[Thực hành] Sử dụng thymeleaf cho ứng dụng Quản lý Dữ liệu Khách hàng

**Mục tiêu**

Luyện tập sử dụng Thymeleaf

Luyện tập config dự án Spring MVC bằng Class JAVA.

**Điều kiện**

Có kiến thức căn bản về Thymeleaf

**Mô tả**

Trong phần này, chúng ta sẽ sử dụng Thymeleaf cho ứng dụng quản lý khách hàng. Đây là một ứng dụng CRUD đơn giản, cho phép người dùng quản lý danh sách khách hàng.

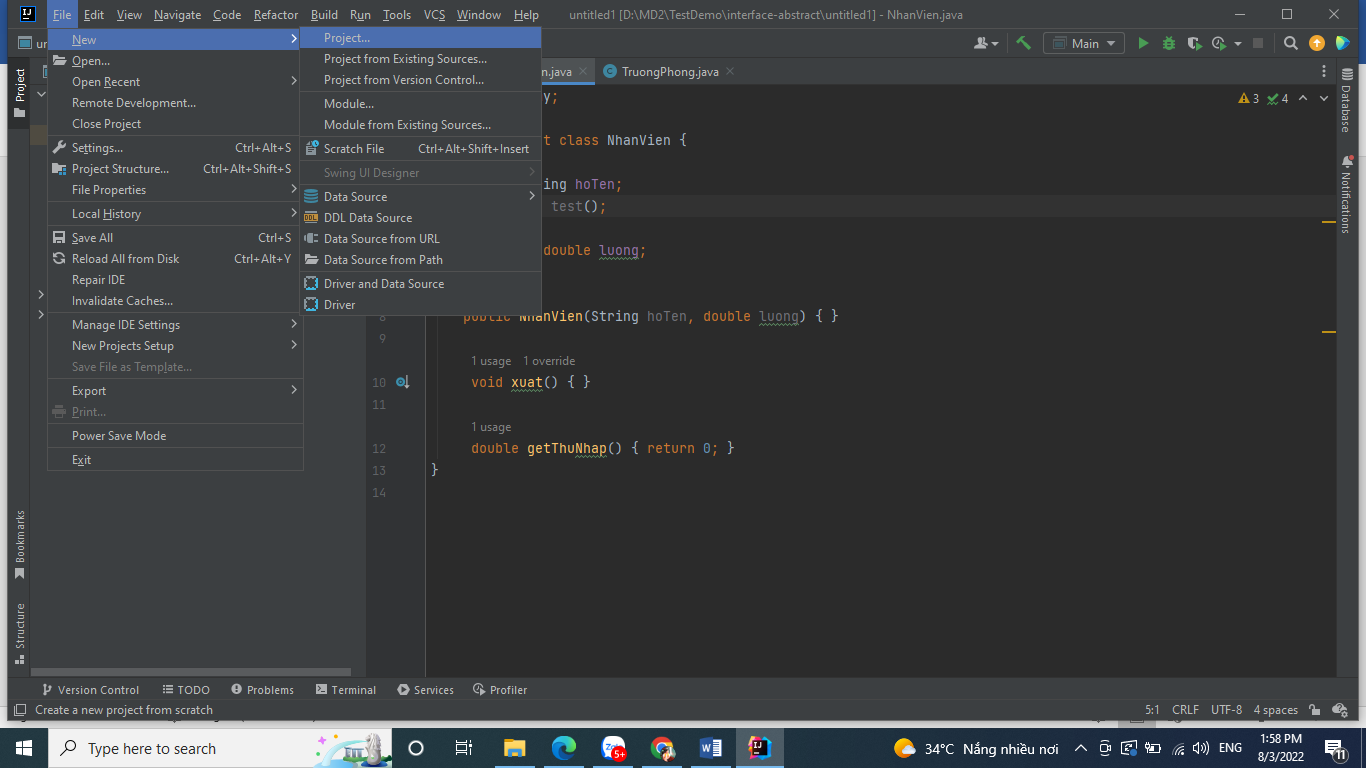
Ứng dụng bao gồm các chức năng:

* Hiển thị danh sách khách hàng
* Thêm mới khách hàng
* Sửa thông tin khách hàng
* Xóa khách hàng khỏi danh sách
* Hiển thị thông tin chi tiết của khách hàng

