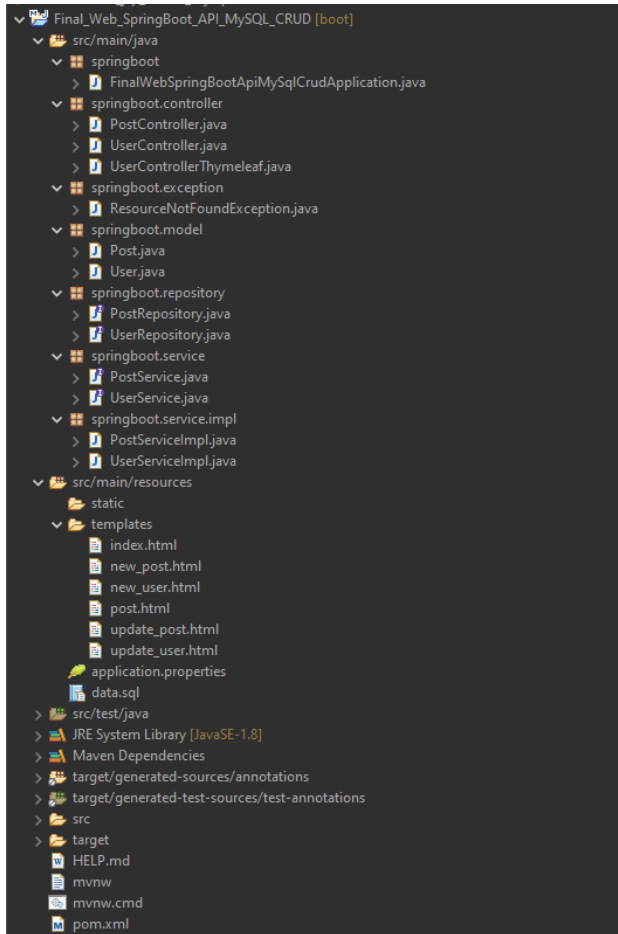
- Nên tách bạch Service rõ ràng. Không nên cho 1 service thực hiện nhiều công việc, nên tách ra nhiều Service.

Các thành phần trên có thể tương tác, gọi lẫn nhau, đó không gì khác ngoài cơ chế dependency injection trong Spring. Cụ thể như sau:

- Trong Controller được inject các Service cần thiết vào
- Trong Service được inject các Repository cần thiết vào

