Bài 12
Kiến trúc MVC

Module: BOOTCAMP WEB-BACKEND DEVELOPMENT

Kiểm tra bài trước

Hỏi và trao đổi về các khó khăn gặp phải trong bài “Tổng quan về ứng dụng WEB"

Tóm tắt lại các phần đã học từ bài “Tổng quan về ứng dụng WEB”

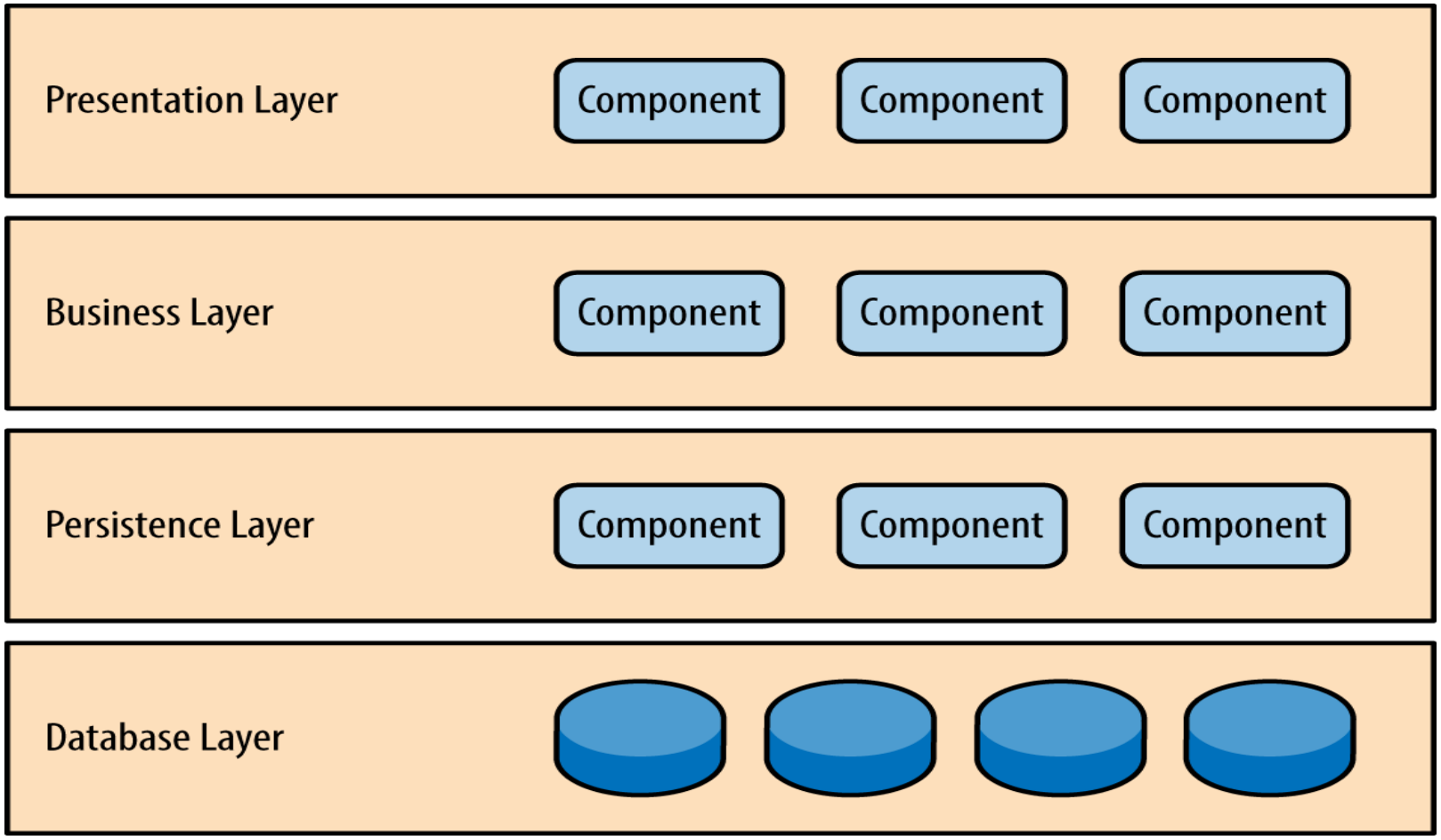
Mục tiêu

- Trình bày được kiến trúc MVC
- Trình bày được kiến trúc phân tầng
- Trình bày được các lợi ích của kiến trúc phân tầng và kiến trúc MVC
- Triển khai được các ứng dụng theo kiến trúc phân tầng
- Triển khai được các ứng dụng theo kiến trúc MVC

Kiến trúc phân tầng

- Kiến trúc phân tầng (multilayered/multitier/n-tier/architecture) là một kiến trúc phần mềm trong đó các thành phần của hệ thống được tổ chức theo các tầng theo chiều ngang, mỗi tầng thực hiện một nhóm nhiệm vụ cụ thể
- Chẳng hạn: Tầng giao diện, tầng điều khiển, tầng dịch vụ, tầng lưu trữ dữ liệu…
- Dạng kiến trúc n-tier phổ biến là 3 tầng (three-tier)