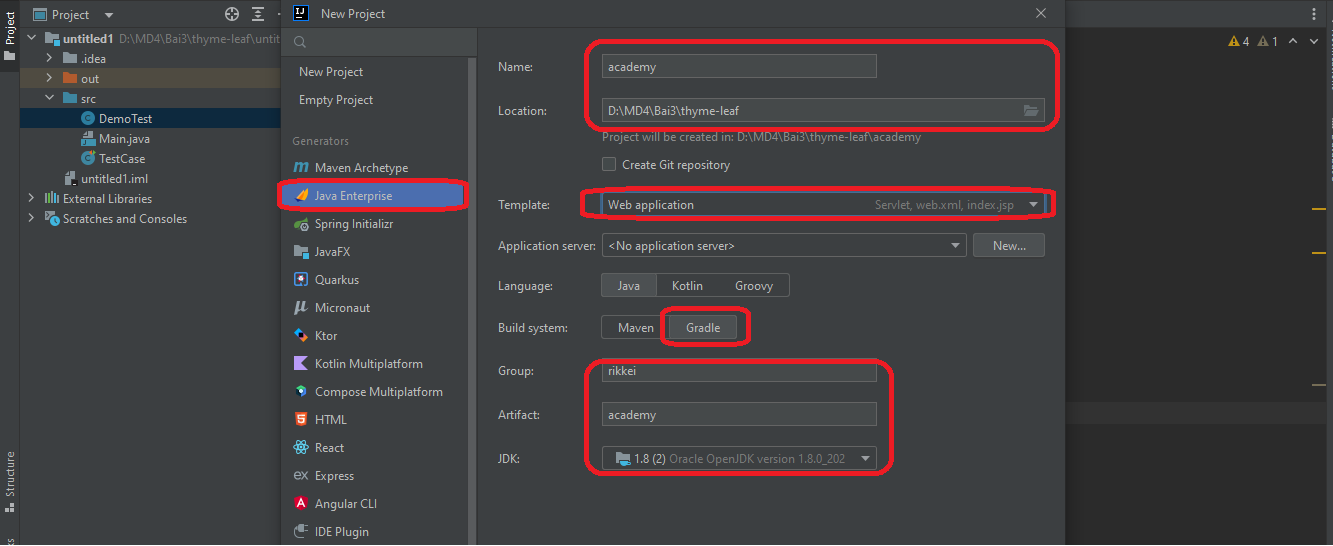
Hướng dẫn

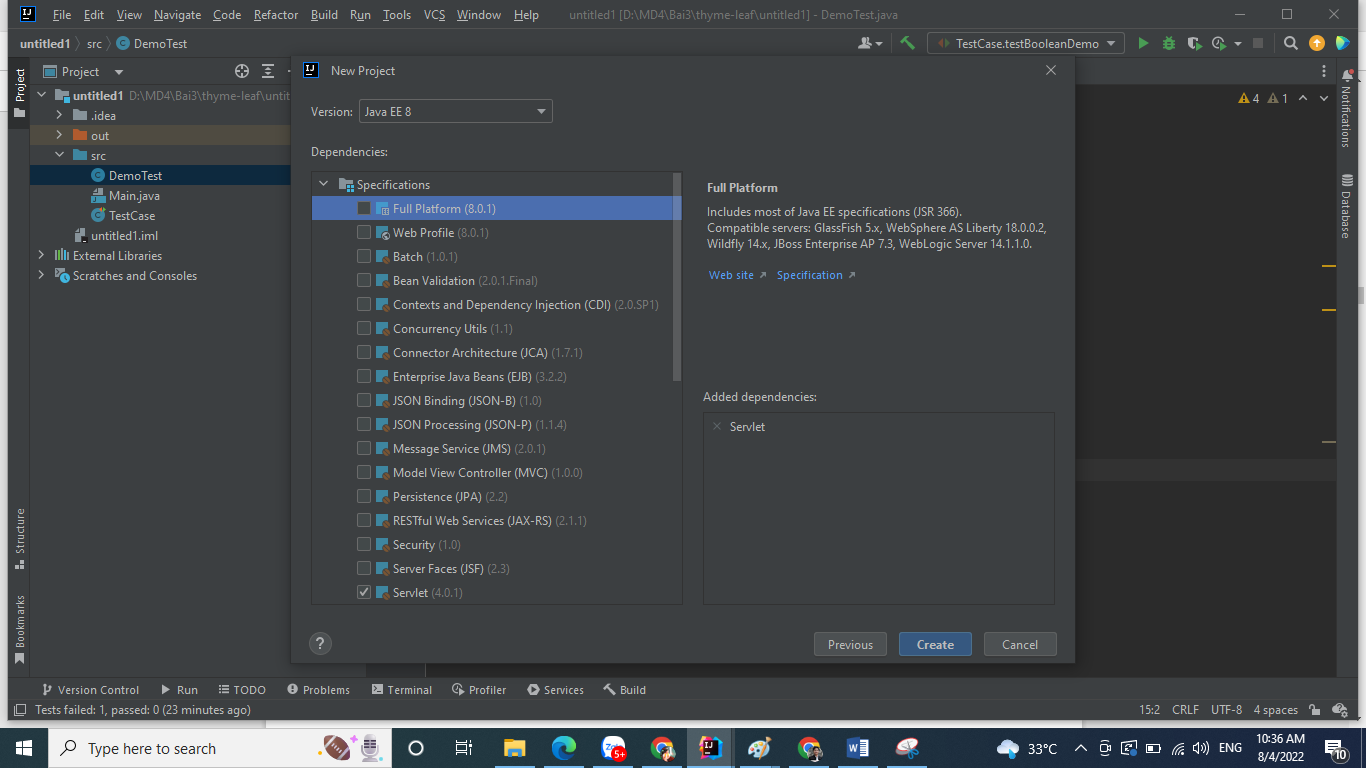
**Bước 1:** Tạo dự án [Spring MVC](https://james.codegym.vn/mod/page/view.php?id=8863) sử dụng Thymeleaf.

Để tạo mới dự án, các bạn chọn File => New => Project.

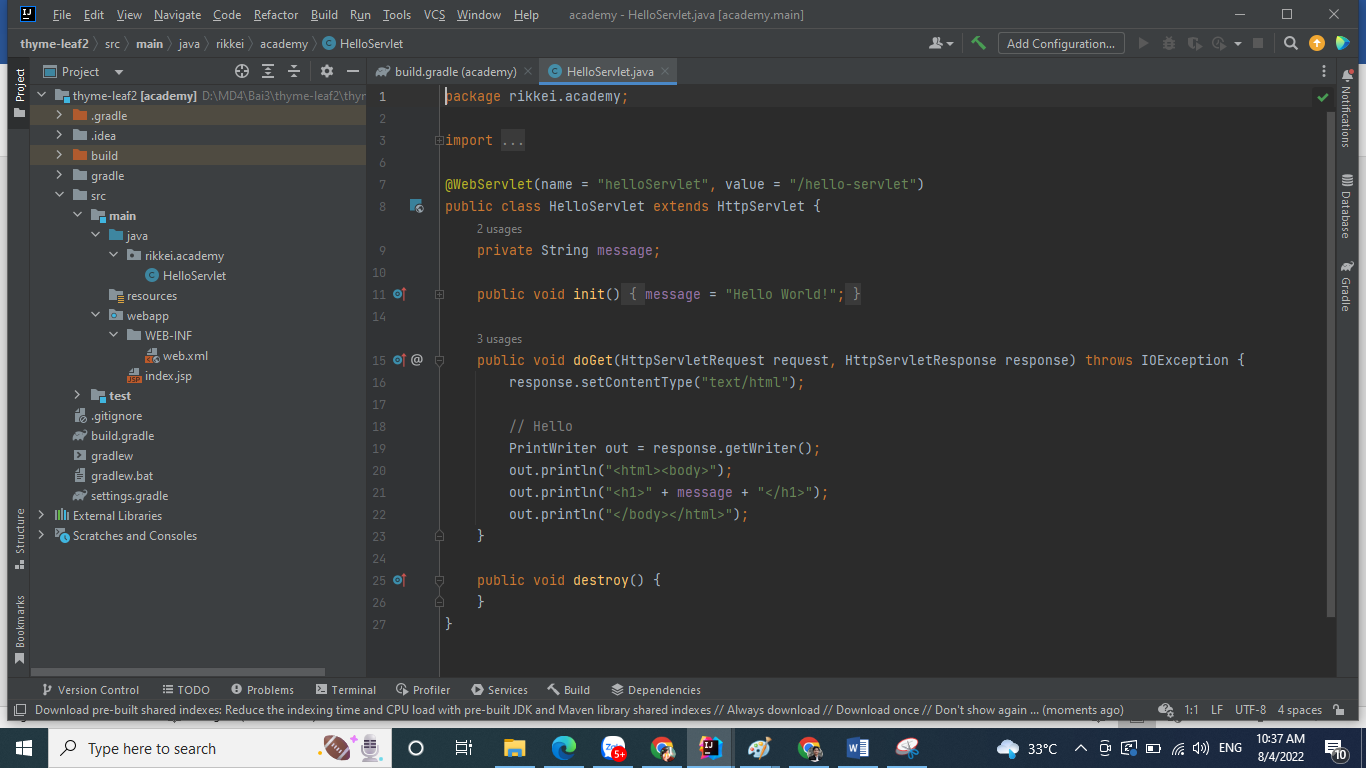


* Tích chọn như hình dưới đây:





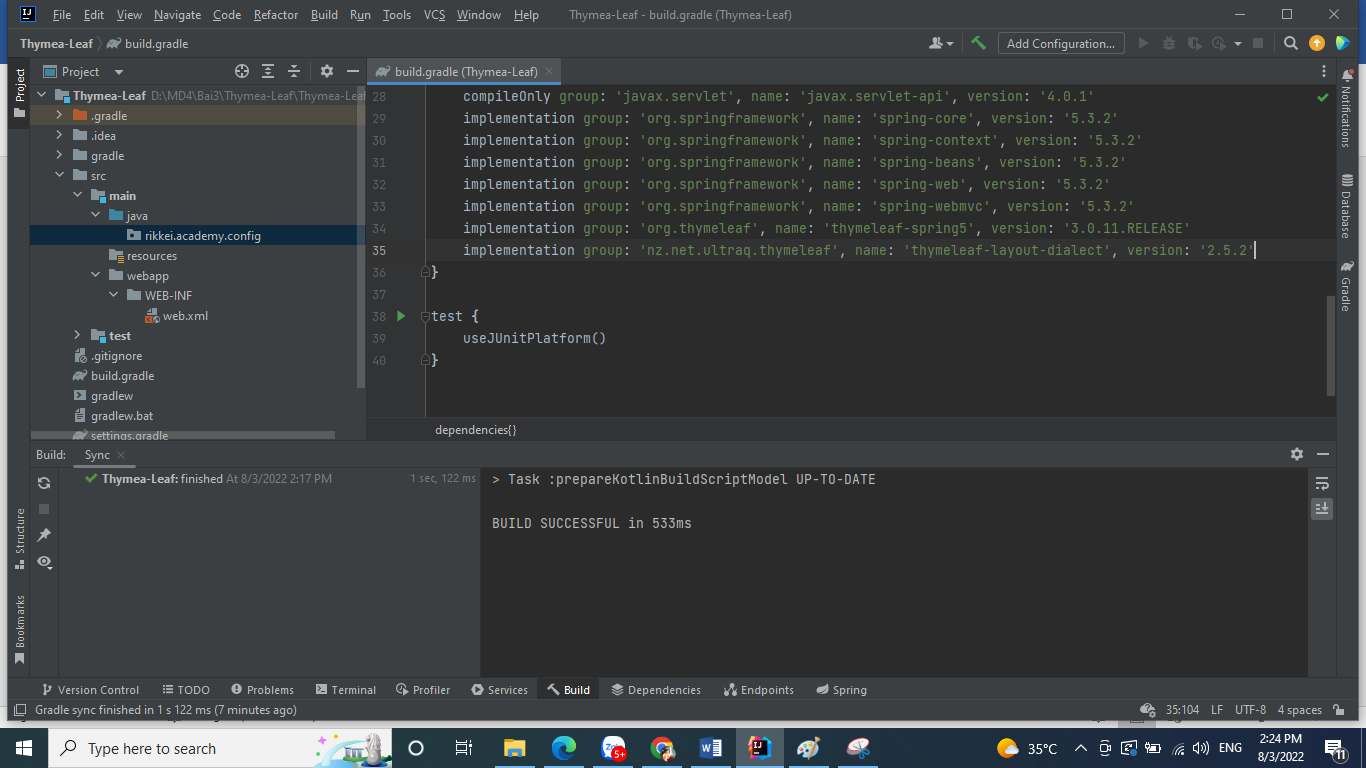
* Cấu trúc dự án sau khi tạo:



* Xóa Class HelloServlet (JAVA Servlet) vs file: index.jsp dưới tầng webapp
* Thêm các thư viện sau vào build.gradle -> dependencies:

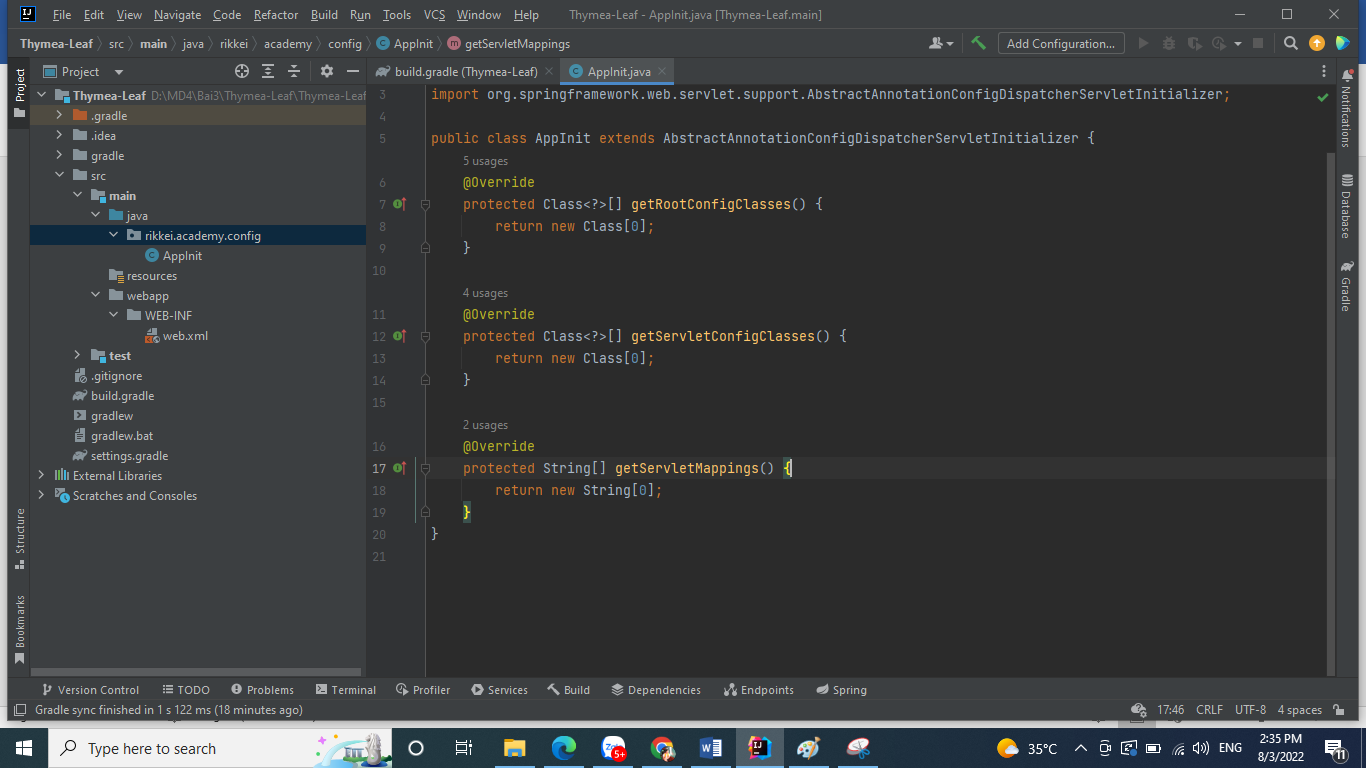
compileOnly group: 'javax.servlet', name: 'javax.servlet-api', version: '4.0.1'  
implementation group: 'org.springframework', name: 'spring-core', version: '5.3.2'  
implementation group: 'org.springframework', name: 'spring-context', version: '5.3.2'  
implementation group: 'org.springframework', name: 'spring-beans', version: '5.3.2'  
implementation group: 'org.springframework', name: 'spring-web', version: '5.3.2'  
implementation group: 'org.springframework', name: 'spring-webmvc', version: '5.3.2'  
implementation group: 'org.thymeleaf', name: 'thymeleaf-spring5', version: '3.0.11.RELEASE'  
implementation group: 'nz.net.ultraq.thymeleaf', name: 'thymeleaf-layout-dialect', version: '2.5.2'