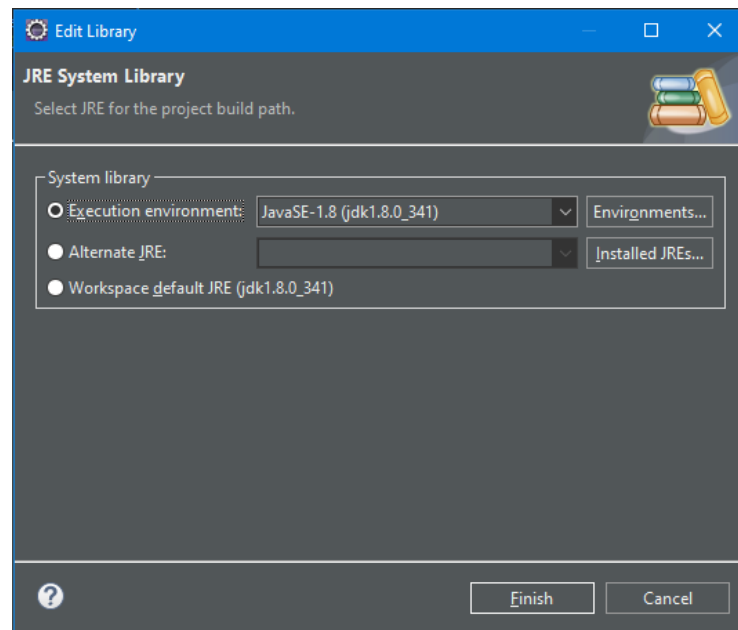
## 2. Trình bày về project

### - Cấu trúc project và file pom.xml:



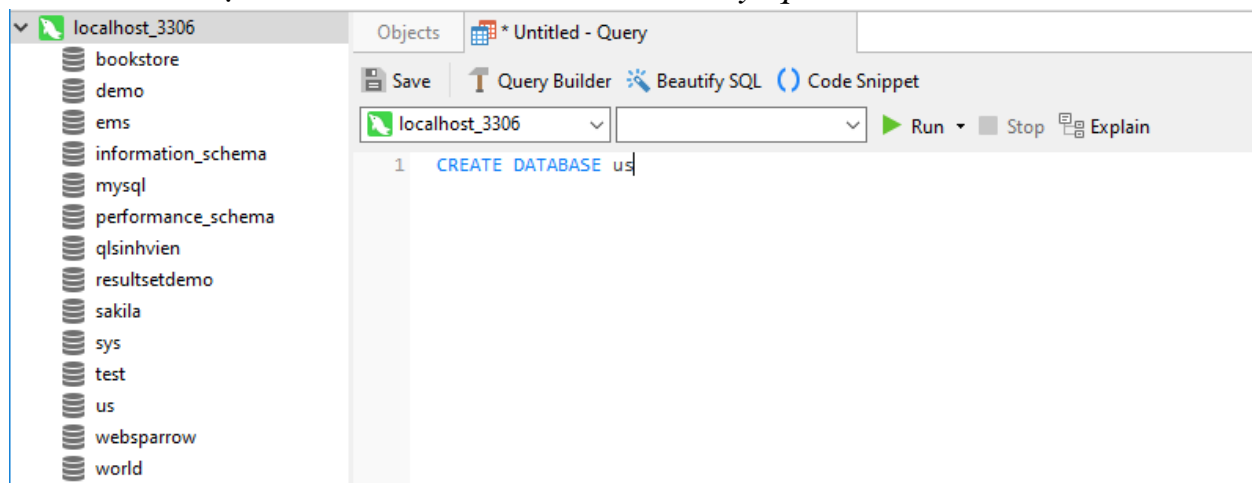
```
Final_Web_SpringBoot_API_MySQL_CRUD/pom.xml X
16 <properties>
17   <java.version>1.8</java.version>
18 </properties>
19 <dependencies>
20
21   <dependency>
22     <groupId>org.springframework.boot</groupId>
23     <artifactId>spring-boot-starter-actuator</artifactId>
24   </dependency>
25
26   <dependency>
27     <groupId>org.springframework.boot</groupId>
28     <artifactId>spring-boot-starter-data-jpa</artifactId>
29   </dependency>
30   <dependency>
31     <groupId>org.springframework.boot</groupId>
32     <artifactId>spring-boot-starter-web</artifactId>
33   </dependency>
34
35   <dependency>
36     <groupId>com.mysql</groupId>
37     <artifactId>mysql-connector-j</artifactId>
38     <scope>runtime</scope>
39   </dependency>
40   <dependency>
41     <groupId>org.projectlombok</groupId>
42     <artifactId>lombok</artifactId>
43     <optional>true</optional>
44   </dependency>
45   <dependency>
46     <groupId>org.springframework.boot</groupId>
47     <artifactId>spring-boot-starter-test</artifactId>
48     <scope>test</scope>
49   </dependency>
50   <dependency>
51     <groupId>org.springframework.boot</groupId>
52     <artifactId>spring-boot-starter-thymeleaf</artifactId>
53   </dependency>
54   <dependency>
55     <groupId>org.springdoc</groupId>
56     <artifactId>springdoc-openapi-ui</artifactId>
57     <version>1.6.12</version>
58   </dependency>
59 </dependencies>
60
61 <build>
62   <plugins>
63     <plugin>
64       <groupId>org.springframework.boot</groupId>
65       <artifactId>spring-boot-maven-plugin</artifactId>
66       <configuration>
67         <excludes>
```

Bao gồm: Spring Web, MySQL Driver, Spring Data JPA, Thymeleaf, Lombok

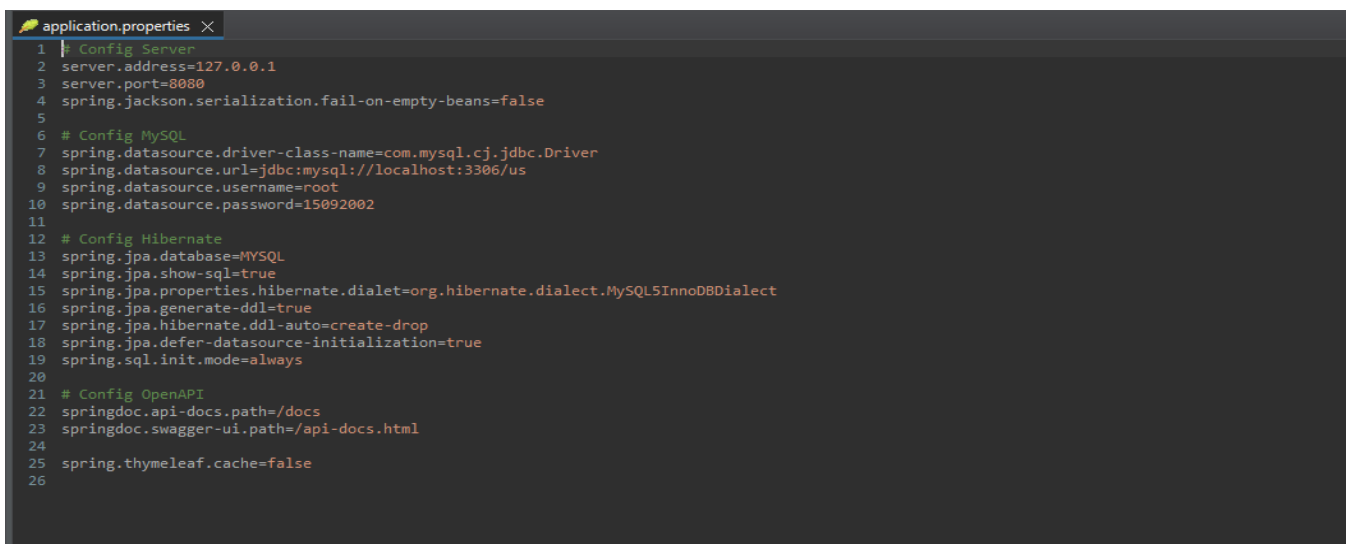


Vì chúng ta sử dụng Java 8 nên ta sẽ dùng jdk 1.8 cho project này

- *Tạo Database và kết nối đến csdl MySql:*



Tạo cơ sở dữ liệu có tên là *us*



Vào file `application.properties` để khai báo các yêu cầu để kết nối đến cơ sở dữ liệu, ở đây ta thấy sẽ kết nối đến csdl Mysql nên port kết nối là 3306 và kết nối đến csdl `us` với `username = root`, `password = 15092002` (tùy cá nhân).

#### - Các model:



#### *User.java*

```
User.java X
1  package springboot.model;
2
3  import java.io.Serializable;
27
28  @Getter
29  @Setter
30  @NoArgsConstructor
31  @AllArgsConstructor
32  @Entity
33  @EntityListeners(AuditingEntityListener.class)
34  @Table(name = "users")
35  public class User implements Serializable {
36
37      private static final long serialVersionUID = 1L;
38
39      @Id
40      @GeneratedValue(strategy = GenerationType.IDENTITY)
41      private long id;
42      @Column(name = "full_name", nullable = false)
43      private String fullName;
44      @Column(name = "email")
45      private String email;
46      @Column(name = "phone")
47      private String phone;
48
49      @CreatedDate
50      @Column(name = "created_at", nullable = false, updatable = false)
51      private Instant createdAt;
52
53      @LastModifiedDate
54      @Column(name = "updated_at")
55      private Instant updatedAt;
56
57      @JsonIgnoreProperties("user")
58      @OneToMany(
59          mappedBy = "user",
60          cascade = CascadeType.ALL,
61          orphanRemoval = true
62      )
63      private List<Post> posts;
64
65      //getter and setter
66
```

Model *User* có các biến như là: *id*, *fullName*, *email*, *phone*, *createdAt*, *updatedAt*, *posts* (1 list các bài *Post*), *getter & setter*. Cũng như có các ràng buộc như not null, khóa chính, không thể update. Và có mối quan hệ *@OneToMany* (một user có thể có nhiều bài posts).