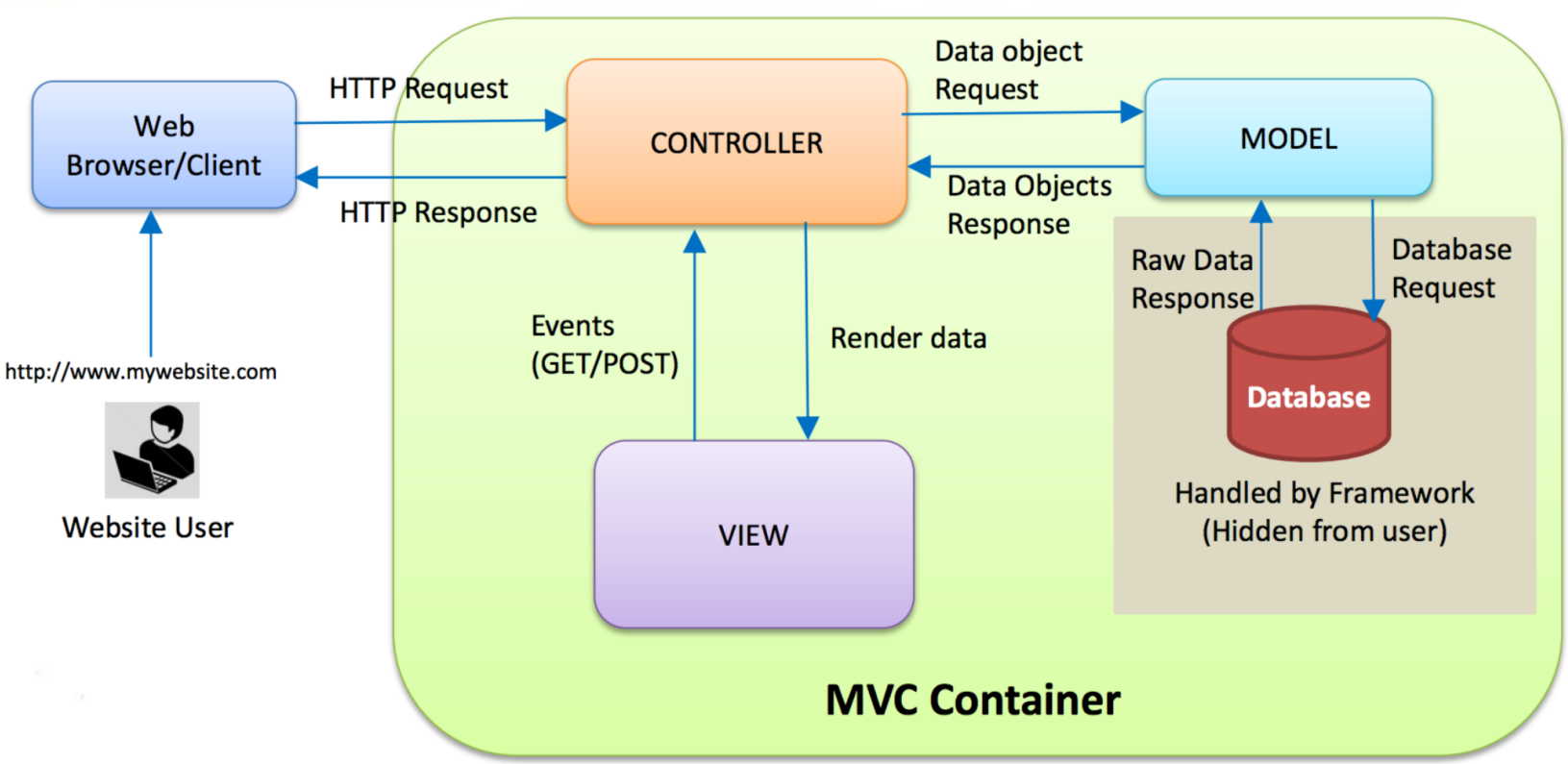
Kiến trúc phân tầng: Ví dụ



Kiến trúc MVC

- MVC là viết tắt của Model-View-Controller
- Kiến trúc MVC tổ chức các thành phần của hệ thống vào 3 tầng riêng biệt và có kết nối đến nhau:
  - Tầng Model: Biểu diễn dữ liệu và các logic nghiệp vụ
  - Tầng View: Hiển thị dữ liệu và là giao diện tương tác với người dùng
  - Tầng Controller: Xử lý các thao tác từ người dùng, kết nối giữa Model và View

MVC là gì


Web
Browser/Client
HTTP Request
HTTP Response
CONTROLLER
Data object
Request
Data Objects
Response
MODEL
Raw Data
Response
Database
Request
Database
Handled by Framework
(Hidden from user)
Events
(GET/POST)
Render data
VIEW
http://www.mywebsite.com
Website User
MVC Container

Lợi ích của kiến trúc phân tầng

- Tách các trọng tâm của từng thành phần trong ứng dụng (separation of concerns)
  - Các thành phần trong một tầng thì chỉ thực hiện các nhiệm vụ liên quan đến tầng đó
- Dễ phát triển
- Dễ kiểm thử
- Dễ quản lý
- Dễ bảo trì

Lợi ích của kiến trúc MVC

- Dễ tái sử dụng
- Dễ mở rộng
- Tách phần view và phần nghiệp vụ riêng biệt
- Cho phép các Lập trình viên làm việc trên các thành phần khác nhau trong cùng một thời điểm
- Dễ bảo trì