* Tạo package: rikkei.academy.config trong thư mục java như hình:



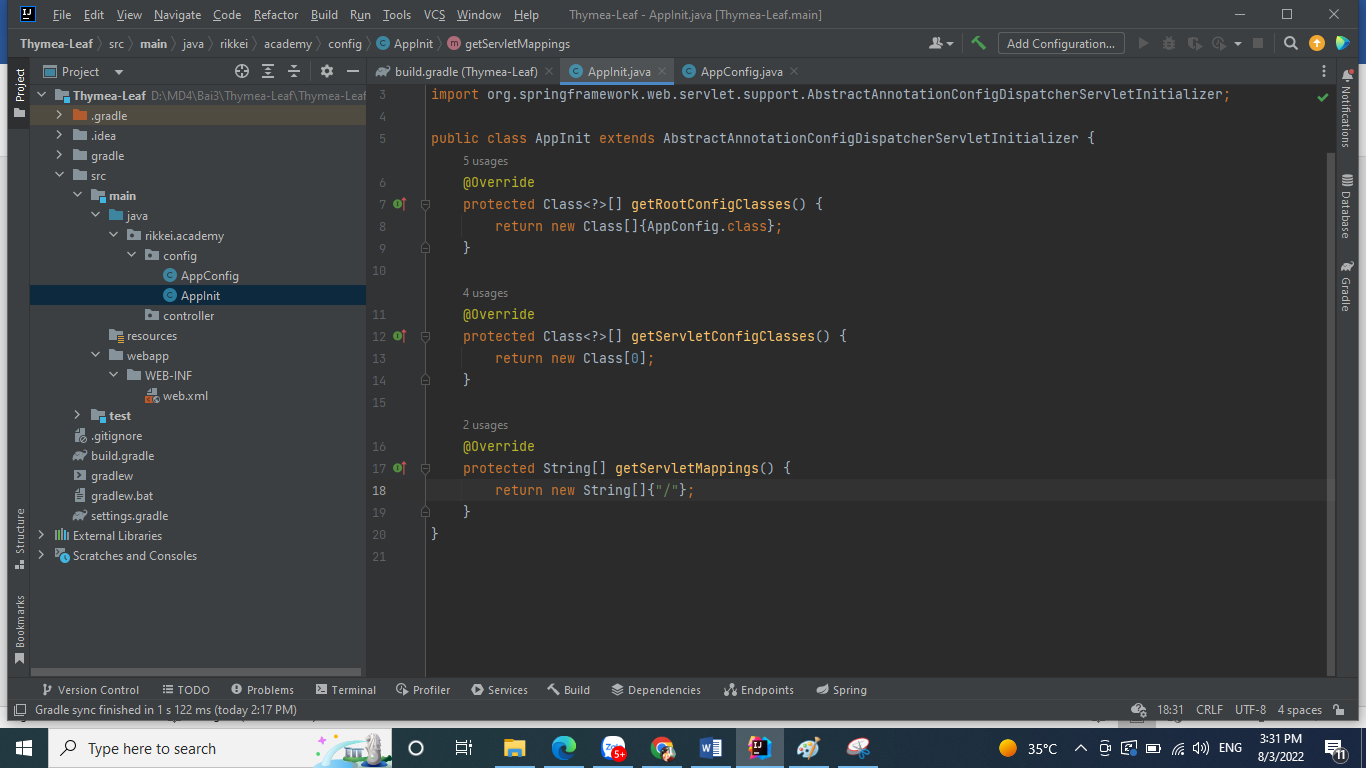
Tạo Class: AppInit trong package: config -> Cho Class này extend AbstractAnnotationConfigDispatcherServletInitializer

* Triển khai các phương thức implimention ta được kết quả như sau:
* Đây chính là suối nguồn của luồng chạy trong dự án Spring MVC nơi tạo ra 3 luồng chạy từ 3 phương thức triển khai.