## Post.java

```
1 package.springboot.model;
2
3 import java.io.Serializable;
4
23
24 @Data
25 @Entity
26 @EntityListeners(AuditingEntityListener.class)
27 @Table(name = "posts")
28 public class Post implements Serializable {
29
30     private static final long serialVersionUID = 1L;
31
32     @Id
33     @GeneratedValue(strategy = GenerationType.IDENTITY)
34     private long id;
35
36     @Column(name = "title")
37     private String title;
38     @Column(name = "content")
39     private String content;
40
41     @JsonIgnoreProperties("posts")
42     @ManyToOne(fetch = FetchType.LAZY)
43     @JoinColumn(name = "user")
44     private User user;
45
46     @CreatedDate
47     @Column(name = "created_at", nullable = false, updatable = false)
48     private Instant createdAt;
49
50     @LastModifiedDate
51     @Column(name = "updated_at")
52     private Instant updatedAt;
53
54     //getter and setter
55 }
```

Model *Post* có các biến như là: *id*, *title*, *content*, *user* (một model *User*, chủ sở hữu bài *Post*), *createdAt*, *updatedAt*, *getter & setter*. Cũng như có các ràng buộc như not null, khóa chính, không thể update. Và có mối quan hệ *@ManyToOne* (nhiều bài *Post* có thể được sở hữu bởi 1 *User*).

Chúng ta sẽ tạo *Entity* để định nghĩa các thuộc tính cho *User*, *Post* bao gồm các *getter & setter*. Các tag *@Data* và *@Entity* là của Spring giúp chúng ta định nghĩa cho *Entity*. Tag *@Table(name= " ")* sẽ giúp chúng ta tạo ra bảng có tên theo ý muốn với primary key là *Id*.

### - Repository

```
springboot.repository
├── PostRepository.java
└── UserRepository.java
```

## UserRepository.java

```
1 package.springboot.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
7 @Repository
8 public interface UserRepository extends JpaRepository<User, Long> {
9
10 }
11 }
```



## *PostRepository.java*

```
PostRepository.java X
1 package springboot.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5 @Repository
6 public interface PostRepository extends JpaRepository<Post, Long> {
7
8 }
9
10
11
```

Các file trong package *springboot.repository* cho phép chúng ta thao tác trực tiếp với csdl với các hàm có sẵn như là *findAll()*, *save()*, *findAllById()*, ...

### - Service & ServiceImpl

Service:

```
springboot.service
> PostService.java
> UserService.java
```

## *UserService.java*

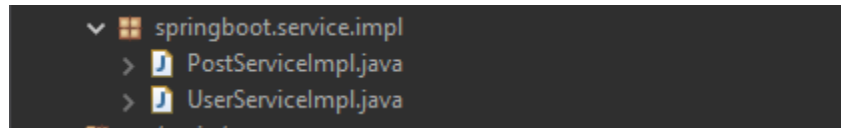
```
UserService.java X
1 package springboot.service;
2
3 import java.util.List;
4
5 public interface UserService {
6     User saveUser(User user);
7     void saveuser(User user);
8     List<User> getAllUsers();
9     User getUserById(long id);
10    User updateUser(User user, long id);
11    void deleteUser(long id);
12 }
13
14
15
```

## *PostService.java*

```
PostService.java X
1 package springboot.service;
2
3 import java.util.List;
4
5 public interface PostService {
6     Post savePost(Post post, long userId);
7     List<Post> getAllPost();
8     List<Post> getAllPostsByUser(long userId);
9     Post getPostById(long postId);
10    Post updatePost(Post post, long id);
11    void deletePost(long id);
12 }
13
14
15
```

Ở đây chúng ta có các hàm cơ bản tương tác với *User* & *Post* như là: Lưu model mới, lấy danh sách *User* & *Post*, tìm kiếm theo *Id*, cập nhật *User* & *Post* theo *Id*, xóa *User* & *Post* theo *Id*.

*ServiceImpl*:



*UserServiceImpl.java*

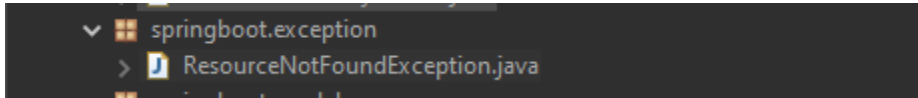
```
UserServiceImpl.java X
1 package springboot.service.impl;
2 import java.util.List;
3
4
5
6
7
8
9
10
11
12 @Service
13 public class UserServiceImpl implements UserService {
14     @Autowired
15     private UserRepository userRepository;
16
17     @Override
18     public User saveUser(User user) {
19         return userRepository.save(user);
20     }
21
22     @Override
23     public void saveuser(User user) {
24         this.userRepository.save(user);
25     }
26
27     @Override
28     public List<User> getAllUsers() {
29         return userRepository.findAll();
30     }
31
32
33     @Override
34     public User getUserById(long id) {
35         return userRepository.findById(id).orElseThrow(() ->
36             new ResourceNotFoundException("User", "Id", id));
37     }
38
39
40     @Override
41     public User updateUser(User user, long id) {
42         User existingUser = userRepository.findById(id).orElseThrow(() ->
43             new ResourceNotFoundException("User", "Id", id));
44
45         existingUser.setFullName(user.getFullName());
46         existingUser.setEmail(user.getEmail());
47         existingUser.setPhone(user.getPhone());
48
49         userRepository.save(existingUser);
50         return existingUser;
51     }
52
53
54     @Override
55     public void deleteUser(long id) {
56         // TODO Auto-generated method stub
57         userRepository.findById(id).orElseThrow(() ->
58             new ResourceNotFoundException("User", "Id", id));
59         userRepository.deleteById(id);
60     }
61 }
```