DEMO MVC

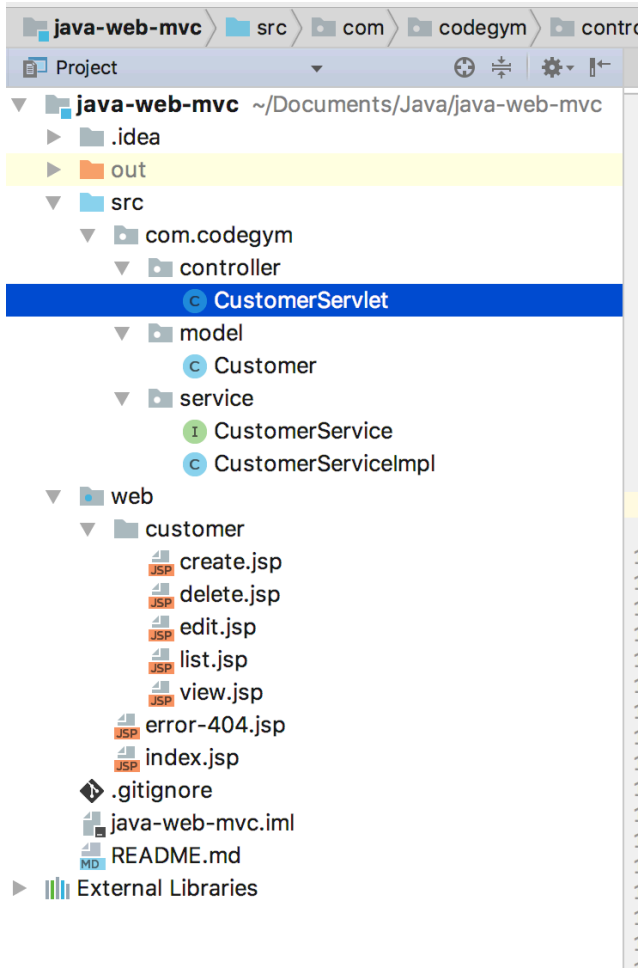
Ví dụ demo Model-View-Controller

Bài toán quản lý danh sách khách hàng

- Phát triển một ứng dụng quản lý khách hàng. Ứng dụng có các chức năng chính sau:
  - Hiển thị danh sách khách hàng
  - Thêm một khách hàng mới
  - Sửa đổi thông tin của khách hàng
  - Xoá một khách hàng
  - Xem thông tin chi tiết của khách hàng

Bước 1: Tạo dự án mới với tên java-web-mvc

- Cấu trúc của các thư mục và file của ứng dụng sau khi hoàn thành sẽ như sau:



Bước 2: Tạo Customer model

- Lớp com.codegym.model.Customer bao gồm các thuộc tính:
  - id: Id của khách hàng
  - name: Tên của khách hàng
  - email: Email của khách hàng
  - address: Địa chỉ của khách hàng

```
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;
    }
}
```

Bước 3: Tạo interface CustomerService

- Interface CustomerService là interface định nghĩa các phương thức ở tầng Service để thao tác với đối tượng Customer.
  - Interface CustomerService bao gồm các phương thức:
  - findAll(): Trả về danh sách tất cả khách hàng
  - save(): Lưu một khách hàng
  - findById(): Tìm một khách hàng theo Id
  - update(): Cập nhật thông tin của một khách hàng
  - remove(): Xoá một khách hàng khỏi danh sách

Bước 3: Tạo interface CustomerService

```
package com.codegym.service;

import com.codegym.model.Customer;

import java.util.List;

public interface CustomerService {
    List<Customer> findAll();

    void save(Customer customer);

    Customer findById(int id);

    void update(int id, Customer customer);

    void remove(int id);
}
```

Bước 4: Tạo lớp CustomerServiceImpl

- CustomerServiceImpl là một lớp triển khai đầy đủ các phương thức của interface CustomerService.
- Trong trường hợp này, chúng ta giả lập lớp CustomerServiceImpl làm đối tượng cung cấp dữ liệu. Danh sách khách hàng sẽ được lưu vào trong một đối tượng Map. Trong một ứng dụng thực tế, danh sách khách hàng sẽ được lưu vào cơ sở dữ liệu. Chúng ta sẽ phát triển tính năng lưu trữ vào CSDL ở trong các phần sau.

Bước 4: Tạo lớp CustomerServiceImpl

```
public class CustomerServiceImpl implements CustomerService{

    private static 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);
    }
}
```

Bước 5: Tạo CustomerServlet để xử lý các request.