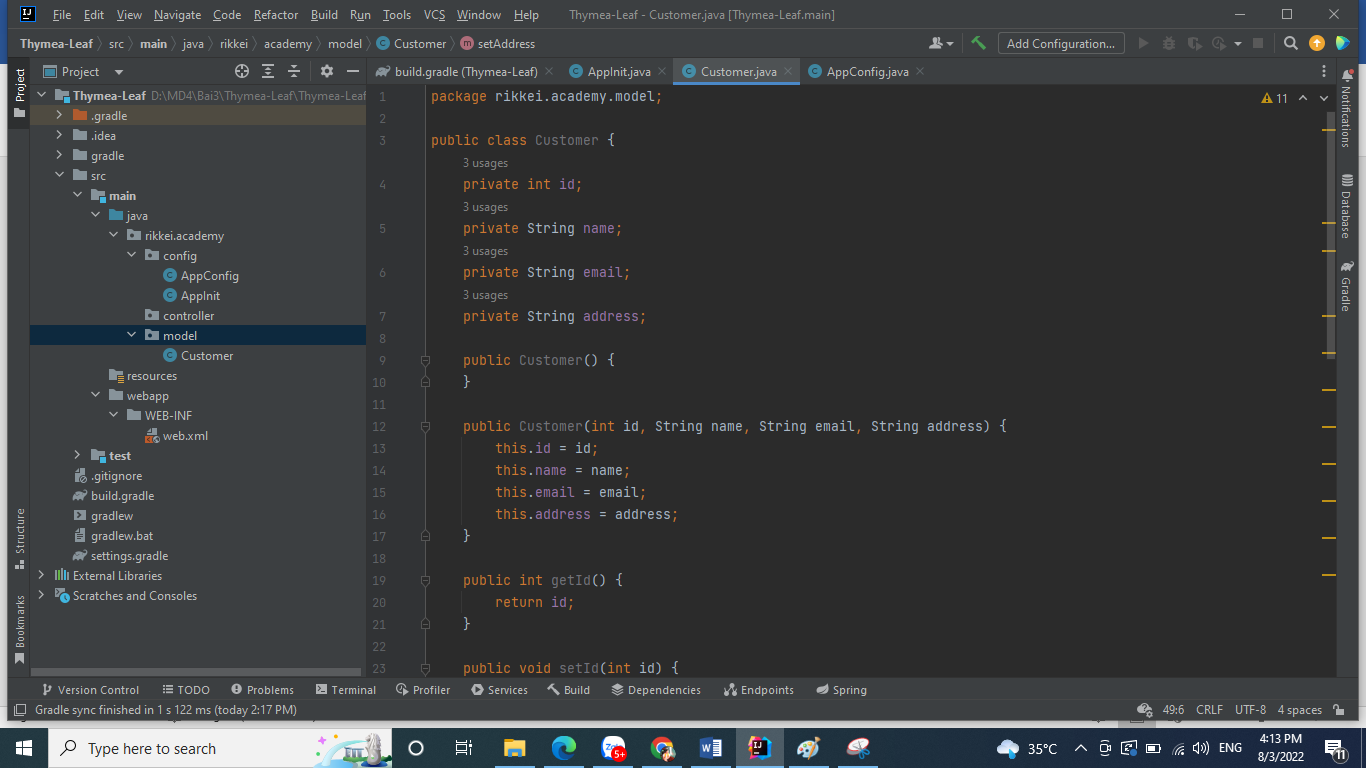


Tiếp tục tạo Class AppConfig vẫn trong package: config cho Class này implements WebMvcConfigurer, ApplicationContextAware

* Thêm code toàn cảnh vào Class này như sau:
* package rikkei.academy.config;  
    
  import org.springframework.beans.BeansException;  
  import org.springframework.context.ApplicationContext;  
  import org.springframework.context.ApplicationContextAware;  
  import org.springframework.context.annotation.Bean;  
  import org.springframework.context.annotation.ComponentScan;  
  import org.springframework.context.annotation.Configuration;  
  import org.springframework.web.servlet.config.annotation.EnableWebMvc;  
  import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;  
  import org.thymeleaf.spring5.SpringTemplateEngine;  
  import org.thymeleaf.spring5.templateresolver.SpringResourceTemplateResolver;  
  import org.thymeleaf.spring5.view.ThymeleafViewResolver;  
  import org.thymeleaf.templatemode.TemplateMode;  
    
  @Configuration  
  @EnableWebMvc  
  @ComponentScan("rikkei.academy.controller")  
  public class AppConfig implements WebMvcConfigurer, ApplicationContextAware {  
    
   private ApplicationContext applicationContext;  
    
   @Override  
   public void setApplicationContext(ApplicationContext applicationContext) throws BeansException {  
   this.applicationContext = applicationContext;  
   }  
    
   //Cấu hình Thymleaf  
   @Bean  
   public SpringResourceTemplateResolver templateResolver() {  
   SpringResourceTemplateResolver templateResolver = new SpringResourceTemplateResolver();  
   templateResolver.setApplicationContext(applicationContext);  
   templateResolver.setPrefix("/WEB-INF/views");  
   templateResolver.setSuffix(".html");  
   templateResolver.setTemplateMode(TemplateMode.*HTML*);  
   templateResolver.setCharacterEncoding("UTF-8");  
   return templateResolver;  
   }  
    
   @Bean  
   public SpringTemplateEngine templateEngine() {  
   SpringTemplateEngine templateEngine = new SpringTemplateEngine();  
   templateEngine.setTemplateResolver(templateResolver());  
   return templateEngine;  
   }  
    
   @Bean  
   public ThymeleafViewResolver viewResolver() {  
   ThymeleafViewResolver viewResolver = new ThymeleafViewResolver();  
   viewResolver.setTemplateEngine(templateEngine());  
   viewResolver.setCharacterEncoding("UTF-8");  
   return viewResolver;  
   }  
    
  }
* Lưu ý: Học viên tìm hiểu về ý nghĩa của từng @Bean , search google tìm hiểu các templage view như: SpringTemplateEgine , ThymeleafViewResolver, … có ý nghĩa gì?
* Tạo kết nối giữa 2 Class: AppInit vs AppConfig để tạo luồng chạy sang Class AppConfig bằng cách sửa code AppInit như sau:



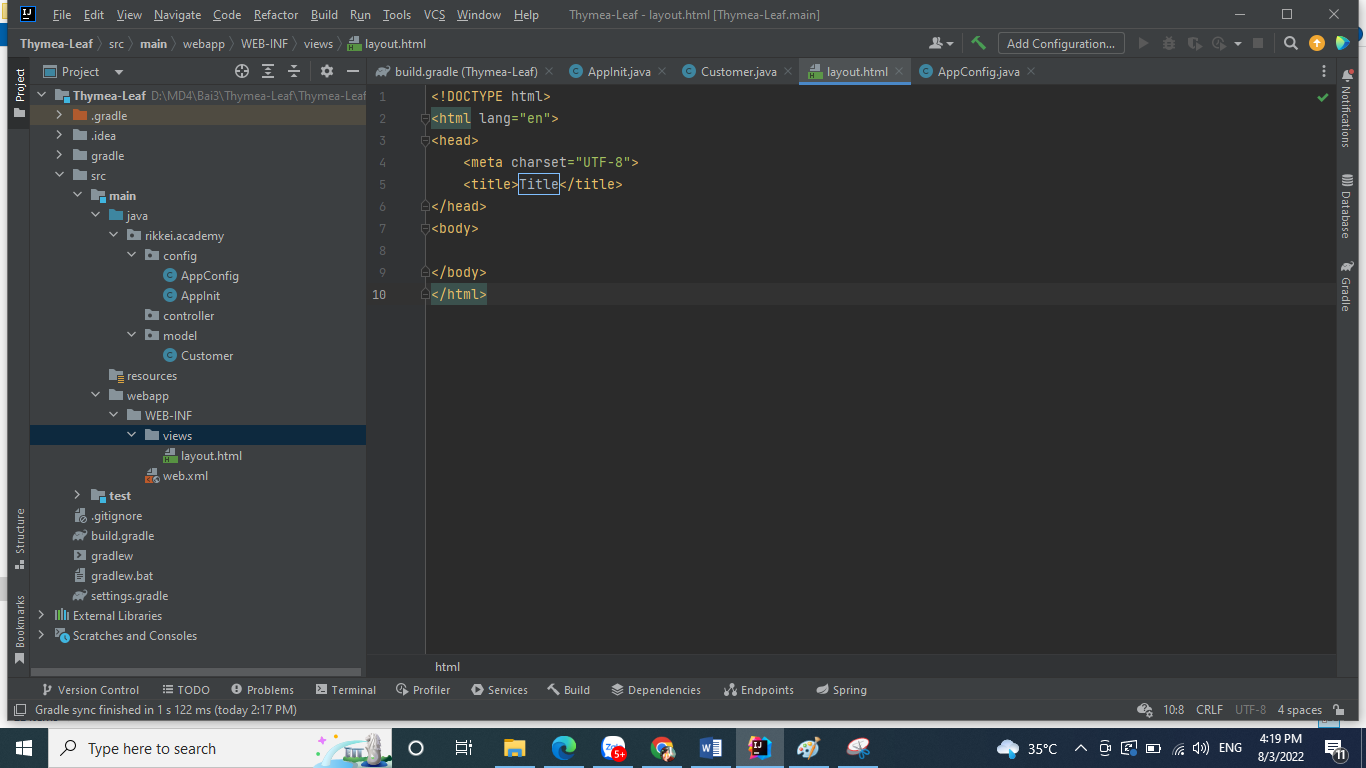
* Tạo package controller từ package: rikkei.academy: (ngang hàng với package config).
* Các Class tạo trong package này được scan luồng chạy thông qua @ComponentScan("rikkei.academy.controller") ở trong Class AppConfig.
* Bước tiếp theo tạo package: model => Tạo Class Customer và thêm code vào như sau:
* package rikkei.academy.model;  
    