## PostServiceImpl.java

```
1 package springboot.service.impl;
2
3 import java.util.List;
4
14
15
16 @Service
17 public class PostServiceImpl implements PostService {
18     @Autowired
19     private PostRepository postRepository;
20     @Autowired
21     private UserRepository userRepository;
22
23     @Override
24     public Post savePost(Post post, long userId) {
25         User user = userRepository.findById(userId).orElseThrow(() ->
26             new ResourceNotFoundException("User", "Id", userId));
27         post.setTitle(post.getTitle());
28         post.setContent(post.getContent());
29         post.setUser(user);
30         postRepository.save(post);
31         return post;
32     }
33     @Override
34     public List<Post> getAllPost() {
35         return postRepository.findAll();
36     }
37     @Override
38     public List<Post> getAllPostsByUser(long userId) {
39         User user = userRepository.findById(userId).orElseThrow(() ->
40             new ResourceNotFoundException("User", "Id", userId));
41         return user.getPosts();
42     }
43     @Override
44     public Post getPostById(long postId) {
45         return postRepository.findById(postId).orElseThrow(() ->
46             new ResourceNotFoundException("Post", "Id", postId));
47     }
48     @Override
49     public Post updatePost(Post post, long id) {
50         Post existingPost = postRepository.findById(id).orElseThrow(() ->
51             new ResourceNotFoundException("Post", "Id", id));
52
53         existingPost.setTitle(post.getTitle());
54         existingPost.setContent(post.getContent());
55         postRepository.save(existingPost);
56         return existingPost;
57     }
58     @Override
59     public void deletePost(long id) {
60         postRepository.findById(id).orElseThrow(() ->
61             new ResourceNotFoundException("Post", "Id", id));
62         postRepository.deleteById(id);
63     }
64 }
```

File trong package *springboot.service.impl* sẽ cho chúng ta điều chỉnh chi tiết các câu truy vấn. Những file này implement lại các file trong *springboot.service* và nó *Override* hết các phương thức lớp cha có. Và nó sử dụng các *interface* trong lớp *springboot.repository* để có thể dễ dàng tương tác với csdl hơn với các hàm có sẵn đã nhắc ở phía trên.

## - Resource Not Found Exception

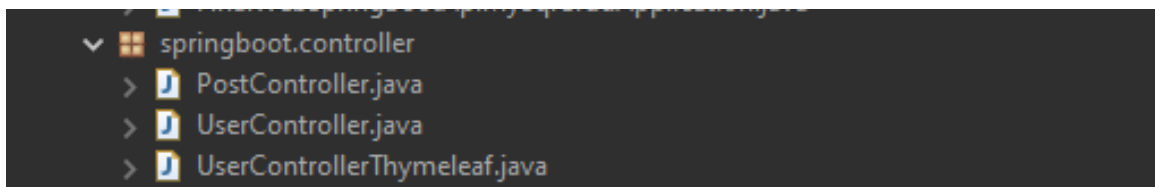


### *ResourceNotFoundException.java*

```
1 package springboot.exception;
2
3
4 import org.springframework.http.HttpStatus;
5
6
7 @ResponseStatus(value = HttpStatus.NOT_FOUND)
8 public class ResourceNotFoundException extends RuntimeException{
9     /**
10      *
11      */
12     private static final long serialVersionUID = 1L;
13     private String resourceName;
14     private String fieldName;
15     private Object fieldValue;
16
17     public ResourceNotFoundException(String resourceName, String fieldName, Object fieldValue) {
18         super(String.format("%s not found with %s : '%s'", resourceName, fieldName, fieldValue));
19         this.resourceName = resourceName;
20         this.fieldName = fieldName;
21         this.fieldValue = fieldValue;
22     }
23
24     public String getResourceName() {
25         return resourceName;
26     }
27
28     public String getFieldName() {
29         return fieldName;
30     }
31
32     public Object getFieldValue() {
33         return fieldValue;
34     }
35 }
36 }
```

Ta thấy đây là file nếu chúng ta không thể tìm ra được dữ liệu hoặc err nào đó.

## - Controller



## UserController.java

```
UserController.java X
1 package springboot.controller;
2
3 import java.util.List;
19
20 @RestController
21 @RequestMapping("/api/users")
22 public class UserController {
23
24     @Autowired
25     private UserService userService;
26
27     //create user
28     @PostMapping
29     public ResponseEntity<User> saveUser(@RequestBody User user) {
30         return new ResponseEntity<User>(userService.saveUser(user),HttpStatus.CREATED);
31     }
32
33     // get all users
34     @GetMapping
35     public List<User> getAllUsers() {
36         return userService.getAllUsers();
37     }
38
39     // select users by id
40     @GetMapping("/{id}")
41     public ResponseEntity<User> getUserById(@PathVariable("id") long userId) {
42         return new ResponseEntity<User>(userService.getUserById(userId), HttpStatus.OK);
43     }
44
45     //update user by id
46     @PutMapping("/{id}")
47     public ResponseEntity<User> updateUser(@PathVariable("id") long id,
48         @RequestBody User user) {
49         return new ResponseEntity<User>(userService.updateUser(user, id), HttpStatus.OK);
50     }
51
52     //delete user by id
53     @DeleteMapping("/{id}")
54     public ResponseEntity<String> deleteUser(@PathVariable("id") long id) {
55         userService.deleteUser(id);
56         return new ResponseEntity<String>("User delete successfully",HttpStatus.OK);
57     }
58 }
59
```