- Lớp CustomerServlet có một đối tượng CustomerServiceImpl dùng để truy xuất đến dữ liệu.

```
@WebServlet(name = "CustomerServlet", urlPatterns = "/customers")
public class CustomerServlet extends HttpServlet {

    private CustomerService customerService = new CustomerServiceImpl();

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

    }
}
```

Bước 6: Điều hướng việc xử lý các tính năng

- Điều hướng việc xử lý các tính năng khác nhau của CustomerServlet thông qua tham số *action*.
- Tham số action được sử dụng để quy định hành động mà CustomerServlet xử lý trong từng request:
  - /customers?action=create: Tạo một khách hàng mới
  - /customers?action=edit: Cập nhật thông tin khách hàng
  - /customers?action=delete: Xoá một khách hàng
  - Các giá trị khác: Hiển thị thông tin khách hàng

Bước 6: Điều hướng việc xử lý các tính năng

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    String action = request.getParameter("action");
    if(action == null){
        action = "";
    }
    switch (action){
        case "create":
            break;
        case "edit":
            break;
        case "delete":
            break;
        case "view":
            break;
        default:
            break;
    }
}
```

Bước 7: Phát triển tính năng hiển thị danh sách khách hàng

- Điều hướng *action* hiển thị danh sách khách hàng về phương thức listCustomers(request, response);

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    String action = request.getParameter("action");
    if(action == null){
        action = "";
    }
    switch (action){
        case "create":
            break;
        case "edit":
            break;
        case "delete":
            break;
        case "view":
            break;
        default:
            listCustomers(request, response);
            break;
    }
}
```

Bước 7: Phát triển tính năng hiển thị danh sách khách hàng

- Phương thức *listCustomers(request, response)* nhận về danh sách khách hàng và chuyển sang view *customer/list.jsp* để hiển thị.

```
private void listCustomers(HttpServletRequest request, HttpServletResponse response) {
    List<Customer> customers = this.customerService.findAll();
    request.setAttribute("customers", customers);

    RequestDispatcher dispatcher = request.getRequestDispatcher("customer/list.jsp");
    try {
        dispatcher.forward(request, response);
    } catch (ServletException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Bước 7: Phát triển tính năng hiển thị danh sách khách hàng

- Tạo file customer/list.jsp để hiển thị danh sách khách hàng.

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Customer List</title>
</head>
<body>
    <h1>Customers</h1>
    <p>
        <a href="/customers?action=create">Create new customer</a>
    </p>
    <table border="1">
        <tr>
            <td>Name</td>
            <td>Email</td>
            <td>Address</td>
            <td>Edit</td>
            <td>Delete</td>
        </tr>
        <c:forEach items='${requestScope["customers"]}' var="customer">
            <tr>
                <td><a href="/customers?action=view&id=${customer.getId()}">${customer.getName()}</a></td>
                <td>${customer.getEmail()}</td>
                <td>${customer.getAddress()}</td>
                <td><a href="/customers?action=edit&id=${customer.getId()}">edit</a></td>
                <td><a href="/customers?action=delete&id=${customer.getId()}">delete</a></td>
            </tr>
        </c:forEach>
    </table>
</body>
</html>
```

Bước 8: Khởi chạy ứng dụng

- Khởi chạy ứng dụng và đi đến đường dẫn http://localhost:8080/customers để quan sát kết quả

Tóm tắt bài học

- Kiến trúc phân tầng (multilayered/multitier/n-tier/architecture) là một kiến trúc phần mềm trong đó các thành phần của hệ thống được tổ chức theo các tầng theo chiều ngang, mỗi tầng thực hiện một nhóm nhiệm vụ cụ thể
- Kiến trúc MVC tổ chức các thành phần của hệ thống vào 3 tầng riêng biệt và có kết nối đến nhau: Tầng Model, Tầng View và Tầng Controller.
- Lợi ích MVC: Dễ tái sử dụng, Dễ mở rộng, Tách phần view và phần nghiệp vụ riêng biệt, Cho phép các Lập trình viên làm việc trên các thành phần khác nhau trong cùng một thời điểm, Dễ bảo trì.

Hướng dẫn

Hướng dẫn làm bài thực hành và bài tập

Chuẩn bị bài tiếp theo: *Tổng quan về Spring MVC*