  public class Customer {  
   private int id;  
   private String name;  
   private String email;  
   private String address;  
    
   public Customer() {  
   }  
    
   public Customer(int id, String name, String email, String address) {  
   this.id = id;  
   this.name = name;  
   this.email = email;  
   this.address = address;  
   }  
    
   public int getId() {  
   return id;  
   }  
    
   public void setId(int id) {  
   this.id = id;  
   }  
    
   public String getName() {  
   return name;  
   }  
    
   public void setName(String name) {  
   this.name = name;  
   }  
    
   public String getEmail() {  
   return email;  
   }  
    
   public void setEmail(String email) {  
   this.email = email;  
   }  
    
   public String getAddress() {  
   return address;  
   }  
    
   public void setAddress(String address) {  
   this.address = address;  
   }  
    
  }
* Cấu trúc dự án hiện tại như ảnh dưới:



* Tạo package service => Tạo interface: ICustomerService trong package: service => Thêm code vào như sau:
* Tạo interface: IcustomerService thêm code vào như sau:

package rikkei.academy.service;  
  
import rikkei.academy.model.Customer;  
  
import java.util.List;  
  
public interface ICustomerService {  
 List<Customer> findAll();  
  
 void save(Customer customer);  
  
 Customer findById(int id);  
  
 void update(int id, Customer customer);  
  
 void remove(int id);  
  
}

* Tạo Class: CustomerServiceIMPL trong package: service cho impliment IcustomerService và triển khai các hàm từ interface như sau:
* package rikkei.academy.service;  
    
  import rikkei.academy.model.Customer;  
    
  import java.util.ArrayList;  
  import java.util.HashMap;  
  import java.util.List;  
  import java.util.Map;  
    
  public class CustomerServiceIMPL implements ICustomerService{  
   private static final Map<Integer, Customer> *customers*;  
    
   static {  
    
   *customers* = new HashMap<>();  
   *customers*.put(1, new Customer(1, "John", "john@codegym.vn", "Hanoi"));  
   *customers*.put(2, new Customer(2, "Bill", "bill@codegym.vn", "Danang"));  
   *customers*.put(3, new Customer(3, "Alex", "alex@codegym.vn", "Saigon"));  
   *customers*.put(4, new Customer(4, "Adam", "adam@codegym.vn", "Beijin"));  
   *customers*.put(5, new Customer(5, "Sophia", "sophia@codegym.vn", "Miami"));  
   *customers*.put(6, new Customer(6, "Rose", "rose@codegym.vn", "Newyork"));  
   }  
    
   @Override  
   public List<Customer> findAll() {  
   return new ArrayList<>(*customers*.values());  
   }  
    
   @Override  
   public void save(Customer customer) {  
   *customers*.put(customer.getId(), customer);  
   }  
    
   @Override  
   public Customer findById(int id) {  
   return *customers*.get(id);  
   }  
    
   @Override  
   public void update(int id, Customer customer) {  
   *customers*.put(id, customer);  
    
   }  
    
   @Override  
   public void remove(int id) {  
   *customers*.remove(id);  
   }  
    
  }
* Tạo thư mục views trong thư mục WEB-INF => Tạo file layout.html ở trong views => Cấu trúc dự án lúc này như sau:



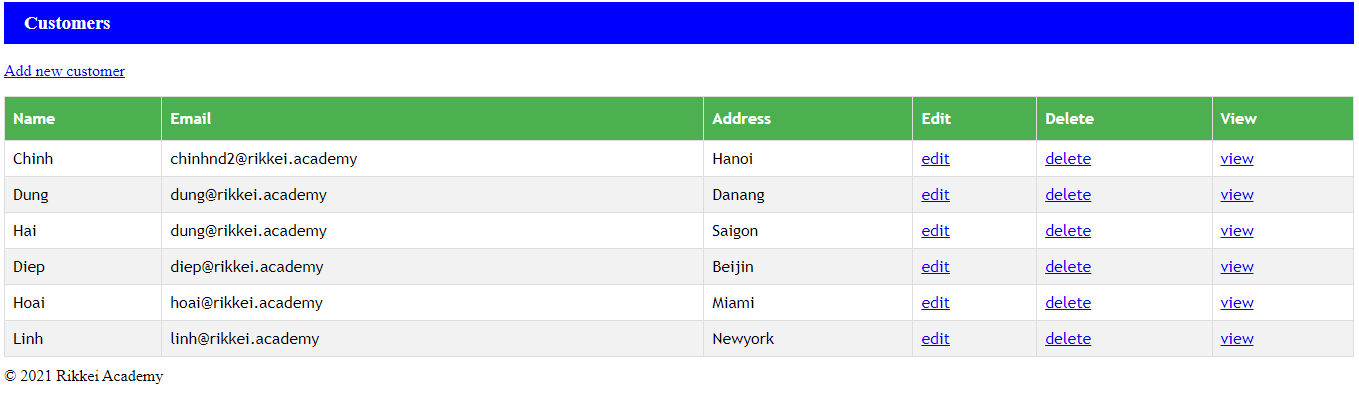
<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org">  
<head th:fragment="head">  
 <meta charset="utf-8" />  
 <meta http-equiv="X-UA-Compatible" content="IE=edge" />  
 <meta name="viewport" content="width=device-width, initial-scale=1" />  
  
 <title>Spring Customer</title>  
  
 <style type="text/css">  
 h3{background-color: blue; color: white; padding: 10px 0 10px 20px;}  
 footer{margin: 10px 0;}  
  