## PostController.java

```
PostController.java X
1 package springboot.controller;
2
3 import java.util.List;
19
20 @RestController
21 @RequestMapping("/api/post")
22 public class PostController {
23
24     @Autowired
25     private PostService postService;
26
27     // create new post
28     @PostMapping("/{userId}")
29     public ResponseEntity<Post> savePost(@RequestBody Post post
30         ,@PathVariable long userId) {
31         return new ResponseEntity<Post>(postService.savePost(post, userId),HttpStatus.OK);
32     }
33
34     @GetMapping
35     public List<Post> getAllPosts() {
36         return postService.getAllPost();
37     }
38
39     @GetMapping("/user/{userId}")
40     public List<Post> get(@PathVariable long userId) {
41         return postService.getAllPostsByUser(userId);
42     }
43
44     //get post by postId
45     @GetMapping("/{postId}")
46     public ResponseEntity<Post> getPostById(@PathVariable long postId) {
47         return new ResponseEntity<Post>(postService.getPostById(postId),HttpStatus.OK);
48     }
49
50     // update post by postId
51     @PutMapping("/{postId}")
52     public ResponseEntity<Post> updatePost(@RequestBody Post post
53         ,@PathVariable long postId) {
54         return new ResponseEntity<Post>(postService.updatePost(post, postId), HttpStatus.OK);
55     }
56
57     // delete post by id
58     @DeleteMapping("/{postId}")
59     public ResponseEntity<String> deletePost(@PathVariable long postId) {
60         postService.deletePost(postId);
61         return new ResponseEntity<String>("Post deleted successfully",HttpStatus.OK);
62     }
63 }
64
```

Đây là các *controller* có tác dụng gọi những câu truy vấn đã được cấu hình ở những lớp dưới (controller -> service -> repository -> database), để có thể trả ra API trên web theo các đường dẫn đã được set chi tiết trong các file. Ở đây ta chỉ trả được file *.json*, ta có thể kiểm tra và kiểm thử với *Postman*,... Nhưng vẫn chưa thể trả ra view, vì thế ta cần có một *controller* khác để có thể cho ra 1 view giúp ta tương tác dễ dàng hơn.

*UserControllerThymeleaf.java*

*Controller với User*

```
UserControllerThymeleaf.java x
22 public class UserControllerThymeleaf {
23     @Autowired
24     private UserService userService;
25
26     @Autowired
27     private PostService postService;
28
29     // list of users
30     @GetMapping("/")
31     public String viewHomePage(Model model) {
32         model.addAttribute("listUsers", userService.getAllUsers());
33         return "index";
34     }
35
36     @GetMapping("/showNewUserForm")
37     public String showNewUserForm(Model model) {
38         User user = new User();
39         model.addAttribute("user", user);
40         return "new_user";
41     }
42
43     @PostMapping("/saveUser")
44     public String saveUser(@ModelAttribute("user") User user) {
45         userService.saveUser(user);
46         return "redirect:/";
47     }
48
49     @PostMapping("/updateUser/{id}")
50     public String updateUser(@PathVariable(value = "id") long id,
51                             @ModelAttribute("user") User user) {
52         userService.updateUser(user, id);
53         return "redirect:/";
54     }
55
56
57     @GetMapping("/showUpdateForm/{id}")
58     public String showUpdateUserForm(@PathVariable(value = "id") long id, Model model) {
59         User user = userService.getUserById(id);
60         model.addAttribute("user", user);
61         return "update_user";
62     }
63
64     @GetMapping("/deleteUser/{id}")
65     public String deleteUser(@PathVariable(value = "id") long id) {
66         this.userService.deleteUser(id);
67         return "redirect:/";
68     }
69
70     @GetMapping("/selectUserById")
71     public String selectUserById(@RequestParam(value = "id") long id, Model model) {
72         User user = userService.getUserById(id);
73         model.addAttribute("listUsers", user);
74         return "index";
75     }
76 }
```

## Controller với Post

```
76 //////////////////////////////////////////////////
77
78 @GetMapping("/posts")
79 public String viewPost(Model model) {
80     model.addAttribute("listPosts", postService.getAllPost());
81     return "post";
82 }
83
84 @GetMapping("/selectPostById")
85 public String selectPostById(@RequestParam(value = "id") long id, Model model) {
86     Post post = postService.getPostById(id);
87     model.addAttribute("listPosts", post);
88     return "post";
89 }
90
91 @GetMapping("/showNewPostForm/{userId}")
92 public String showNewPostForm(Model model, @PathVariable(value = "userId") long userId) {
93     User user = userService.getUserById(userId);
94     Post post = new Post();
95     model.addAttribute("user", user);
96     model.addAttribute("post", post);
97     return "new_post";
98 }
99
100 @PostMapping("/savePost/{userId}")
101 public String savePost(@ModelAttribute("post") Post post, @PathVariable(value = "userId") long userId) {
102     postService.savePost(post, userId);
103     return "redirect:/";
104 }
105
106 @GetMapping("/deletePost/{id}")
107 public String deletePost(@PathVariable(value = "id") long id) {
108     this.postService.deletePost(id);
109     return "redirect:/";
110 }
111
112 @GetMapping("/showPostById/{id}")
113 public String showPostById(@PathVariable(value = "id") long id, Model model) {
114     List<Post> post = postService.getAllPostsByUser(id);
115     model.addAttribute("listPosts", post);
116     return "post";
117 }
118
119 @GetMapping("/showUpdatePostForm/post/{postId}")
120 public String showUpdatePostForm(@PathVariable(value = "postId") long postId, Model model) {
121     Post post = postService.getPostById(postId);
122     model.addAttribute("post", post);
123     return "update_post";
124 }
125
126 @PostMapping("/updatePost/{postId}")
127 public String updatePost(@PathVariable(value = "postId") long postId, @ModelAttribute("post") Post post) {
128     postService.updatePost(post, postId);
129     return "redirect:/";
130 }
131 }
```

Đây là *controller* giúp ta có thể cho ra view thông qua *Thymeleaf* với các đường dẫn được được set chi tiết trong file. Với *controller* này sẽ giúp chúng ta có 1 view bằng cách ở cuối mỗi hàm sẽ trả về 1 file *.html*, vì thế ta có thể xem trên localhost và thực hiện các phương thức *@Get*, *@Post*, *@Put*, *@Delete* dễ dàng hơn.

## - Templates

*Index.html*

```
index.html X
12 <header>
13   <nav class="navbar navbar-expand-md navbar-dark bg-dark">
14     <ul class="ab navbar-nav">
15       <li><a href = ""
16         class="navbar-brand">User List</a></li>
17       <li><a th:href = "@{/showNewUserForm}"
18         class="navbar-brand">Add New User</a></li>
19       <li><a th:href = "@{/posts}"
20         class="navbar-brand">List Posts</a></li>
21     </ul>
22   </nav>
23 </header>
24 <div class="container my-2">
25   <br>
26   <h1 style="text-align:center">User List</h1>
27   <div class="mb-3">
28     <form action="#" th:action="@{/selectUserById}" th:id="id" method="get">
29       <fieldset class="form-group">
30         <input name = "id" type="search"
31           class="form-control" required="required" placeholder="User's ID">
32       </fieldset>
33       <button type="submit" class="btn btn-success">Find</button>
34     </form>
35   </div>
36   <table border="1" class="table table-striped table-responsive-md">
37     <thead>
38       <tr>
39         <th>UserID</th>
40         <th>Full Name</th>
41         <th>Email</th>
42         <th>Phone Number</th>
43         <th>Created At</th>
44         <th>Updated At</th>
45         <th>Actions</th>
46         <th>Posts</th>
47       </tr>
48     </thead>
49     <tbody>
50       <tr th:each="user: ${ListUsers}">
51         <td th:text="${user.id}"></td>
52         <td th:text="${user.fullName}"></td>
53         <td th:text="${user.email}"></td>
54         <td th:text="${user.phone}"></td>
55         <td th:text="${user.createdAt}"></td>
56         <td th:text="${user.updatedAt}"></td>
57         <td class="aa">
58           <a th:href = "@{/showUpdateForm/{id}(id=${user.id})}" class="btn btn-primary">Update</a>
59           <a th:href = "@{/deleteUser/{id}(id=${user.id})}" class="btn btn-danger">Delete</a>
60         </td>
61       </tr>
62     </tbody>
63   </table>
64   <a th:href = "@{/showNewPostForm/{id}(id=${user.id})}" class="btn btn-primary">New</a>
65   <a th:href = "@{/showPostById/{id}(id=${user.id})}" class="btn btn-danger">Update</a>
66 </div>
```

Đây là trang chính của chúng ta với *header*, và *body* với *Thymeleaf*, *Bootstrap*. Phần *body* sẽ tạo ra một bảng, chúng ta sẽ lấy từng *user* trong *listUser* thông qua *th:each* của *Thymeleaf*, và lấy từng thông tin của *user* đó thông qua *th:text*, ở cuối mỗi dòng sẽ có các *Actions* là các nút: *Update* (dùng để cập nhật *User*), *Delete* (dùng để xóa *User*), các tương tác với *Posts*: *New* (dùng để tạo ra một *Post* mới thuộc sở hữu của *User* thuộc dòng này), *Update* (dùng để cập nhật các bài *Post* mà *User* này đang sở hữu). Và ta cũng sẽ có khung tìm kiếm ở dưới đầu danh sách, chỉ cần nhập vào *Id* thì ta sẽ được trả dòng thông tin của *User* có *Id* được nhập vào. Và có các nút: *UserList*, *Add New User*, *List Posts* ở phần *Header*.



## User List

| User's ID |                |               |              |                      |                      |  |   |
|-----------|----------------|---------------|--------------|----------------------|----------------------|--|---|
| Find      |                |               |              |                      |                      |  |   |
| UserID    | Full Name      | Email         | Phone Number | Created At           | Updated At           | Actions  | Posts   |
| 1         | Ho Xuan Hieu 2 | xh2@gmail.com | 147852369    | 2022-12-18T08:40:20Z | 2022-12-18T08:40:20Z | <a href="#">Update</a><br><a href="#">Delete</a> | <a href="#">New</a><br><a href="#">Update</a> |

## new\_user.html

```

1 <!DOCTYPE html>
2 <html lang="en" xmlns:th="http://www.thymeleaf.org">
3 <head>
4 <meta charset="ISO-8859-1">
5 <title>User Management System</title>
6 <link rel="stylesheet"
7 href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"
8 integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaAoApmYm81iuxoPkF0JwJ8ERdknLPwQ"
9 crossorigin="anonymous">
10 </head>
11 <header>
12 <nav class="navbar navbar-expand-md navbar-dark bg-dark">
13 <ul class="navbar-nav">
14 <li><a th:href = "@{/}"
15 class="navbar-brand">User List</a></li>
16 <li><a th:href = "@{/showNewUserForm}"
17 class="navbar-brand">Add New User</a></li>
18 <li><a th:href = "@{/posts}"
19 class="navbar-brand">List Posts</a></li>
20 </ul>
21 </nav>
22 </header>
23 <body>
24 <div class="container my-2">
25 <h1 style="text-align:center">New User Form</h1>
26 <form action = "#" th:action="@{/saveUser}" th:object = "${user}" method="POST" >
27 <input type = "text" th:field = "{fullName}"
28 placeholder="User's Full Name" class="form-control mb-4 col-4">
29
30 <input type = "text" th:field = "{email}"
31 placeholder="User's Email" class="form-control mb-4 col-4">
32
33 <input type = "text" th:field = "{phone}"
34 placeholder="User's Phone Number" class="form-control mb-4 col-4">
35
36 <button type="submit" class="btn btn-info col-2">Save User</button>
37 </form>
38
39 </div>
40 </body>
41 </html>

```

Khi bấm vào nút *Add New User* ở trang chính ta sẽ được chuyển đến trang này, có *form* để nhập vào thông tin của *User* mới cần được tạo, khi ta nhấn *Save User* sẽ thực hiện hành động *th:action="@{/saveUser}"*. Và như thế ta sẽ Lưu thành công một *User* mới.