 #customers {  
 font-family: "Trebuchet MS", Arial, Helvetica, sans-serif;  
 border-collapse: collapse;  
 width: 100%;  
 }  
  
 #customers td, #customers th {  
 border: 1px solid #ddd;  
 padding: 8px;  
 }  
  
 #customers tr:nth-child(even){background-color: #f2f2f2;}  
  
 #customers tr:hover {background-color: #ddd;}  
  
 #customers th {  
 padding-top: 12px;  
 padding-bottom: 12px;  
 text-align: left;  
 background-color: #4CAF50;  
 color: white;  
 }  
  
 input[type=text],input[type=email]{  
 width: 100%;  
 padding: 12px 0px;  
 margin: 8px 0;  
 display: inline-block;  
 border: 1px solid #ccc;  
 border-radius: 4px;  
 box-sizing: border-box;  
 }  
  
 input[type=submit] {  
 width: 300px;  
 background-color: #4CAF50;  
 color: white;  
 padding: 14px 20px;  
 margin: 8px 0;  
 border: none;  
 border-radius: 4px;  
 cursor: pointer;  
 text-align: center;  
 }  
  
 input[type=submit]:hover {  
 background-color: #45a049;  
 }  
  
 #form {  
 width: 500px;  
 border-radius: 5px;  
 background-color: #f2f2f2;  
 padding: 20px;  
 }  
  
 .notify span{font-size: 15px; color: blue;}  
 </style>  
</head>  
<body>  
  
<footer th:fragment="footer">  
 &copy; 2022 Rikkei Academy  
</footer>  
</body>  
</html>

* Tạo file: index.html trong thư mục views và thêm code vào như sau:
* <!DOCTYPE html>  
  <html xmlns:th="http://www.thymeleaf.org">  
  <head th:replace="/layout :: head">  
  </head>  
  <body>  
  <h3>Customers</h3>  
  <p>  
   <a th:href="@{/customer/create}">  
   Add new customer  
   </a>  
  </p>  
    
  <table id = "customers">  
   <thead>  
   <tr>  
   <th>Name</th>  
   <th>Email</th>  
   <th>Address</th>  
   <th>Edit</th>  
   <th>Delete</th>  
   <th>View</th>  
   </tr>  
   </thead>  
   <tbody>  
   <tr th:each="row,rowStat : ${customers}">  
   <td th:text="${row.getName()}"></td>  
   <td th:text="${row.getEmail()}"></td>  
   <td th:text="${row.getAddress()}"></td>  
   <td><a th:href="@{/customer/{id}/edit(id=${row.getId()})}">edit</a></td>  
   <td><a th:href="@{/customer/{id}/delete(id=${row.getId()})}">delete</a></td>  
   <td><a th:href="@{/customer/{id}/view(id=${row.getId()})}">view</a></td>  
   </tr>  
   </tbody>  
  </table>  
  <footer th:replace="/layout :: footer"></footer>  
  </body>  
  </html>
* Tạo Class: CustomerController và thêm code vào như sau:

package rikkei.academy.controller;  
  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RequestMapping;  
import rikkei.academy.model.Customer;  
import rikkei.academy.service.CustomerServiceIMPL;  
import rikkei.academy.service.ICustomerService;  
  
import java.util.List;  
@Controller  
@RequestMapping(value = {"/","/customer"})  
public class CustomerController {  
 private final ICustomerService customerService = new CustomerServiceIMPL();  
 @GetMapping  
 public String index(Model model) {  
 List<Customer> customerList = customerService.findAll();  
 model.addAttribute("customers", customerList);  
 return "/index";  
 }  
  
}

Trong đó:

* Annotation @Controller giúp Spring xác định lớp hiện tại là một Controller.
* Annotation @GetMapping xác định phương thức Index sẽ đón nhận các request có HTTP method là GET và URI pattern là "/"
* Phương thức Index được truyền vào một tham số có kiểu dữ liệu là Model. Model có nhiệm vụ là truyền dữ liệu từ Controller tới View. Ở đây, chúng ta sẽ lấy ra danh sách các khách hàng thông qua customerService.findAll(). Sau đó gắn danh sách này vào Model thông qua phương thức addAttribute(). customers chính là tên biến đại diện cho danh sách mà ta sẽ dùng ở View sau này.
* Phương thức Index sẽ trả về một String, từ String này mà Spring MVC sẽ suy ra View nào sẽ nhận dữ liệu từ Controller (return "index"), vậy View sẽ nhận dữ liệu ở đây là **index.html**
* Add Tomcat, chạy trương trình và quan sát kết quả:



Trong đó:

Khai báo namespace thymeleaf thông qua thuộc tính th:

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

Gắn phần head và footer đã được khai báo trong file **layout.html**:

<head th:replace="/layout :: head"></head

<footer th:replace="/layout :: footer"></footer>

Thuộc tính **th:href**khai báo đường link trong thẻ a.

Thuộc tính **th:if** sử dụng để kiểm tra điều kiện.

Thuộc tính **th:each** tương ứng với câu lệnh for each.

Thuộc tính **th:text** dùng để đổ dữ liệu dưới dạng text vào thẻ HTML.

**${customers}** là danh sách khách hàng sẽ được lặp, chính là model customers được controller truyền lên view.

**row** đại diện cho một khách hàng tại một bước lặp cụ thể.

**rowStat** là một biến trạng thái, giúp chúng ta theo dõi vòng lặp.

**${...}** là biểu thức đánh giá các biến , biểu thức hay model.

* Thêm code vào Class: **CustomerController**
* @GetMapping("/create")  
  public String create(Model model) {  
   model.addAttribute("customer", new Customer());  
   return "/create";  
  }
* Tạo file: **create.html** dưới thư mục views và thêm code vào như sau:

<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org">  
<head th:replace="/layout :: head"></head>  
<body>  
<h3>Add Customer</h3>  
<p>  
 <a th:href="@{/customer}">  
 Back to customer list  
 </a>  
</p>  
<div id="form">  
 <form th:action="@{/customer/save}" th:object="${customer}"  
 method="POST" novalidate="novalidate">  
 <input type="hidden" th:field="\*{id}"/>  
 <div>  
 <label>Name</label>  
 <input type="text" th:field="\*{name}"/>  
 </div>  
 <div>  
 <label>Email</label>  
 <input type="email" th:field="\*{email}"/>  
 </div>  
 <div>  
 <label>Address</label>  
 <input type="text" th:field="\*{address}"/>  
 </div>  
 <input type="submit" value="Save"/>  
 </form>  
</div>  
<footer th:replace="/layout :: footer"></footer>  
</body>  
</html>

Trong đó:

* Giá trị trong **@GetMapping** ở **CustomerController** tương ứng đường link của button "Add new customer".
* Chúng ta sẽ truyền sang view create.html một model Customer có tên là customer. Mỗi thuộc tính của customer sẽ tương ứng với một input trong form.