## New User Form

User's Full Name

User's Email

User's Phone Number

Save User

## update\_user.html

```
1 <!DOCTYPE html>
2 <html lang="en" xmlns:th="http://www.thymeleaf.org">
3 <head>
4 <meta charset="ISO-8859-1">
5 <title>User Management System</title>
6 <link rel="stylesheet"
7 href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"
8 integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFrFOJwJ8ERdknLPMO"
9 crossorigin="anonymous">
10 </head>
11 <header>
12 <nav class="navbar navbar-expand-md navbar-dark bg-dark">
13 <ul class="navbar-nav">
14 <li><a th:href = "@{/}"
15 class="navbar-brand">User List</a></li>
16 <li><a th:href = "@{/showNewUserForm}"
17 class="navbar-brand">Add New User</a></li>
18 <li><a th:href = "@{/posts}"
19 class="navbar-brand">List Posts</a></li>
20 </ul>
21 </nav>
22 </header>
23 <body>
24 <div class="container my-2">
25 <h1 style="text-align:center">Update User Form</h1>
26 <form th:action="@{/updateUser/{id}(id = ${user.getId()})}" th:object = "${user}" method="POST">
27
28 <input type="hidden" th:field="*{id}" />
29
30 <label class="form-label"><h4>User's Full Name</h4></label>
31 <input type = "text" th:field = "*{fullName}" class="form-control mb-4 col-4">
32
33 <label class="form-label"><h4>User's Email</h4></label>
34 <input type = "text" th:field = "*{email}" class="form-control mb-4 col-4">
35
36 <label class="form-label"><h4>User's Phone</h4></label>
37 <input type = "text" th:field = "*{phone}" class="form-control mb-4 col-4">
38
39 <button type="submit" class="btn btn-info col-2">Update User</button>
40 </form>
41 <hr>
42 </div>
43 </body>
44 </html>
```

Khi ta nhấn vào nút *Update* ở cuối mỗi dòng thông tin ở trang chủ, ta sẽ được chuyển đến trang này với id tương ứng của *User* đó. Một form để có thể cập nhật lại thông tin. Sau khi nhấn nút *Update User*, sẽ thực hiện `th:action="@{/updateUser/{id}(id = ${user.getId()})}"` với đầu vào chính là *Id* mà ta cần *Update*. Và như thế ta đã cập nhật thành công *User*.

User List Add New User List Posts

## Update User Form

User's Full Name

User's Email

User's Phone

*new\_post.html*

```
new_post.html x
1 <!DOCTYPE html>
2 <html lang="en" xmlns:th="http://www.thymeleaf.org">
3 <head>
4 <meta charset="ISO-8859-1">
5 <title>User Management System</title>
6 <link rel="stylesheet"
7 href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css">
8 </head>
9 <header>
10 <nav class="navbar navbar-expand-md navbar-dark bg-dark">
11 <ul class="navbar-nav">
12 <li><a th:href = "@{/}"
13 class="navbar-brand">User List</a></li>
14 <li><a th:href = "@{/showNewUserForm}"
15 class="navbar-brand">Add New User</a></li>
16 <li><a th:href = "@{/posts}"
17 class="navbar-brand">List Posts</a></li>
18 </ul>
19 </nav>
20 </header>
21 <body>
22 <div class="container my-2">
23 <h1 style="text-align:center">New Post Form</h1>
24 <form action="#" th:action = "@{/savePost/{userId}}(userId = ${user.getId()})"
25 th:object = "${post}" method = "POST" >
26
27 <input type = "text" th:field = "${title}"
28 placeholder="Title of the post" class="form-control mb-4 col-4">
29
30 <input type = "text" th:field = "${content}"
31 placeholder="Content of the post" class="form-control mb-4 col-4">
32
33 <button type="submit" class="btn btn-info col-2" >Save Post</button>
34
35 </form>
36 </div>
37 </body>
38 </html>
```

Khi ta nhấn nút New ở cột Posts của một User ở trang chủ, ta sẽ được chuyển đến một form để có thể thêm 1 bài post mới với chủ sở hữu chính là User đó. Sau khi nhấn nút Save Post sẽ thực hiện hành động `th:action="@{/savePost/{userId}}(userId = ${user.getId()})"` với đầu vào chính là `userId` của chủ sở hữu. Và như thế ta được tạo mới thành công.

User List Add New User List Posts

## New Post Form

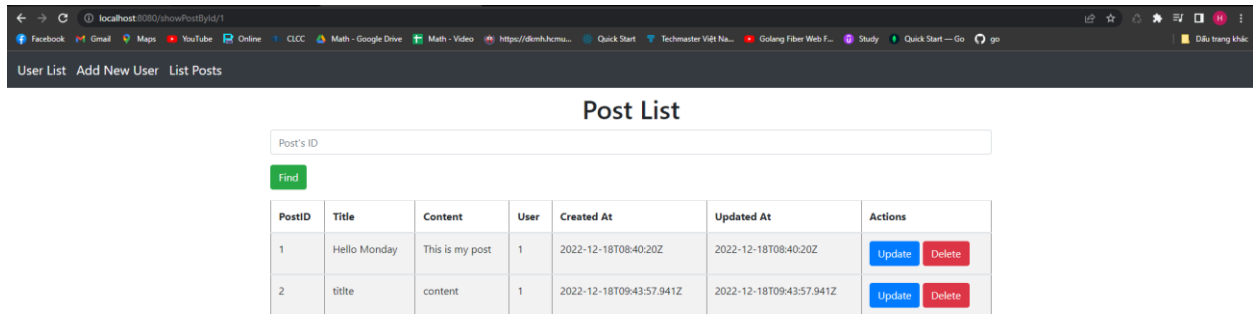
Save Post

## post.html

```
12 <header>
13   <nav class="navbar navbar-expand-md navbar-dark bg-dark">
14     <ul class="navbar-nav">
15       <li><a th:href = "@{/}"
16         class="navbar-brand">User List</a></li>
17       <li><a th:href = "@{/showNewUserForm}"
18         class="navbar-brand">Add New User</a></li>
19       <li><a th:href = "@{/posts}"
20         class="navbar-brand">List Posts</a></li>
21     </ul>
22   </nav>
23 </header>
24 <div class="container my-2">
25   <h1 style="text-align:center">Post List</h1>
26   <div class="mb-3">
27     <form action="#" th:action="@{/selectPostById}" th:id="id" method="get">
28       <fieldset class="form-group">
29         <input name = "id" type="search"
30           class="form-control" required="required" placeholder="Post's ID">
31       </fieldset>
32       <button type="submit" class="btn btn-success">Find</button>
33     </form>
34   </div>
35   <table border="1" class="table table-striped table-responsive-md">
36     <thead>
37       <tr>
38         <th>PostID</th>
39         <th>Title</th>
40         <th>Content</th>
41         <th>User</th>
42         <th>Created At</th>
43         <th>Updated At</th>
44         <th>Actions</th>
45       </tr>
46     </thead>
47     <tbody th:each = "post: ${listPosts}">
48       <tr>
49         <td th:text = "${post.id}"></td>
50         <td th:text = "${post.title}"></td>
51         <td th:text = "${post.content}"></td>
52         <td th:text = "${post.user.getId()}"></td>
53         <td th:text = "${post.createdAt}"></td>
54         <td th:text = "${post.updatedAt}"></td>
55
56         <td>
57           <a th:href = "@{/showUpdatePostForm/post/{postId}(postId=${post.id})}" class="btn btn-primary">Update</a>
58           <a th:href = "@{/deletePost/{id}(id=${post.id})}" class="btn btn-danger">Delete</a>
59         </td>
60       </tr>
61     </tbody>
62   </table>
63 </div>
```

Khi bấm vào nút ListPosts trên thanh ở trang chủ ta sẽ được thấy danh sách tất cả các bài Post hiện có trong csdl. Ở trang này sẽ có thanh tìm kiếm, ta sẽ tìm kiếm bằng cách nhập vào Id bài viết và nhấn nút Find. Cuối mỗi dòng dữ liệu có Actions: Update (dùng để Update bài viết) và Delete (dùng để xóa bài viết).

Ở trang chủ, khi nhấn vào nút *Update* ở cột *Posts*, bạn sẽ được trả ra một trang có danh sách các bài *Posts* thuộc sở hữu của *User* đó, tại đây bạn có thể tùy ý tương tác với bài *Post* như *Delete*, *Update*.



| PostID | Title        | Content         | User | Created At               | Updated At               | Actions   |
|--------|--------------|-----------------|------|--------------------------|--------------------------|---|
| 1      | Hello Monday | This is my post | 1    | 2022-12-18T08:40:20Z     | 2022-12-18T08:40:20Z     | <button>Update</button> <button>Delete</button> |
| 2      | titlle       | content         | 1    | 2022-12-18T09:43:57.941Z | 2022-12-18T09:43:57.941Z | <button>Update</button> <button>Delete</button> |

Sau khi nhấn vào nút *Update* ở cột *Posts*, ta thấy được rằng đối với *User* có *Id* = 1 thì sẽ có 2 bài viết, và ở đây ta có thể tùy ý tương tác với bài *Posts*.

*update\_post.html*

```
update_post.html
1 <!DOCTYPE html>
2 <html lang="en" xmlns:th="http://www.thymeleaf.org">
3 <head>
4 <meta charset="ISO-8859-1">
5 <title>User Management System</title>
6 <link rel="stylesheet"
7 href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"
8 integrity="sha384-MCw98/SFNgE8fJ3TGXwE0ngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdnLP"
9 crossorigin="anonymous">
10 </head>
11 <body>
12 <header>
13 <nav class="navbar navbar-expand-md navbar-dark bg-dark">
14 <ul class="navbar-nav">
15 <li><a th:href = "@{/}"
16 class="navbar-brand">User List</a></li>
17 <li><a th:href = "@{/showNewUserForm}"
18 class="navbar-brand">Add New User</a></li>
19 <li><a th:href = "@{/posts}"
20 class="navbar-brand">List Posts</a></li>
21 </ul>
22 </nav>
23 </header>
24 <div class="container my-2">
25 <h1 style="text-align:center">Update Post Form</h1>
26 <form th:action="@{/updatePost/{postId}(postId = ${post.getId()})}"
27 th:object="${post}" method="POST">
28
29 <input type="hidden" th:field="*{id}" />
30 <label class="form-label"><h4>Title</h4></label>
31 <input type="text" th:field="*{title}" class="form-control mb-4 col-4">
32 <label class="form-label"><h4>Content</h4></label>
33 <input type="text" th:field="*{content}" class="form-control mb-4 col-4">
34
35
36 <button type="submit" class="btn btn-info col-2">Update Post</button>
37 </form>
38 <hr>
39 </div>
40 </body>
41 </html>
```

Sau khi nhấn vào nút *Update* ở cột *Actions* trong *Post List*, ta sẽ được trả về 1 form tại đây sẽ nhận vào *postId*, sau khi điền các thông tin và nhấn nút *Update Post*, sẽ thực hiện *th:action="@{/updatePost/{postId}(postId = \${post.getId()})}"*, nhận vào *postId* để tìm ra bài post có *Id* giống như vậy và bắt đầu *Update*.

## Update Post Form

### Title

### Content