**th:action** chỉ ra đường dẫn sẽ xử lý submit form, **${customer}** ở **th:object** chính là biến customer.

Như đã nói ở trên, mỗi thuộc tính của customer sẽ tương ứng với một input trong form, nên chúng ta sẽ thêm thuộc tính **th:field="\*{fieldname}"**vào các input. Do khi submit form chúng ta cần phải chỉ ra entity nào sẽ được gửi lên cho nên chúng ta phải thêm:

<input type="hidden" th:field="\*{id}"/>

để chỉ ra ID của entity.

* Do request gửi lên có HTTP method là POST, nên ta sẽ sử dụng @PostMapping.
* Đối tượng customer được truyền vào save() chính là đối tượng customer chúng ta đã truyền từ phương thức create() sang view create.html. Đối tượng này sẽ lưu thông tin của người dùng nhập vào.
* customer.setId() sẽ tạo Id ngẫu nhiên cho đối tượng customer.
* Sau khi lưu customer, chúng ta sẽ redirect về trang danh sách khách hàng. Chuỗi đằng sau "redirect:" là đường dẫn của trang mà mình muốn redirect. Đồng thời, chúng ta cũng sẽ gửi một Flash message về trang danh sách khách hàng để thông báo lưu thành công, bằng cách sử dụng redirect.addFlashAttribute(messageName, messageContent).
* **Hiển thị form sửa thông tin khách hàng:**

Đoạn mã sau đây trên trang **index.html** sẽ điều hướng đến trang edit.html:

<td><a th:href="@{/customer/{id}/edit(id=${row.getId()})}">edit</a></td>

Trong class CustomerController, ta viết thêm phương thức edit để hiển thị trang edit.html:

@GetMapping("/{id}/edit")  
public String edit(@PathVariable int id, Model model) {  
 model.addAttribute("customer", customerService.findById(id));  
 return "/edit";  
}

Trong đó:

* Tham số @PathVariable int id lấy id của customer từ đường dẫn rồi gán vào biến id.
* hàm customerService.findById(id) sẽ lấy customer theo id rồi truyền sang view **edit.html**

Trang **edit.html** sẽ có dạng như sau: <!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org">  
<head th:replace="/layout :: head"></head>  
<body>  
<h3>Edit Customer</h3>  
<p>  
 <a th:href="@{/customer}">  
 Back to customer list  
 </a>  
</p>  
<div id="form">  
 <form action="#" th:action="@{/customer/update}" th:object="${customer}"  
 method="POST" novalidate="novalidate">  
 <input type="hidden" th:field="\*{id}"/>  
 <div>  
 <label>Name</label>  
 <input type="text" th:field="\*{name}"/>  
 </div>  
 <div>  
 <label>Email</label>  
 <input type="email" th:field="\*{email}"/>  
 </div>  
 <div>  
 <label>Address</label>  
 <input type="text" th:field="\*{address}"/>  
 </div>  
 <input type="submit" value="Edit"/>  
 </form>  
</div>  
<footer th:replace="/layout :: footer"></footer>  
</body>  
</html>

Trong controller, ta sẽ tạo phương thức update() để lưu một sửa đổi, như sau:

@PostMapping("/update")  
public String update(Customer customer) {  
 customerService.update(customer.getId(), customer);  
 return "redirect:/customer";  
}

**12.1. Hiển thị form xóa khách hàng:**

Đoạn mã sau đây trên trang index.html sẽ điều hướng đến trang delete.html:

<td><a th:href="@{/customer/{id}/delete(id=${row.getId()})}">delete</a></td>

Trong class CustomerController, ta viết thêm phương thức  GET delete() để hiển thị trang delete.html:

@GetMapping("/{id}/delete")  
public String delete(@PathVariable int id, Model model) {  
 model.addAttribute("customer", customerService.findById(id));  
 return "/delete";  
}

Trang delete.html sẽ có dạng như sau:

<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org">  
<head th:replace="/layout :: head"></head>  
<body>  
<h3>Remove Customer</h3>  
<p>  
 <a th:href="@{/customer}">  
 Back to customer list  
 </a>  
</p>  
<div id="form">  
 <form action="#" th:action="@{/customer/delete}" th:object="${customer}"  
 method="POST" novalidate="novalidate">  
 <input type="hidden" th:field="\*{id}"/>  
 <div>  
 <label>Name</label>  
 <input type="text" th:field="\*{name}"/>  
 </div>  
 <div>  
 <label>Email</label>  
 <input type="email" th:field="\*{email}"/>  
 </div>  
 <div>  
 <label>Address</label>  
 <input type="text" th:field="\*{address}"/>  
 </div>  
 <input type="submit" value="Remove"/>  
 </form>  
</div>  
<footer th:replace="/layout :: footer"></footer>  
</body>  
</html>

Trong controller, ta sẽ tạo phương thức POST delete() để xóa khách hàng, như sau:

@PostMapping("/delete")  
public String delete(Customer customer, RedirectAttributes redirect) {  
 customerService.remove(customer.getId());  
 redirect.addFlashAttribute("success", "Removed customer successfully!");  
 return "redirect:/customer";  
}

* **View chi tiết khách hàng**

Đoạn mã sau đây trên trang index.html sẽ điều hướng đến trang view.html:

<td><a th:href="@{/customer/{id}/view(id=${row.getId()})}">view</a></td>

Trong class CustomerController, ta viết thêm phương thức  GET view() để hiển thị trang view.html:

@GetMapping("/{id}/view")  
public String view(@PathVariable int id, Model model) {  
 model.addAttribute("customer", customerService.findById(id));  
 return "/view";  
}

Trang **view.html** sẽ có dạng như sau:

<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org">  
<head th:replace="/layout :: head"></head>  
<body>  
<h3>View Customer</h3>  
<p>  
 <a th:href="@{/customer}">  
 Back to customer list  
 </a>  
</p>  
<div id="detail">  
 <div class="cusdetail">  
 <label>Name:</label>  
 <label th:text="${customer.name}"></label>  
 </div>  
 <div class="cusdetail">  
 <label>Email:</label>  
 <label th:text="${customer.email}"></label>  
 </div>  
 <div class="cusdetail">  
 <label>Address:</label>  
 <label th:text="${customer.address}"></label>  
 </div>  
</div>  
<footer th:replace="/layout :: footer"></footer>  
</body>  
</html>

* **Chạy chương trình test các chức năng:**

Các bạn chạy chương trình rồi quan sát kết quả.

Lúc này dự án của chúng ta sẽ chạy trên cổng 8080: [http://localhost:8080](http://localhost:8080/contact)

Mã nguồn tham khảo:

https://github.com/rikkei-academy/MD4-Bai3-TH-CRUD-Thyme-Leaf.git

Hướng dẫn nộp bài:

Up code lên github.

Paste link github vào phần nộp bài